

CAP6671 Intelligent Systems: Robots, Agents, and Humans

Lecture 1: Introduction

Instructor: Dr. Gita Sukthankar
Email: gitars@eeecs.ucf.edu

Outline

- Administrative details
- What are intelligent systems?
- Course outline
- Web site:
 - Webcourses: CAP6671
 - <http://www.eecs.ucf.edu/~gitars/cap6671/>
- Complete assignment: opinion on singularity
- Reading:
 - Katia Sycara, “Multiagent Systems”, AI Magazine 19(2): Summer 1998, 79-92.
 - Rod Brooks, “I, Rod Brooks, Am a Robot” IEEE Spectrum
 - Jennings et al., “Human-Agent Collectives” CACM

Administrative Details

- Phone: 407-823-4305 (email preferred)
- Email: gitars@eeecs.ucf.edu
- Office hours over Zoom: T 3:30-5pm and Th 5:00-6:30pm
- Website:
 - Currently being mirrored on an external website and webcourses
 - <http://www.eecs.ucf.edu/~gitars/cap6671/>
- No in person attendance required
 - Class presentation can be done over videoconference
 - Exam is delivered as an open book Webcourses quiz.
 - No exam proctor required for virtual sections

Lectures

- Slides available on webcourses site under Lectures
- Videos available through Panopto
 - Will be posted immediately after class
 - Please let me know if there are technical problems with the videos!
- Your feedback is important and welcome

Prerequisites

- CAP5610: Machine Learning (or comparable experience in AI/machine learning/robotics)
- Take look at the reinforcement learning papers on last year's website to gauge whether you have sufficient experience to take the course
- Programming experience in Python helpful for the reinforcement learning agent section of the course
- Will occasionally recommend that you view background videos from the Coursera/Udacity AI and machine learning courses

Evaluation

- Homework: 30%
- Class presentation: 10%
 - Can be done in class, over videoconference during office hours or at a separately scheduled time
- Final exam: 30%
- Final project: 30%

Class Presentation

- Select a paper written for an AI conference the last 3 years (between 6-15 pages) (get my approval)
- Create a PowerPoint presentation
- Do 15 minute class presentation
- Grade will be based on:
 - Your understanding of the paper
 - The quality of your presentation slides
 - Clarity of spoken presentation
- Presentations will be done either during class time or over videoconference by appointment
- Presentations will start in Mar

Assignments

- Opinion about the singularity (due Friday)
- Homework 1: Agent competition literature review (Feb)
- Homework 2: Program a reinforcement learning agent (Mar)

Note: this course has very little work at the beginning and a lot of work at the end so please remember this when scheduling your time.

Final Project

- Topic options:
 - Extend one of the homework assignments
 - Link project to your current research
 - Implement system described in research paper
- Writeup in conference format

Final Exam

- To be completed as an online timed webcourses quiz
- Open book, no collaboration
- Based solely on readings and lectures
- Mostly short-answer questions
- Mainly a mechanism to make sure that you get the most out of the readings

Assignment Policies

- Must abide by UCF's code on academic integrity
- All external sources (code, URLs, papers) must be correctly cited in your work.
- You will be able to request one week extension on the homework.
- Final project can be completed by teams of students; other work must be completed individually.
- Submissions through webcourses
- Final projects must be submitted by the cut off date at the end of the semester.

How to Read/Write/Review a Paper

- Read the title and the abstract carefully
- Look up unfamiliar terms in the paper
- Where was it published?
- What sources does it reference?
- What testbed/software does it use?
- What are the inputs/outputs of any new algorithms that the author is describing?
- What fundamental claim is the author making?
- How do they evaluate the claim?
- What is the novelty/technical contribution?

Introductions

- Introduce yourself in person or over email.

My Research

- Direct the Intelligent Agents Lab at UCF

<http://ial.eecs.ucf.edu/projects.php>

Background

- What is an intelligent system?

Background

- What is an intelligent system?
 - Webster's definition of intelligence:
 - a (1): the ability to **learn** or understand or to **deal with new or trying situations** : the skilled use of reason (2): the ability to apply **knowledge to manipulate one's environment** or to **think abstractly** as measured by objective criteria b: mental acuteness
 - AI: “study and design of intelligent agents”
 - Agent: “system that perceives its environment and takes actions which maximize its chances of success”
(Webster, Wikipedia, Russell and Norvig)

Background

- What are some examples of intelligent systems?

RAP Systems

Examples of intelligent systems:

- **Robots**
 - Physically-embodied agent
 - Characterized by sense-cognition-action loop
- **Agents**
 - Software system that inhabits an information environment and has a sense-cognition-action loop
- **People (Animals)**
 - Biological agents
- Also “bots”
 - Agent that inhabits a simulation of a physical-environment

AI Intelligent Systems



Autonomous Vehicles



IBM Watson/ChatGPT



Apple Siri/OK Google



Companion NPCs

Background

- What are the desiderata of an intelligent systems?

Background

- Some commonly mentioned desiderata:
 - Deliberative
 - Reactive
 - Capable of learning from past experiences
 - Capable of adapting to new situations
 - Social: capable of communicating/cooperating with other agents
 - Creative: capable of finding new solutions to specified problems
 - Rational: maximizes chance of success, avoids risk, consistent preference model

Background

- How do robotic systems differ from other types of intelligent systems?

Background

- How do robotic systems differ from other types of intelligent systems?
 - Physically-embodied
 - Must be capable of reasoning about uncertainty
 - Specialized capabilities for vision/localization
 - Difficult for robots to have as complete and detailed a world model as a virtual agent

Background

- What are computational techniques that we can use to create intelligent behavior?

Background

- What are computational techniques that we can use to create intelligent behavior?
 - Search
 - A*, dynamic programming, UCT, many others
 - Optimization
 - Numerical methods for finding optima in high-dimensional state spaces (e.g. linear/quadratic programming)
 - Machine Learning
 - Reinforcement learning
 - Inference (process of drawing conclusions and changing beliefs)
 - Bayesian filters
 - Monte Carlo localization

3 Different Approaches

- Classical AI (Knowledge Representation + Search)
 - Represent the problem
 - Search for solution
- Machine learning (Error Metrics + Optimization)
 - Use data to guide search
 - Minimize penalty function
 - Use statistical techniques to characterize the data sets
- Robotics (Control Systems, State Estimation)
 - Satisficing search---reasonable solution quickly!
 - Robust control—make stability guarantees

Background

- What are metrics that can be used to evaluate the performance of intelligent systems?

Background

- What are metrics that can be used to evaluate the performance of intelligent systems?
 - Tend to be domain-specific
 - Accuracy of producing “the right answer”
 - Metrics often used by the machine learning community: classification accuracy, confusion matrices, precision/recall
 - Producing an answer that matches subject matter experts
 - Speed/ability to handle problems with larger state spaces
 - Benchmarking against other existing techniques and showing statistically significant gains
 - User studies

Background

- If a program can use machine learning to learn what to do, does it make the programmer's life easier? (i.e., does it require fewer lines of code including libraries and less debugging effort)

Background

- If a program can use machine learning to learn what to do, does it make the programmer's life easier? (i.e., does it require fewer lines of code including libraries and less debugging effort)
 - Generally, adding AI and machine learning to systems makes them more complicated and adds many lines of code (not necessarily written by you) to the system requirements.
 - However, the hope is that you end up with a more flexible, capable, and reusable system.

Background

- Does artificial intelligence have to mimic human intelligence?

Background

- Does artificial intelligence have to mimic human intelligence?
 - Thinking humanly
 - “automation of activities that we associate with human thinking, activities such as decision-making, problem-solving, learning” (Bellman)
 - Acting humanly
 - “art of creating machines that perform functions that require intelligence when performed by people” (Kurzweil)
 - Thinking rationally
 - “study of computations that make it possible to perceive, reason, and act” (Winston)
 - Acting rationally
 - “the study of the design of intelligent agents” (Poole)

Background

- Will AI ever reach the point of singularity?
 - The singularity is supposed to begin after engineers build the first computer with greater than human intelligence.
 - The first machine will trigger a series of cycles in which that smart machine will rapidly beget other smarter machines

Background

- Will AI ever reach the point of singularity?
 - Rod Brooks (MIT) lays out these options:
 - AI scenario: we create super human AI in computers
 - Intelligence amplification: enhance human intelligence through human-computer interfaces
 - Biomedical: improve the neurological operation of our brains
 - Internet Scenario: network of humans, computers, and robots become sufficiently effective to be regarded as a superhuman being

Background

- What are limitations of current AI systems?

Background

- What are limitations of current AI systems?
 - ``Intelligence'' is typically confined to one aspect of the agent---the rest of the agent is often relatively simple or non-existent
 - Incapable of handling large state spaces
 - Designed to live in sandboxes---only operate within a limited set of conditions
 - Cannot transfer existing learning to new situations
 - Brittle: cannot recover gracefully from errors
 - Opaque/non self-reflective: cannot communicate problems to users, difficult to debug

Course Reading

- Overview articles from AI magazine
- Research papers from a variety of conferences and journals:
 - AAMAS: Autonomous Agents and Multi-agent Systems
 - IVA: Intelligent Virtual Agents
 - AAAI/IJCAI: main AI conferences
 - AIIDE: AI in Digital Entertainment and Games
 - IROS/ICRA: robotics conferences
 - UAI: Uncertainty in AI
 - ICAPS: Automated Planning and Scheduling
 - AGI: Artificial General Intelligence
 - ICML: Machine Learning
 - SBP: Social Behavior and Prediction

Course Outline

- Agents (AI challenge problems)
- Robotics
- Modeling human intelligence/ethics
- Algorithms for creating intelligent behavior
(interspersed throughout the course)

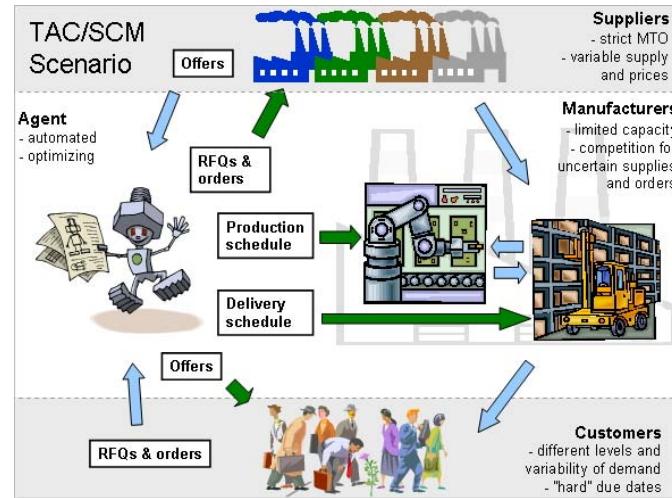
Course Outline

- Planning: search-based methods for deciding on a course of action
 - Hierarchical task network planning
 - Markov Decision Processes
- Learning: creating an agent policy based on a set of exemplars
 - Reinforcement learning
 - Transfer learning
- Coordination mechanisms
 - Multi-agent planning
 - Auction-based mechanisms

AI Challenge Problems



Starcraft Competition



Trading Agent Competition



Urban Rescue Robots



Robocup Soccer

New Domain



New challenge problem: AI assisting human teams playing Minecraft

Course Outline

Robotic Systems

- System architectures
- Path planning
 - Refinements on A*
 - High-dimensional search strategies (RRT)
- Localization
 - Probabilistic inference methods for determining exact position based on uncertain measurements
 - Particle filters (Monte Carlo localization)
 - SLAM: Simultaneous Localization and Mapping

Course Outline

Modeling Humans

- Virtual humans
- Agent-based social simulations
- Human-agent and human-robot interfaces
- Ethics

Reminder...

- Reading:
 - Katia Sycara, [Multiagent Systems](#), AI Magazine 19(2): Summer 1998, 79-92
 - Rod Brooks, “I, Rod Brooks, Am a Robot” IEEE Spectrum
 - Jennings et al., “Human-Agent Collectives” CACM
- Complete assignment: Opinion about the Singularity

CAP6671 Intelligent Systems: Robots, Agents, and Humans

Lecture 2: Multi-agent Systems

Instructor: Dr. Gita Sukthankar
Email: gitars@eecs.ucf.edu

Announcements

- Reading:
 - D. Nau, Current Trends in Automated Planning, AI Magazine
 - Kelly et al., Offline Planning with HTNs in Video Games, AIIDE 2008

More on Singularity

- Best book on topic: N. Bostrom,
Superintelligence

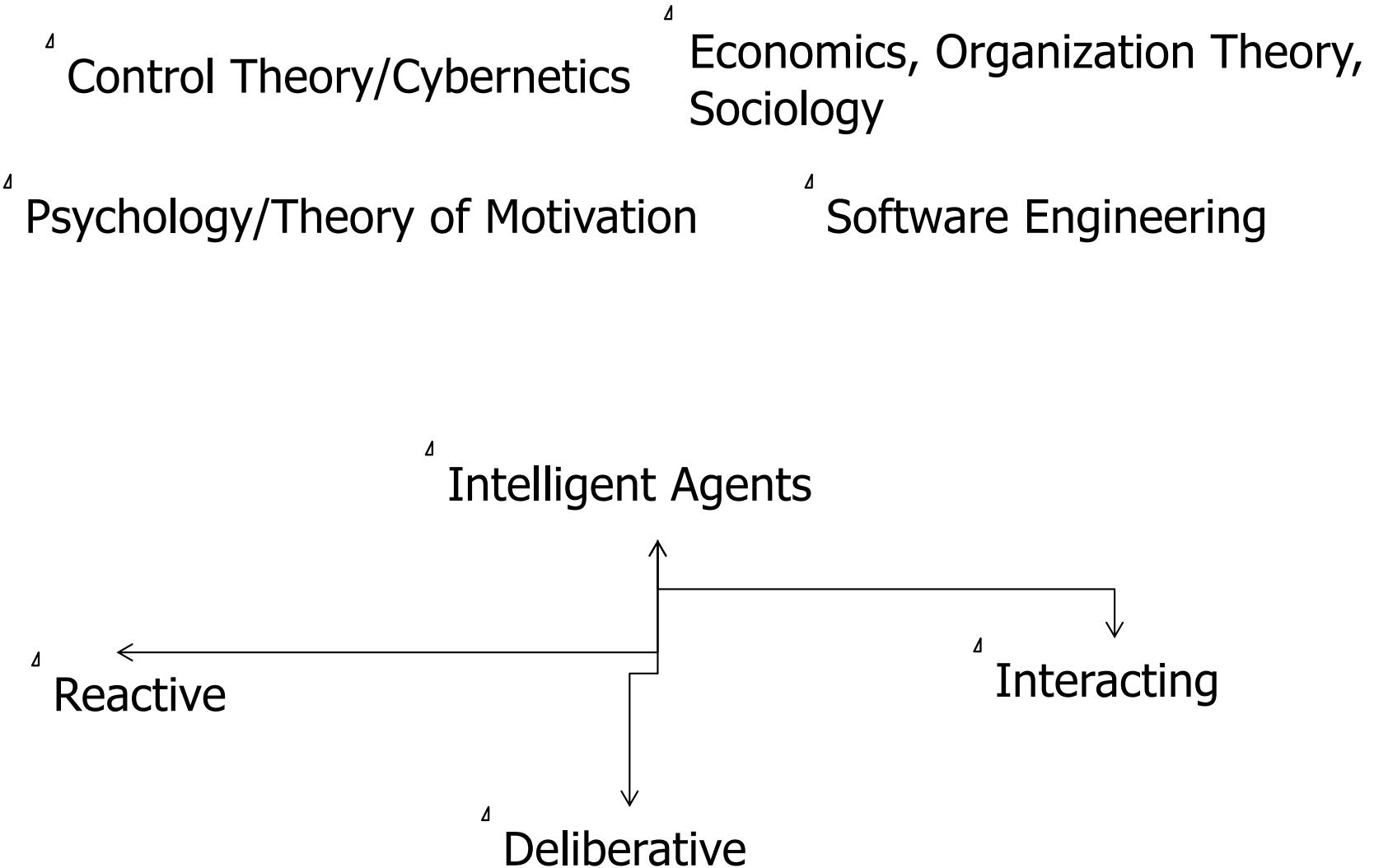
What happens when machines surpass human intelligence?

<https://youtu.be/pywF6ZzsghI>

Overview

- Purpose of this paper:
 - Introduce research problems that we are going to discuss in this course
 - Ground everyone in the same terminology
 - Outline components of a multi-agent system
 - Case study of multi-agent systems
 - How to build a multi-agent system

Intellectual Heritage



Agent Characteristics (Sycara)

An agent is a computational entity that acts on behalf of a human, organization or other agent. An agent's key characteristics are:

Situatedness: agent receives sensory inputs from its environment and outputs actions (physical environments, Internet)

Autonomy: agent is goal-directed and has control over its internal state (can reason and act without direct intervention of others)

Interactivity/Sociability: agents interact in peer to peer fashion with humans and/or other agents in pursuit of their goals (individual or collective)

Adaptivity: agents respond reactively or proactively to changes in their environment

What is a multi-agent system?

- MAS: “loosely coupled network of problem solvers that interact to solve problems that are beyond the individual capabilities or knowledge of each problem solver”
- More generally, often the systems are RAP systems (Robots, Agents, and People) in which agents can refer to any of the above entities.

BDI Agent

- Belief-Desire-Intention (Bratman, 1987)
- Describe internal processing state of an agent by a set of mental categories
- **Belief:** current state of the world
- **Desire:** preferences over future world states or courses of action
- **Goals:** consistent subset of desires that an agent might pursue
- **Intention:** selection of goals that the agent has a **committed** stance towards
- **Plans:** mechanism for achieving intentions
- BDI logics include techniques to logically reason about these concepts.

Agent Architectures

- Deliberative (e.g. STRIPS (Fikes)):
 - Attempt to attain a set of goals
 - Planner is a tool for determining the optimal course of action for goal achievement.
- Reactive (e.g. subsumption (Brooks)):
 - Respond to the current environment using a stimulus/response paradigm
 - Little or no internal representation of state
 - Agents are individually dumb and “intelligence” is an emergent property of the way that the system is connected
- Hybrid architectures (e.g. 3T (Bonasso)):
 - Agents contain deliberative and reactive components
 - RETSINA is an example of this type of architecture
- Cognitive architectures (SOAR, ACT-R)
 - Based on human models of memory, thought, information processing

Anatomy of an Agent

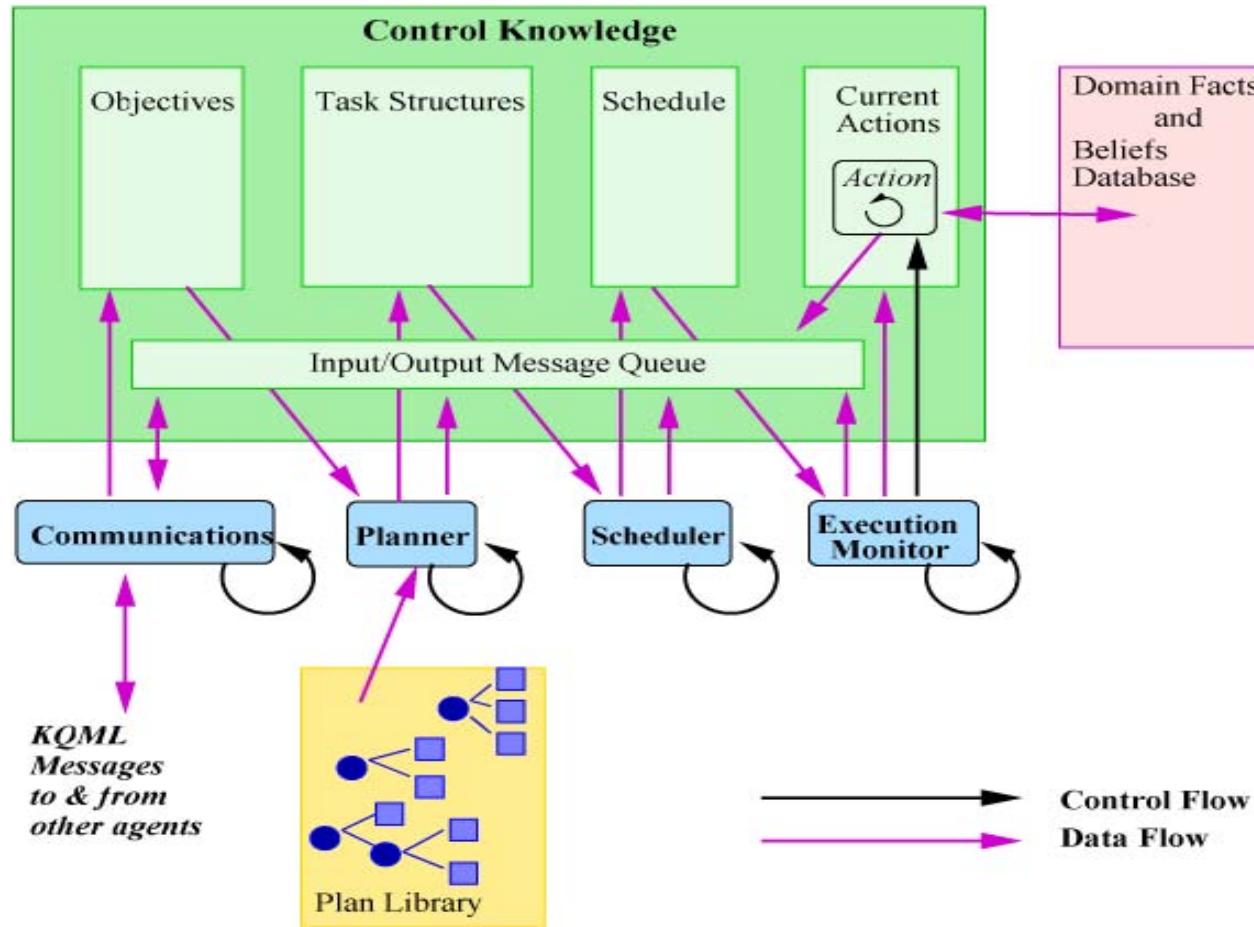
- What should an agent contain?

Anatomy of an Agent

- Sense-cognition-action loop
- Communication protocol/language
- Representation of world state
- Deliberative component/planner
- Reactive behaviors (more important for robots)

Different agent research groups spend more or less effort on the various components (some might be entirely omitted depending on the application).

RETSINA Agent



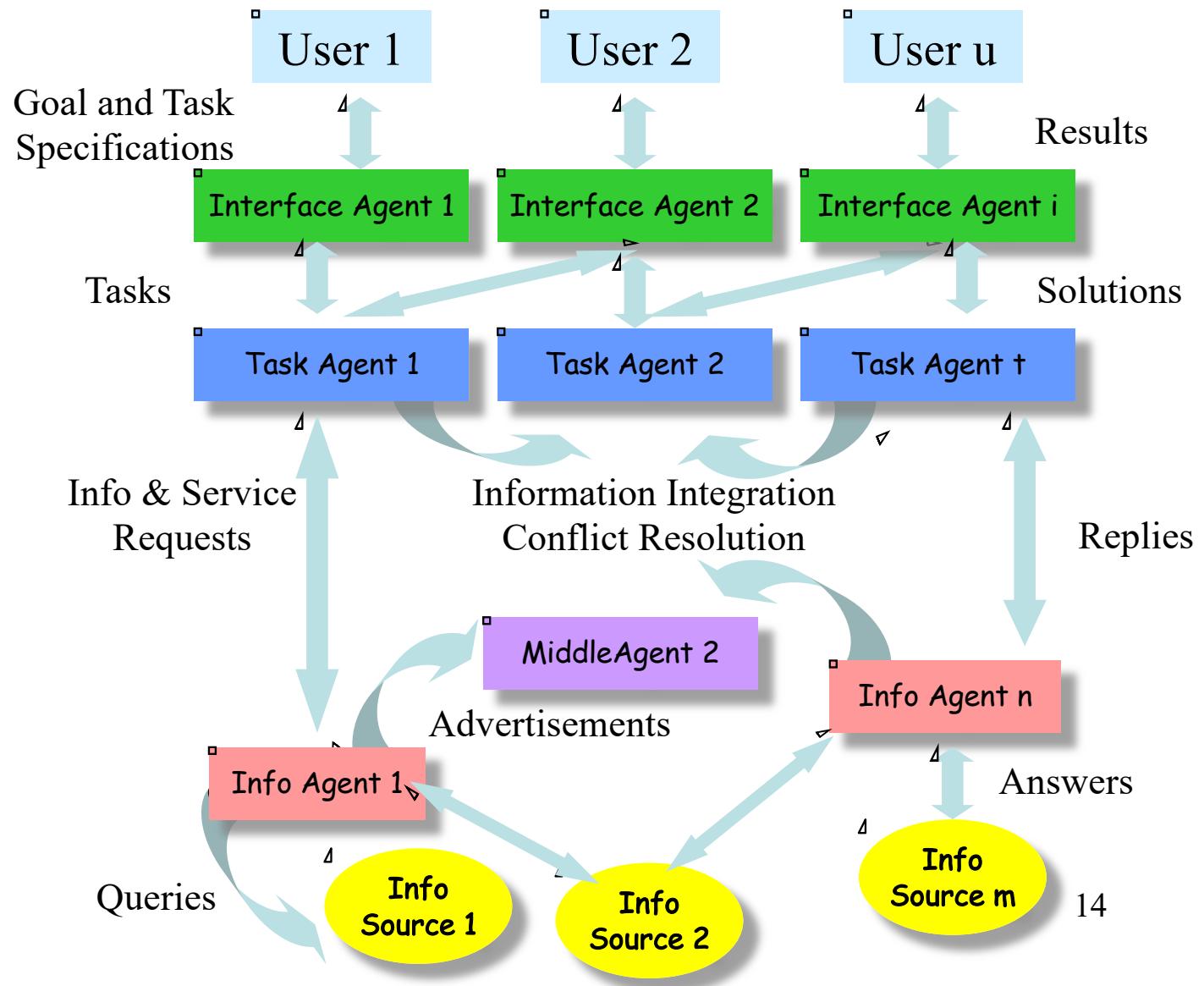
KQML: knowledge query machine language

RETSINA Architecture

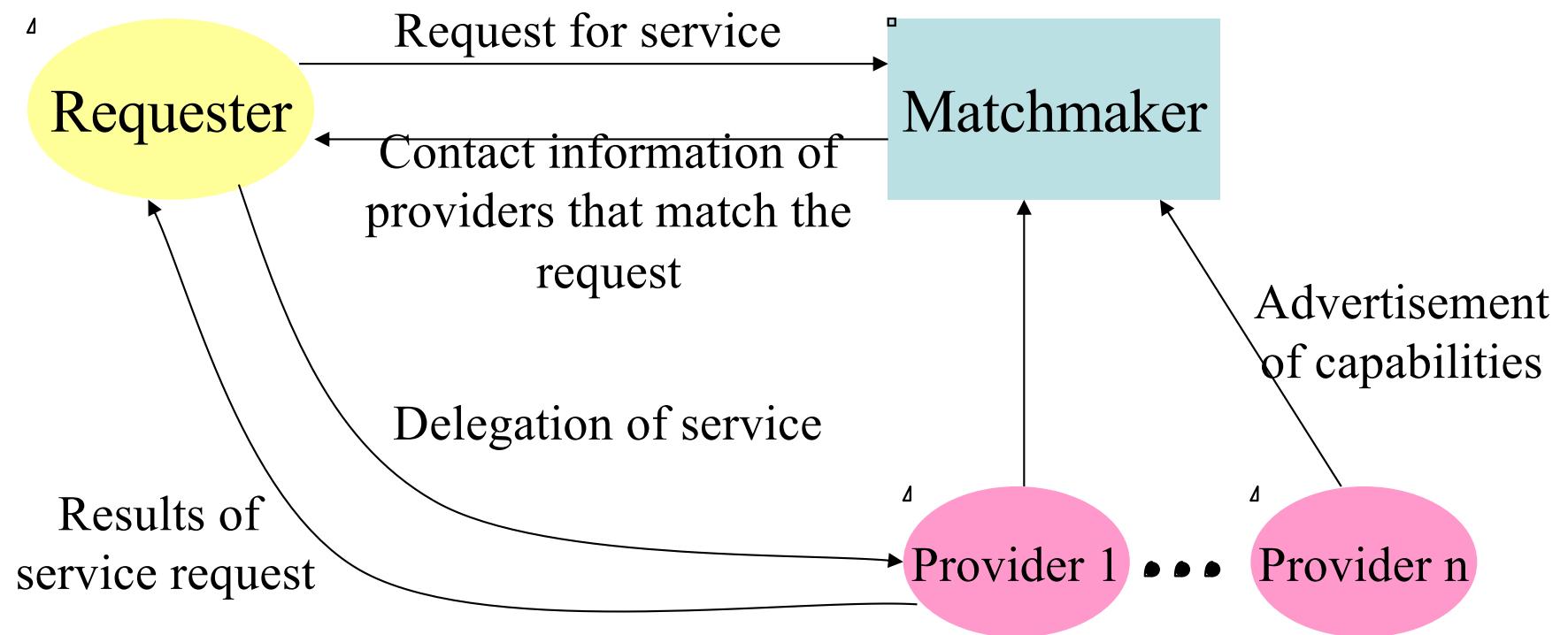
- Reusable Task-Structured Intelligent Networked Agent Architecture
- Originally implemented in Perl and converted to Java
- Developed around 1995
- Categories of agents:
 - Interface
 - Information
 - Task
 - Middle

Multi-Agent Organization

distributed adaptive collections of agents that help users by retrieving information, providing advice and anticipate user's information needs.



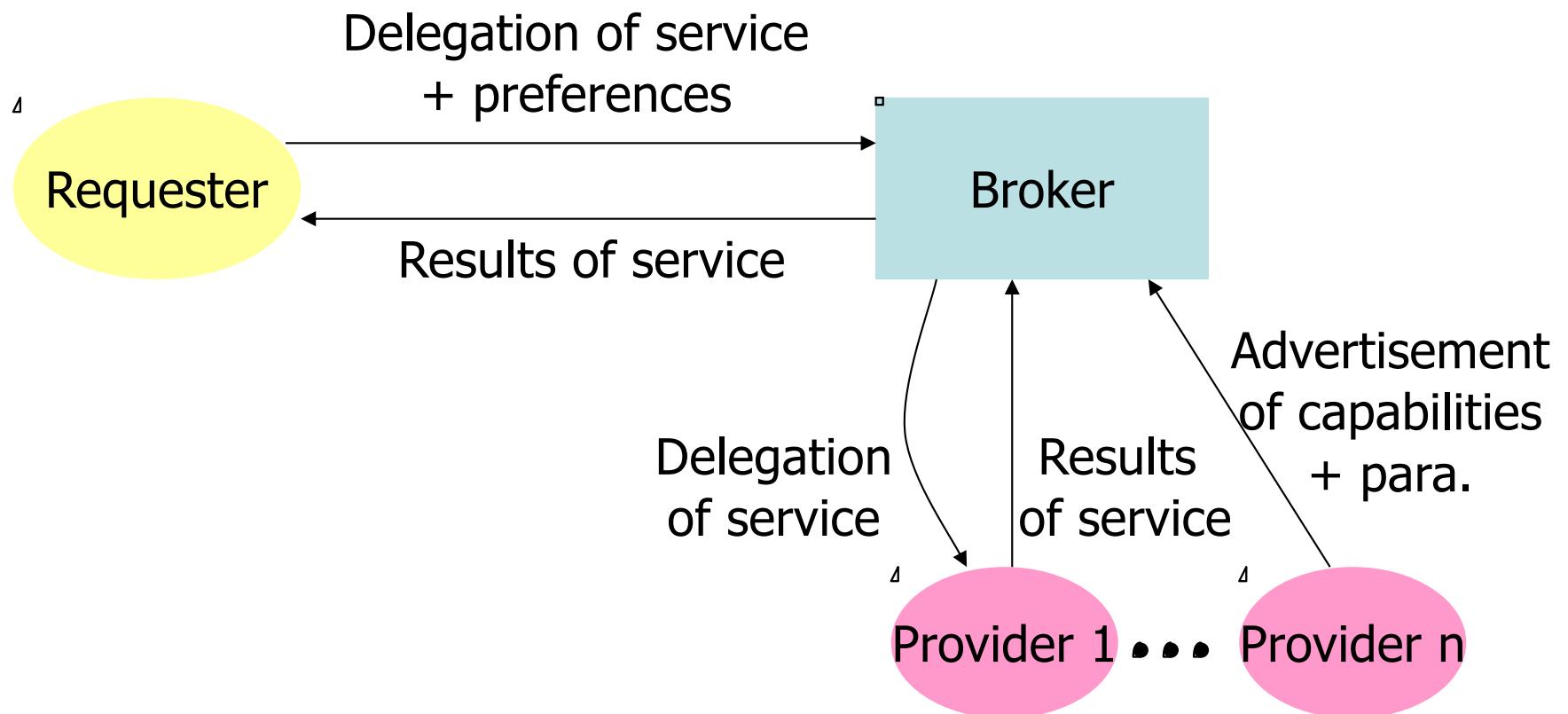
Matchmaker (Middle Agent)



Yellow-page system

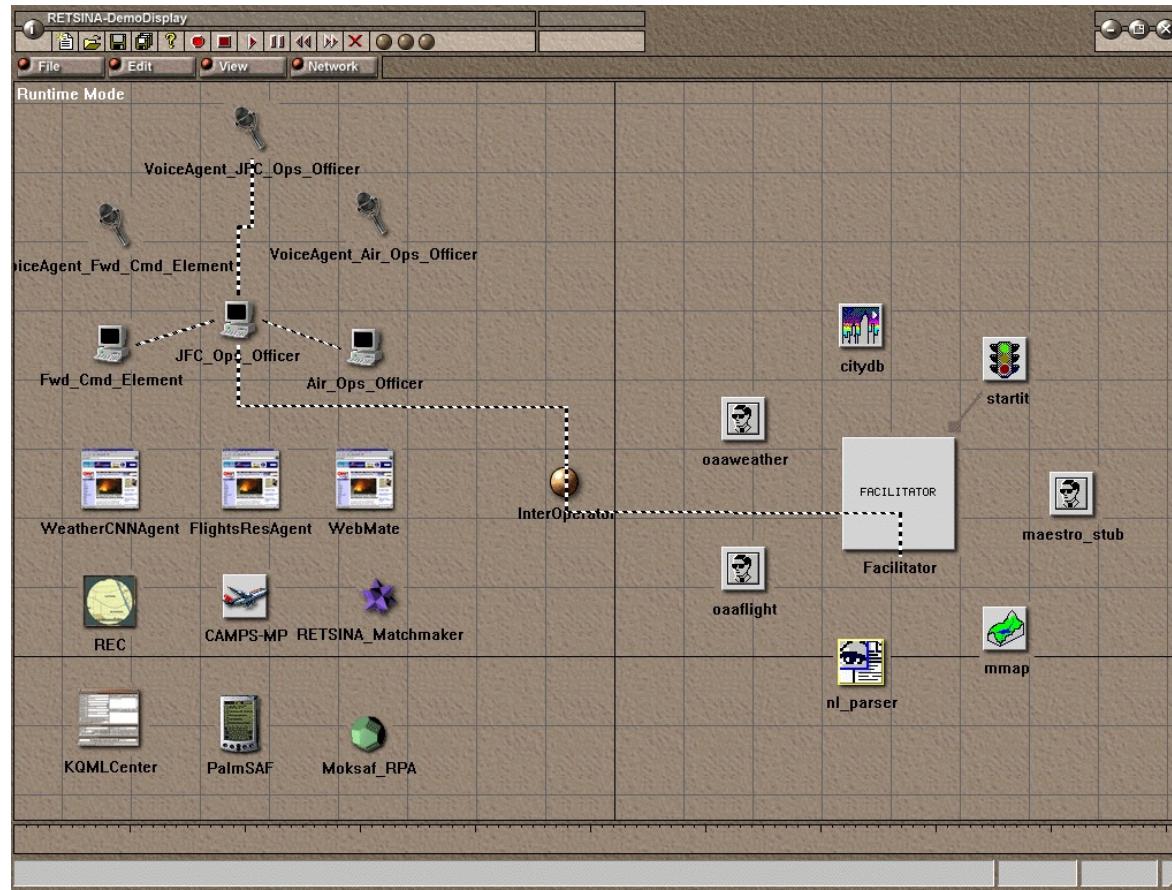
Specialized matchmaker language: LARKS

Broker (Middle Agent)



Visualization Agent

- Way of visualizing the state of a multi-agent system
- Useful for demos and debugging



Characteristics of an MAS

- Incomplete information
- No global control
- Data is decentralized
- Computation is asynchronous

MAS Coordination

How are interactions between different agents structured?

MAS Coordination

Teamwork: Agents share a common set of goals and each contributes to the fulfillment of these goals through teamwork. No explicit modeling of individual agent utility

Coalitions: Agents seek to maximize individual utility and group utility (coalition stability is an issue)

Coordination: Agents pursue their individual goals and utilities; coordination with others is done only to avoid harmful interactions (e.g. traffic)

Negotiation: Agents seek to maximize their individual utility but are willing to compromise (i.e. better off if they reach agreement than not)

Game Theoretic Interactions: Agents seek to maximize individual utility while taking into consideration other's options

Adversarial Interactions/Zero Sum Games: Agents seek to maximize own utility while minimizing utility of opponent

Why use an MAS?

Why use an MAS?

- Solve problems too huge for one agent
- Interoperation of existing legacy systems
- Preserve privacy of user data
- Use spatially separated information sources
- Distributed expertise

Some RETSINA Applications

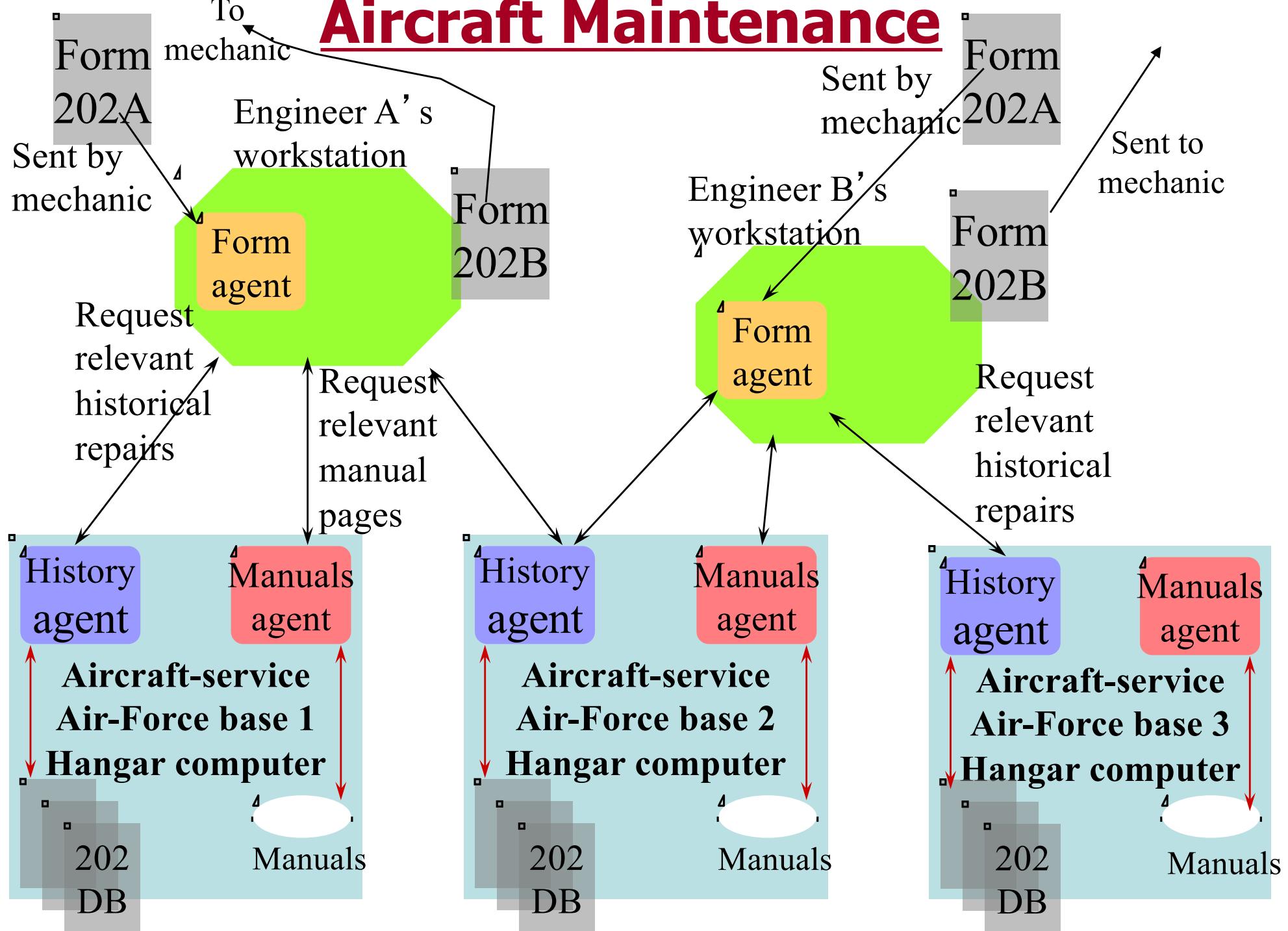
- Agent-based financial portfolio management
- Aiding Human Teams in team planning
- Agent-aided aircraft maintenance
- E-commerce in wholesale markets (agent-based auctions and negotiation)
- Agent-based Supply Chain Management
- Robot teams for Search and Rescue
- Team Rescue Scenario (NEO)
- Agent-based “on the move” collaboration on mobile devices

Sycara, K., Paolucci, M., Van Velsen, M. and Giampapa, J. “The RETSINA Multiagent Infrastructure”, *Journal of Autonomous Agents and Multiagent Systems*, vol. 7, nos. 1-2, July/September, 2003.

Agent-based Aircraft Maintenance

- Shehory, Sukthankar, and Sycara 1998
- Why use an MAS?
 - User interface agents needed to be lightweight and run on either a tablet computer or desktop
 - Utilize legacy systems that the military already had in place

Aircraft Maintenance



Agent-based Programming

- Does it improve system performance?

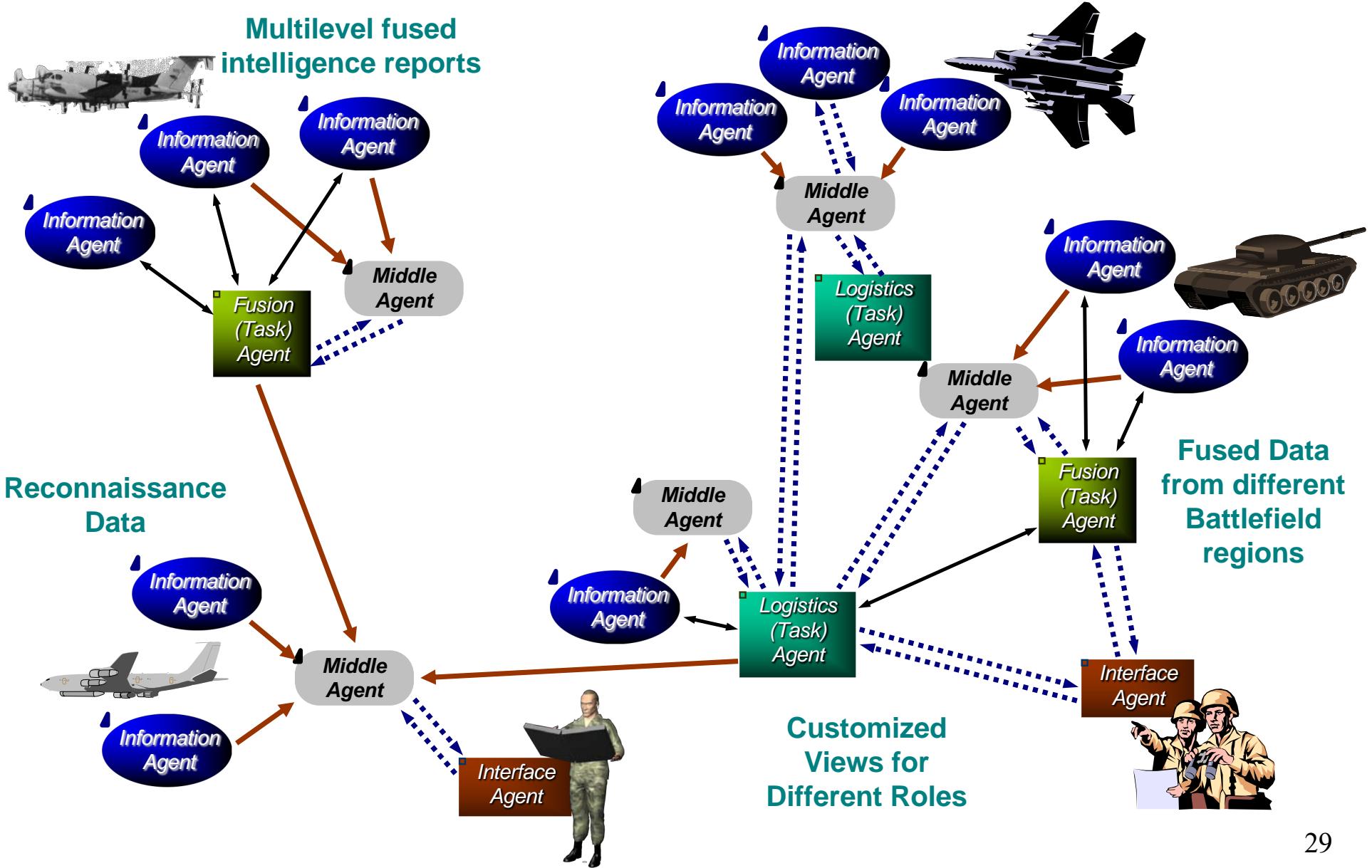
Claims

- Computational efficiency
- Reliability
- Extensibility
- Robustness
- Maintainability
- Responsiveness
- Flexibility
- Reuse
- *Does agent-based programming really offer all those advantages?*

Examples of MAS Applications

- Sensor networks
- Smart buildings
- Shopping bots/financial agents
- Information-gathering
- Manufacturing
- Personal information management agents
- Transportation management
 - Air traffic control
- Virtual humans/embodied conversational agents

Information Fusion with MAS



TIE Video Demo

<http://youtu.be/OvtoKueGO84>

Building an MAS

- Consider your application---do your agents have to interact with a set of users, a simulation environment, robots?
- Computational demands: how many agents will be operating simultaneously? Will they all be on the same machine or network?
- Where is the intelligence---in the agent or the coordination mechanism?
- Design a sense-cognition-action loop for a single agent
- Consider various coordination mechanisms
- Pick an appropriate programming language/agent toolkit
 - Features to look for: support for communication, threading, debugging tools, agent launching

Research Areas

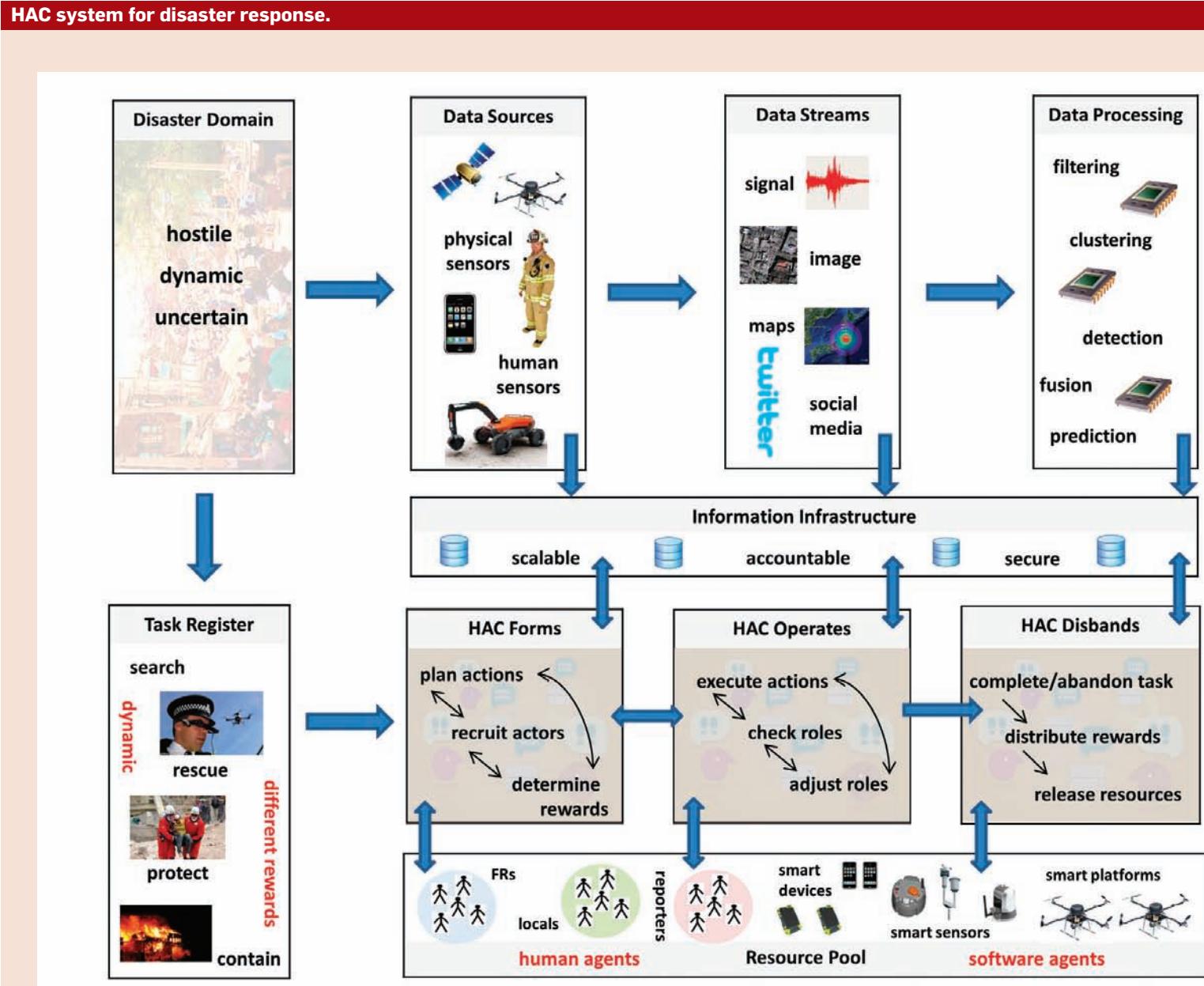
- Task allocation
 - Contract net protocol, auction-based frameworks
- Resource/communication limitations
 - Distributed constraint optimization
- Resolving sub-goal interactions in multi-agent planning
 - Decentralized MDP, team planning
- Obtaining resources (getting what you want)
 - Auctions, negotiation, coalitions
- Modeling other agents
 - Plan recognition, game theory
- Adaptation and learning
 - Multi-agent RL

What remains unsolved?

- When this article was written:
 - Lack of systematic methodology
 - Few industrial-strength MAS toolkits
 - More emphasis on logical vs. probabilistic inference
- Now:
 - Larger-scale systems
 - *Human-agent interfaces*
 - Global interoperability
 - Dynamic planning/problem-solving
 - Lifelong learning

Human-Agent Collectives

HAC system for disaster response.



Human-Agent Collectives

- Jennings et al. 2014
- Symbiotic interleaving of humans and computers performing tasks together
 - “constellations of people, resources, and information must come together, operate in a coordinated fashion, and then disband.”
- Similar to the Sycara concept but with more emphasis on human contribution
- Example: crowdsourcing, citizen science, and participatory sensing frameworks

Crowdsourcing



- Break large task into many small tasks that can be completed by a “crowd” of individuals
- Computer is responsible for assessing quality and combining output across people
- Used to label large datasets
- Amazon Mechanical Turk

References

- Jorg P. Muller, “The Design of Intelligent Agents: A Layered Approach”
- K. Sycara, MAS School Presentation

CAP6671 Intelligent Systems: Robots, Agents, and Humans

Lecture 3: Planning

Instructor: Dr. Gita Sukthankar
Email: gitars@eecs.ucf.edu

AI Writing Code

This AI Could Bring Us Computers That Can Write Their Own Software

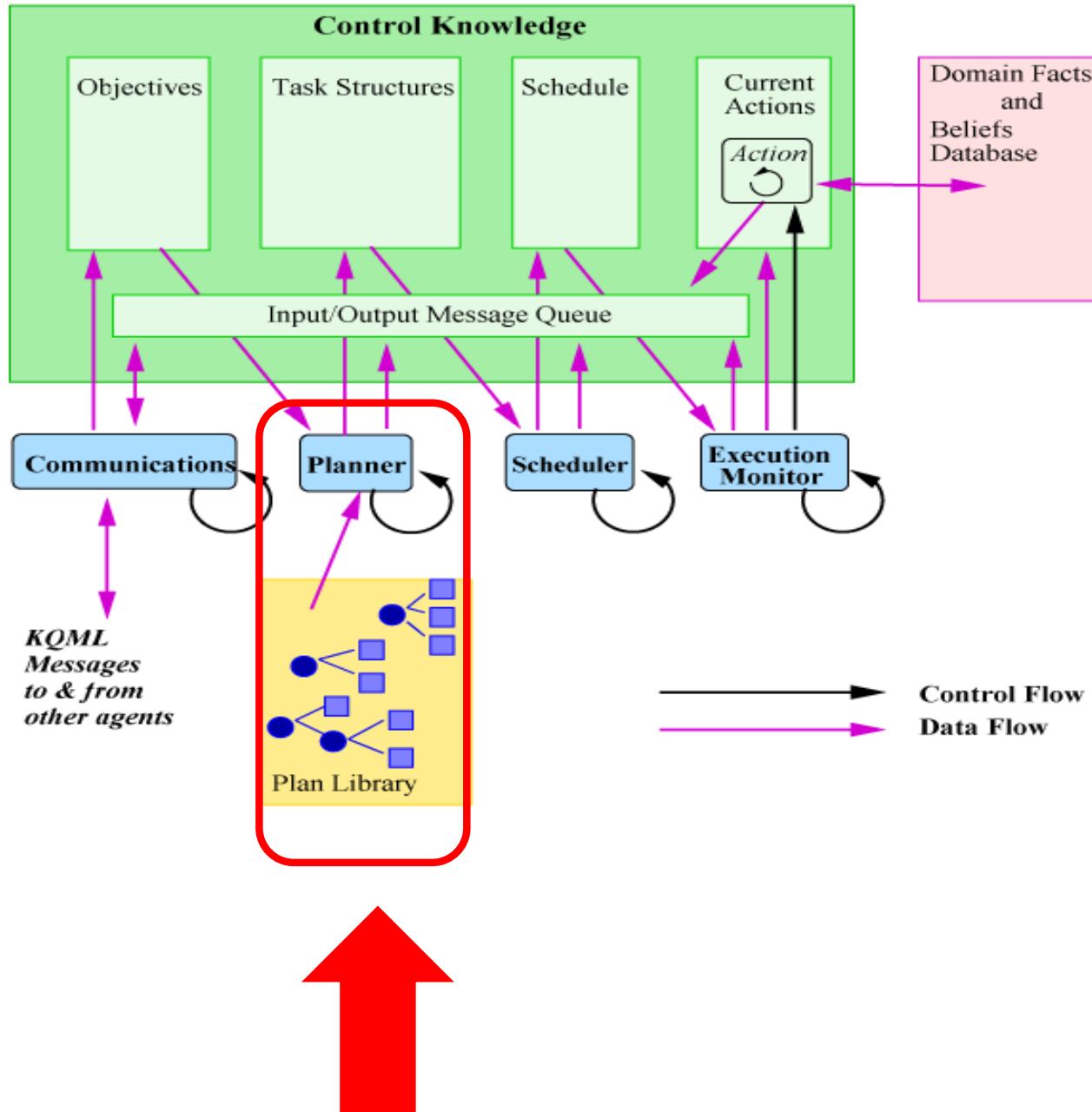
- GPT-3 can kind of write code when trained on GitHub
- Machine Inferred Code Similarity: identify what code is supposed to do and find similar pieces of code (recommender system)

[https://singularityhub.com/2020/08/02/
this-ai-could-bring-us-computers-that-can-write-software/](https://singularityhub.com/2020/08/02>this-ai-could-bring-us-computers-that-can-write-software/)

Announcements

- Reading:
 - A. Blum and M. Furst, Fast planning through graph plan analysis, AI 90: 281-300
 - K. Lavers and G. Sukthankar, A Monte Carlo Approach for Football Play Generation, AIIDE 2010

Intro to Planning



Planning Reading

- Nau paper: overview of the state of the art on planning
- Koenig paper: interesting practical applications on HTN planning
- Blum paper: speeding up planning through reachability analysis
- Our paper: Monte Carlo search in adversarial domains

[a representation] of future behavior ... usually a set of actions, with temporal and other constraints on them, for execution by some agent or agents. - Austin Tate

[MIT Encyclopedia of the Cognitive Sciences, 1999]

00 B E€ 000	048	01 Setup	02 Sp ead p o tress tfr m 1800 RM pine
00 C E€ 000	200	01 Setup	02 Ph o llog a b y o f p b r eis us ng p b t oo n "realis"
00 D E€ 000	000	01 Setup	02 Et big o f c ope
00 T E€ 000	577	01 Total time on E€	
00 A M€ 000	457	01 Setup	02 Pre a eboard fo eldina
00 B N		A portion of a manufacturing process plan	02 Sceen in ts fo dr stoon board

Generating Plans of Action

- Computer programs to aid human planners
 - Project management (consumer software)
 - Plan storage and retrieval
 - e.g., ***variant process planning*** in manufacturing
 - Automatic schedule generation
 - various OR and AI techniques
- For some problems, we would like generate plans (or pieces of plans) automatically
 - Much more difficult
 - Automated-planning research is starting to pay off

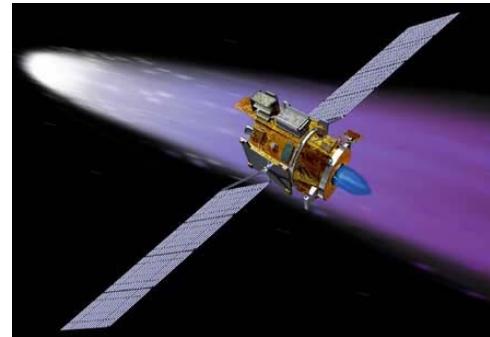


```
PICKUP:
08 AEW 800 100 01000000000000000000000000000000
08 AVMC 200 000 02000000000000000000000000000000
08 BVMC 010 043 01000000000000000000000000000000
08 CVMC 010 027 01000000000000000000000000000000
08 DVMC 010 027 01000000000000000000000000000000
08 EVMC 220 120 01000000000000000000000000000000
08 AVMC 200 000 01000000000000000000000000000000
08 BVMC 010 043 01000000000000000000000000000000
08 CVMC 010 027 01000000000000000000000000000000
08 DVMC 010 027 01000000000000000000000000000000
08 EVMC 250 427 01000000000000000000000000000000
08 A EC 800 229 01000000000000000000000000000000
08 B EC 800 048 01000000000000000000000000000000
08 C EC 800 200 01000000000000000000000000000000
08 D EC 800 800 01000000000000000000000000000000
08 T EC 800 427 01000000000000000000000000000000
08 A MC 800 457 01000000000000000000000000000000
08 B MC 800 029 01000000000000000000000000000000
08 C MC 800 750 01000000000000000000000000000000
08 D MC 800 571 01000000000000000000000000000000
08 E MC 800 427 01000000000000000000000000000000
[...]
08 G TE 800 809 01000000000000000000000000000000
08 H TC 800 467 01000000000000000000000000000000
08 I 1970 4837 01000000000000000000000000000000
```

What are planners useful for?

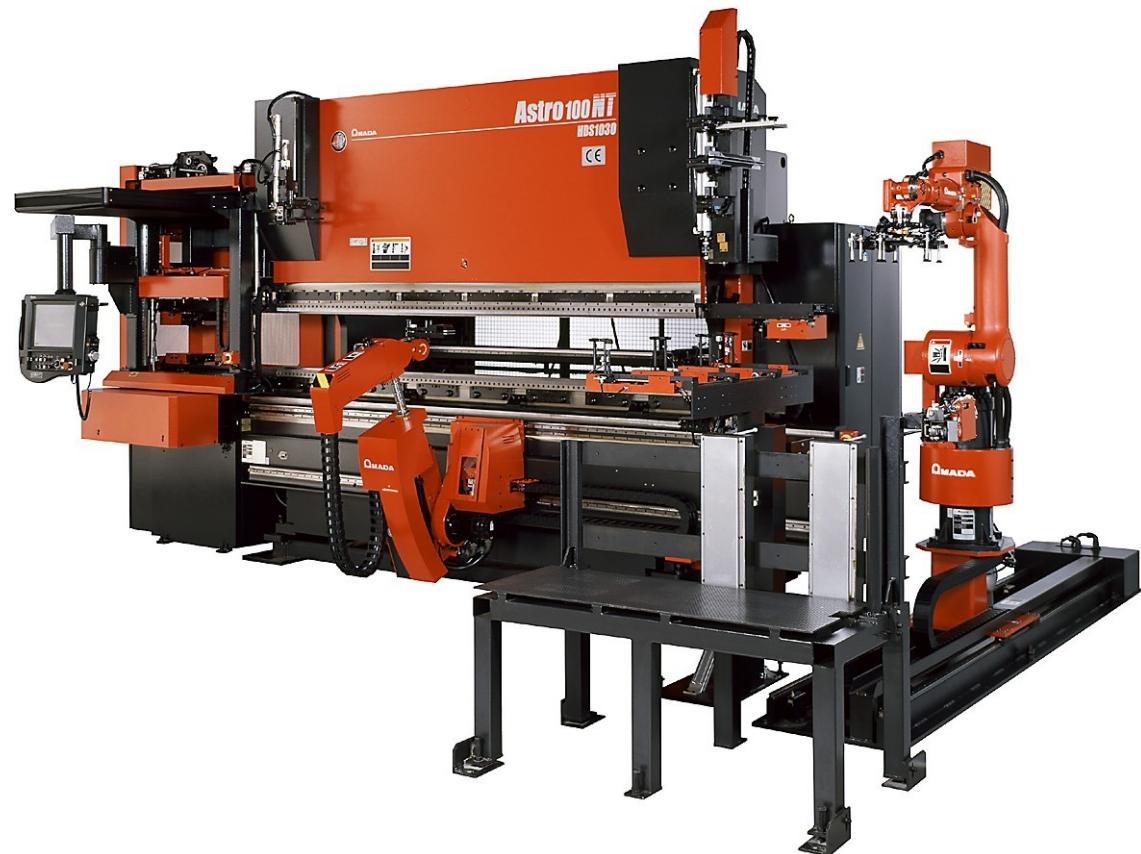
Space Exploration

- Autonomous planning, scheduling, control
 - NASA: JPL and Ames
- Remote Agent Experiment (RAX)
 - Deep Space 1
- Mars Exploration Rover (MER)



Manufacturing

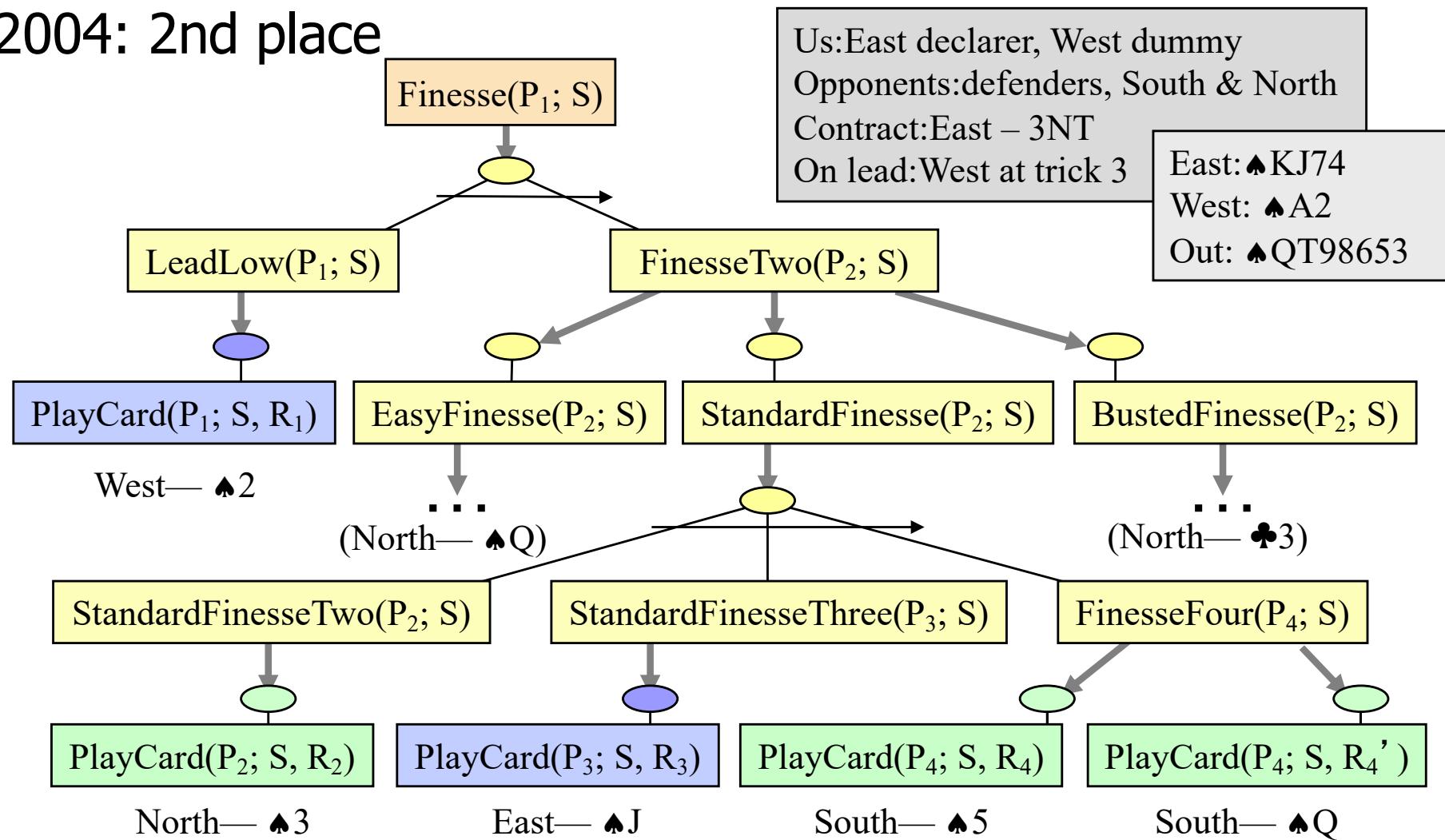
- Sheet-metal bending machines - Amada Corporation
 - Software to plan the sequence of bends
[Gupta and Bourne, *J. Manufacturing Sci. and Engr.*, 1999]



Games

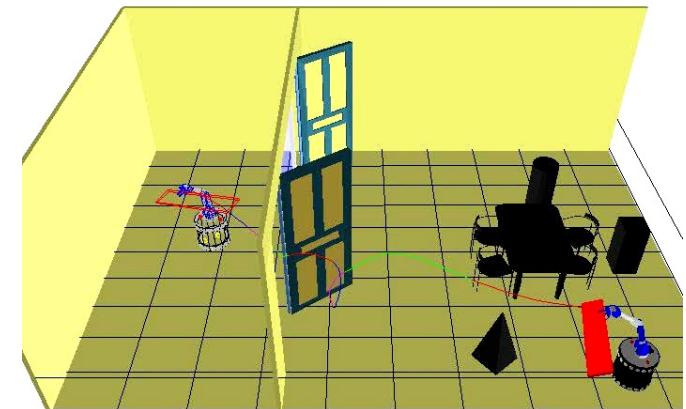
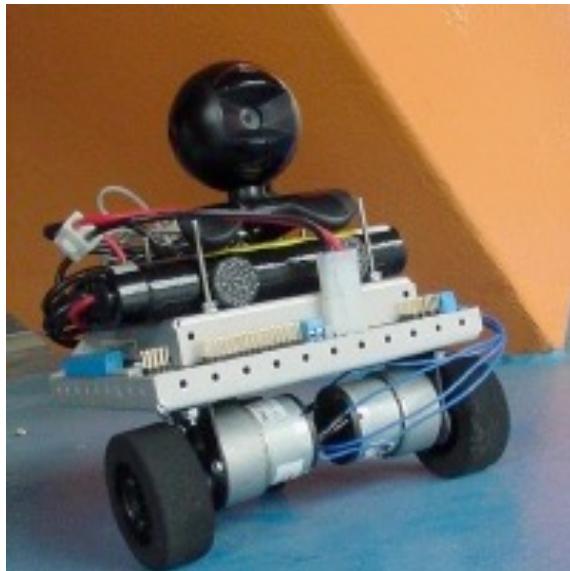
- *Bridge Baron* - Great Game Products

- 1997 world champion of computer bridge
[Smith, Nau, and Throop, *AI Magazine*, 1998]
- 2004: 2nd place



Planning in Robotics

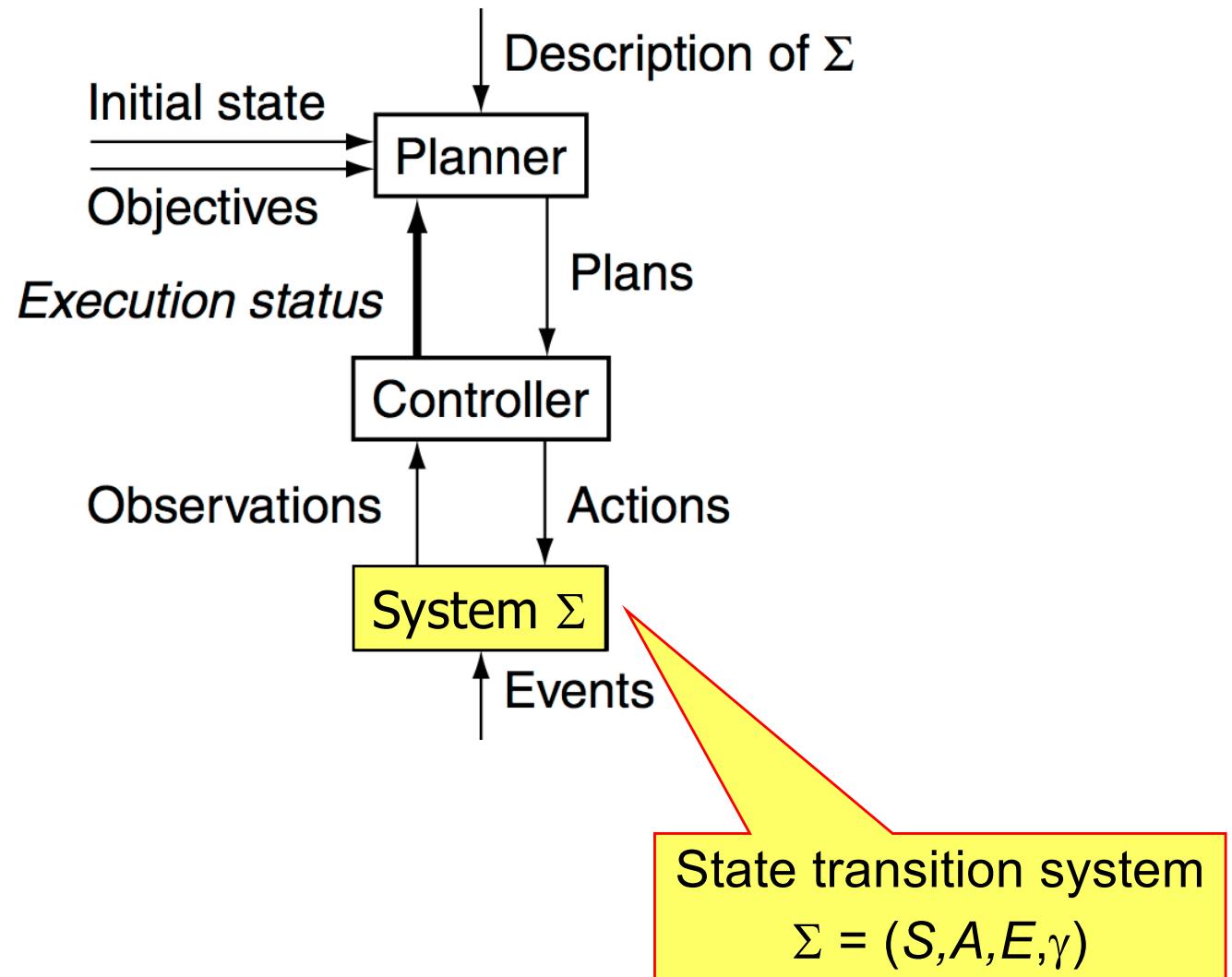
- Path planning
- Step planning (for humanoid robots)
- Kinodynamic planning (arms, legs)
- Mission planning



https://youtu.be/b_INOZ5NyMk

Conceptual Model

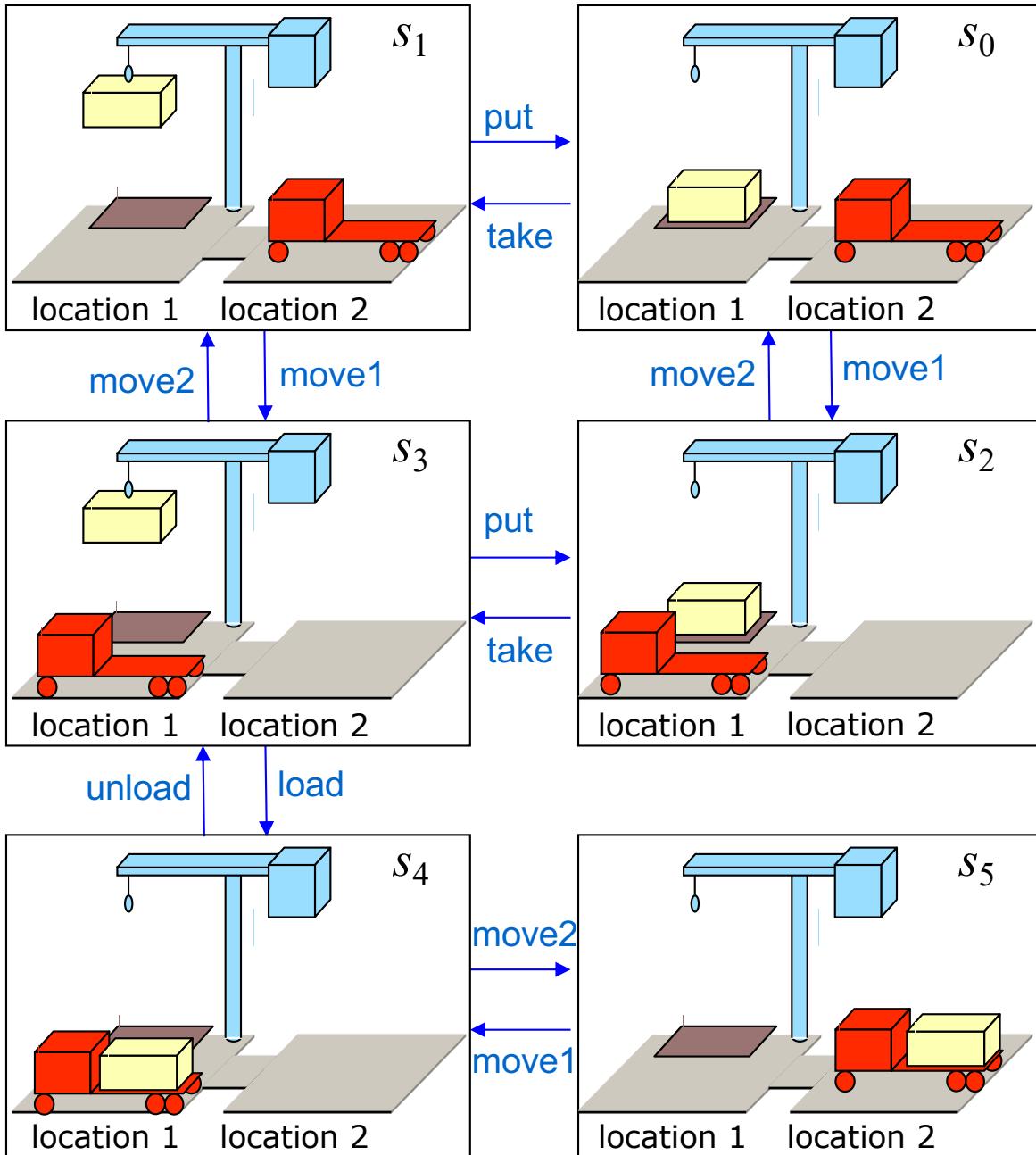
1. Environment



State Transition System

$$\Sigma = (S, A, E, \gamma)$$

- $S = \{\text{states}\}$
- $A = \{\text{actions}\}$
- $E = \{\text{exogenous events}\}$
- State-transition function
 $\gamma: S \times (A \cup E) \rightarrow S$
 - ◆ $S = \{s_0, \dots, s_5\}$
 - ◆ $A = \{\text{move1, move2, put, take, load, unload}\}$
 - ◆ $E = \{\}$
 - ◆ γ : see the arrows



The Dock Worker Robots (DWR) domain

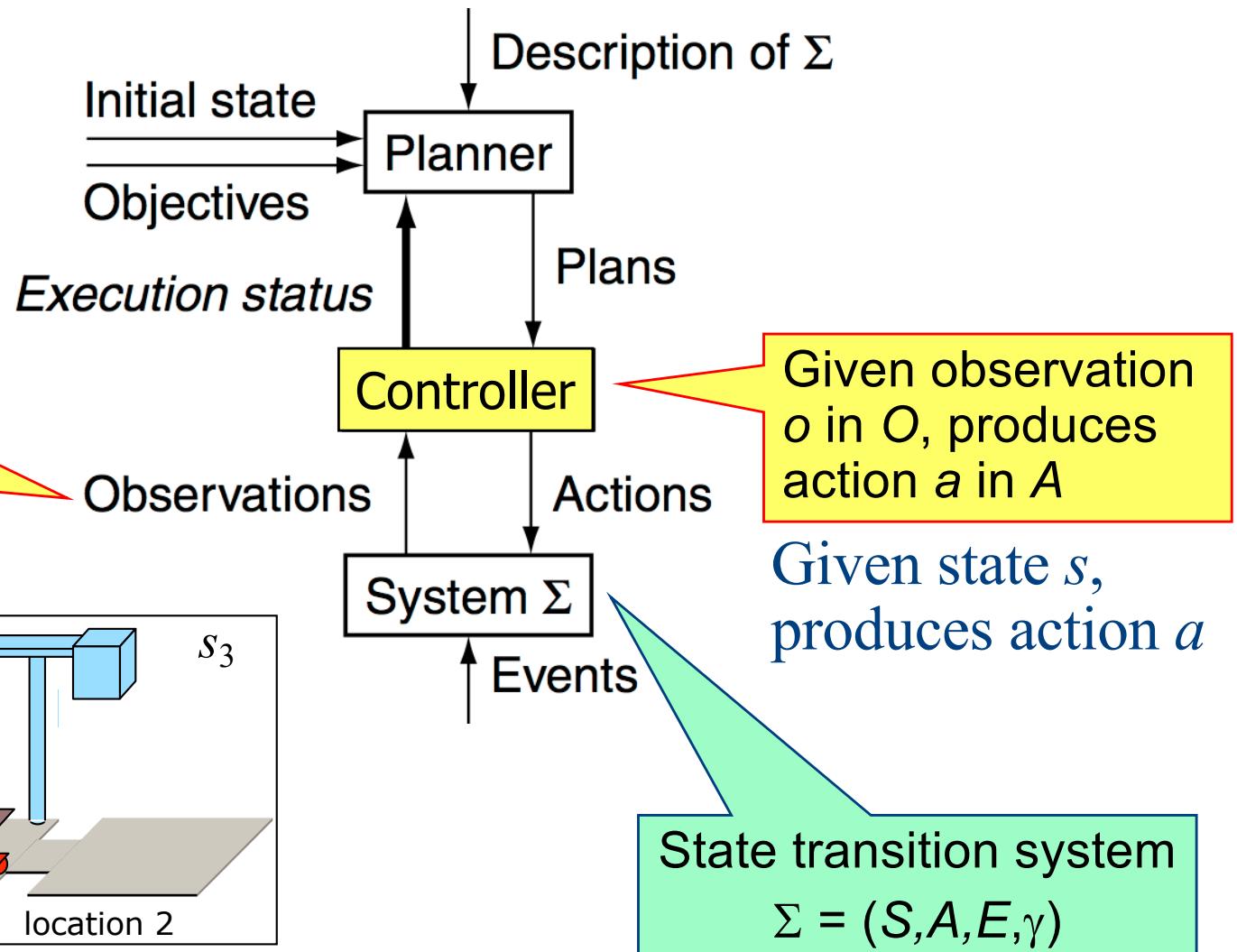
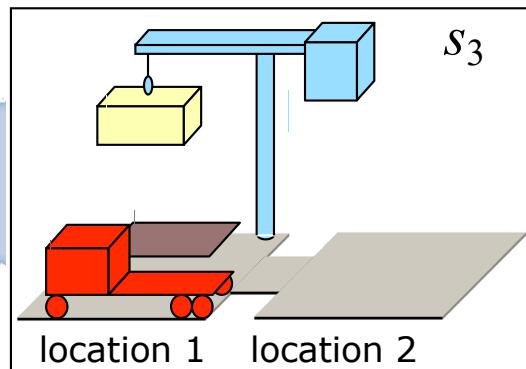
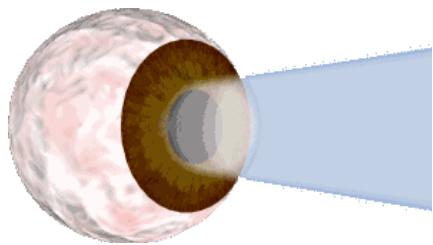
Conceptual Model

2. Controller

Complete observability:

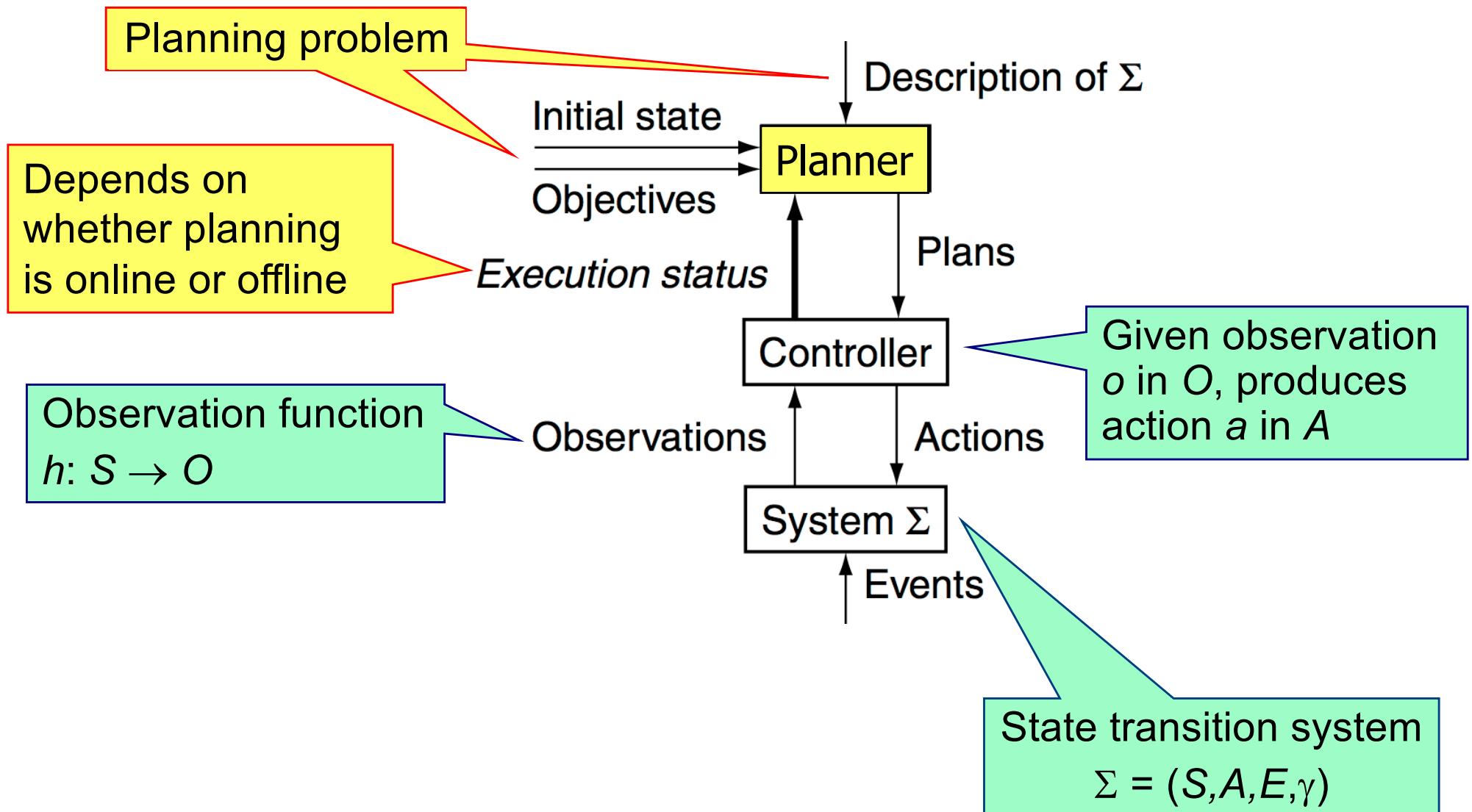
$$h(s) = s$$

Observation function
 $h: S \rightarrow O$



Conceptual Model

3. Planner's Input



Planning Problem

Description of Σ

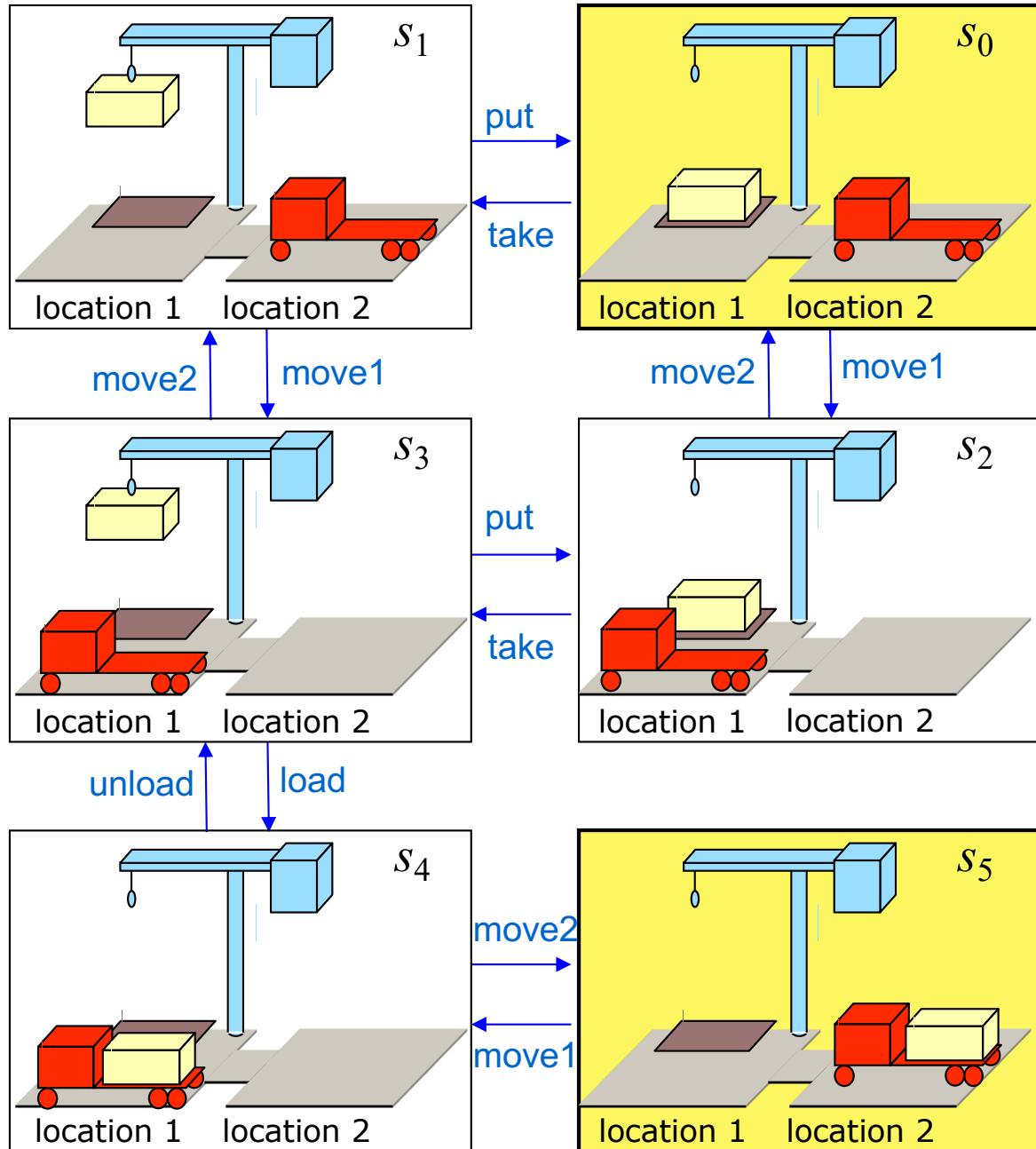
Initial state or set of states

Initial state = s_0

Objective

Goal state, set of goal states, set of tasks, “trajectory” of states, objective function, ...

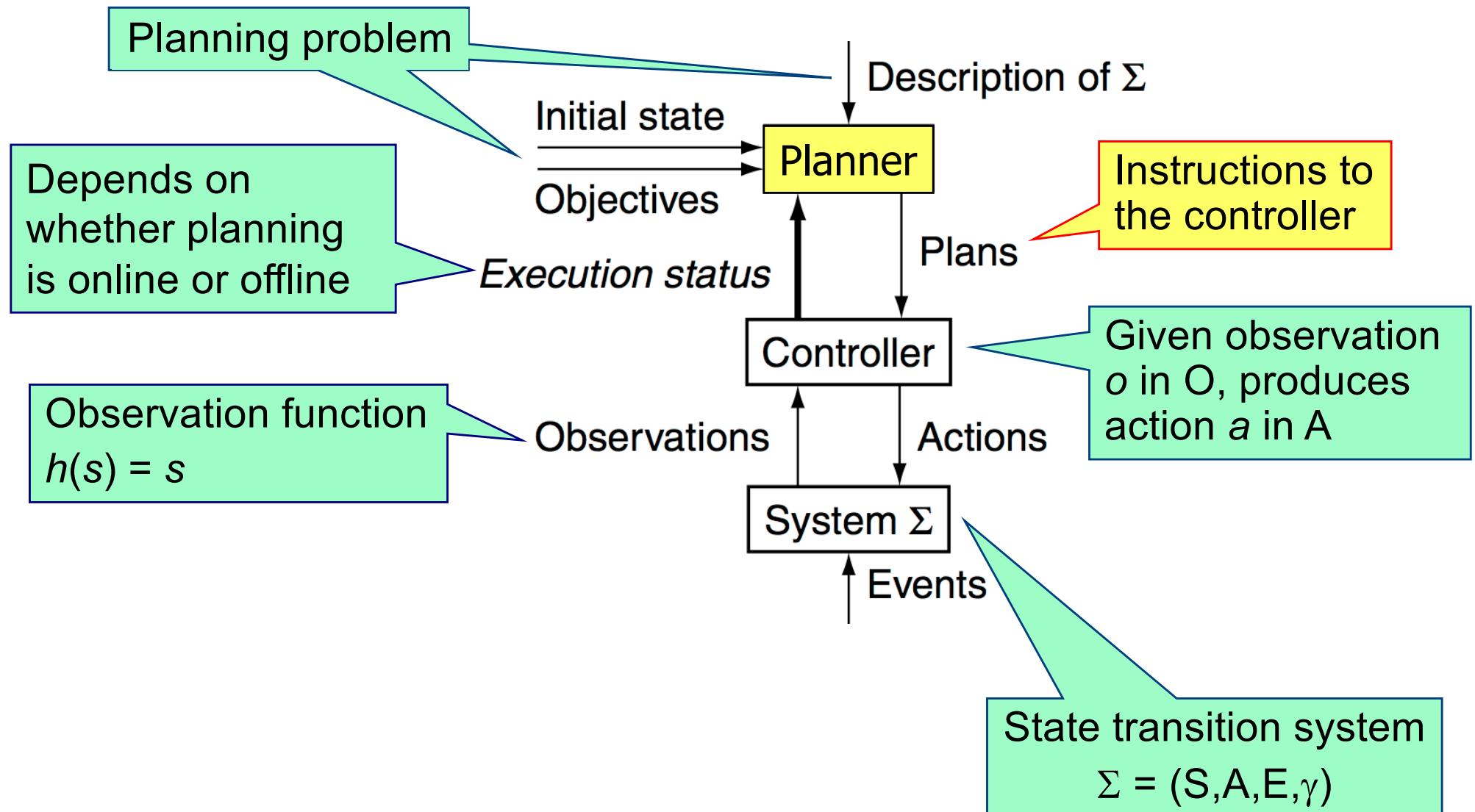
Goal state = s_5



The Dock Worker Robots (DWR) domain

Conceptual Model

4. Planner's Output



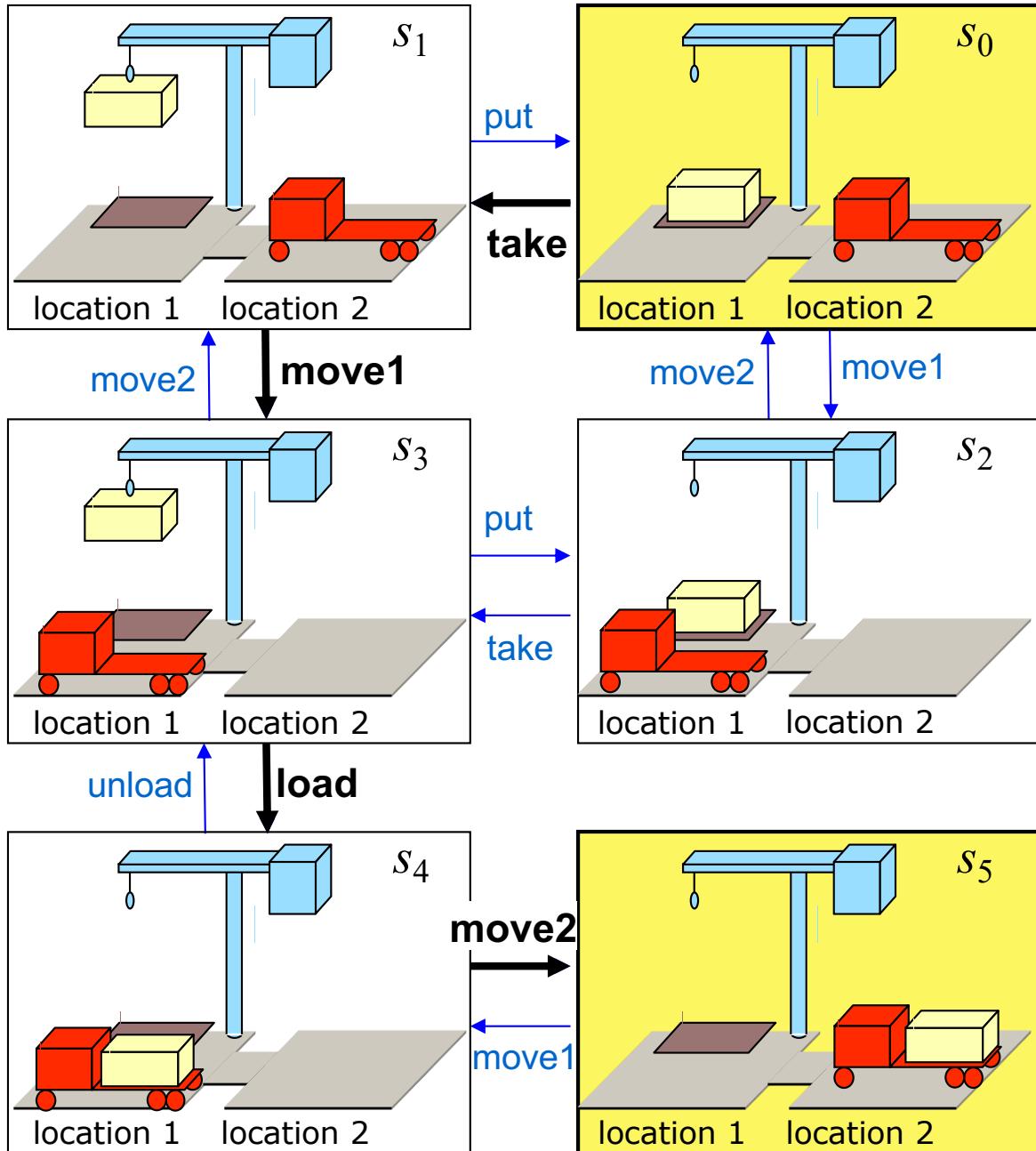
Plans

Classical plan: a sequence of actions

$\langle \text{take}, \text{move1}, \text{load}, \text{move2} \rangle$

Policy: partial function from S into A

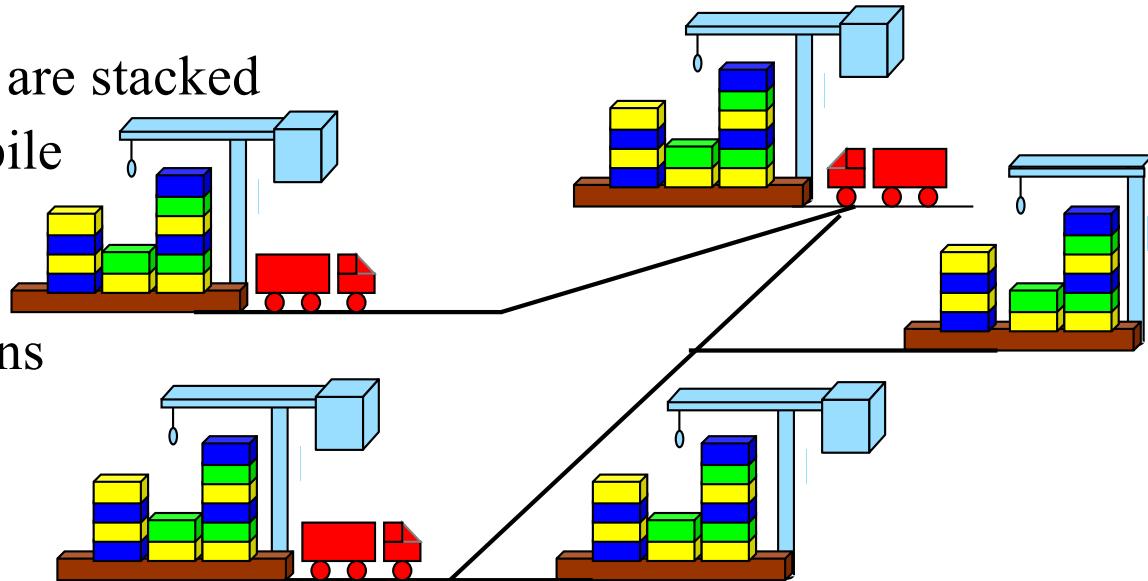
$\{(s_0, \text{take}),$
 $(s_1, \text{move1}),$
 $(s_3, \text{load}),$
 $(s_4, \text{move2})\}$



The Dock Worker Robots (DWR) domain

A running example: Dock Worker Robots

- **Locations:** l_1, l_2, \dots
- **Containers:** c_1, c_2, \dots
 - ◆ can be stacked in piles, loaded onto robots, or held by cranes
- **Piles:** p_1, p_2, \dots
 - ◆ fixed areas where containers are stacked
 - ◆ pallet at the bottom of each pile
- **Robot carts:** r_1, r_2, \dots
 - ◆ can move to adjacent locations
 - ◆ carry at most one container
- **Cranes:** k_1, k_2, \dots
 - ◆ each belongs to a single location
 - ◆ move containers between piles and robots
 - ◆ if there is a pile at a location, there must also be a crane there



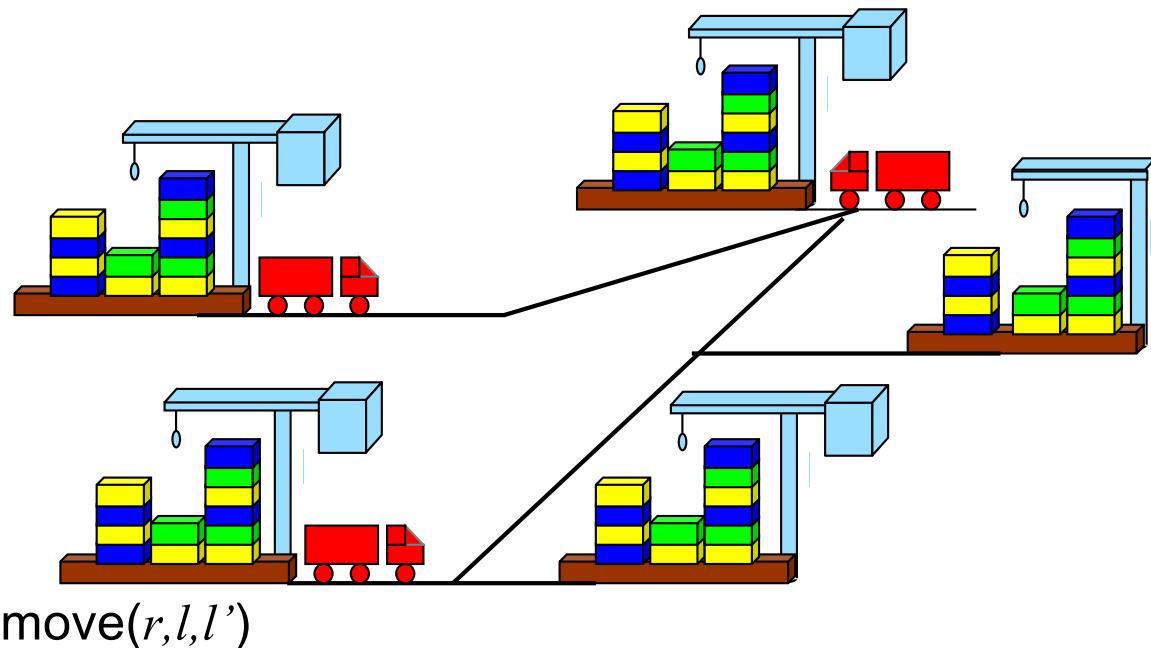
A running example: Dock Worker Robots

- Fixed relations: same in all states

$\text{adjacent}(l, l')$ $\text{attached}(p, l)$ $\text{belong}(k, l)$

- Dynamic relations: differ from one state to another

$\text{occupied}(l)$	$\text{at}(r, l)$
$\text{loaded}(r, c)$	$\text{unloaded}(r)$
$\text{holding}(k, c)$	$\text{empty}(k)$
$\text{in}(c, p)$	$\text{on}(c, c')$
$\text{top}(c, p)$	$\text{top}(\text{pallet}, p)$



- Actions:

$\text{take}(c, k, p)$	$\text{put}(c, k, p)$
$\text{load}(r, c, k)$	$\text{unload}(r)$

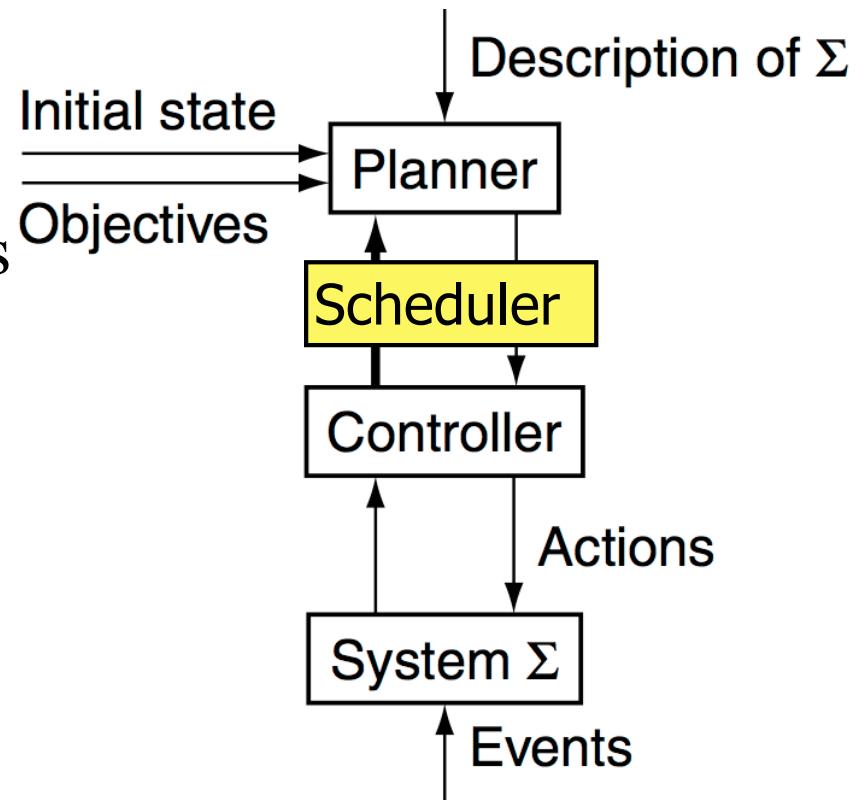
$\text{move}(r, l, l')$

Planning Versus Scheduling

- What is the difference between these two types of problems?

Planning Versus Scheduling

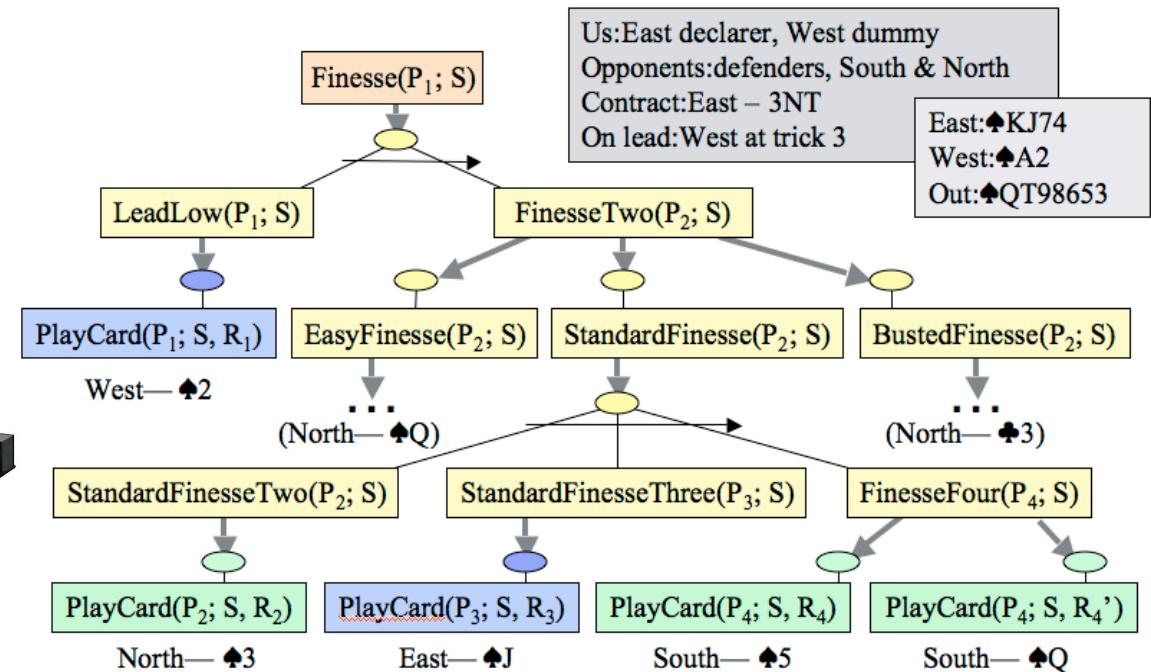
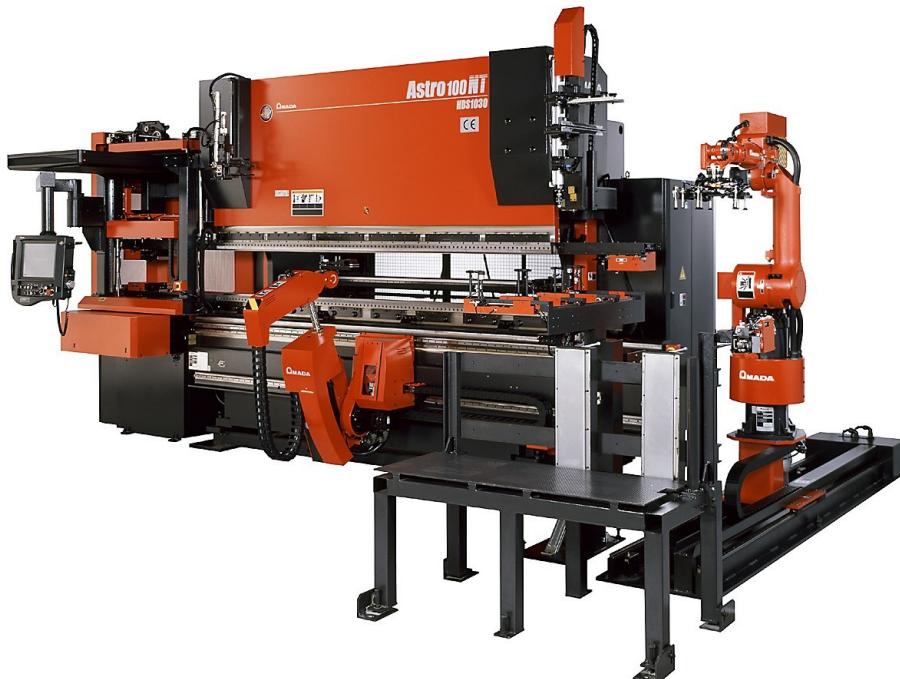
- Scheduling
 - ◆ Decide when and how to perform a given set of actions
 - » Time constraints
 - » Resource constraints
 - » Objective functions
 - ◆ Typically NP-complete
- Planning
 - ◆ Decide what actions to use to achieve some set of objectives
 - ◆ Can be much worse than NP-complete; worst case is undecidable



Types of Planners:

1. Domain-Specific

- Made or tuned for a specific domain
- Won't work well (if at all) in any other domain
- Most successful real-world planning systems work this way



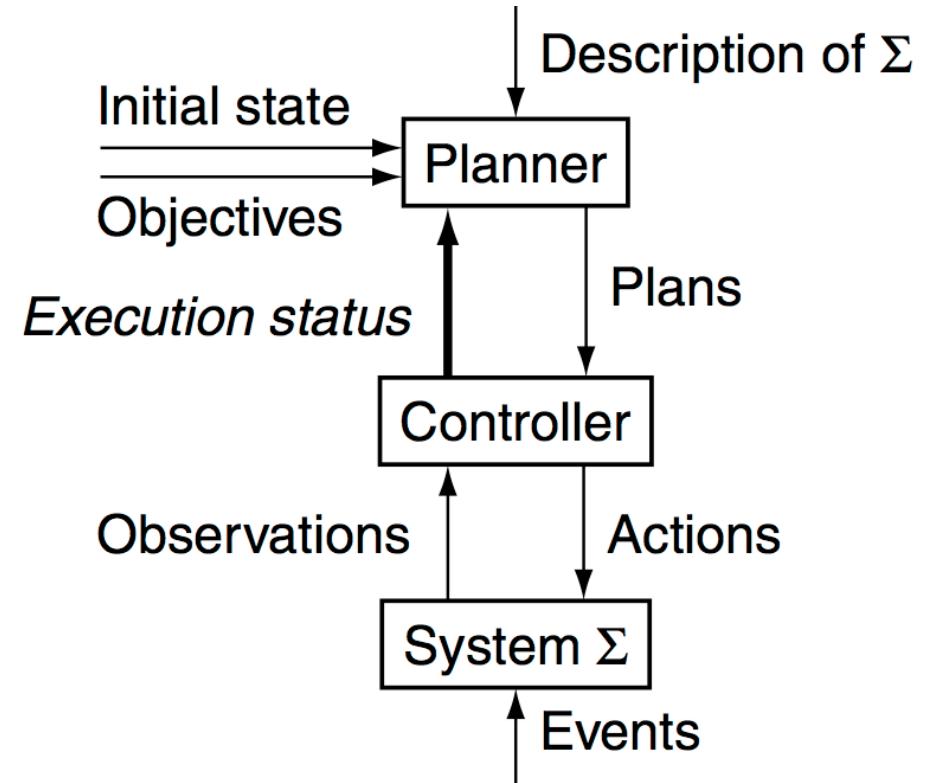
Types of Planners

2. Domain-Independent

- In principle, a domain-independent planner works in any planning domain
- Uses no domain-specific knowledge except the definitions of the basic actions
- Representation written in a STRIPS or PDDL-type formalism
- In practice,
 - ◆ Not feasible to develop domain-independent planners that work in *every* possible domain
- Make simplifying assumptions to restrict the set of domains
 - ◆ *Classical planning*
 - ◆ Historical focus of most automated-planning research

Restrictive Assumptions

- A0: Finite system
 - ◆ finitely many states, actions, and events
- A1: Fully observable
 - ◆ the controller always knows what state Σ is in
- A2: Deterministic
 - ◆ each action or event has only one possible outcome
- A3: Static
 - ◆ No exogenous events: no changes except those performed by the controller



Restrictive Assumptions

A4: Attainment goals

- ◆ a set of goal states S_g

A5: Sequential plans

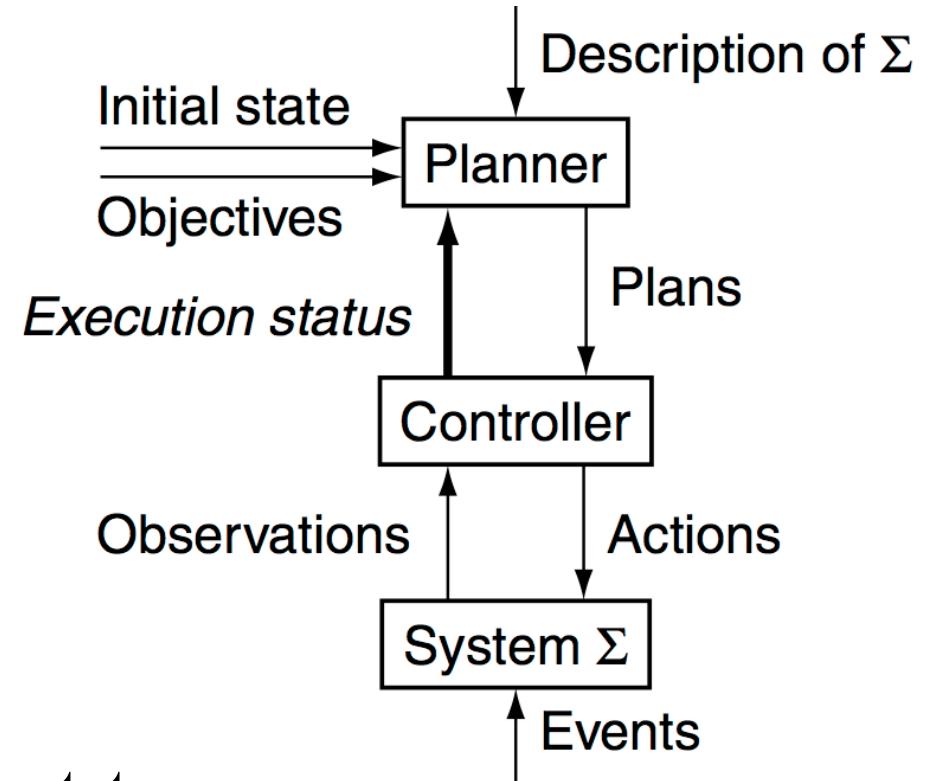
- ◆ a plan is a linearly ordered sequence of actions (a_1, a_2, \dots, a_n)

A6 :Implicit time

- ◆ no time durations
- ◆ linear sequence of instantaneous states

A7: Off-line planning

- ◆ planner doesn't know the execution status

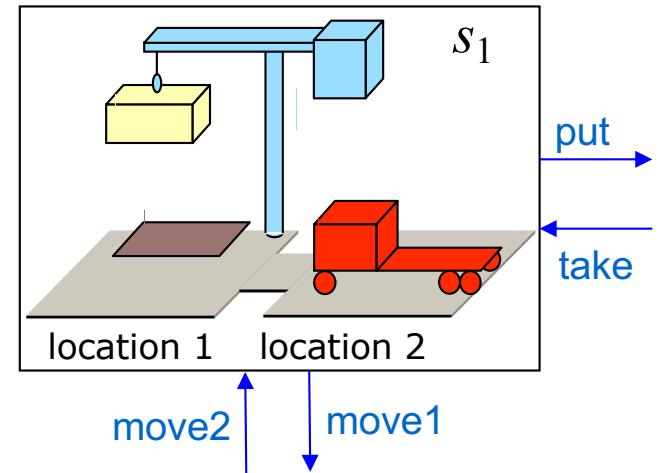


Classical Planning

- Classical planning requires all eight restrictive assumptions
 - ◆ Offline generation of action sequences for a deterministic, static, finite system, with complete knowledge, attainment goals, and implicit time
- Reduces to the following problem:
 - ◆ Given (Σ, s_0, S_g)
 - ◆ Find a sequence of actions (a_1, a_2, \dots, a_n) that produces a sequence of state transitions (s_1, s_2, \dots, s_n) such that s_n is in S_g .
- This is just path-searching in a graph
 - ◆ Nodes = states
 - ◆ Edges = actions
- *Is this trivial?*

Classical Planning

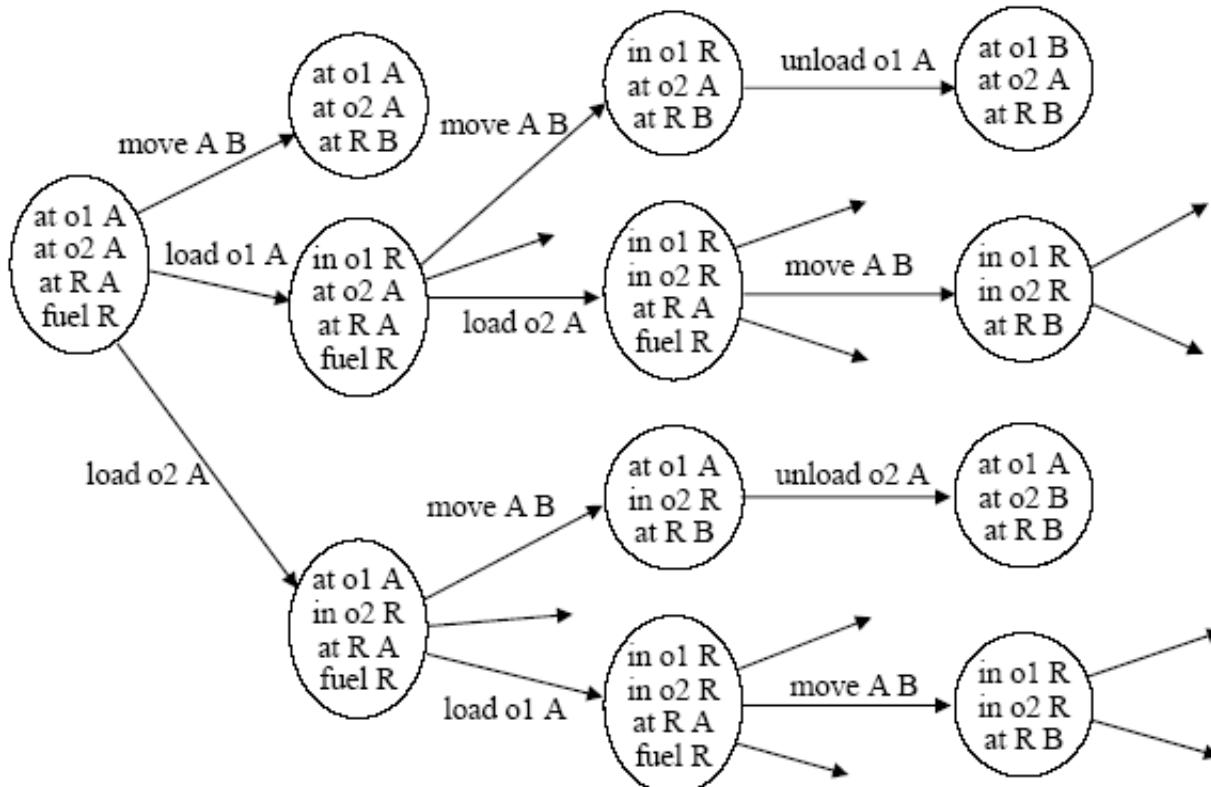
- Generalize the earlier example:
 - ◆ Five locations, three robot carts, 100 containers, three piles
 - » Then there are 10^{277} states
- Number of particles in the universe is only about 10^{87}
 - ◆ The example is more than 10^{190} times as large!
- Automated-planning research has been heavily dominated by classical planning
 - ◆ Dozens (hundreds?) of different algorithms



Plan Definition Language

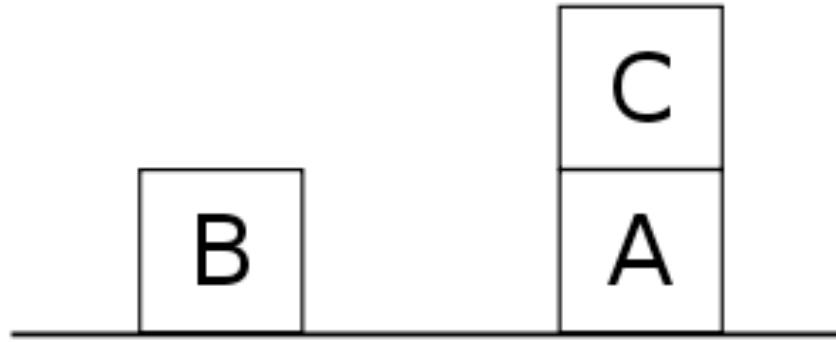
- PDDL=planning domain definition language
- Attempt to standardize the input of planner
- Domain definition
 - ◆ Predicates
 - ◆ Operators
- Problem definition
 - ◆ Objects present
 - ◆ Initial state description
 - ◆ Goal
- STRIPS: commonly used subset of PDDL

State Space Planning



- Conceptually simple form of planning
- Does not discover parallelizable actions

Goal Interaction: Sussman Anomaly

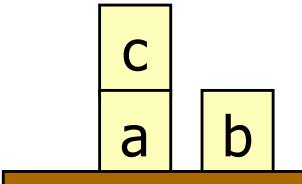


2 subgoals

- A on B
- B on C

Planner must interleave goal achievement.

Completing either goal in entirety will render the other goal impossible



Plan-Space Planning (UCPOP)

$\text{clear}(x)$, with $x = a$

$\text{unstack}(x,a)$

$\text{clear}(b)$,
handempty

$\text{clear}(a)$

$\text{pickup}(b)$

$\text{holding}(a)$

$\text{putdown}(x)$

handempty

$\text{pickup}(a)$

$\text{holding}(a)$

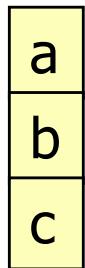
$\text{stack}(b,c)$

$\text{clear}(b)$

$\text{stack}(a,b)$

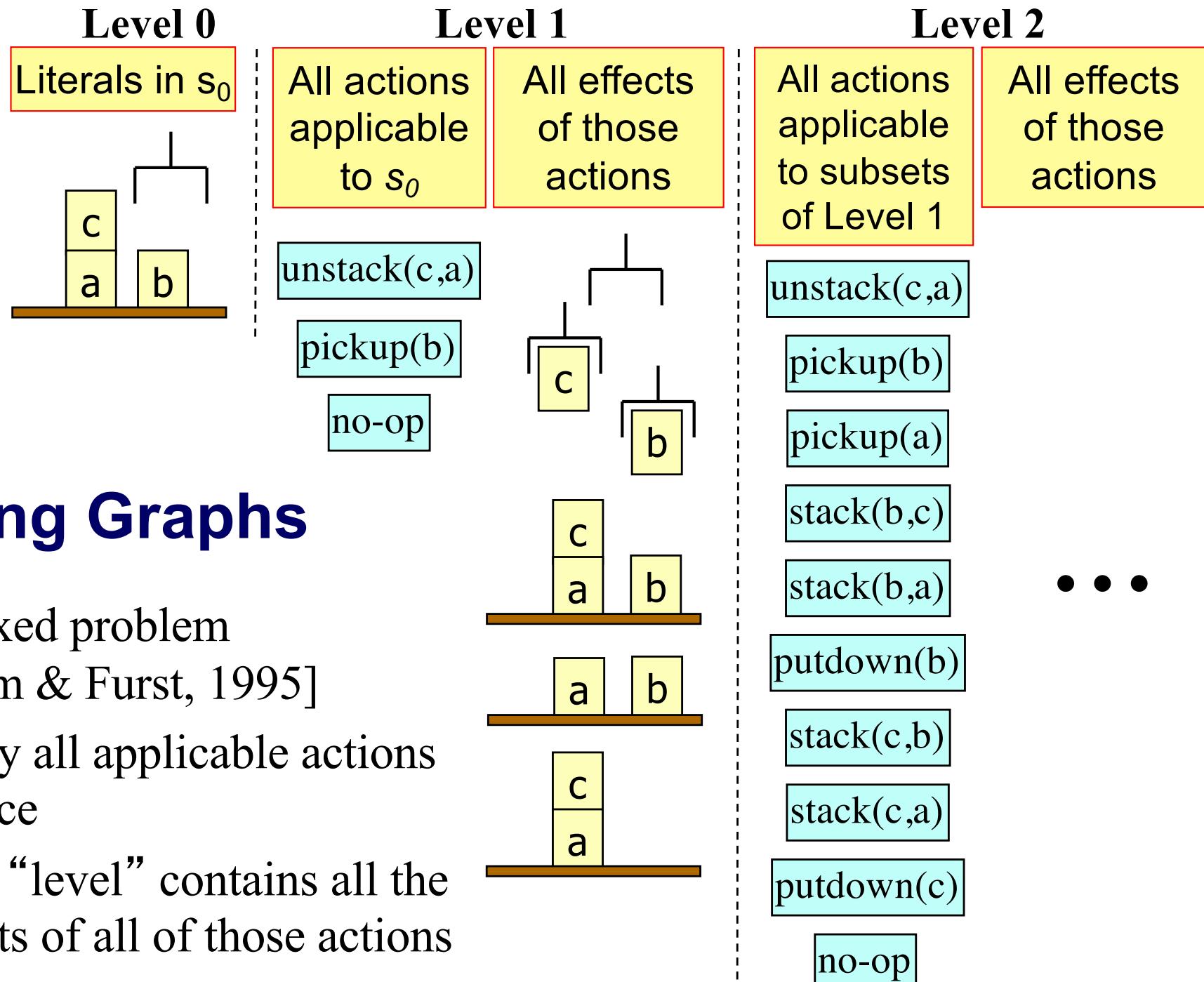
$\text{on}(a,b)$
 $\text{on}(b,c)$

Goal:
 $\text{on}(a,b) \& \text{on}(b,c)$



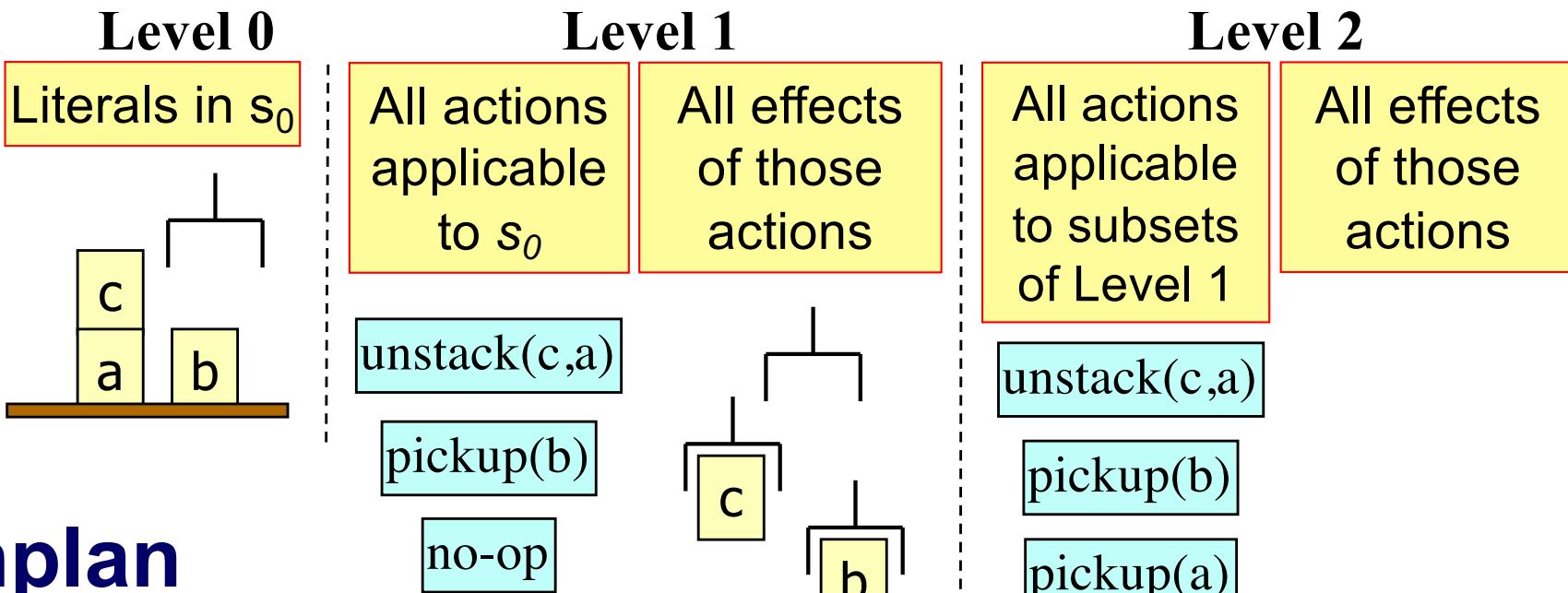
For classical planning,
not used very much any more
RAX and MER use
temporal-planning extensions of it

- Decompose sets of goals into the individual goals
- Plan for them separately
 - ◆ Bookkeeping info to detect and resolve interactions



Planning Graphs

- Relaxed problem
[Blum & Furst, 1995]
- Apply all applicable actions at once
- Next “level” contains all the effects of all of those actions



Graphplan

- For $n = 1, 2, \dots$
 - ◆ Make planning graph of n levels (*polynomial time*)
 - ◆ State-space search *within the planning graph*
- Graphplan's many children
 - ◆ IPP, CGP, DGP, LGP, PGP, SGP, TGP, ...

Heuristic Search

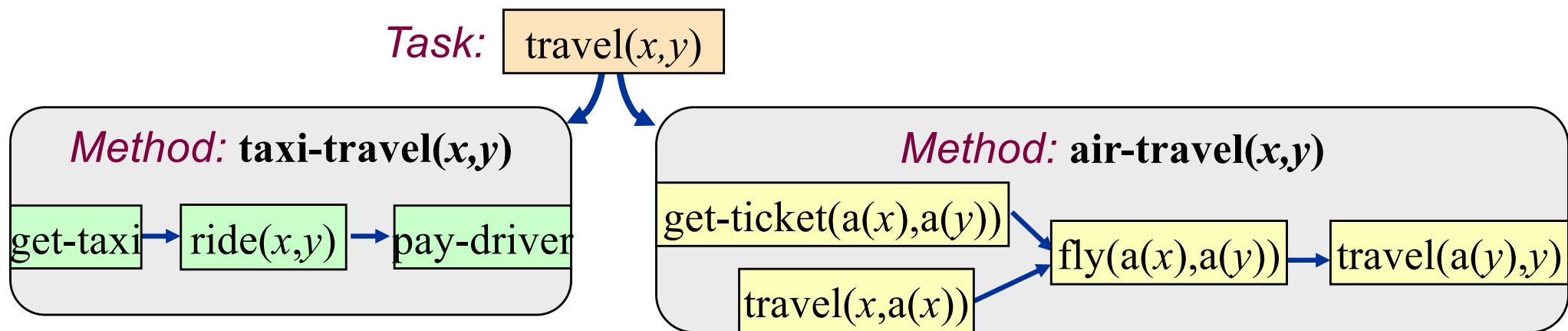
- Can we do an A*-style heuristic search?
- For many years, nobody could come up with a good h function
 - ◆ But planning graphs make it feasible
 - » Can extract h from the planning graph
- Problem: A* quickly runs out of memory
 - ◆ So do a greedy search
- Greedy search can get trapped in local minima
 - ◆ Greedy search plus local search at local minima
- HSP [Bonet & Geffner]
- FastForward [Hoffmann]

Translation to Other Domains

- Translate the planning problem or the planning graph into another kind of problem for which there are efficient solvers
 - ◆ Find a solution to that problem
 - ◆ Translate the solution back into a plan
- Satisfiability solvers, especially those that use local search
 - ◆ Satplan and Blackbox [Kautz & Selman]
- Integer programming solvers such as Cplex
 - ◆ [Vossen *et al.*]

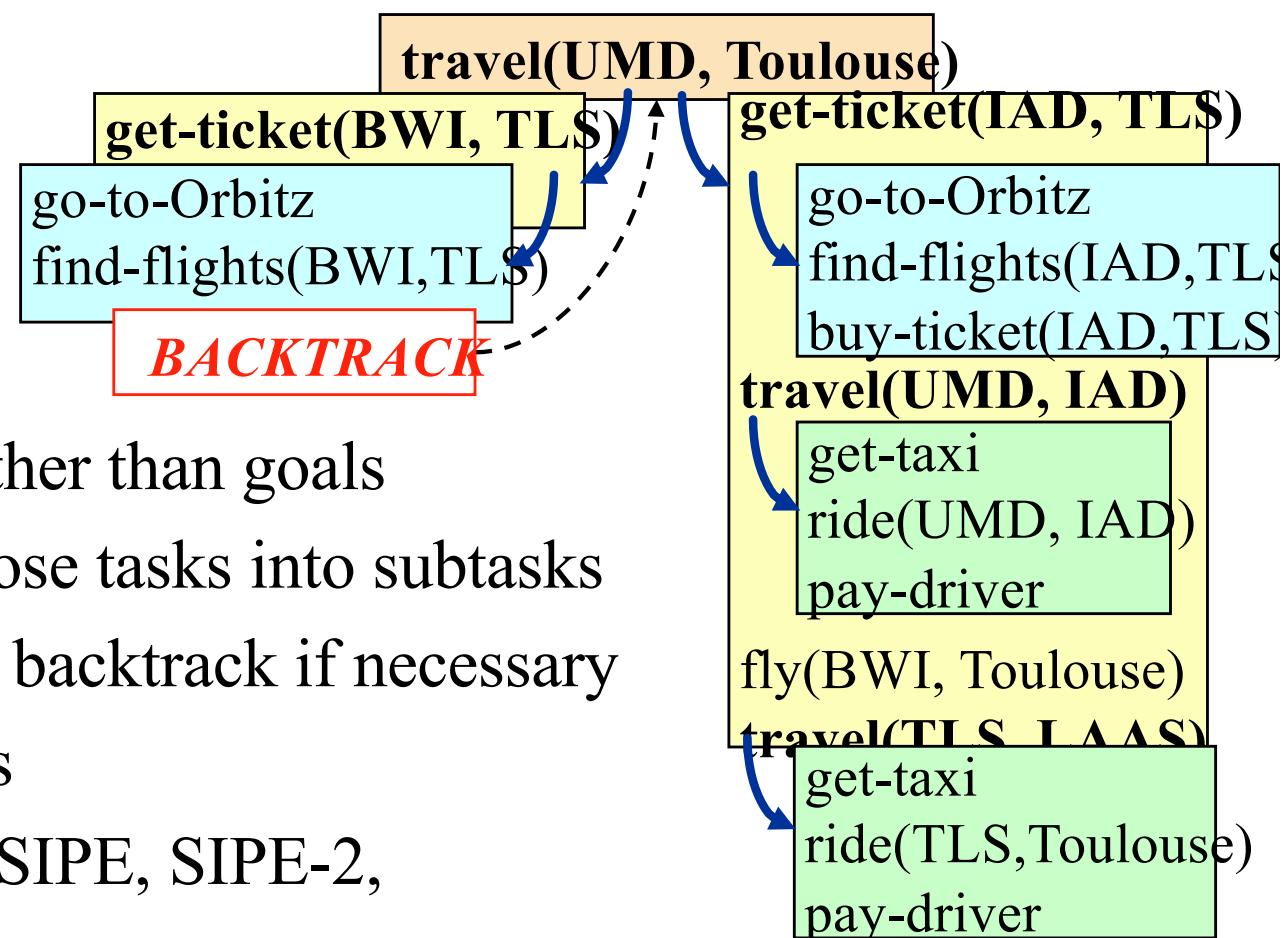
Types of Planners

- Domain-independent planners are quite slow compared with domain-specific planners
 - ◆ Blocks world in linear time [Slaney and Thiébaux, *A.I.*, 2001]
 - ◆ Can get analogous results in many other domains
- But we don't want to write a whole new planner for every domain!
- **Configurable planners**
 - ◆ Domain-independent planning engine
 - ◆ Input includes info about how to solve problems in the domain
 - » Hierarchical Task Network (HTN) planning
 - » Planning with control formulas

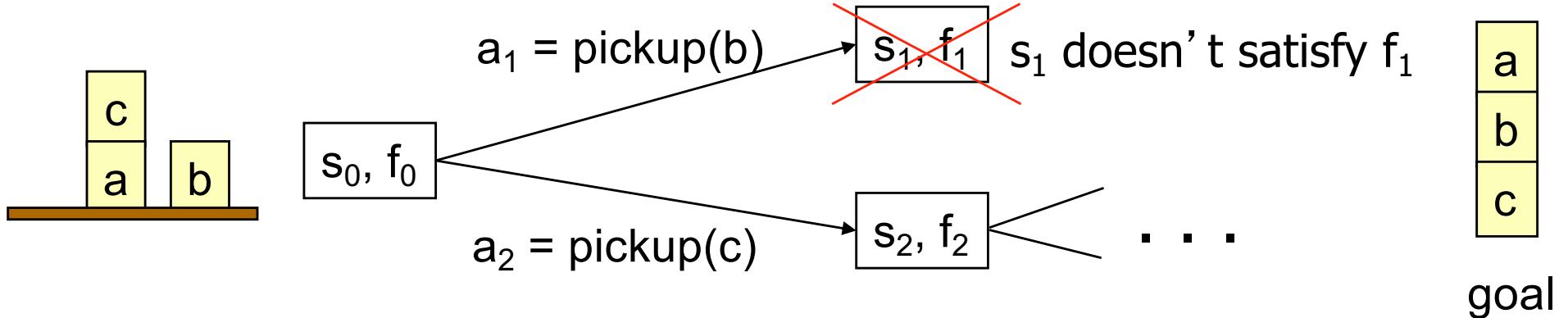


HTN Planning

- Problem reduction
 - ◆ *Tasks* (activities) rather than goals
 - ◆ *Methods* to decompose tasks into subtasks
 - ◆ Enforce constraints, backtrack if necessary
- Real-world applications
- Noah, Nonlin, O-Plan, SIPE, SIPE-2, SHOP, SHOP2



Planning with Control Formulas

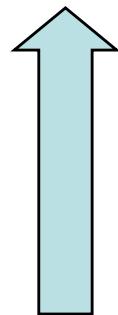


- At each state s_i we have a *control formula* f_i in temporal logic
$$\text{ontable}(x) \wedge \neg \exists [y: \text{GOAL}(\text{on}(x, y))] \Rightarrow \Diamond(\neg \text{holding}(x))$$

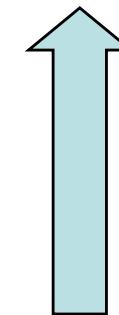
“never pick up x from table unless x needs to be on another block”
- For each successor of s , derive a control formula using *logical progression*
- Prune any successor state in which the progressed formula is false
 - ◆ TLPlan [Bacchus & Kabanza]
 - ◆ TALplanner [Kvarnstrom & Doherty]

Comparisons

up-front
human effort



Domain-specific
Configurable
Domain-independent

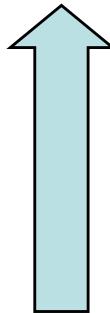


performance

- Domain-specific planner
 - ◆ Write an entire computer program - lots of work
 - ◆ Lots of domain-specific performance improvements
- Domain-independent planner
 - ◆ Just give it the basic actions - not much effort
 - ◆ Not very efficient

Comparisons

coverage



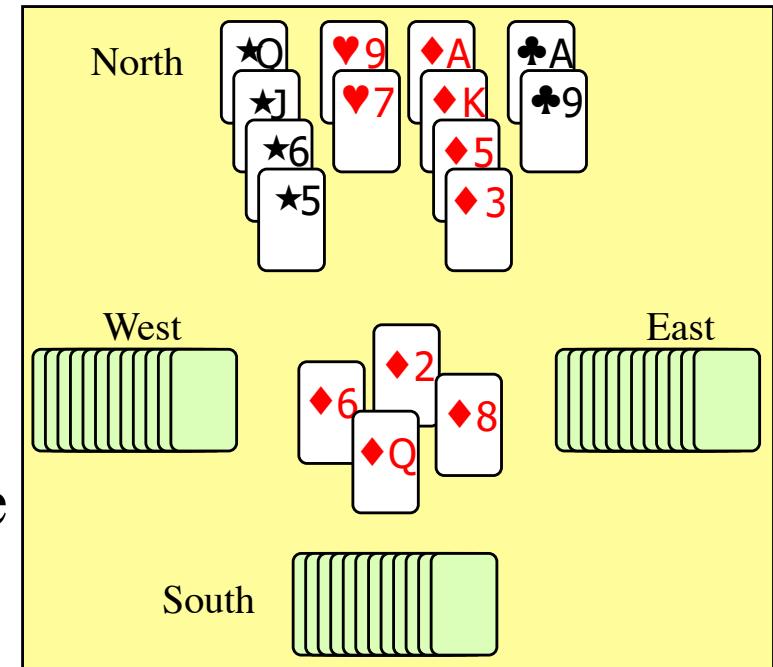
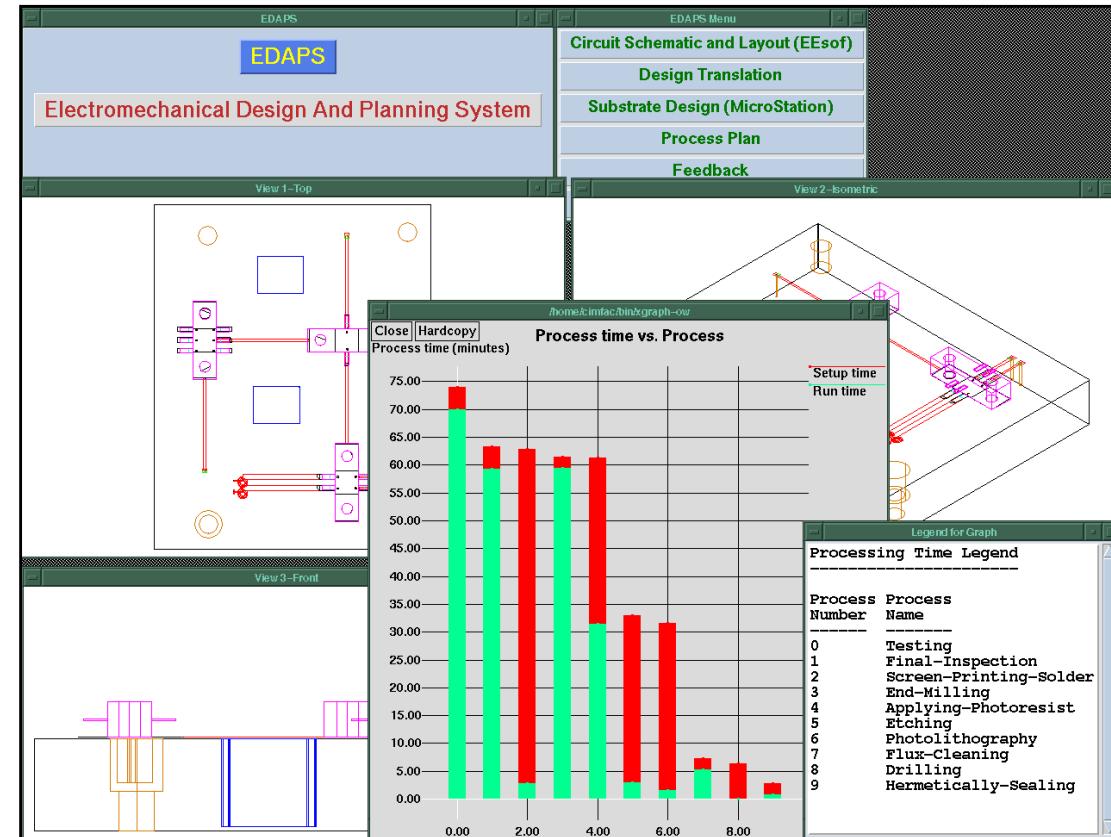
Configurable
Domain-independent
Domain-specific

- A domain-specific planner only works in one domain
- **In principle**, configurable and domain-independent planners should both be able to work in any domain
- **In practice**, configurable planners work in a larger variety of domains
 - ◆ Partly due to efficiency
 - ◆ Partly due to expressive power

What are classical planners bad at?

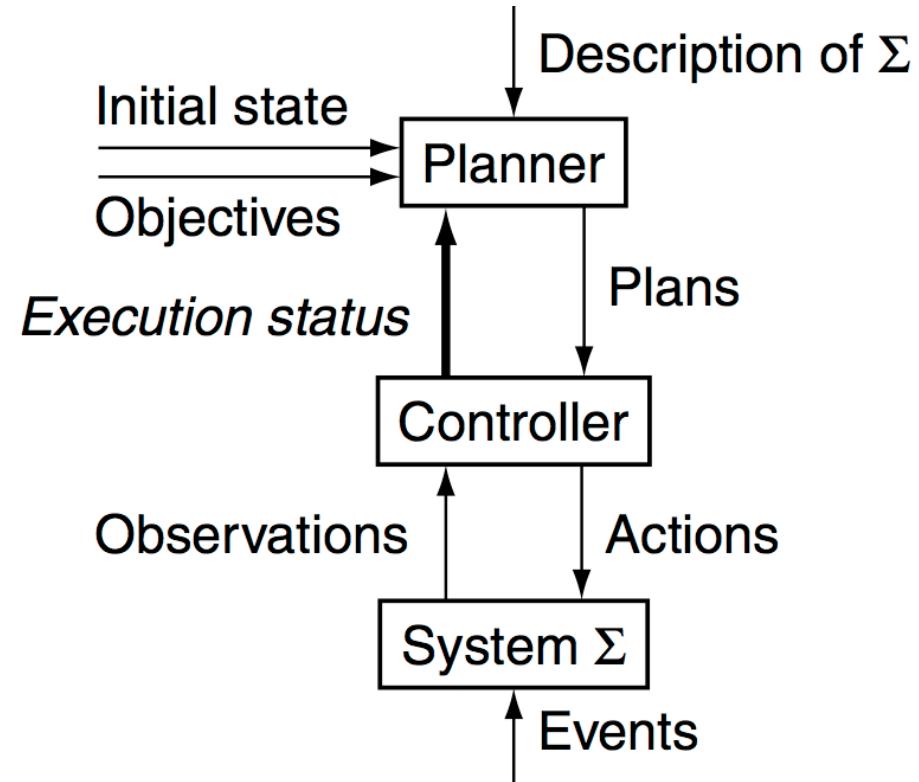
Example

- Typical characteristics of application domains
 - Dynamic world
 - Multiple agents
 - Imperfect/uncertain info
 - External info sources
 - users, sensors, databases
- Durations, time constraints, asynchronous actions
- Numeric computations
 - geometry, probability, etc.
- Classical planning excludes all of these



Relax the Assumptions

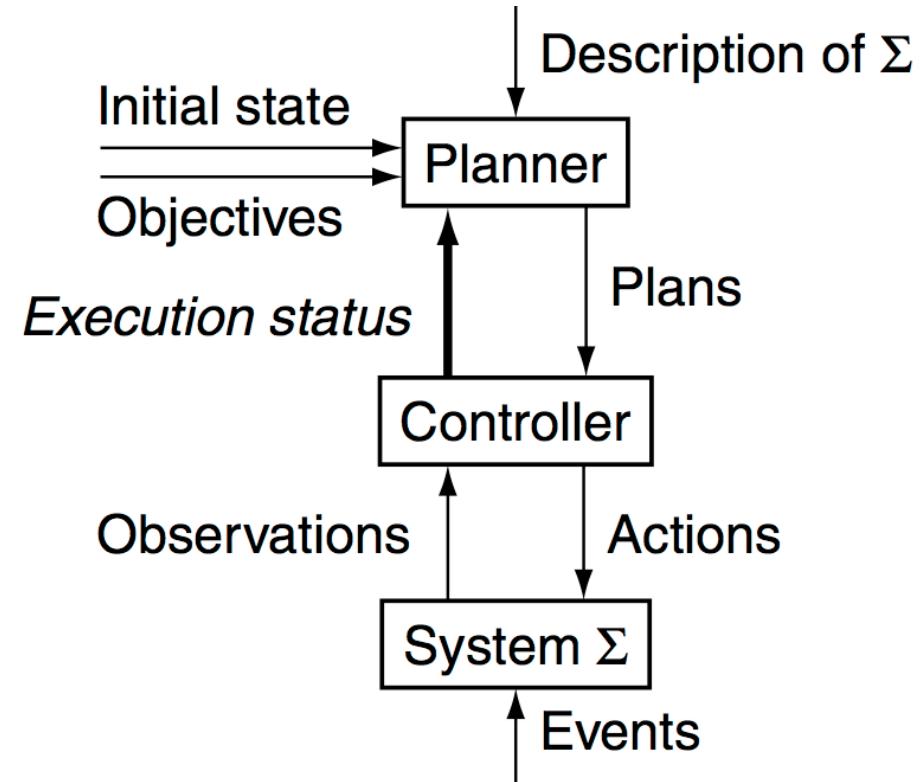
- Relax A2 (deterministic Σ):
 - ◆ Actions have more than one possible outcome
 - ◆ Seek policy or contingency plan
 - ◆ With probabilities:
 - » Discrete Markov Decision Processes (MDPs)
 - ◆ Without probabilities:
 - » Nondeterministic transition systems



$$\begin{aligned}\Sigma &= (S, A, E, \gamma) \\ S &= \{\text{states}\} \\ A &= \{\text{actions}\} \\ E &= \{\text{events}\} \\ \gamma: S \times (A \cup E) &\rightarrow 2^S\end{aligned}$$

Relax the Assumptions

- Relax A1 and A2:
 - ◆ Finite POMDPs
 - » Plan over *belief states*
 - » Exponential time & space
- Relax A0 and A2:
 - ◆ Continuous or hybrid MDPs
 - » Control theory
- Relax A0, A1, and A2:
 - ◆ Continuous or hybrid POMDPs
 - » Robotics



$$\Sigma = (S, A, E, \gamma)$$

$$S = \{\text{states}\}$$

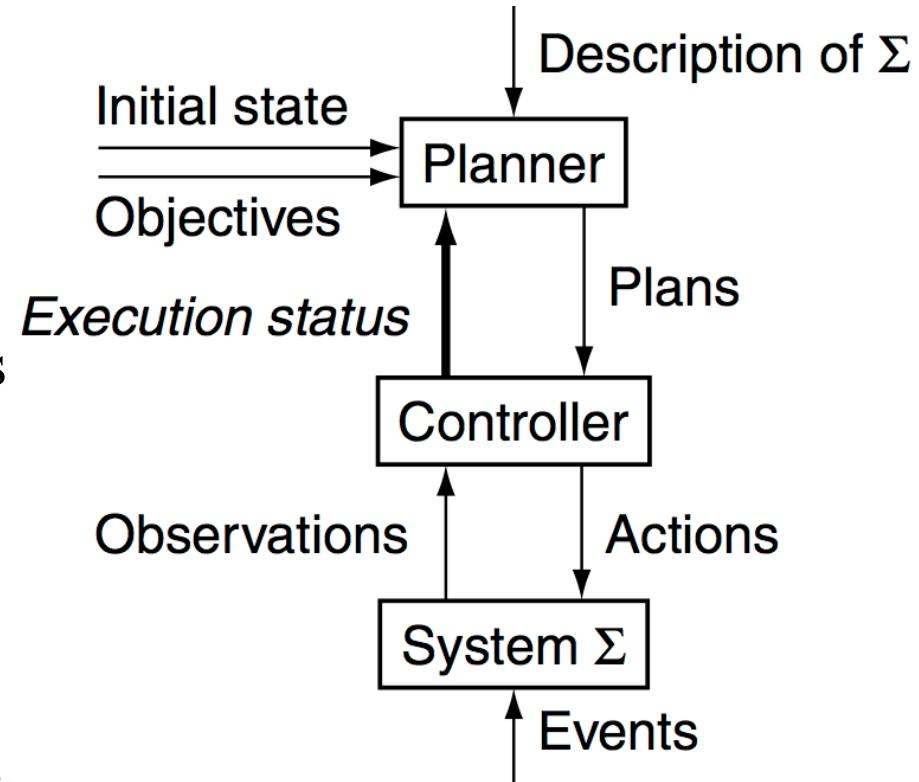
$$A = \{\text{actions}\}$$

$$E = \{\text{events}\}$$

$$\gamma: S \times (A \cup E) \rightarrow 2^S$$

Relax the Assumptions

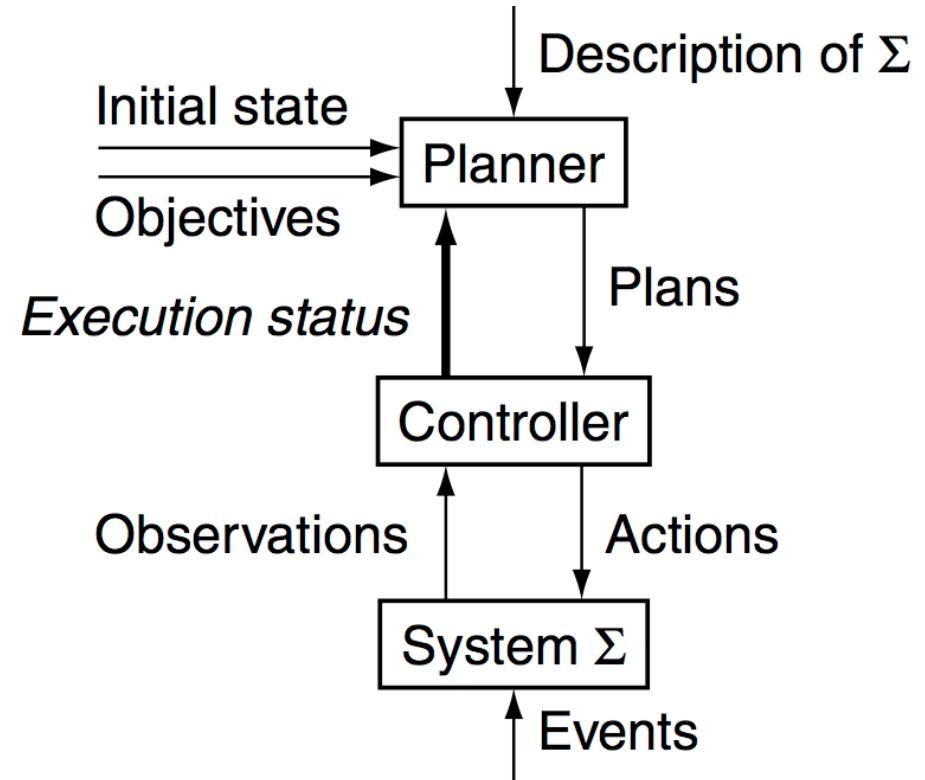
- Relax A3 (static Σ):
 - ◆ Other agents or dynamic environment
 - » Finite perfect-info zero-sum games (introductory AI courses)
 - ◆ Randomly behaving environment
 - » Decision analysis (business, operations research)
 - » Can sometimes map this into MDPs or POMDPs
- Relax A1 and A3
 - ◆ Imperfect-information games
 - ◆ Case study: Chapter 23 (bridge)



$$\begin{aligned}\Sigma &= (S, A, E, \gamma) \\ S &= \{\text{states}\} \\ A &= \{\text{actions}\} \\ E &= \{\text{events}\} \\ \gamma: S \times (A \cup E) &\rightarrow 2^S\end{aligned}$$

Relax the Assumptions

- Relax A5 (sequential plans) and A6 (implicit time):
 - ◆ Temporal planning
- Relax A0, A5, A5
 - ◆ Planning and resource scheduling



$$\begin{aligned}\Sigma &= (S, A, E, \gamma) \\ S &= \{\text{states}\} \\ A &= \{\text{actions}\} \\ E &= \{\text{events}\} \\ \gamma: S \times (A \cup E) &\rightarrow 2^S\end{aligned}$$

International Planning Competition

- Held at the ICAPS conference to find the fastest planner
- <http://ipc.icaps-conference.org/>
- 2020 IPC:
 - ◆ Hierarchical planning
- SHOP2 (described in article) won one of the top prizes in the 2002 competition

References

- Ghallab, Nau, Traverso, Automated Task Planning: Theory and Practice

CAP6671 Intelligent Systems: Robots, Agents, and Humans

Lecture 4: Planning Systems; AI in Games

**Instructor: Dr. Gita Sukthankar
Email: gitars@eecs.ucf.edu**

Planning Problem

Description of Σ

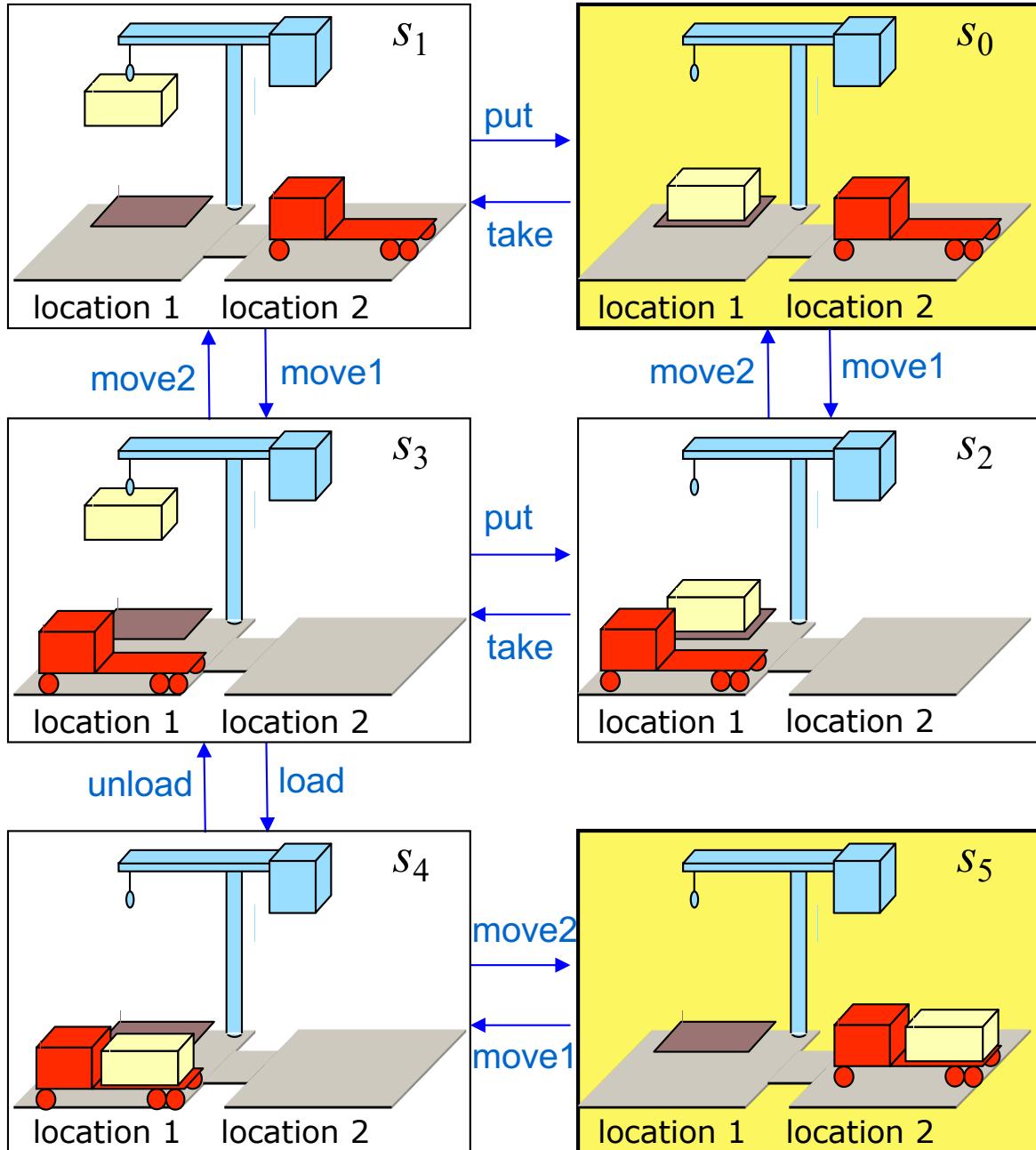
Initial state or set of states

Initial state = s_0

Objective

Goal state, set of goal states, set of tasks, “trajectory” of states, objective function, ...

Goal state = s_5



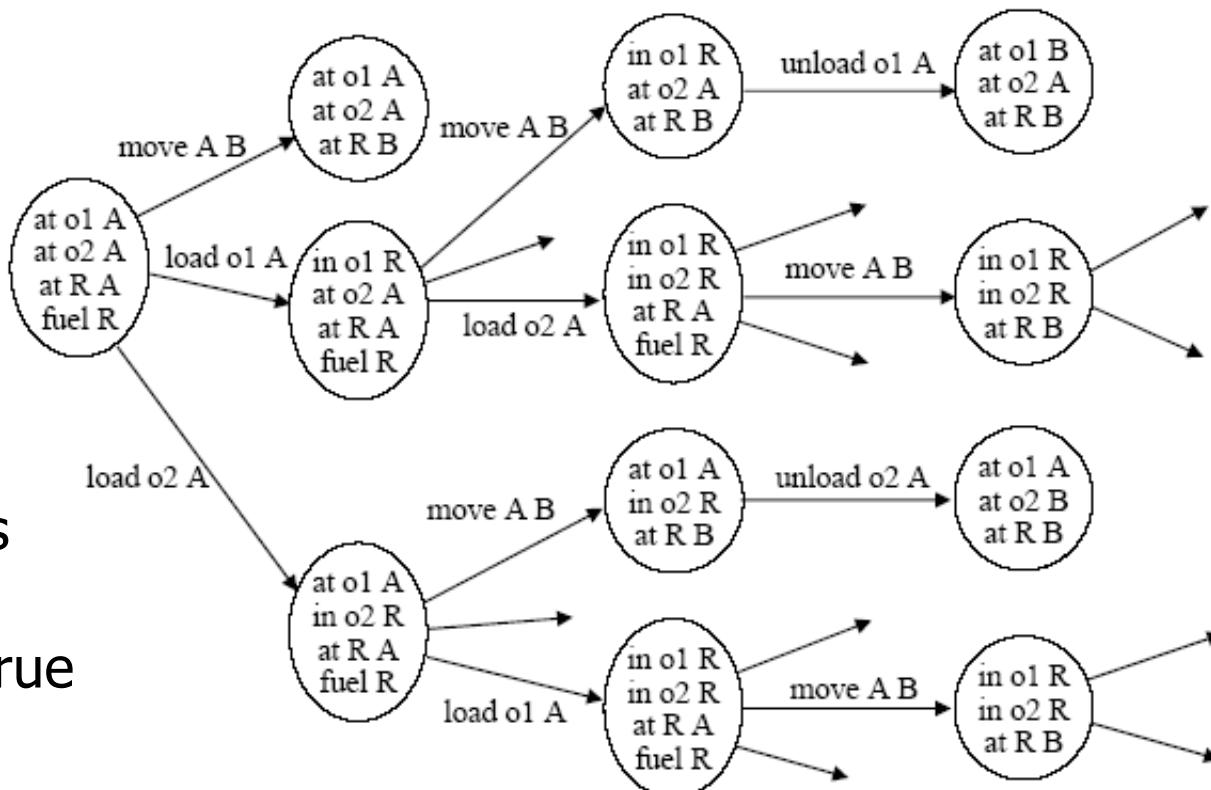
The Dock Worker Robots (DWR) domain

Planners

- *State-space planning*
 - ◆ Each node represents a state of the world
 - » A plan is a path through the space
- *Plan-space planning (e.g. UCPOP)*
 - ◆ Each node is a set of partially-instantiated operators, plus some constraints
 - » Impose more and more constraints, until we get a plan
- *Graphplan*
 - ◆ Create a planning graph that solves a less constrained version of the problem
- *Hierarchical Task Network (HTN)*
 - ◆ Create and compose more complex operators

State Space Planning

Each state is defined by a list of true propositions



- Conceptually simple form of planning
- Does not discover parallelizable actions

Forward-search(O, s_0, g)

$s \leftarrow s_0$

$\pi \leftarrow$ the empty plan

loop

 if s satisfies g then return π

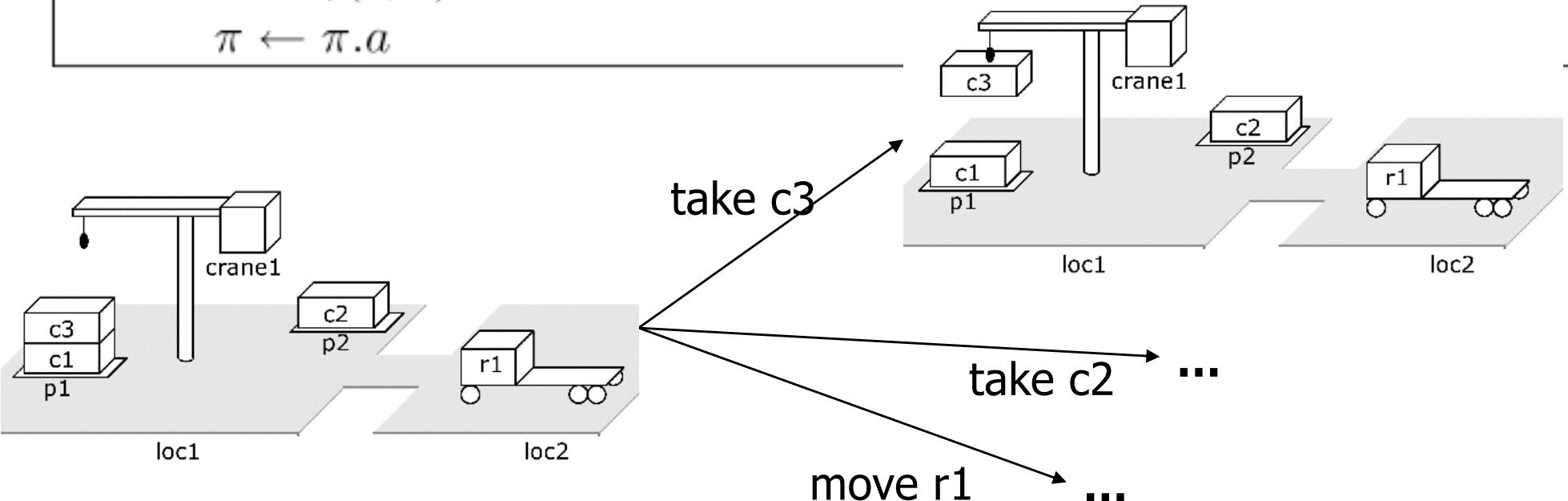
$E \leftarrow \{a | a \text{ is a ground instance an operator in } O,$
 and $\text{precond}(a)$ is true in $s\}$

 if $E = \emptyset$ then return failure

 nondeterministically choose an action $a \in E$

$s \leftarrow \gamma(s, a)$

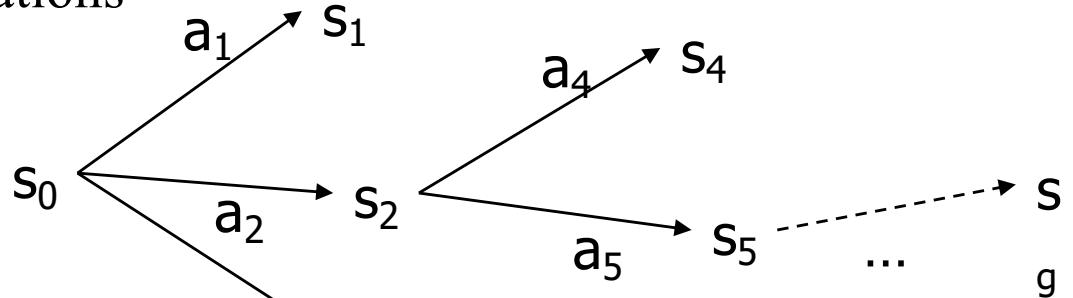
$\pi \leftarrow \pi.a$



Deterministic Implementations

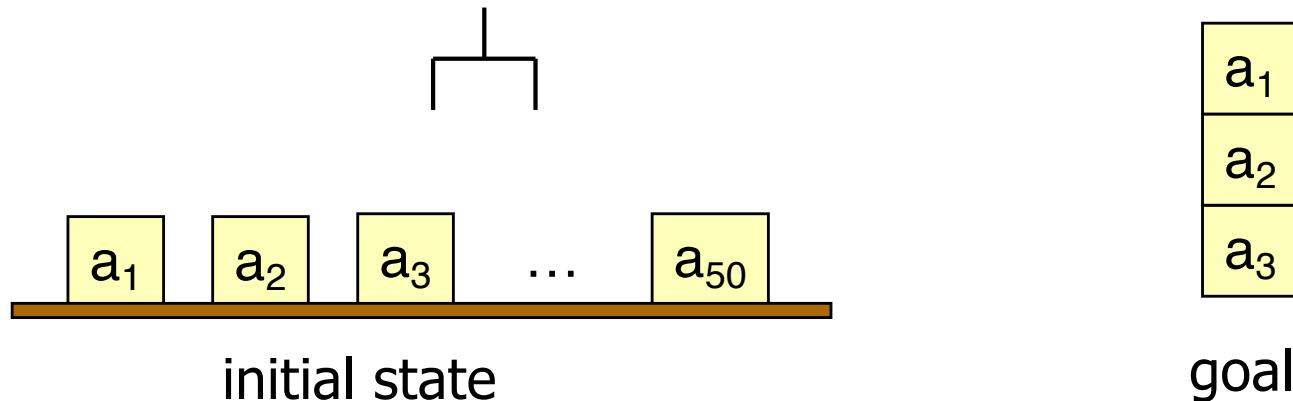
- Some deterministic implementations of forward search:

- ◆ breadth-first search
- ◆ depth-first search
- ◆ best-first search (e.g., A*)
- ◆ greedy search



- Breadth-first and best-first search are sound and complete
 - ◆ But they usually aren't practical because they require too much memory
 - ◆ Memory requirement is exponential in the length of the solution
- In practice, more likely to use depth-first search or greedy search
 - ◆ Worst-case memory requirement is linear in the length of the solution
 - ◆ In general, sound but not complete
 - » But classical planning has only finitely many states
 - » Thus, can make depth-first search complete by doing loop-checking

Branching Factor of Forward Search



- Forward search can have a very large branching factor
 - ◆ E.g., many applicable actions that don't progress toward goal
- Why this is bad:
 - ◆ Deterministic implementations can waste time trying lots of irrelevant actions
- Need a good heuristic function and/or pruning procedure

Backward Search

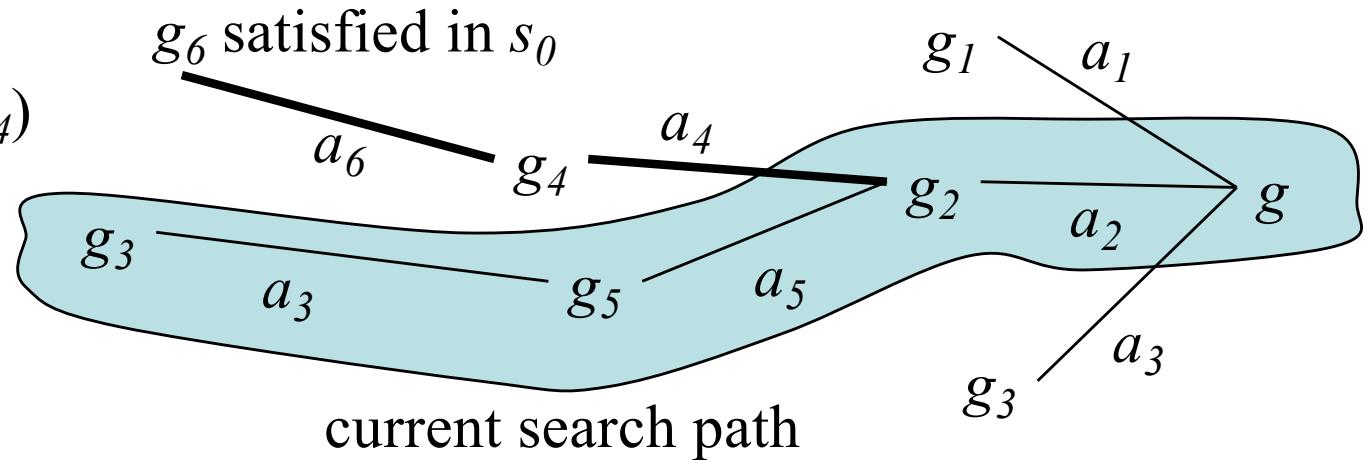
- For forward search, we started at the initial state and computed state transitions
 - ◆ new state = $\gamma(s, a)$
- For backward search, we start at the goal and compute inverse state transitions
 - ◆ new set of subgoals = $\gamma^{-1}(g, a)$
- To define $\gamma^{-1}(g, a)$, must first define *relevance*:
 - ◆ An action a is relevant for a goal g if
 - » a makes at least one of g 's literals true
 - $g \cap \text{effects}(a) \neq \emptyset$
 - » a does not make any of g 's literals false
 - $g^+ \cap \text{effects}^-(a) = \emptyset$ and $g^- \cap \text{effects}^+(a) = \emptyset$

STRIPS

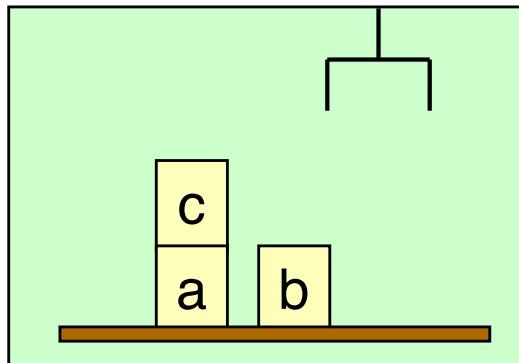
- $\pi \leftarrow$ the empty plan
- do a modified backward search from g
 - ◆ each new set of subgoals is just $\text{precond}(a)$
 - ◆ whenever you find an action that's executable in the current state, then go forward on the current search path as far as possible, executing actions and appending them to π
 - ◆ repeat until all goals are satisfied

$$\pi = \langle a_6, a_4 \rangle$$

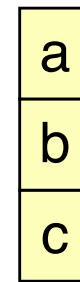
$$s = \gamma(\gamma(s_0, a_6), a_4)$$



The Sussman Anomaly

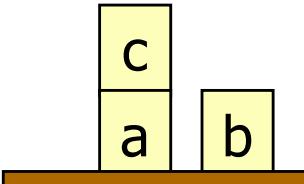


Initial state

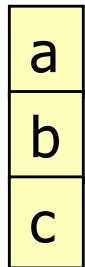
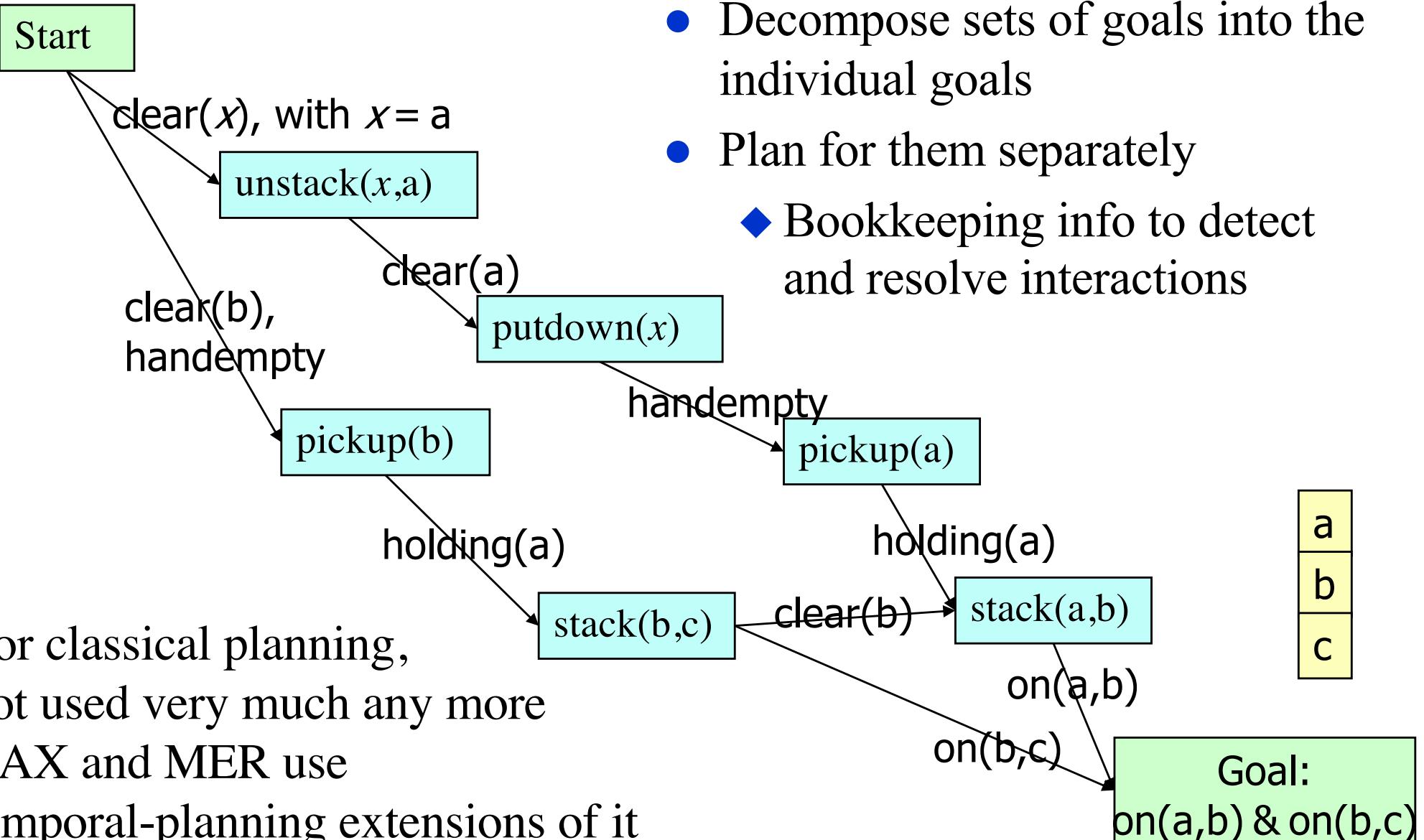


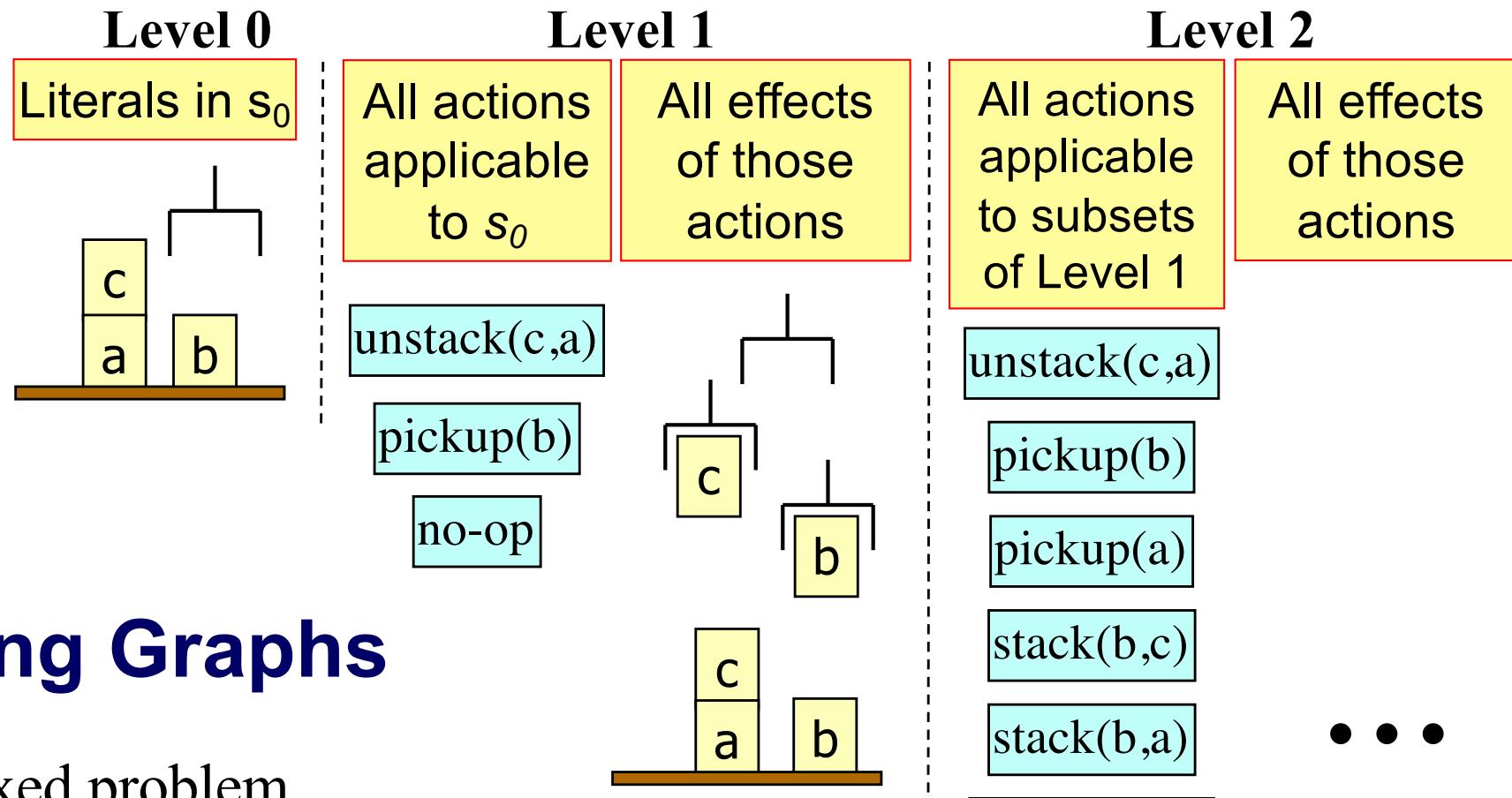
goal

- On this problem, STRIPS can't produce an irredundant solution



Plan-Space Planning (UCPOP)





Planning Graphs

- Relaxed problem
[Blum & Furst, 1995]
- Apply all applicable actions at once
- Next “level” contains all the effects of all of those actions

• • •

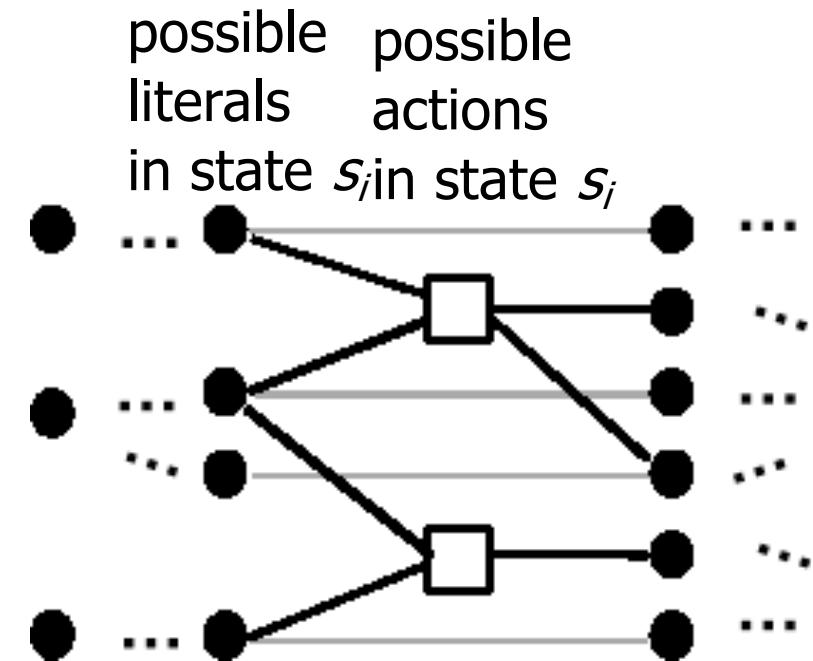
Graphplan

procedure Graphplan:

- for $k = 0, 1, 2, \dots$
 - ◆ *Graph expansion:*
 - » create a “planning graph” that contains k “levels”
 - ◆ Check whether the planning graph satisfies a necessary (but insufficient) condition for plan existence

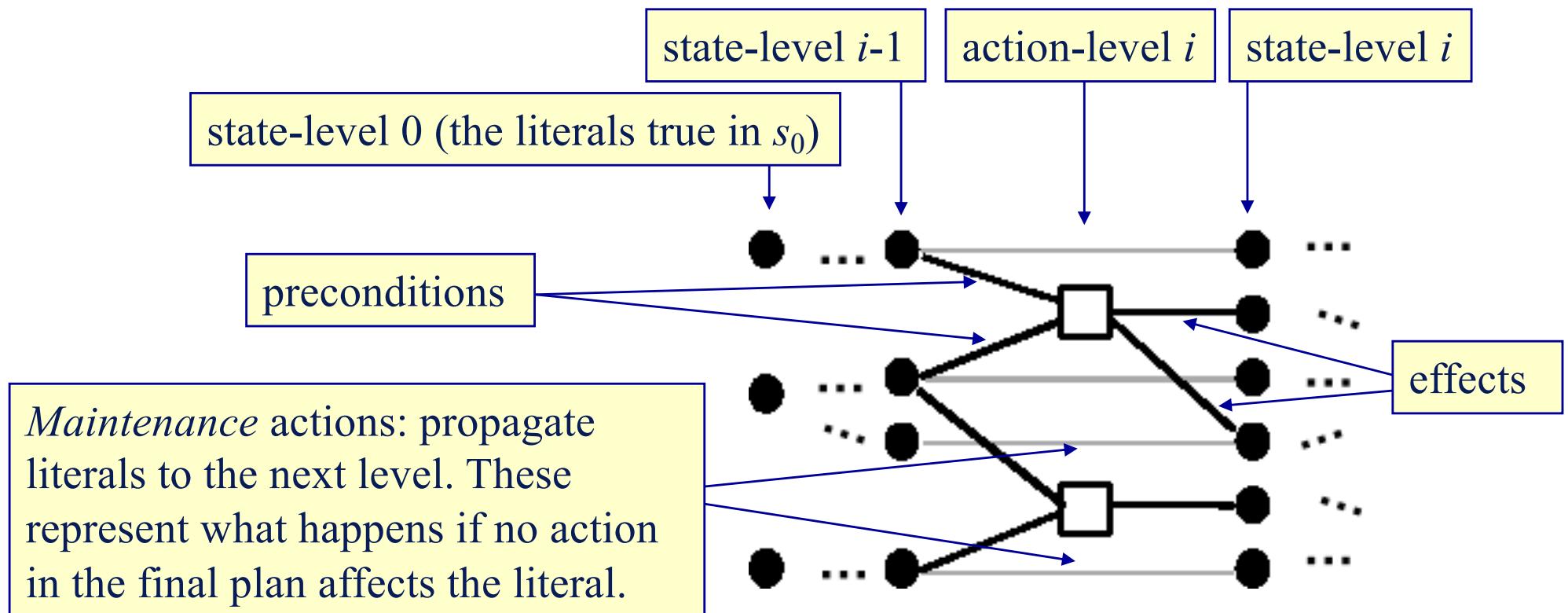
relaxed problem

- ◆ If it does, then
 - » do *solution extraction:*
 - backward search,
modified to consider
only the actions in
the planning graph
 - if we find a solution,
then return it



The Planning Graph

- Alternating layers of ground literals and actions
 - All actions that might possibly occur at each time step
 - All of the literals asserted by those actions



Example

» due to Dan Weld (U. of Washington)

- Suppose you want to prepare dinner as a surprise for your sweetheart (who is asleep)

$$s_0 = \{\text{garbage}, \text{cleanHands}, \text{quiet}\}$$

$$g = \{\text{dinner}, \text{present}, \neg\text{garbage}\}$$

Action	Preconditions	Effects
cook()	cleanHands	dinner
wrap()	quiet	present
carry()	<i>none</i>	$\neg\text{garbage}, \neg\text{cleanHands}$
dolly()	<i>none</i>	$\neg\text{garbage}, \neg\text{quiet}$

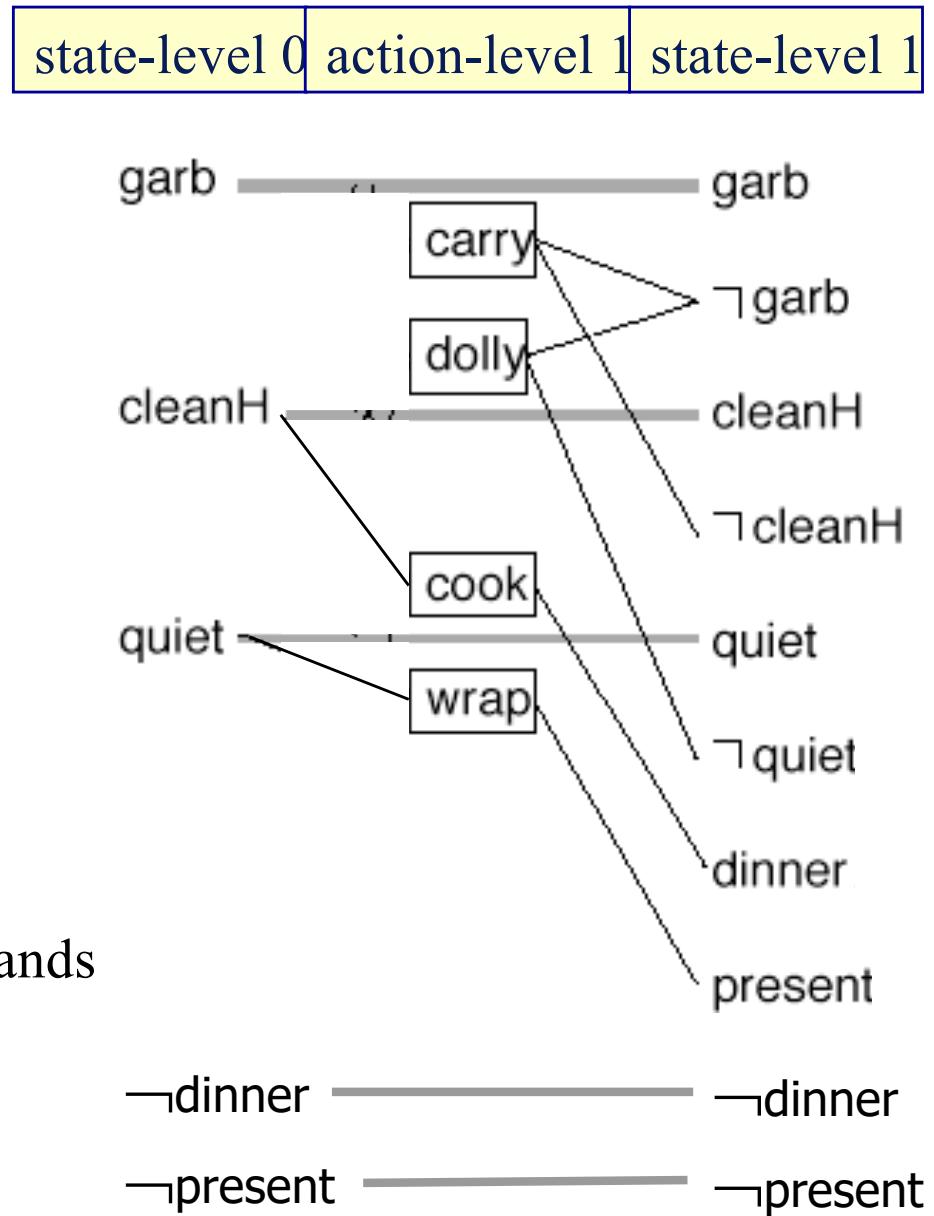
Also have the maintenance actions: one for each literal

Example (continued)

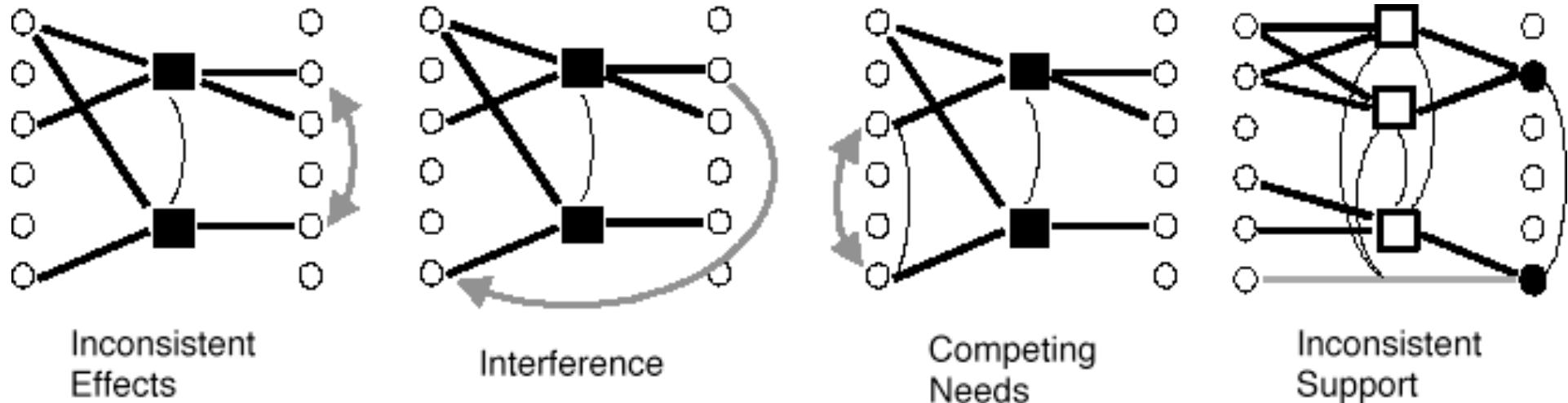
- state-level 0:
 $\{\text{all atoms in } s_0\} \cup$
 $\{\text{negations of all atoms not in } s_0\}$
 - action-level 1:
 $\{\text{all actions whose preconditions are satisfied and non-mutex in } s_0\}$
 - state-level 1:
 $\{\text{all effects of all of the actions in action-level 1}\}$

Action	Preconditions	Effects
cook()	cleanHands	dinner
wrap()	quiet	present
carry()	<i>none</i>	\neg garbage
dolly()	<i>none</i>	\neg garbage

Also have the maintenance actions



Mutual Exclusion



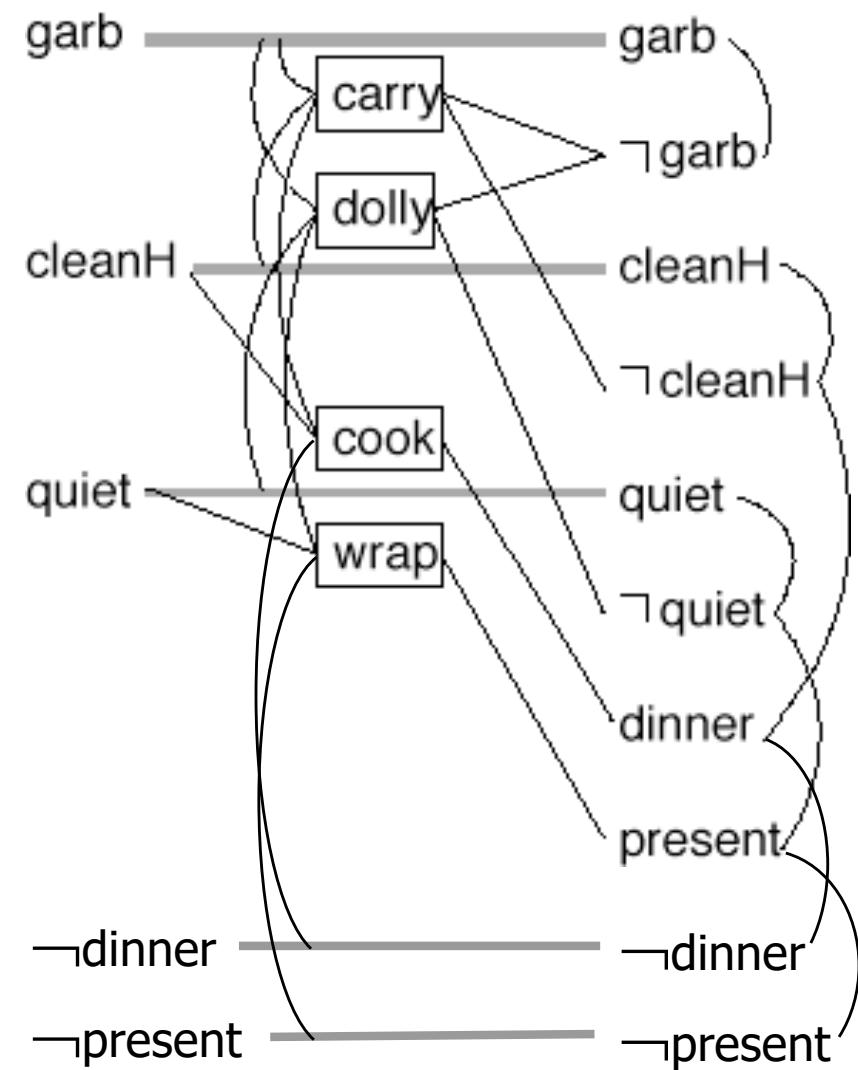
- Two actions at the same action-level are mutex if
 - ◆ *Inconsistent effects*: an effect of one negates an effect of the other
 - ◆ *Interference*: one deletes a precondition of the other
 - ◆ *Competing needs*: **they have mutually exclusive preconditions**
- Otherwise they don't interfere with each other
 - ◆ Both may appear in a solution plan
- Two literals at the same state-level are mutex if
 - ◆ *Inconsistent support*: one is the negation of the other, **or all ways of achieving them are pairwise mutex**

Example (continued)

- Augment the graph to indicate mutexes
- carry* is mutex with the maintenance action for *garbage* (inconsistent effects)
- dolly* is mutex with *wrap*
 - interference
- $\sim\text{quiet}$ is mutex with *present*
 - inconsistent support
- each of *cook* and *wrap* is mutex with a maintenance operation

Action	Preconditions	Effects
cook()	cleanHands	dinner
wrap()	quiet	present
carry()	<i>none</i>	$\neg\text{garbage}$, $\neg\text{cleanHands}$
dolly()	<i>none</i>	$\neg\text{garbage}$, $\neg\text{quiet}$
Also have the maintenance actions		

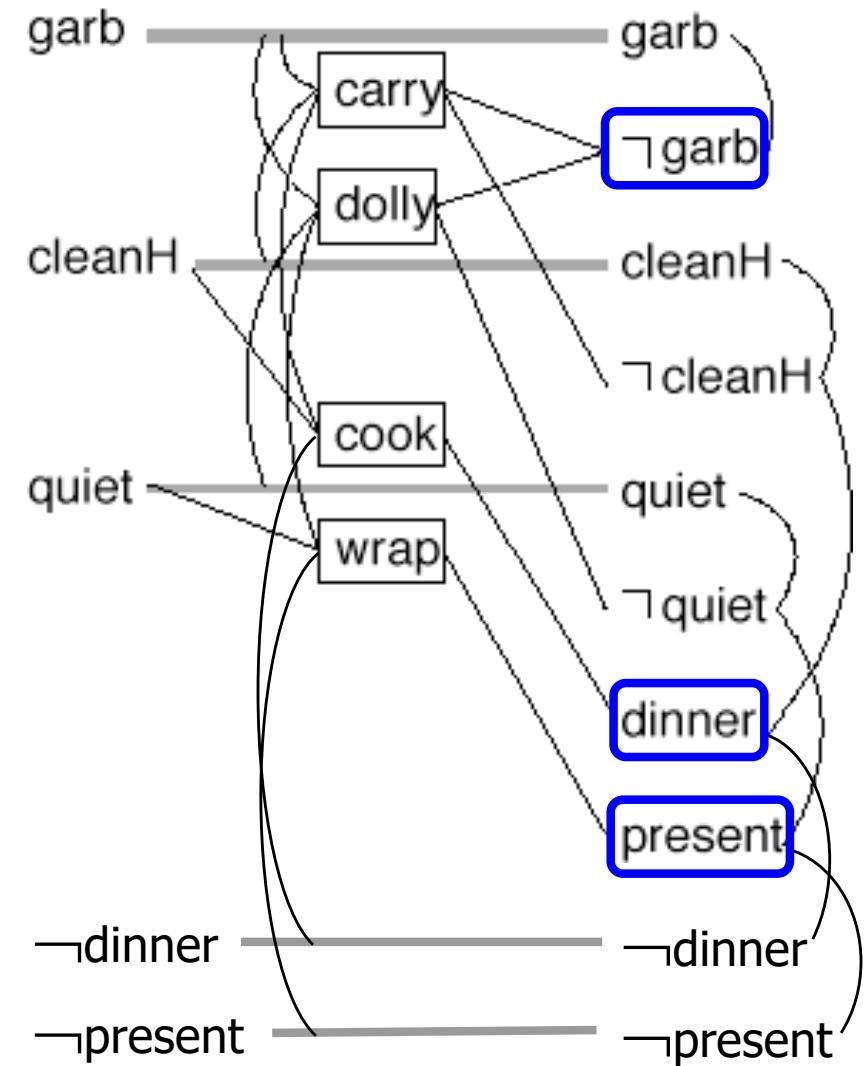
state-level 0	action-level 1	state-level 1
---------------	----------------	---------------



Example (continued)

- Check to see whether there's a possible plan
- Recall that the goal is
 - ◆ $\{\neg\text{garbage}, \text{dinner}, \text{present}\}$
- Note that
 - ◆ All are possible in s_1
 - ◆ None are mutex with each other
- Thus, there's a chance that a plan exists
- Try to find it
 - ◆ Solution extraction

state-level 0	action-level 1	state-level 1
---------------	----------------	---------------



Solution Extraction

The set of goals we are trying to achieve

The level of the state s_j

procedure Solution-extraction(g, j)

if $j=0$ then return the solution

for each literal l in g

nondeterministically choose an action

to use in state s_{j-1} to achieve l

if any pair of chosen actions are mutex

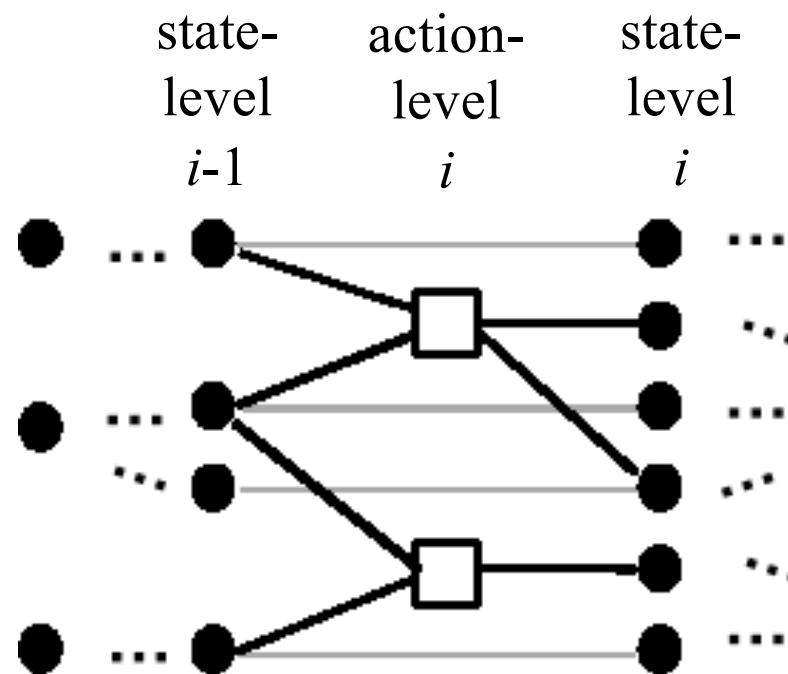
then backtrack

$g' := \{\text{the preconditions of the chosen actions}\}$

Solution-extraction($g', j-1$)

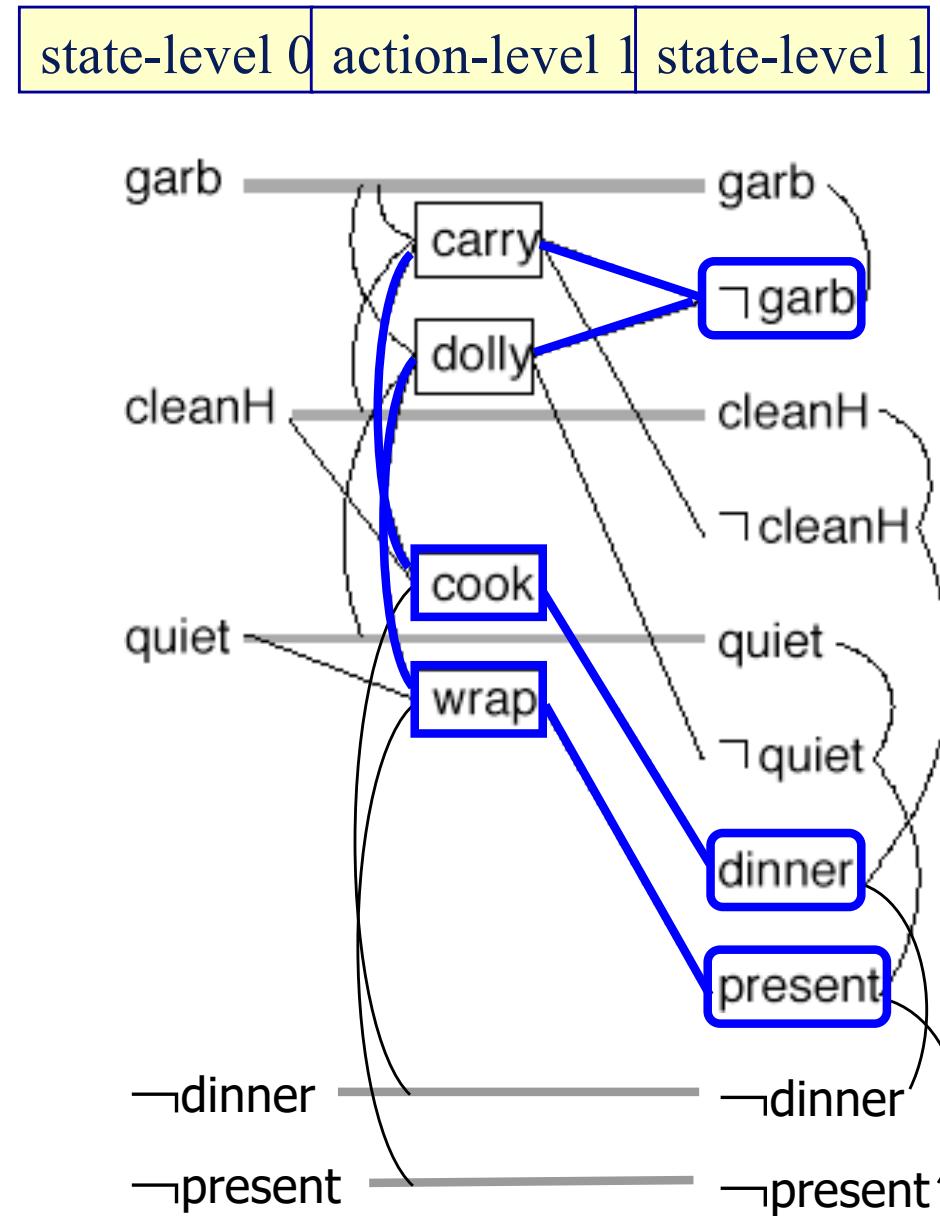
end Solution-extraction

A real action or a maintenance action



Example (continued)

- Two sets of actions for the goals at state-level 1
- Neither works: both sets contain actions that are mutex



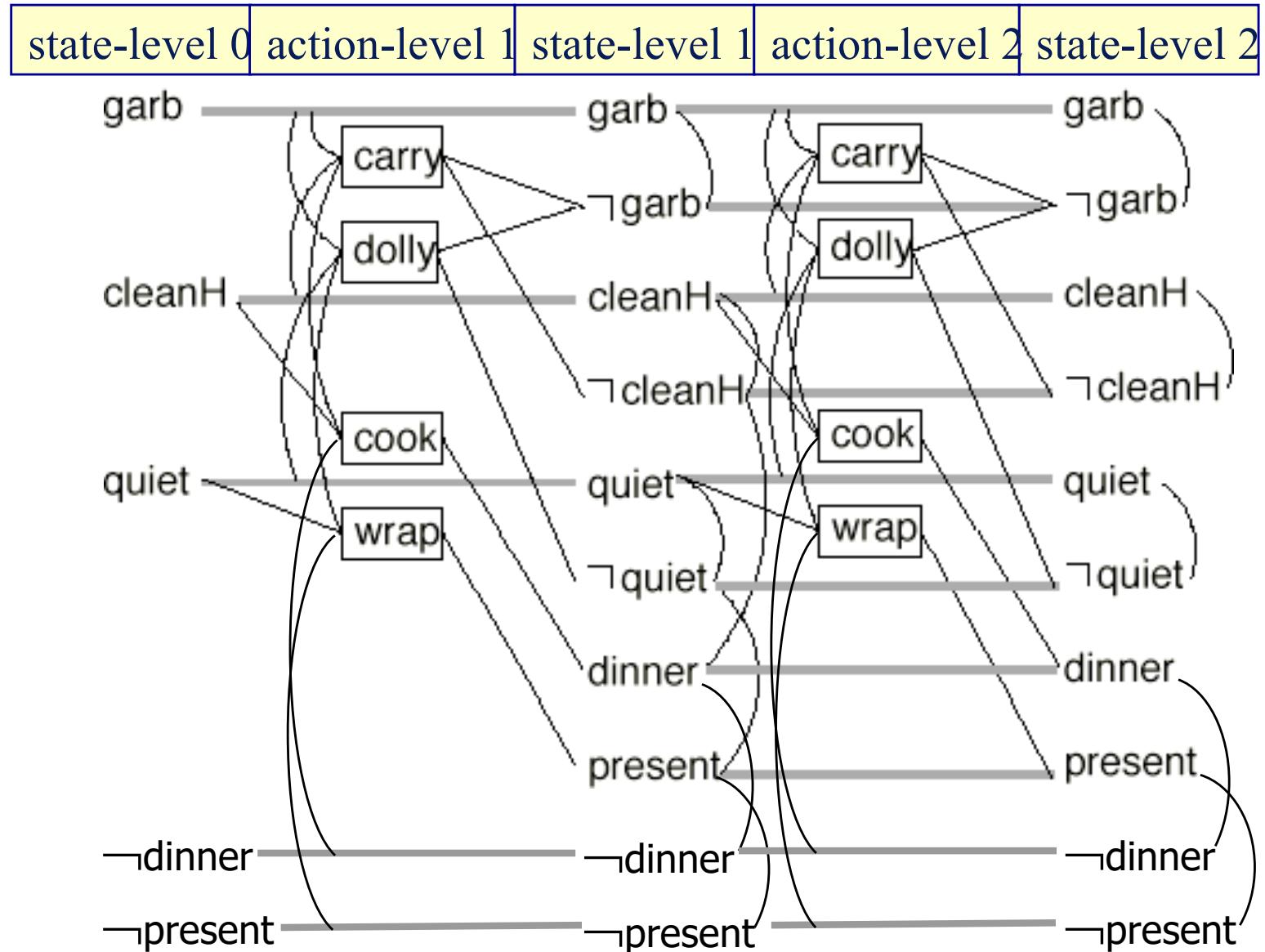
Recall what the algorithm does

procedure Graphplan:

- for $k = 0, 1, 2, \dots$
 - ◆ *Graph expansion:*
 - » create a “planning graph” that contains k “levels”
 - ◆ Check whether the planning graph satisfies a necessary (but insufficient) condition for plan existence
 - ◆ If it does, then
 - » do *solution extraction:*
 - backward search,
modified to consider
only the actions in
the planning graph
 - if we find a solution,
then return it

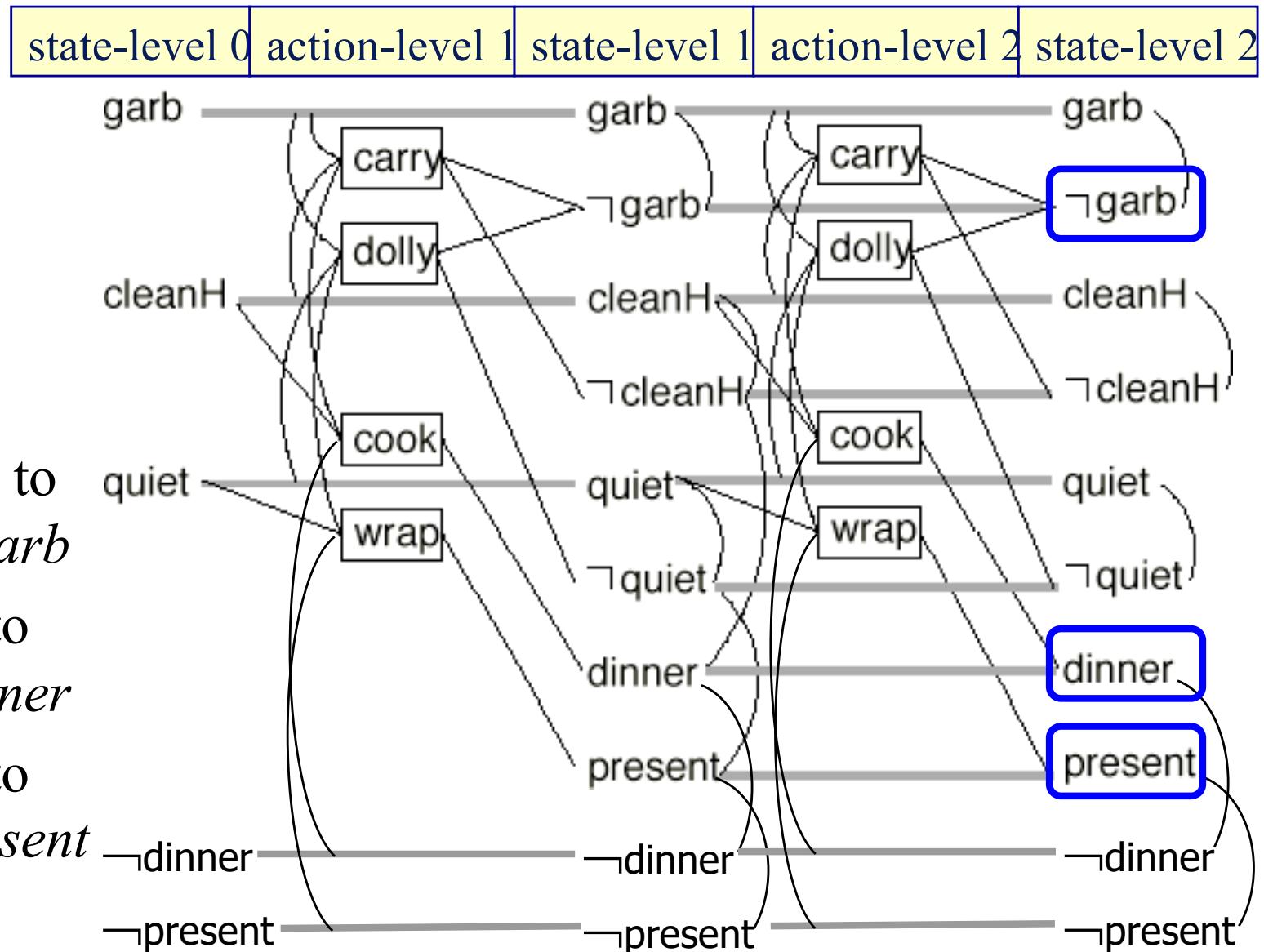
Example (continued)

- Go back and do more graph expansion
- Generate another action-level and another state-level



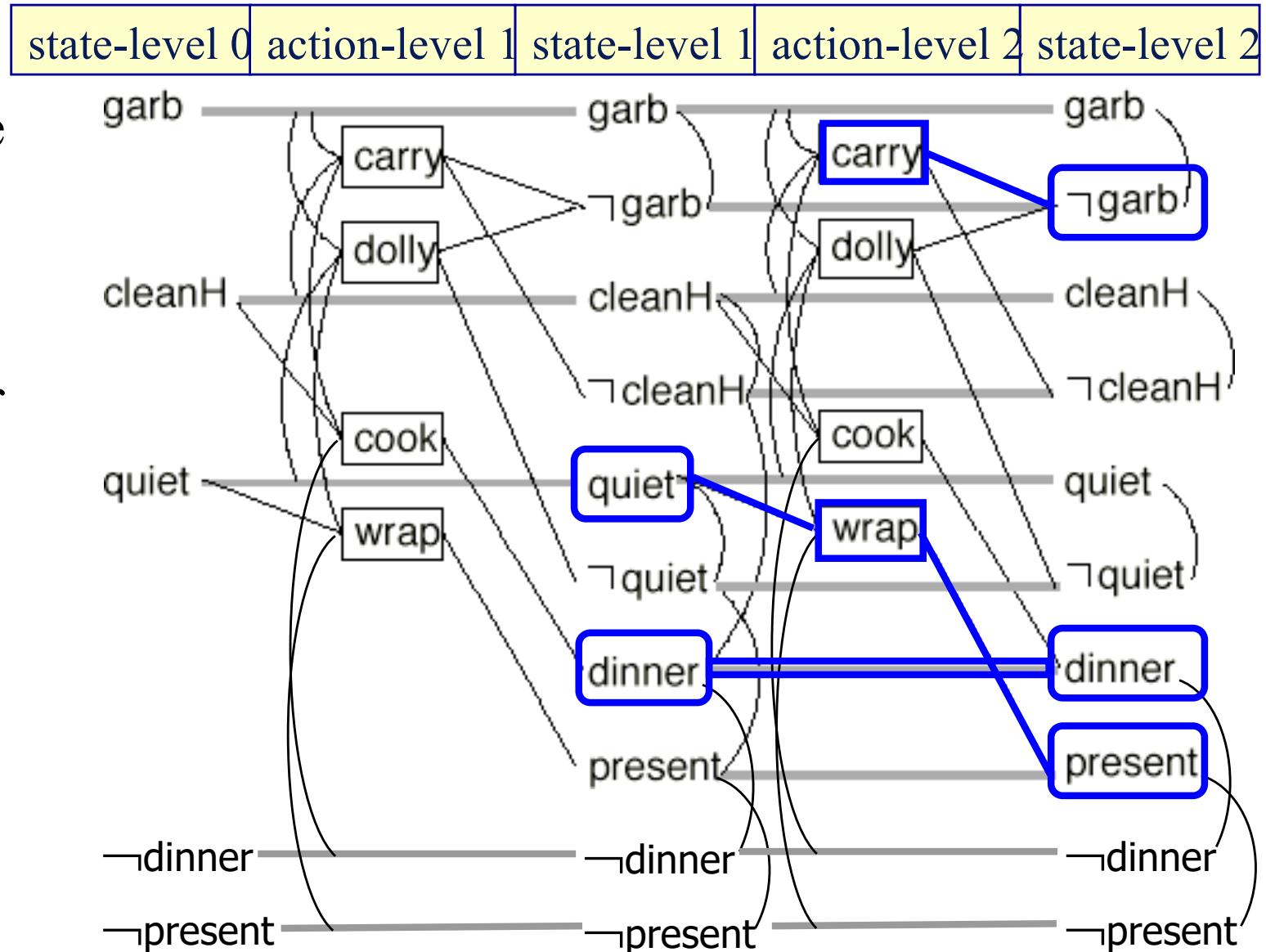
Example (continued)

- Solution extraction
- Twelve combinations at level 4
 - ◆ Three ways to achieve *¬garb*
 - ◆ Two ways to achieve *dinner*
 - ◆ Two ways to achieve *present*



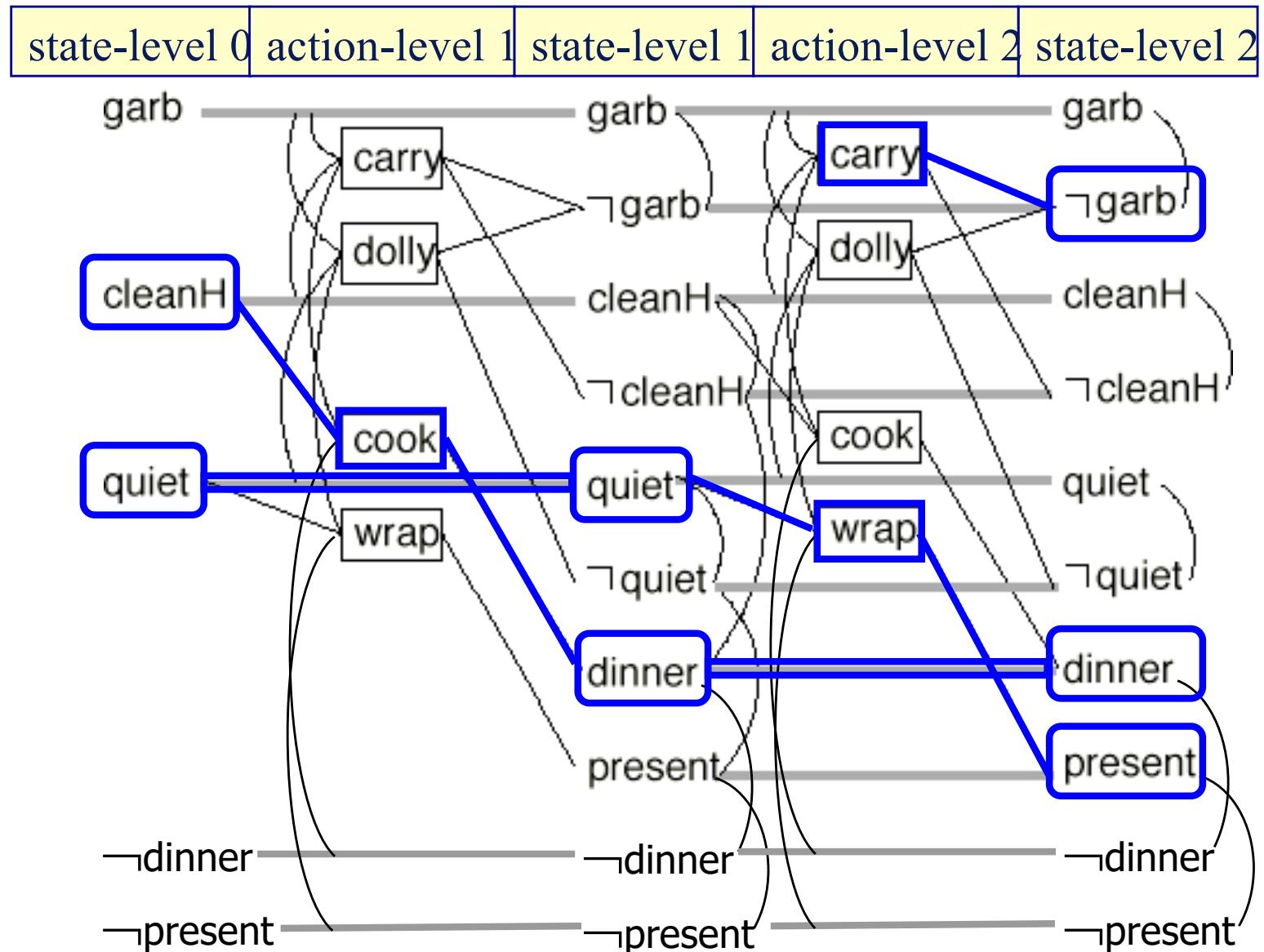
Example (continued)

- Several of the combinations look OK at level 2
- Here's one of them



Example (continued)

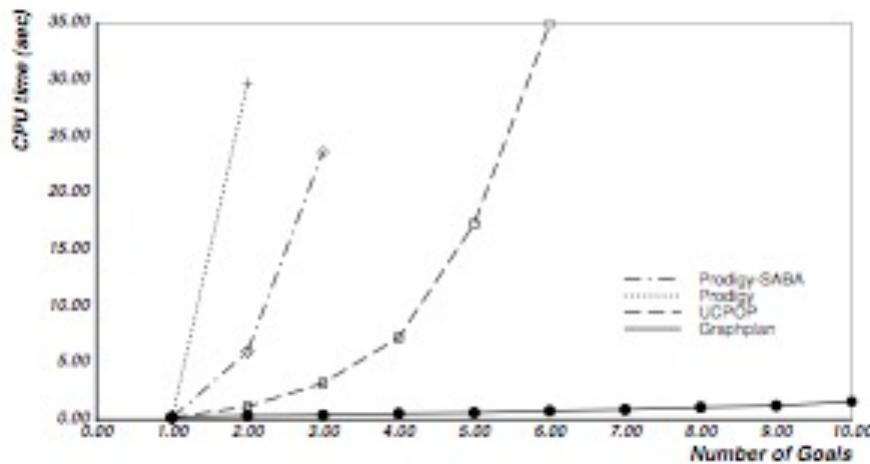
- Call Solution-Extraction recursively at level 2
- It succeeds
- Solution whose *parallel length* is 2



Comparison with PSP

- Advantage: the backward-search part of Graphplan—which is the hard part—will only look at the actions in the planning graph
 - ◆ smaller search space than PSP; thus faster
- Disadvantage: to generate the planning graph, Graphplan creates a huge number of ground atoms
 - ◆ Many of them may be irrelevant
- Can alleviate (but not eliminate) this problem by assigning data types to the variables and constants
 - ◆ Only instantiate variables to terms of the same data type

Results



- Graphplan avoids unnecessary backtracking of the other planners.
- Computation time stays relatively flat even as the number of goals increases

Application Area: AI in Games

- Why do AI in games?

Application Area: AI in Games

- Why do AI in games?
- Games are good for AI
 - Wonderful testing environment—more complicated than the simplified grid world problems but with a manageable amount of complexity
 - Sensor problem is trivial compared to robotics
 - Rich opportunities for human interaction
- AI is good for games (and game developers).
 - Create believable and autonomous opponents
 - Improve on commonly used software development techniques like finite state automata and scripting
 - Create variability in bot behavior
 - Good for other in-game tasks such as directing narrative flow and camera placement

AI in Games

- Which AI techniques are applicable to these problems?

IEEE COG Competitions 2021

https://ieee-cog.org/2021/#competitions_section

AI Snakes

Space Invaders

Bot Bowl III

Carle's Game Competition

ColorShapeLinks AI

DOTA2 5v5

Fighting Game AI

General Strategy Game AI

General Video Game AI

Tactile Games

Ludii

Microorts

Starcraft

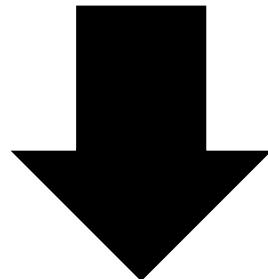
Strategy Card Games

Angry Birds

These competitions can make good final project ideas!

Problems

- How do we create long-lasting bots that act autonomously over days in games?
 - J-P. Kelly, A. Boea, and S. Koenig, Offline planning with HTNs in Video Games, AIIDE 2008



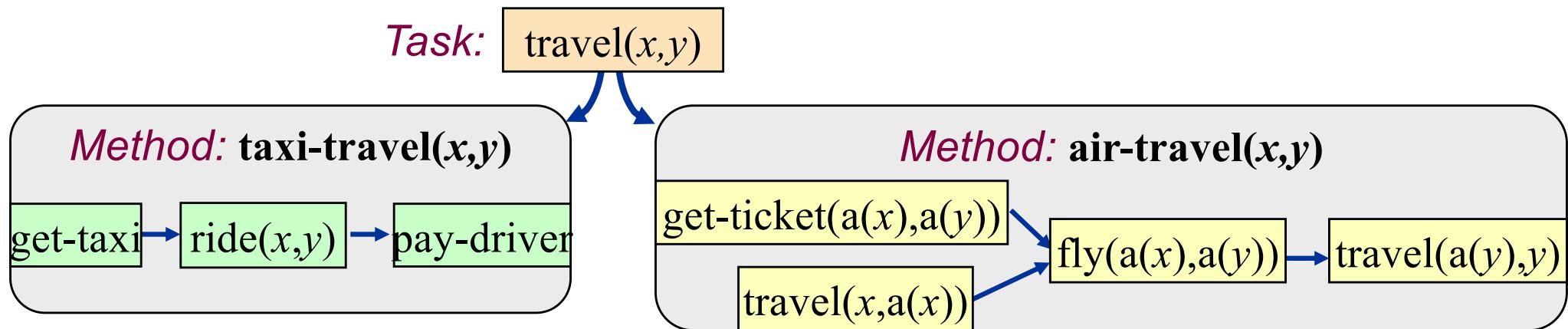
HTN Planning

HTN Planning

- We may already have an idea how to go about solving problems in a planning domain
- Example: travel to a destination that's far away:
 - ◆ Domain-independent planner:
 - » many combinations of vehicles and routes
 - ◆ Experienced human: small number of “recipes”
 - e.g., flying:
 1. buy ticket from local airport to remote airport
 2. travel to local airport
 3. fly to remote airport
 4. travel to final destination
- How to enable planning systems to make use of such recipes?

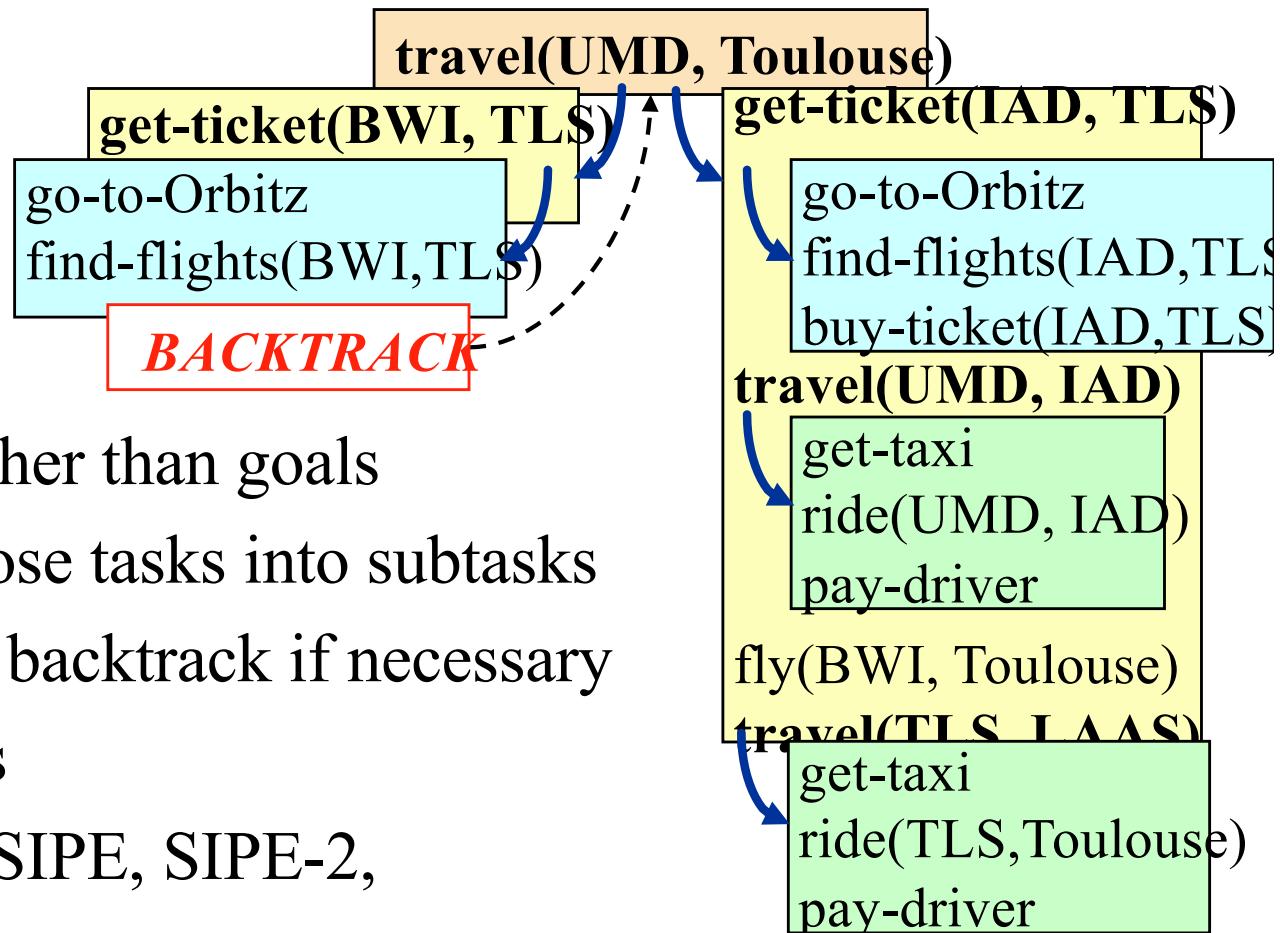
Two Approaches

- Control rules :
 - ◆ Write rules to prune every action that *doesn't* fit the recipe
- Hierarchical Task Network (HTN) planning:
 - ◆ Describe the actions and subtasks that *do* fit the recipe



HTN Planning

- Problem reduction
 - ◆ *Tasks* (activities) rather than goals
 - ◆ *Methods* to decompose tasks into subtasks
 - ◆ Enforce constraints, backtrack if necessary
- Real-world applications
- Noah, Nonlin, O-Plan, SIPE, SIPE-2, SHOP, SHOP2



Methods

- Totally ordered method: a 4-tuple

$$m = (\text{name}(m), \text{task}(m), \text{precond}(m), \text{subtasks}(m))$$

- ◆ $\text{name}(m)$: an expression of the form $n(x_1, \dots, x_n)$

» x_1, \dots, x_n are parameters - variable symbols

- ◆ $\text{task}(m)$: a nonprimitive task

- ◆ $\text{precond}(m)$: preconditions (literals)

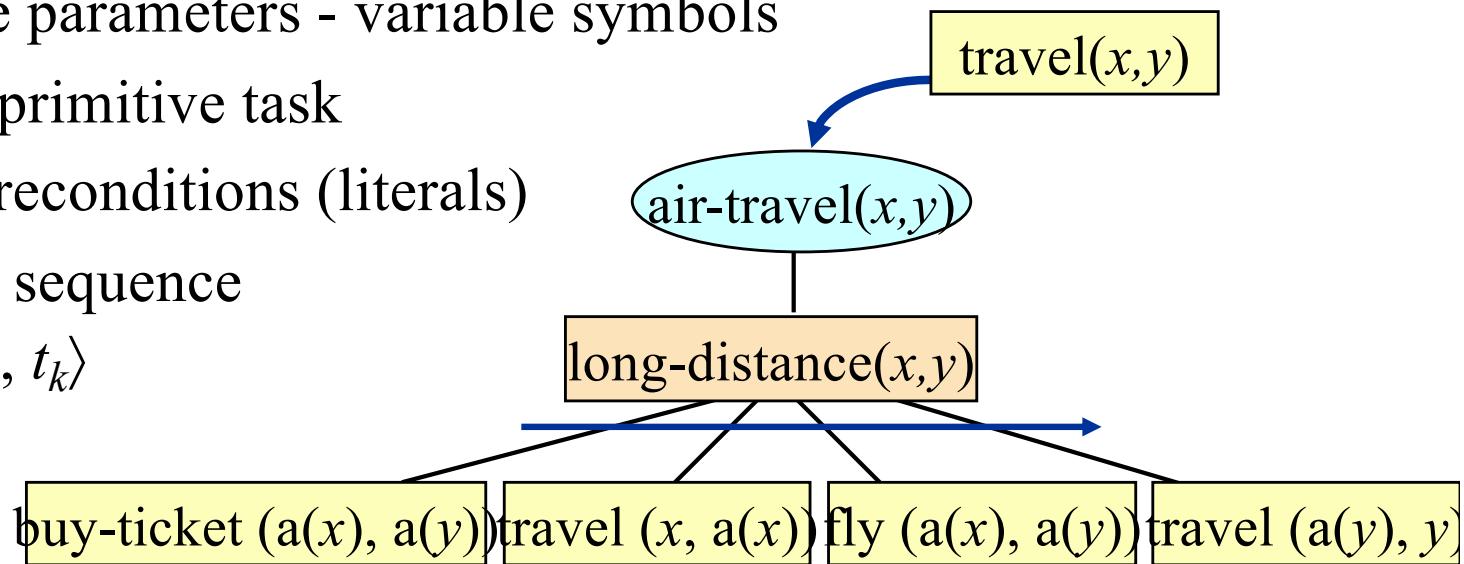
- ◆ $\text{subtasks}(m)$: a sequence
of tasks $\langle t_1, \dots, t_k \rangle$

$\text{air-travel}(x, y)$

task: $\text{travel}(x, y)$

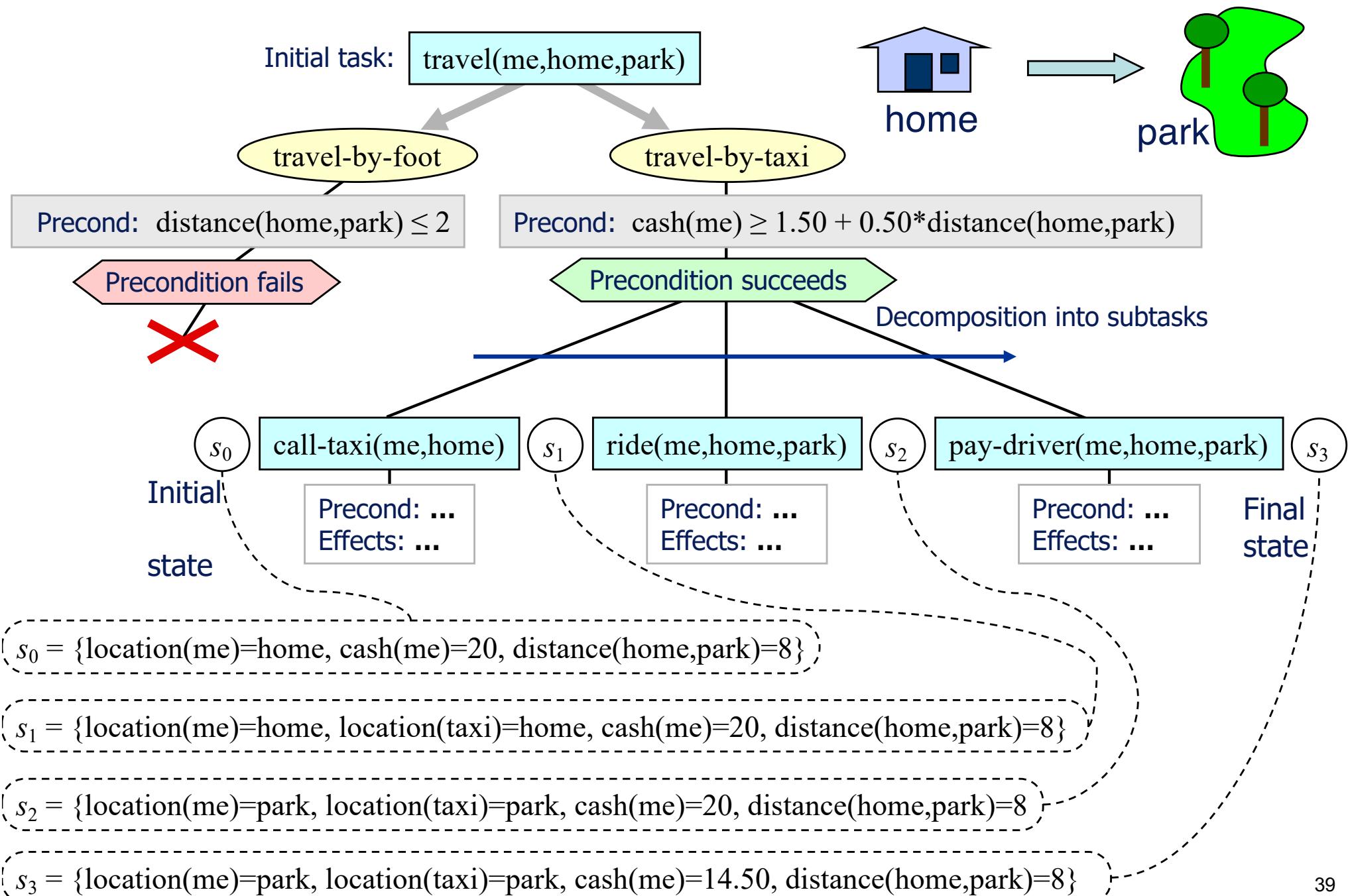
precond: $\text{long-distance}(x, y)$

subtasks: $\langle \text{buy-ticket}(a(x), a(y)), \text{travel}(x, a(x)), \text{fly}(a(x), a(y)),$
 $\text{travel}(a(y), y) \rangle$

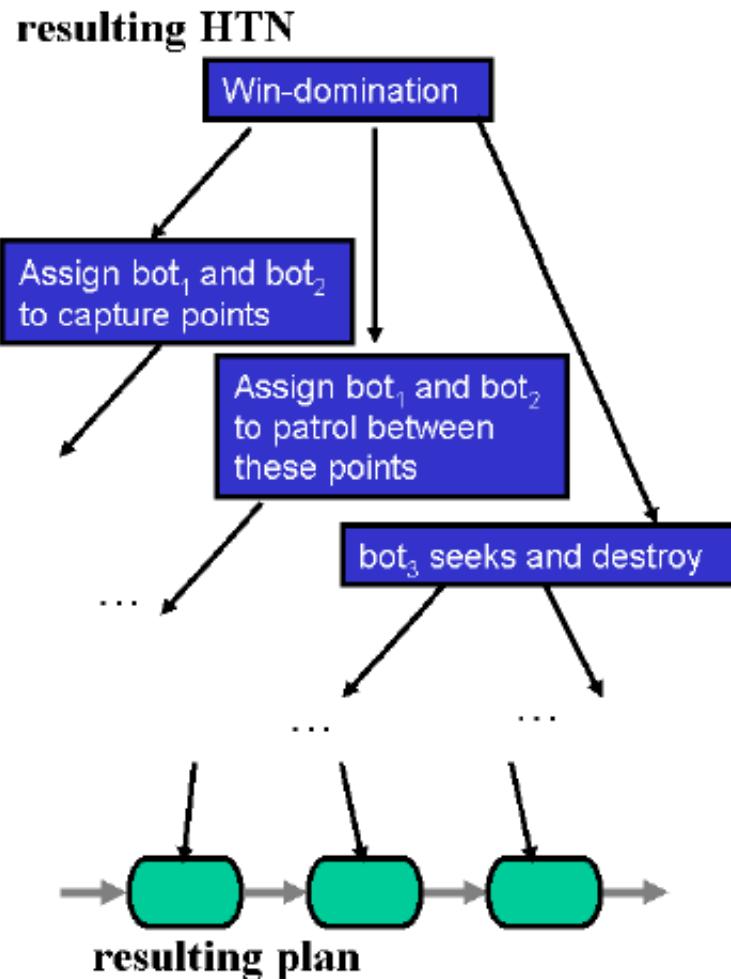


Planning Problem:

I am at home, I have \$20,
I want to go to a park 8 miles away



Hierarchical Task Network

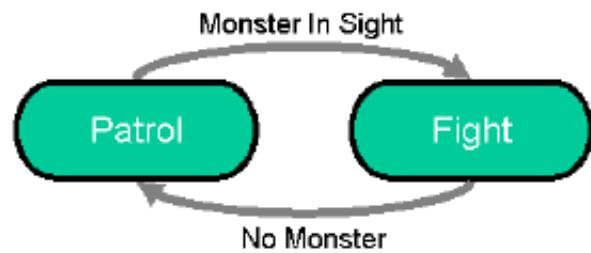


Planning Methods

- **Control All Points**
 - Task:
win-domination
 - Preconditions:
 - The team consists at least of 2 members
 - Subtasks:
 - Capture all domination points
 - Assign 2 members to patrol between those points
 - Assign remaining team to search and destroy task

Finite State Machine

FSM:



A resulting plan:



Planning Operators

•**Patrol**

- Preconditions:
No Monster
- Effects:
patrolled

•**Fight**

- Preconditions:
Monster in sight
- Effects:
No Monster

Results

NPCs	Time (sec.)	Nodes	Plan Length
3	6.12	1108	293
4	6.49	1452	381
5	7.07	1798	469
6	7.56	2142	557
7	7.94	2486	645
15	10.83	6332	1351
20	12.47	8062	1797
40	28.57	14982	3566

Table 1: Summary of results for 24 game hours when the problem size varies from 3 to 40 NPCs.

Game Hours	Time (sec.)	Nodes	Plan Length
4	12.31	2634	700
8	15.80	4570	1416
12	17.49	6566	2134
16	20.45	9086	2857
20	24.01	11126	3591
24	25.32	13614	4308

Table 2: Summary of results for 40 NPCs when the covered time period varies from 4 to 24 game hours.

- Planning scales well in numbers of hours and number of NPCs managed using the system
- What other metrics could the authors have evaluated?

Alternate Approach: Narratoria

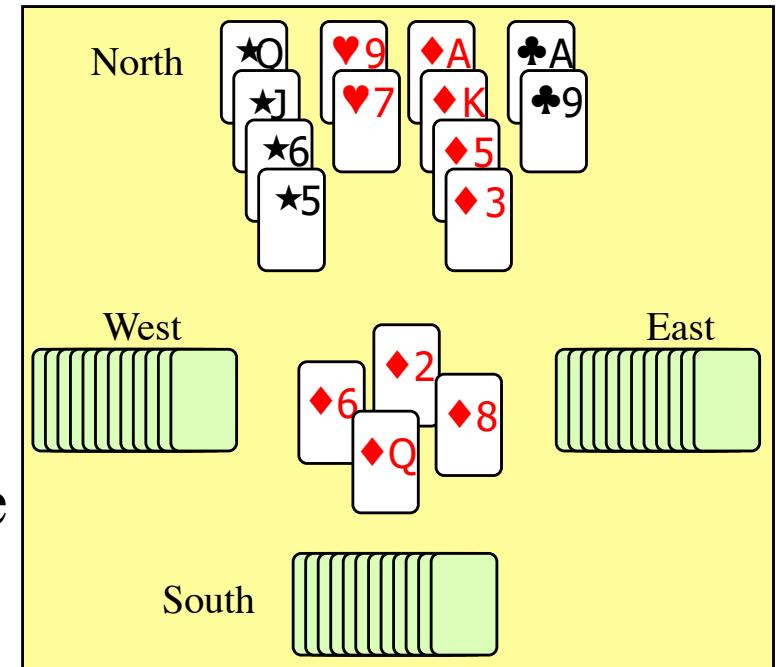
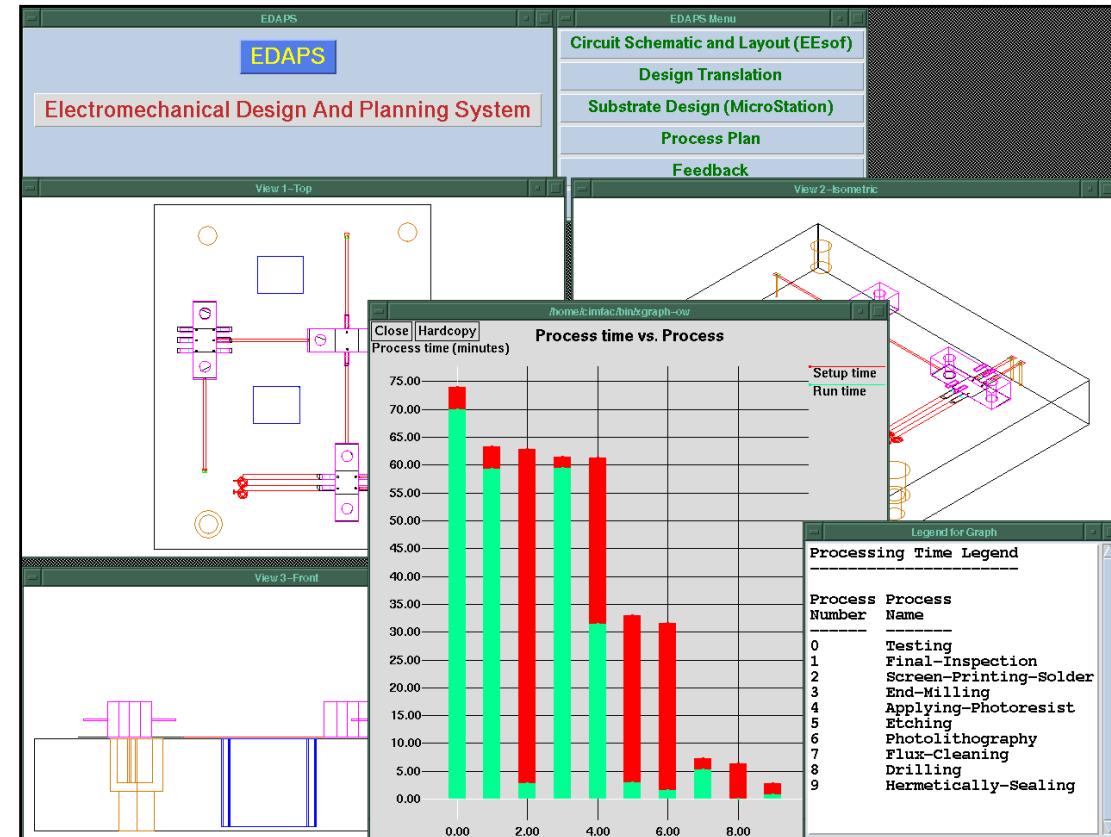
- Example of AI aiding the game designer rather than replacing the designer
- Narrative planning: relatively new area of AI in which planning is used to improve the story content of the game sequences.
- The goal is to make scenes memorable and interesting for the players rather than to optimize the NPC's combat capabilities.

<https://www.youtube.com/watch?v=rm473LmLcwU&feature=youtu.be>

Where does classical planning fail?

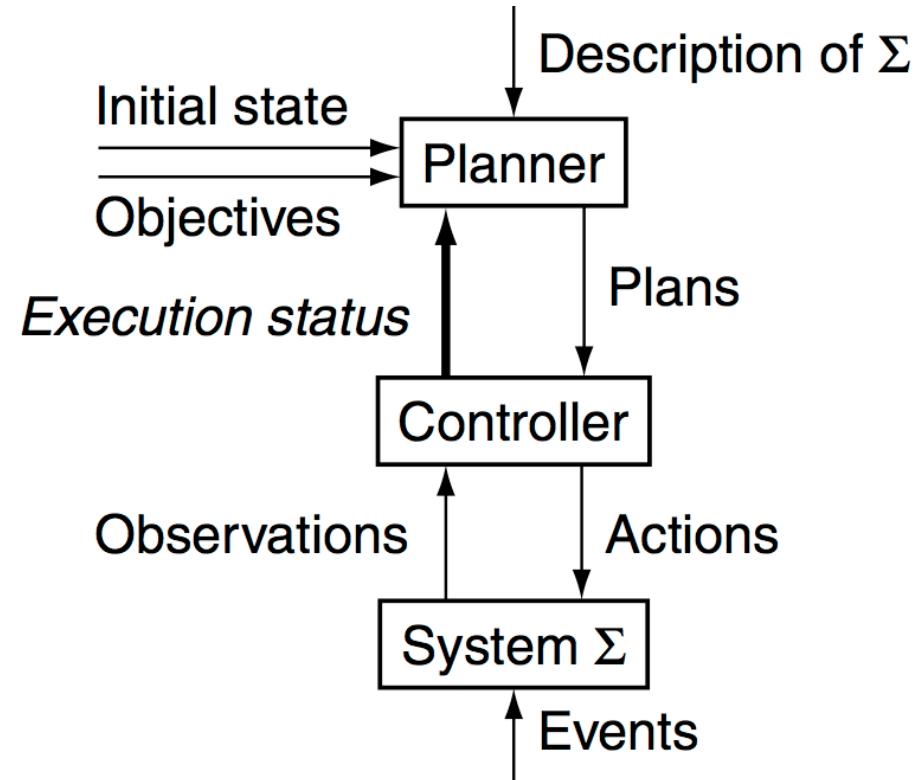
Example

- Typical characteristics of application domains
 - Dynamic world
 - Multiple agents
 - Imperfect/uncertain info
 - External info sources
 - users, sensors, databases
- Durations, time constraints, asynchronous actions
- Numeric computations
 - geometry, probability, etc.
- Classical planning excludes all of these



Relax the Assumptions

- Relax A2 (deterministic Σ):
 - ◆ Actions have more than one possible outcome
 - ◆ Seek policy or contingency plan
 - ◆ With probabilities:
 - » Discrete Markov Decision Processes (MDPs)
 - ◆ Without probabilities:
 - » Nondeterministic transition systems



$$\begin{aligned}\Sigma &= (S, A, E, \gamma) \\ S &= \{\text{states}\} \\ A &= \{\text{actions}\} \\ E &= \{\text{events}\} \\ \gamma: S \times (A \cup E) &\rightarrow 2^S\end{aligned}$$

International Planning Competition

- Held at the ICAPS conference to find the fastest planner
- <https://icaps23.icaps-conference.org/competitions/>
 - ◆ Classical
 - ◆ Learning
 - ◆ Probabilistic
 - ◆ Numeric
 - ◆ HTN

Next Time:

- Monte Carlo search for non classical planning domains

CAP6671 Intelligent Systems

Lecture 5: Markov Decision Processes; Monte Carlo Search

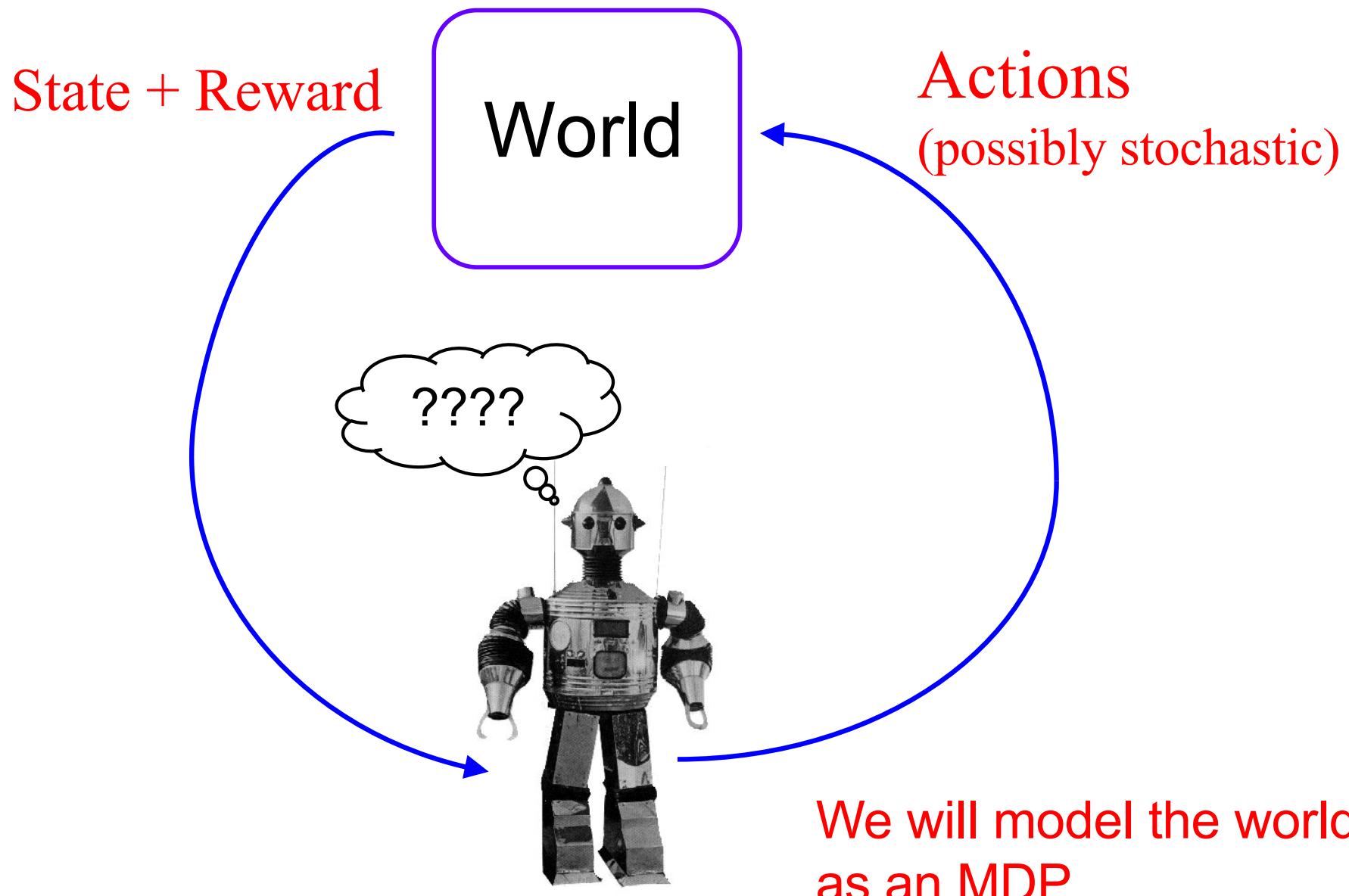
Instructor: Dr. Gita Sukthankar

Email: gitars@eeecs.ucf.edu

Planning

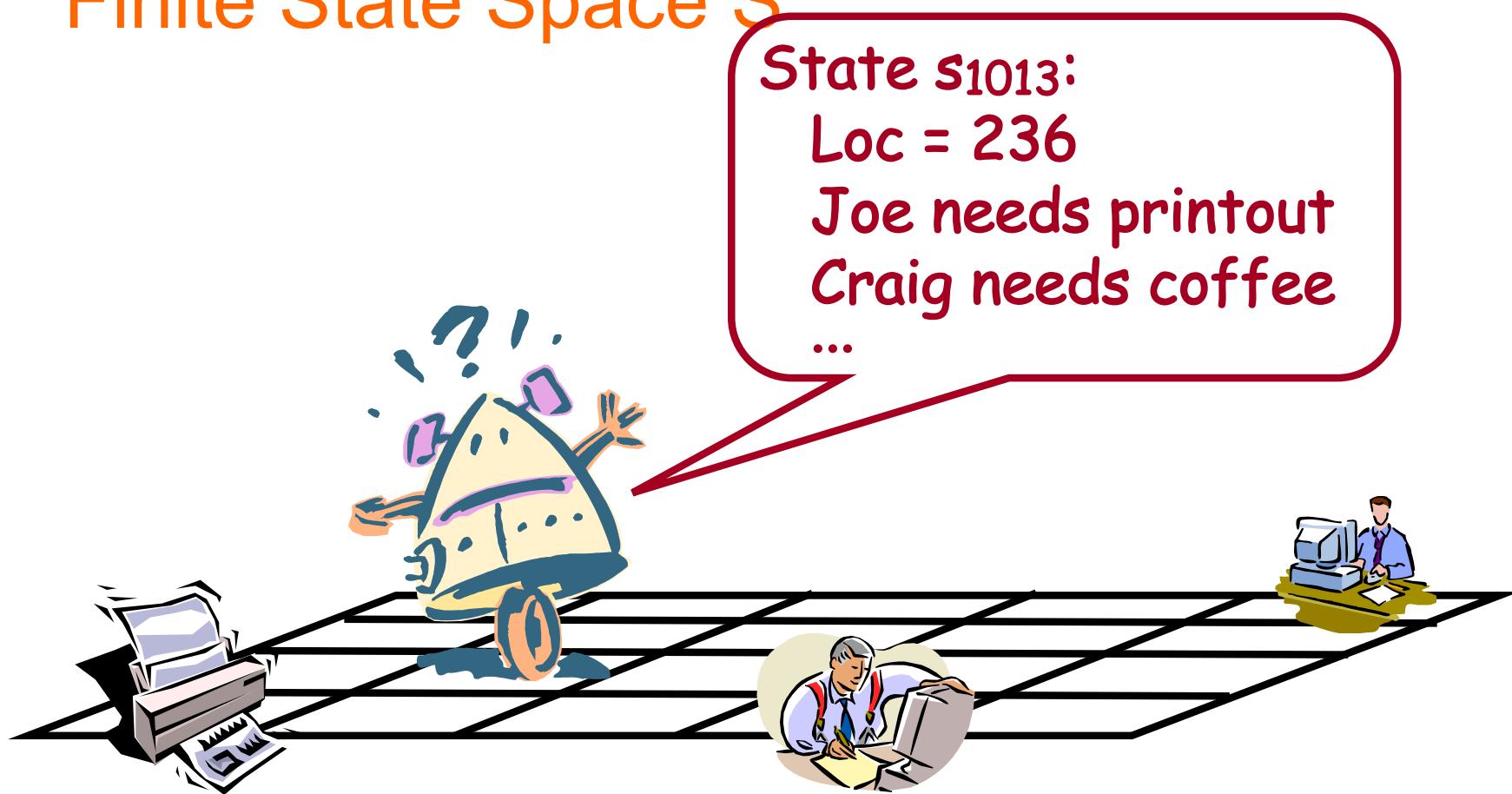
- Finished discussion of classical planning in deterministic worlds
 - Old assumption: actions have one outcome
 - New assumption: actions can have multiple outcomes (relax A2)
 - General model: discrete Markov Decision Process
-
- This will be the key planning model that we will assume for the rest of the semester.

Stochastic/Probabilistic Planning: Markov Decision Process (MDP) Model

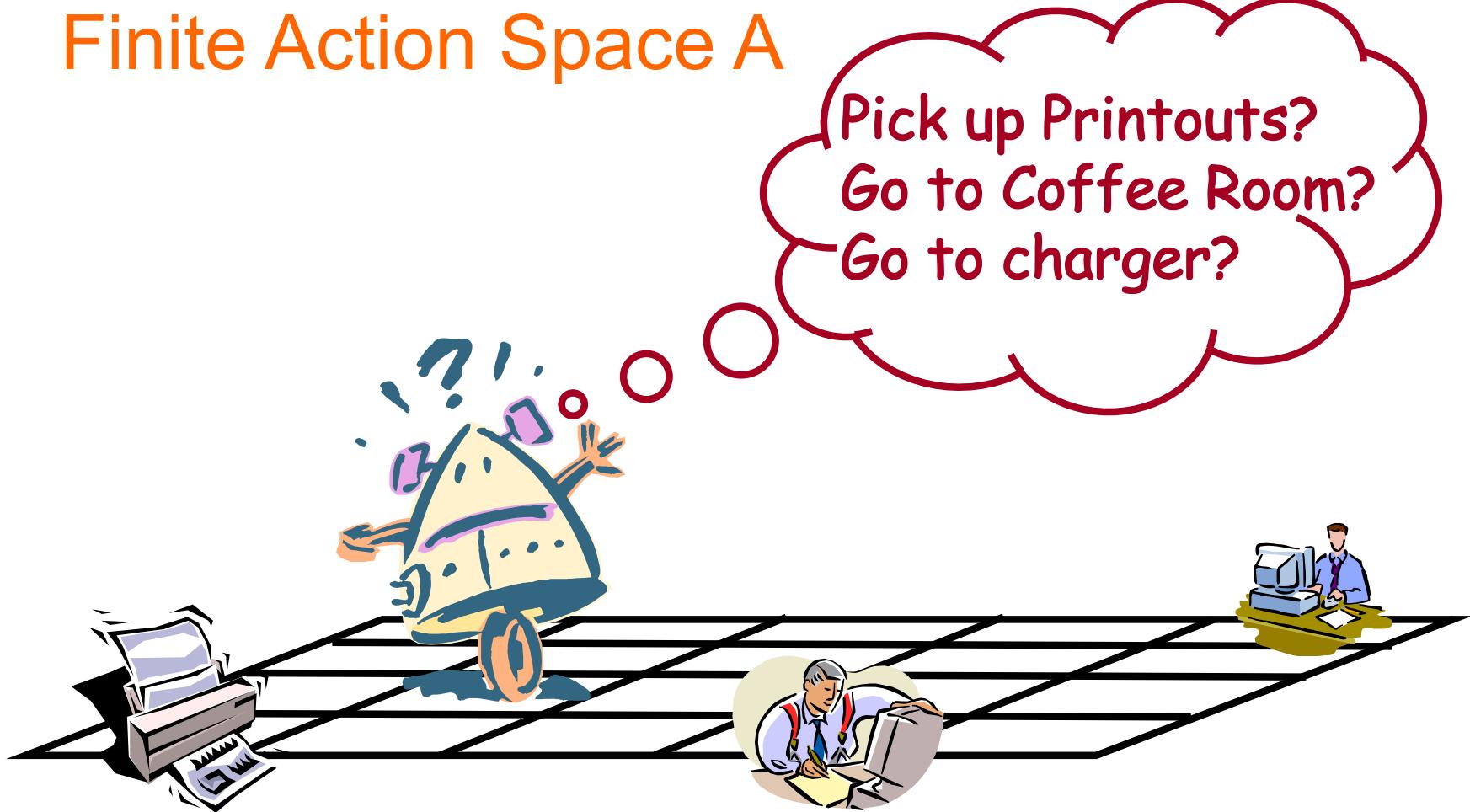


System Dynamics

Finite State Space S



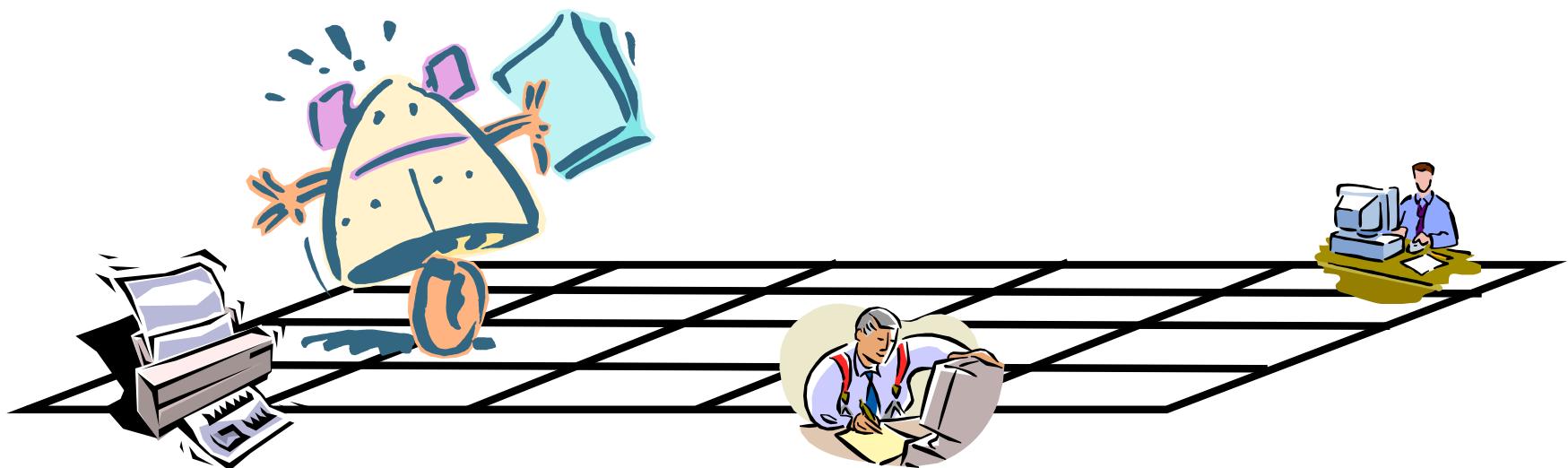
System Dynamics



System Dynamics

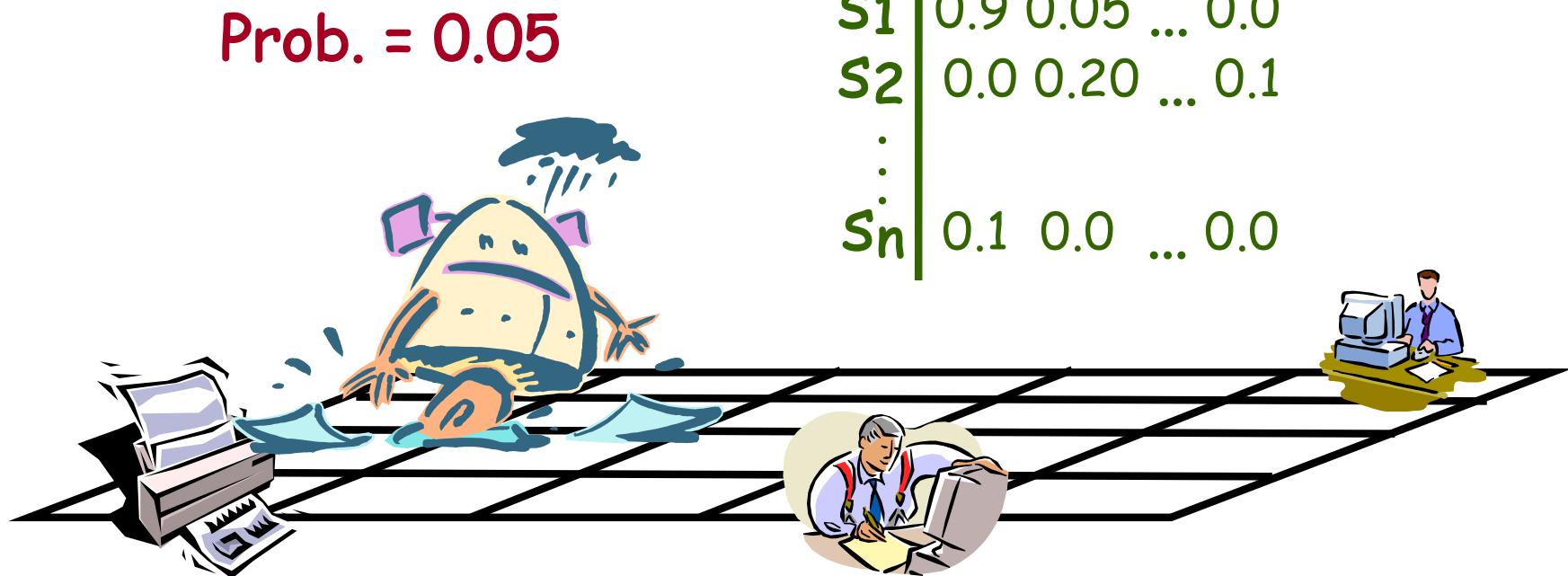
Transition Probabilities: $\Pr(s_i, a, s_j)$

Prob. = 0.95



System Dynamics

Transition Probabilities: $\Pr(s_i, a, s_k)$

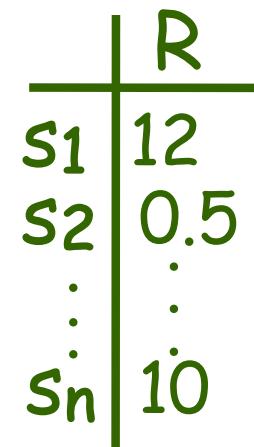
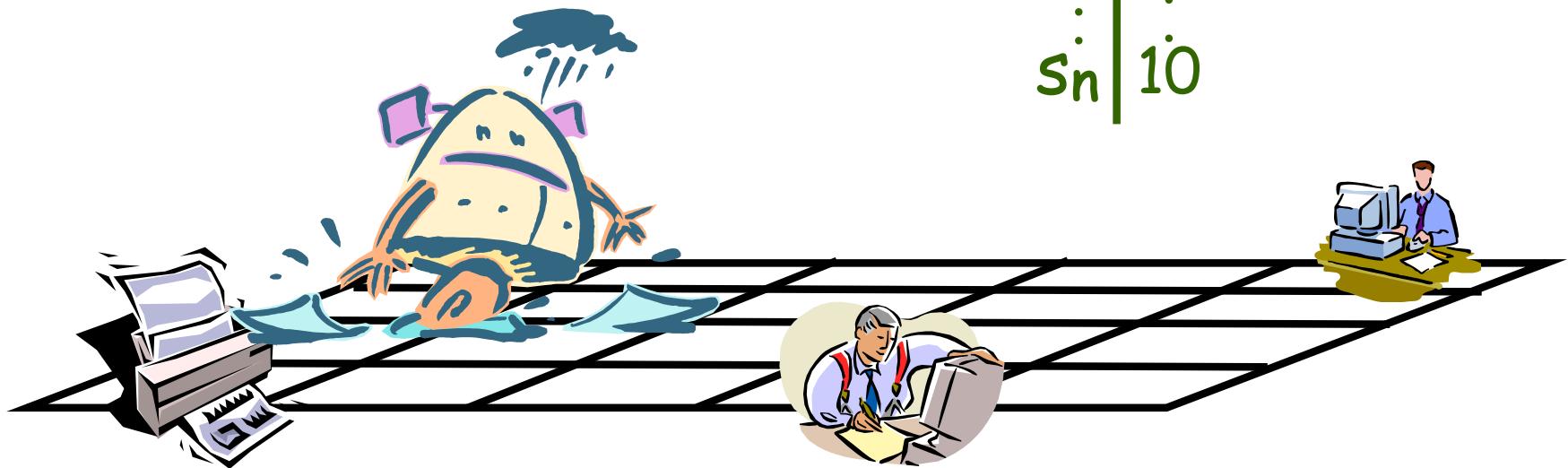


Reward Process

Reward Function: $R(s_i)$

- action costs possible

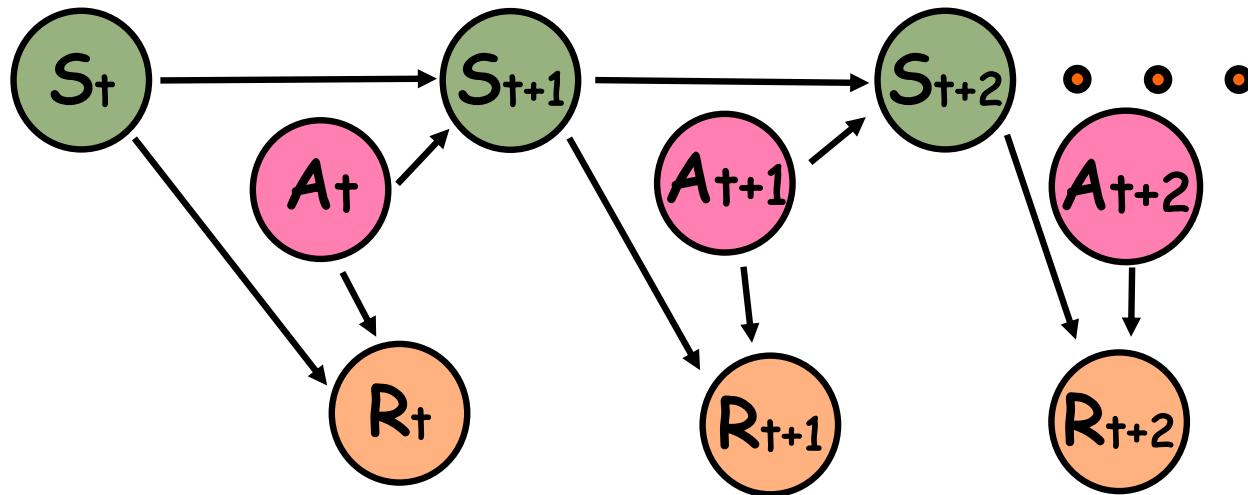
Reward = -10



Markov Decision Processes

- Agents' actions can affect the state of the world
- Most popular model is the Markov Decision Process
- An MDP has four components, S , A , R , Pr :
 - (finite) state set S ($|S| = n$)
 - (finite) action set A ($|A| = m$)
 - transition function $\text{Pr}(s,a,t)$
 - each $\text{Pr}(s,a,-)$ is a distribution over S
 - represented by set of $n \times n$ stochastic matrices
 - bounded, real-valued reward function $R(s)$
 - represented by an n -vector
 - can be generalized to include action costs: $R(s,a)$
 - can be stochastic (but replacable by expectation)
- Model easily generalizable to countable or continuous state and action spaces

Graphical View of MDP



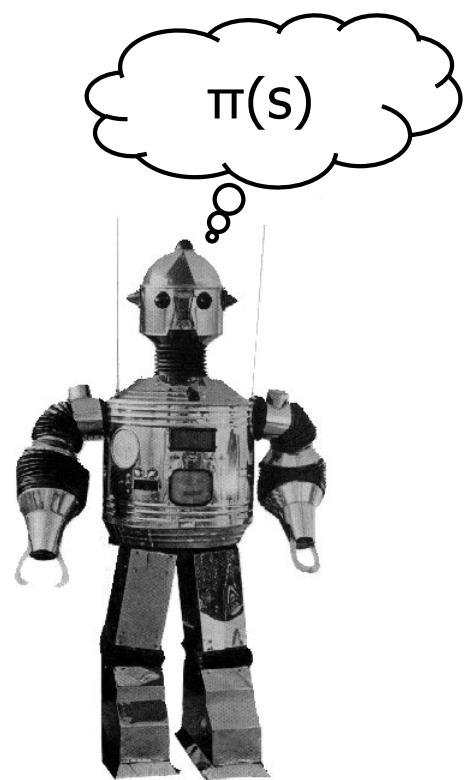
- **First-Order Markovian dynamics** (history independence)
 - ▲ Next state only depends on current state and current action
 - ▲ $\Pr(S_{t+1}|A_t, S_t, A_{t-1}, S_{t-1}, \dots, S^0) = \Pr(S_{t+1}|A_t, S_t)$
- **First-Order Markovian reward process**
 - ▲ Reward only depends on current state and action
 - ▲ $\Pr(R_t|A_t, S_t, A_{t-1}, S_{t-1}, \dots, S^0) = \Pr(R_t|A_t, S_t)$

Assumptions

- Markovian dynamics (history independence)
 - $\Pr(S^{t+1}|A^t, S^t, A^{t-1}, S^{t-1}, \dots, S^0) = \Pr(S^{t+1}|A^t, S^t)$
- Markovian reward process
 - $\Pr(R^t|A^t, S^t, A^{t-1}, S^{t-1}, \dots, S^0) = \Pr(R^t|A^t, S^t)$
- Stationary dynamics and reward
 - $\Pr(S^{t+1}|A^t, S^t) = \Pr(S^{t'+1}|A^{t'}, S^{t'})$ for all t, t'
- **Full observability**
 - though we can't predict what state we will reach when we execute an action, once it is realized, we know what it is

Policies (“plans” for MDPs)

- Given an MDP we wish to compute a **policy**
 - ▲ Could be computed offline or online.
- A policy is a possibly stochastic mapping from states to actions
 - ▲ $\pi:S \rightarrow A$
 - ▲ $\pi(s)$ is action to do at state s
 - ▲ specifies a continuously reactive controller



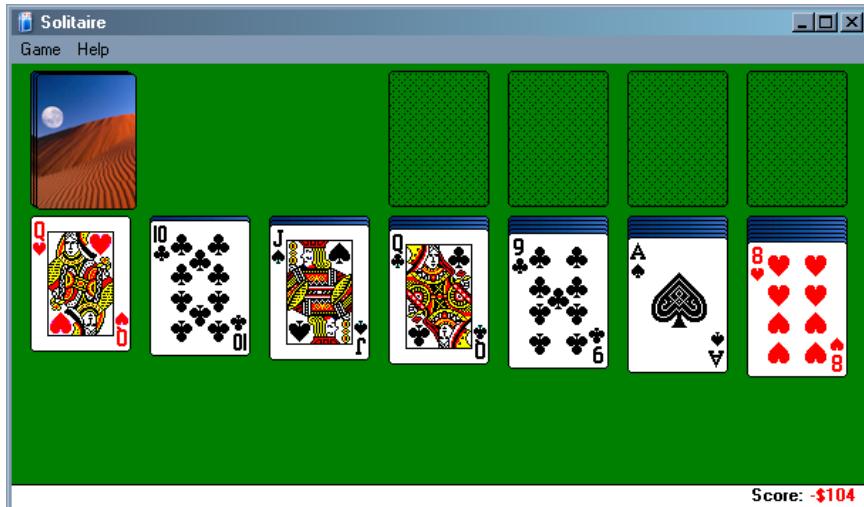
Solution Strategies

- Planning/optimization approaches:
 - Use search strategies/linear programming to calculate values for states over longer time horizons
 - Example: value iteration, policy iteration
- Reinforcement learning
 - Use data from the environment to learn unknown MDP parameters
- Monte Carlo approaches
 - Use simulators/models to stochastically calculate values and policies

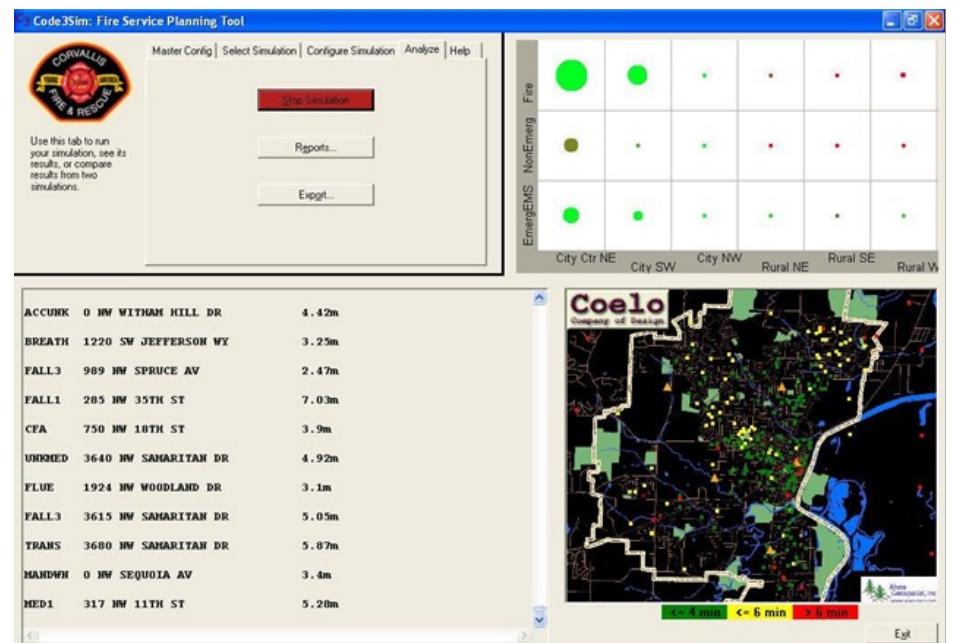
Large Worlds: Monte-Carlo Approach

- Often a **simulator** of a planning domain is available or can be learned from data
 - Even when domain can't be expressed via MDP language

Klondike Solitaire



Fire & Emergency Response



Example Domains with Simulators

- Traffic simulators
- Robotics simulators
- Military campaign simulators
- Computer network simulators
- Emergency planning simulators
 - ▲ large-scale disaster and municipal
- Sports domains (Madden Football)
- Board games / Video games
 - ▲ Go / RTS

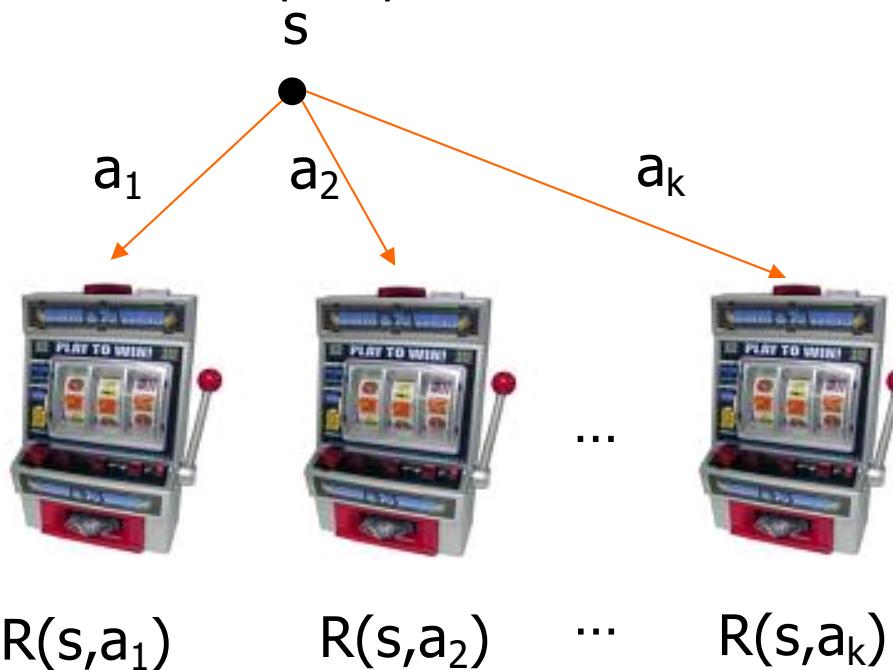
In many cases Monte-Carlo techniques yield state-of-the-art performance. Even in domains where model-based planning is applicable.

MDP: Simulation-Based Representation

- A simulation-based representation gives: S , A , R , T :
 - ▲ finite state set S (generally very large)
 - ▲ finite action set A
 - ▲ Stochastic, real-valued, bounded reward function $R(s,a) = r$
 - Stochastically returns a reward r given input s and a
 - Can be implemented in arbitrary programming language
 - ▲ Stochastic transition function $T(s,a) = s'$ (i.e. a simulator)
 - Stochastically returns a state s' given input s and a
 - Probability of returning s' is dictated by $\Pr(s' | s,a)$ of MDP
 - T can be implemented in an arbitrary programming language

Single State Monte-Carlo Planning

- Suppose MDP has a single state and k actions
 - ▲ Figure out which action has best expected reward
 - ▲ Can sample rewards of actions using calls to simulator
 - ▲ Sampling a is like pulling slot machine arm with random payoff function $R(s,a)$

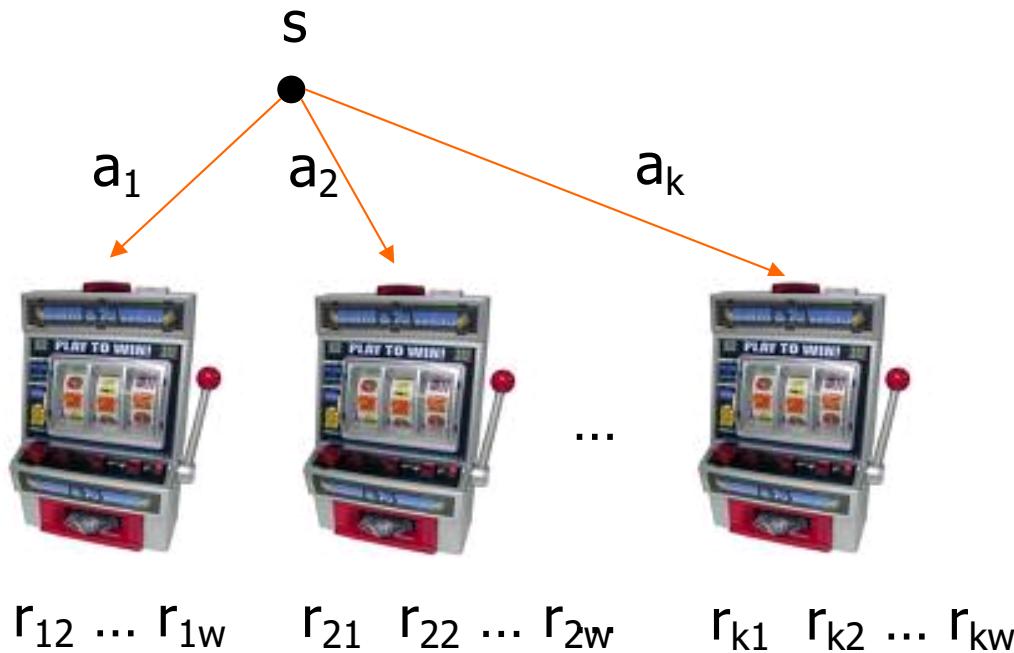


Multi-Armed Bandit Problem

Uniform Bandit Algorithm

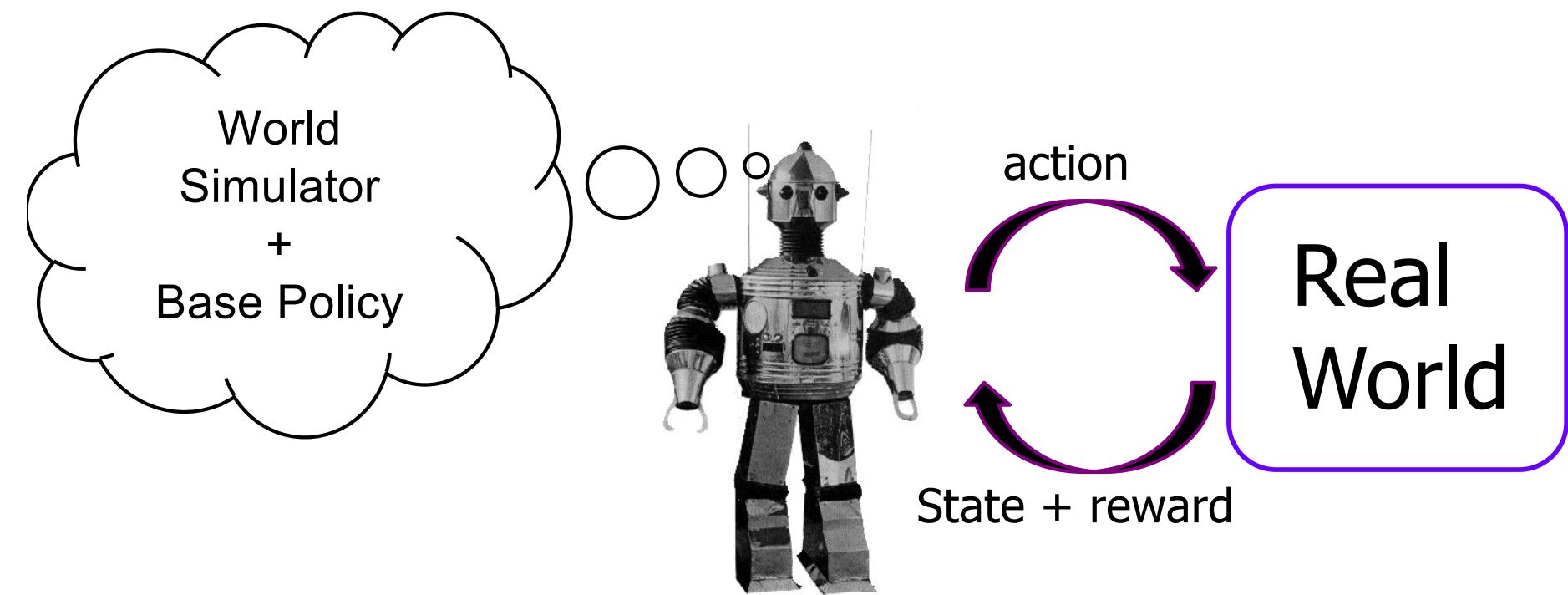
NaiveBandit from [Even-Dar et. al., 2002]

1. Pull each arm w times (uniform pulling).
2. Return arm with best average reward.

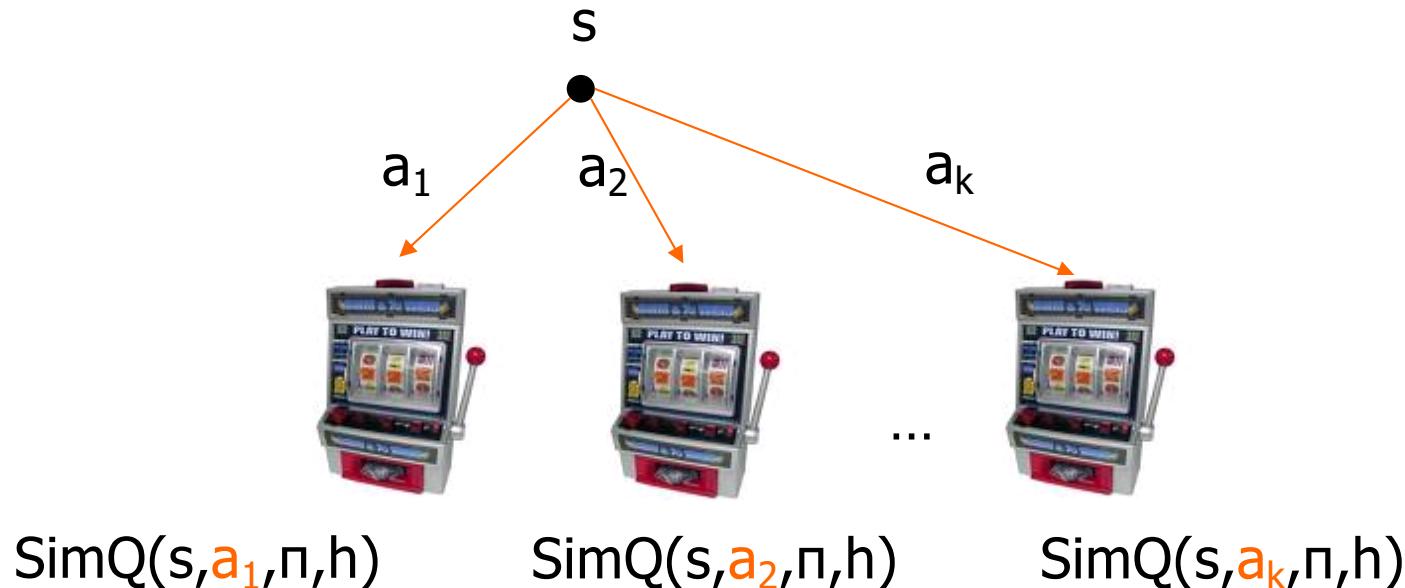


Policy Improvement via Monte-Carlo

- Now consider a multi-state MDP.
- Suppose we have a simulator and a non-optimal policy
 - ▲ E.g. policy could be a standard heuristic or based on intuition
- Can we somehow compute an improved policy?



Policy Improvement via Bandits



- **Idea:** define a stochastic function **SimQ(s,a,π,h)** that we can implement and whose expected value is $Q_\pi(s,a,h)$
- Use Bandit algorithm to select improved action

How to implement SimQ?

Policy Improvement via Bandits

```
SimQ(s,a,π,h)
```

```
    r = R(s,a)
```

```
    s = T(s,a)
```

```
    for i = 1 to h-1
```

```
        r = r + βi R(s, π(s))
```

```
        s = T(s, π(s))
```

```
    Return r
```

simulate a in s

simulate h-1 steps
of policy

- Simply simulate taking **a** in **s** and following policy for $h-1$ steps, returning discounted sum of rewards
- Expected value of $\text{SimQ}(s,a,\pi,h)$ is $Q_\pi(s,a,h)$

Policy Improvement via Bandits

SimQ(s, a, π, h)

$r = R(s, a)$

$s = T(s, a)$

for $i = 1$ to $h-1$

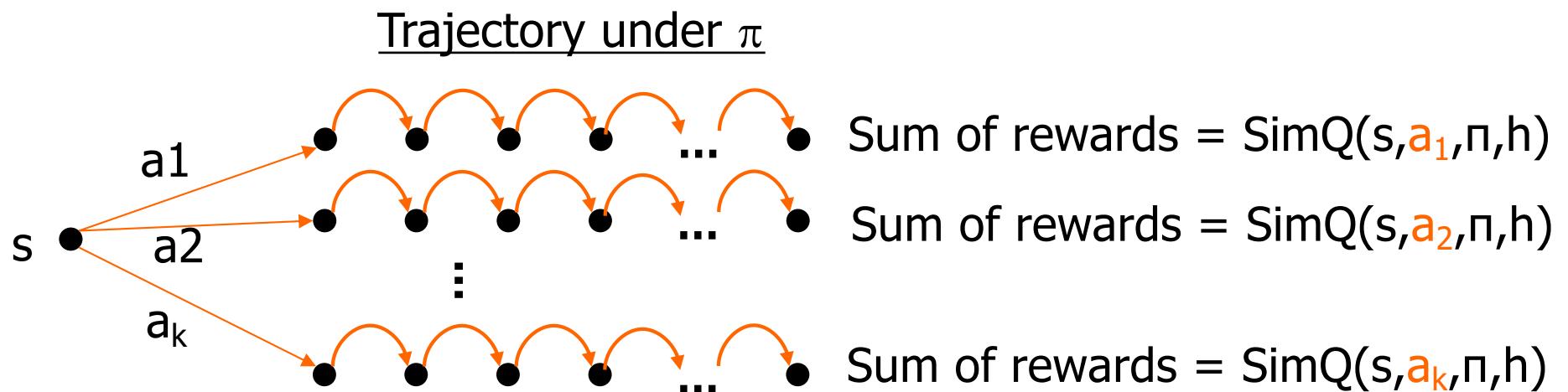
$r = r + \beta^i R(s, \pi(s))$

$s = T(s, \pi(s))$

Return r

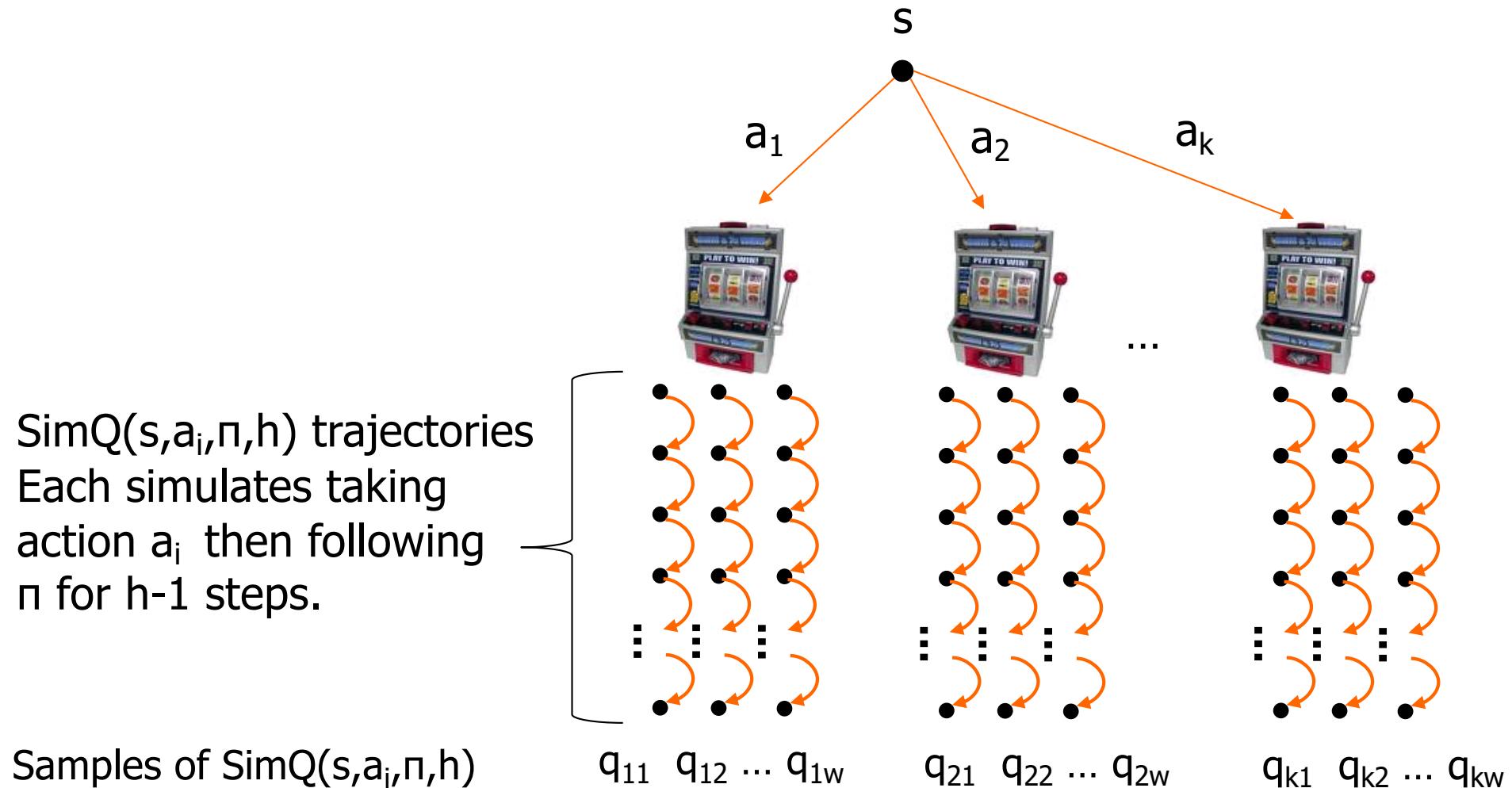
simulate a in s

simulate $h-1$ steps
of policy



Policy Rollout Algorithm

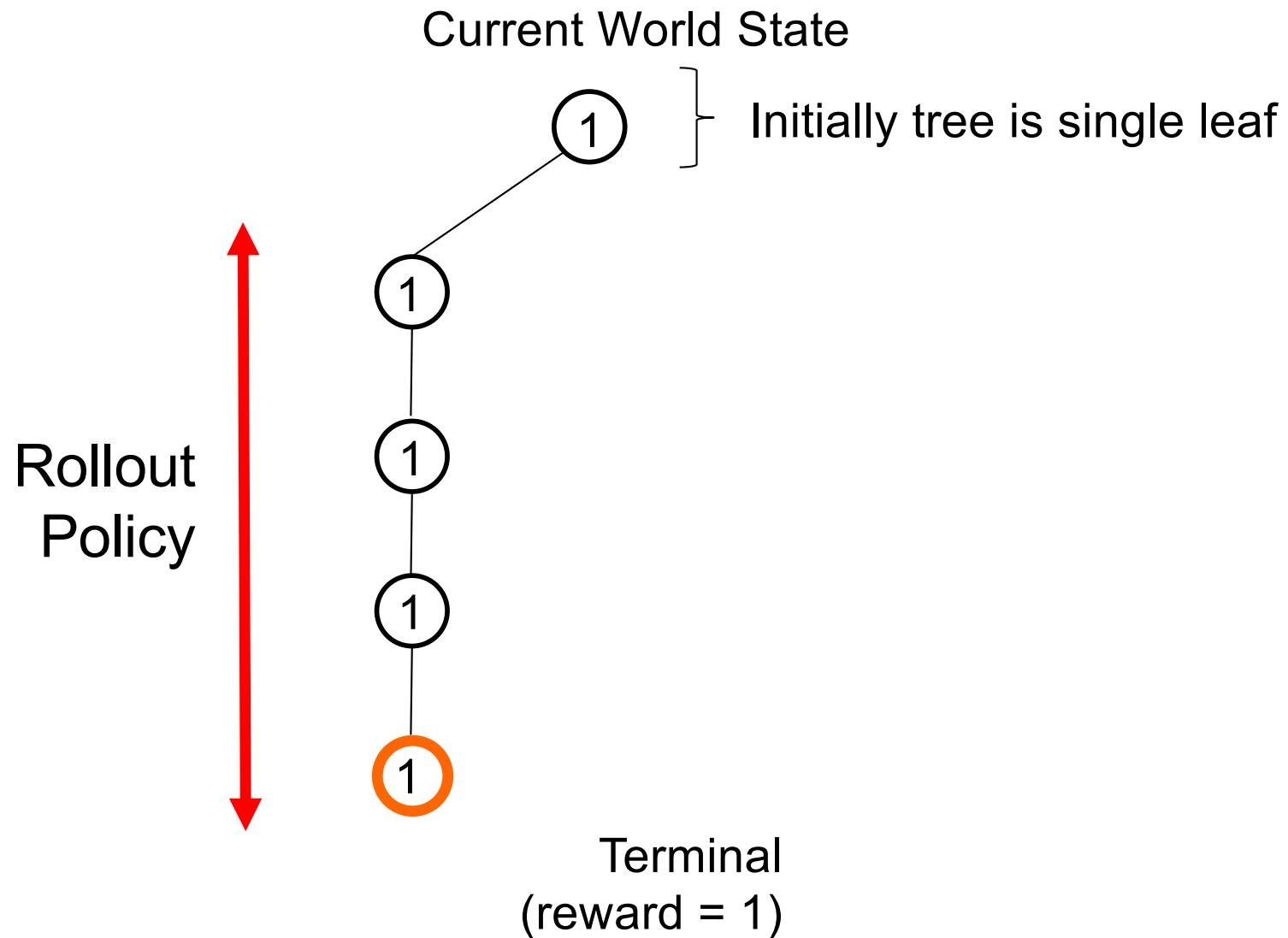
1. For each a_i run $\text{SimQ}(s, a_i, \pi, h)$ w times
2. Return action with best average of SimQ results



UCT Algorithm [Kocsis & Szepesvari, 2006]

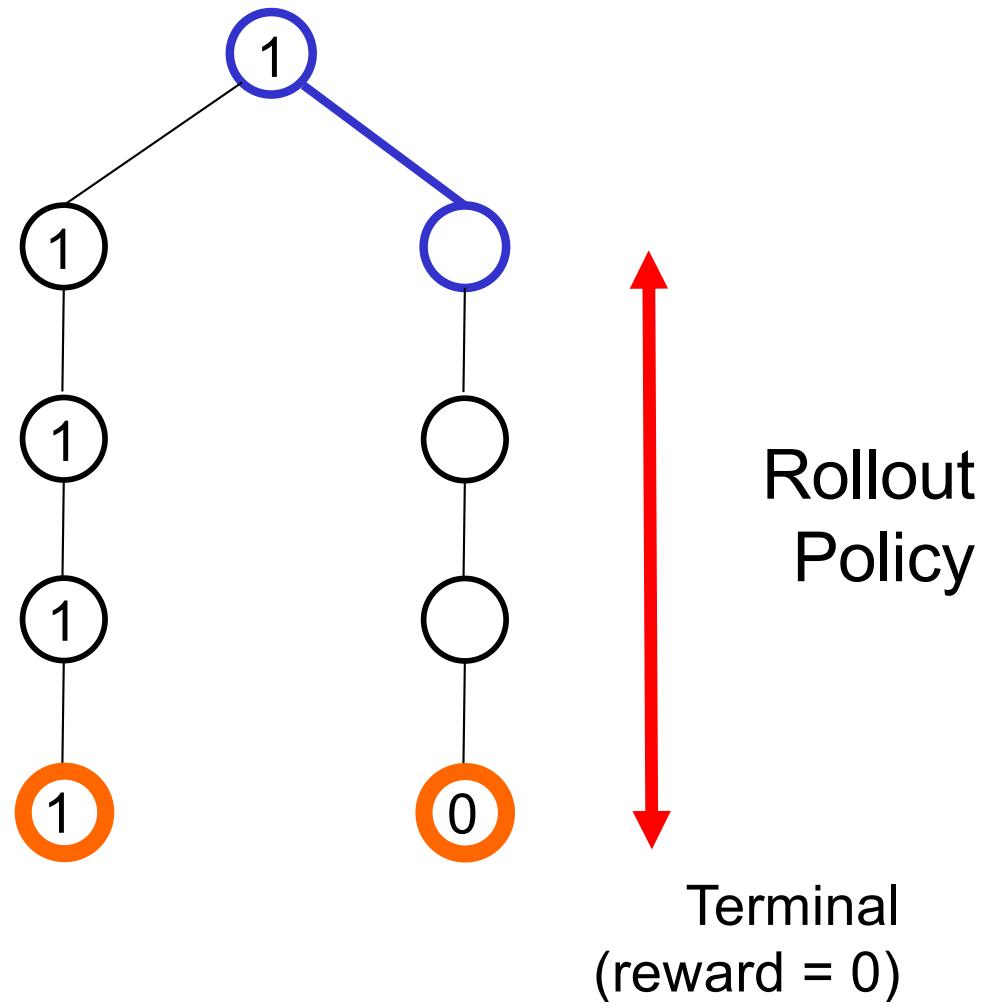
- Improvement on Monte-Carlo Tree Search
 - ▲ Some nice theoretical properties
 - ▲ Much better anytime behavior than sparse sampling
 - ▲ Major advance in computer Go
- Monte-Carlo Tree Search
 - ▲ Repeated Monte Carlo simulation of a rollout policy
 - ▲ Each rollout adds one or more nodes to search tree
- Rollout policy depends on nodes already in tree

At a leaf node perform a random rollout



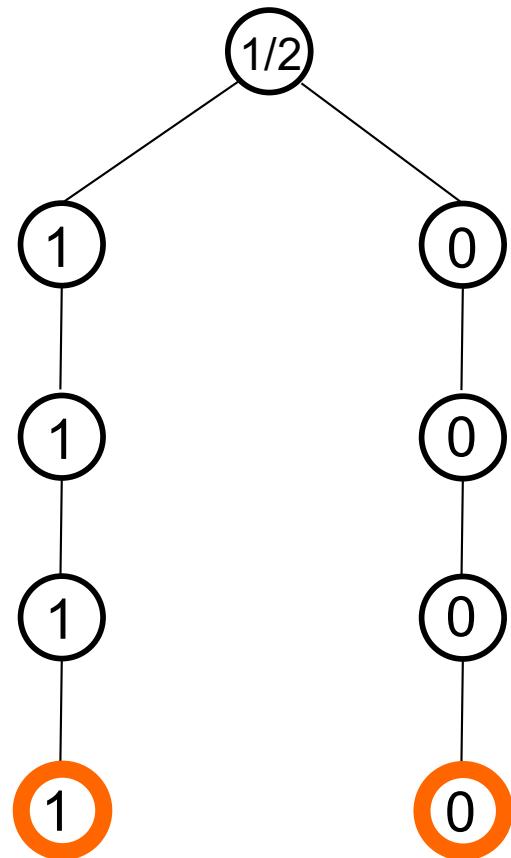
Must select each action at a node at least once

Current World State

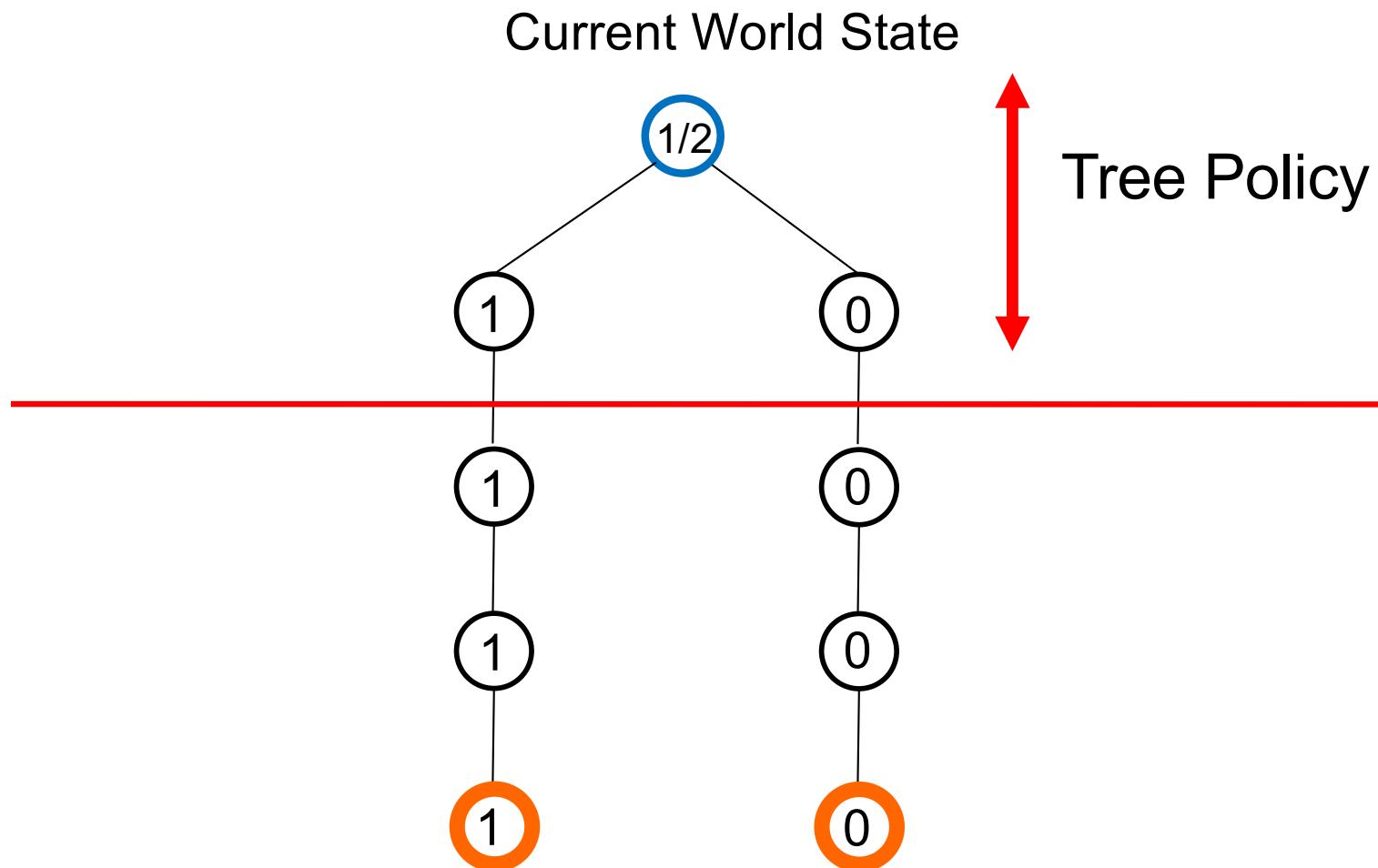


Must select each action at a node at least once

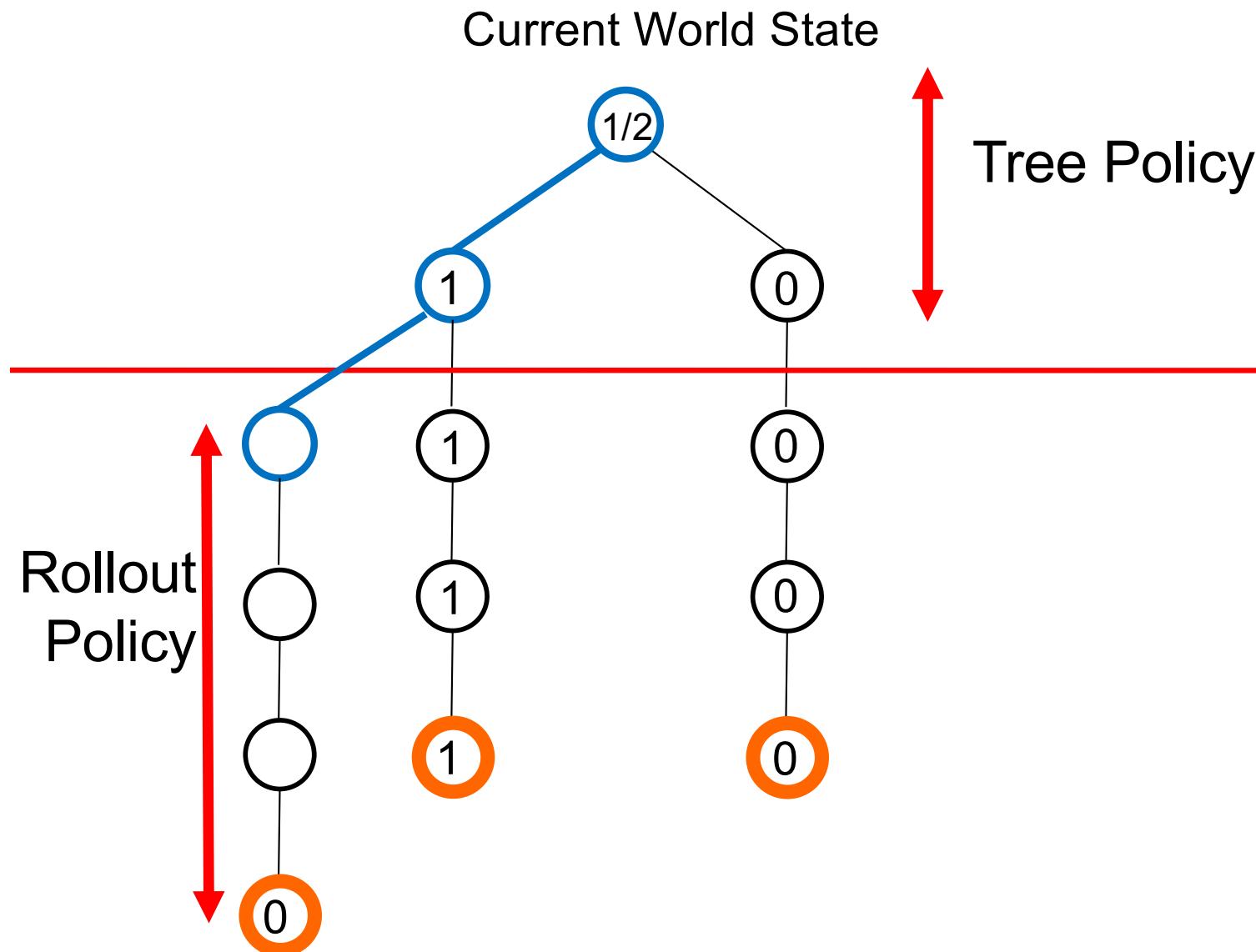
Current World State



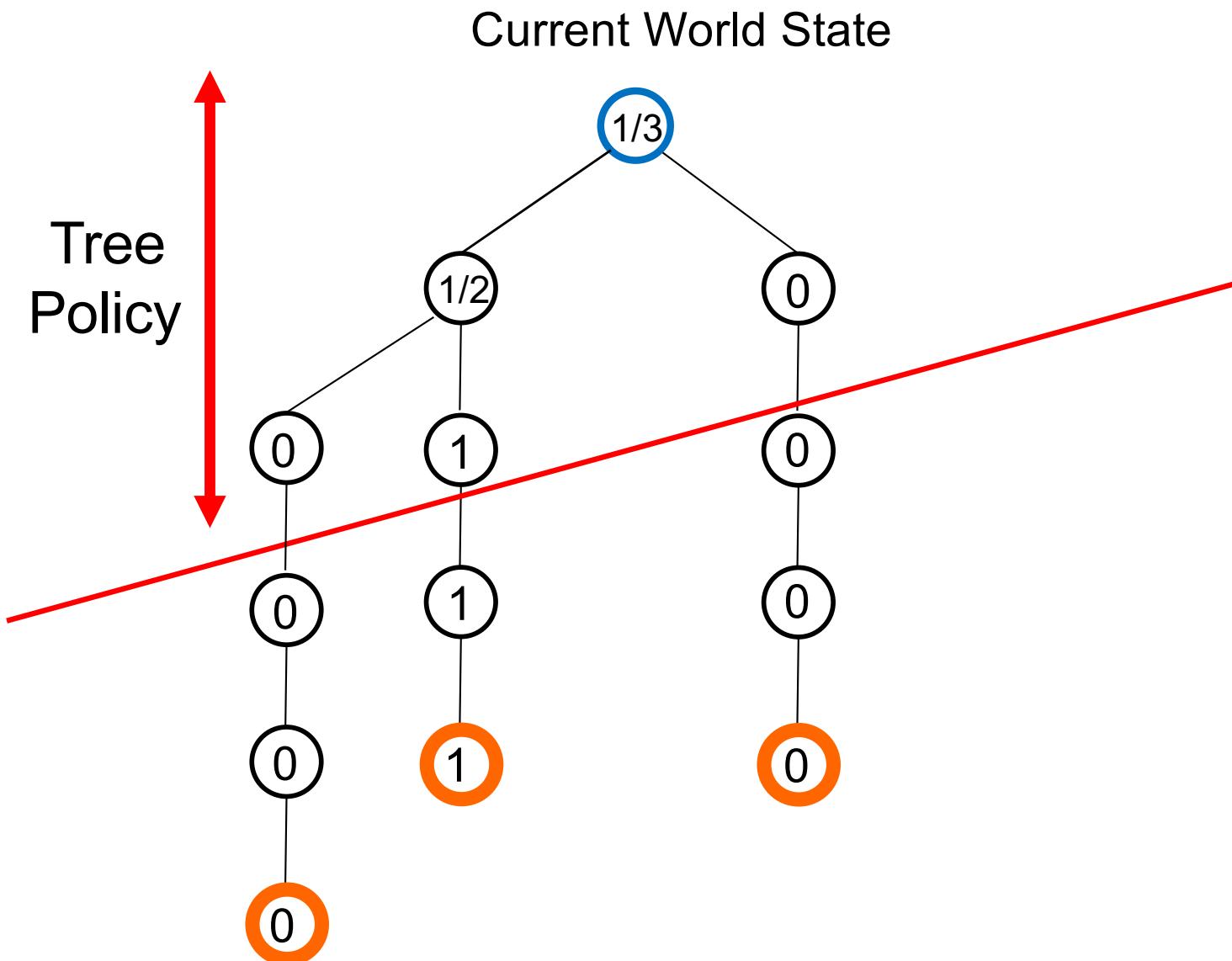
When all node actions tried once, select action according to tree policy



When all node actions tried once, select action according to tree policy



When all node actions tried once, select action according to tree policy



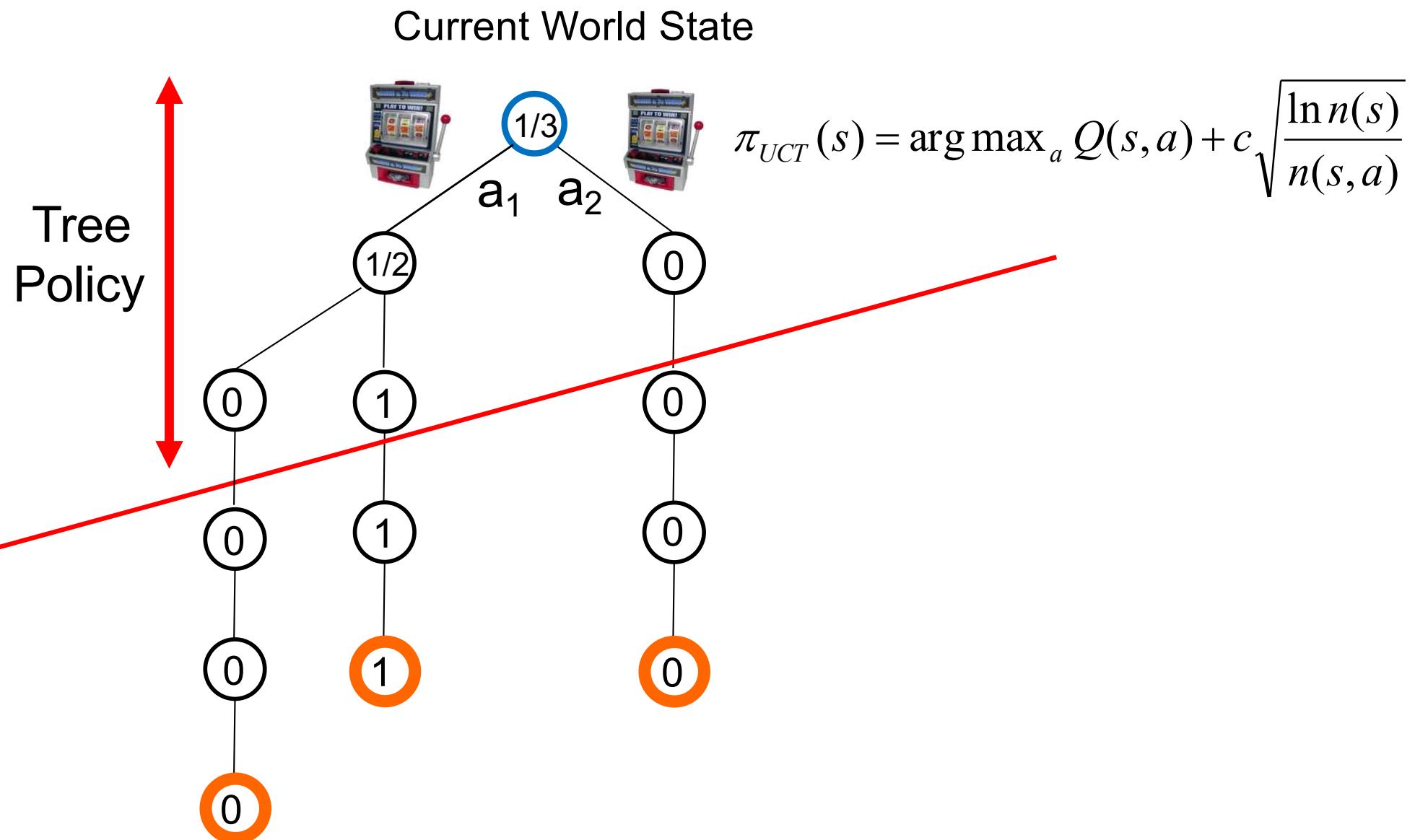
UCT Algorithm [Kocsis & Szepesvari, 2006]

- Basic UCT uses random rollout policy
- Tree policy is based on UCB:
 - ▲ $Q(s,a)$: average reward received in current trajectories after taking action a in state s
 - ▲ $n(s,a)$: number of times action a taken in s
 - ▲ $n(s)$: number of times state s encountered

$$\pi_{UCT}(s) = \arg \max_a Q(s, a) + c \sqrt{\frac{\ln n(s)}{n(s, a)}}$$

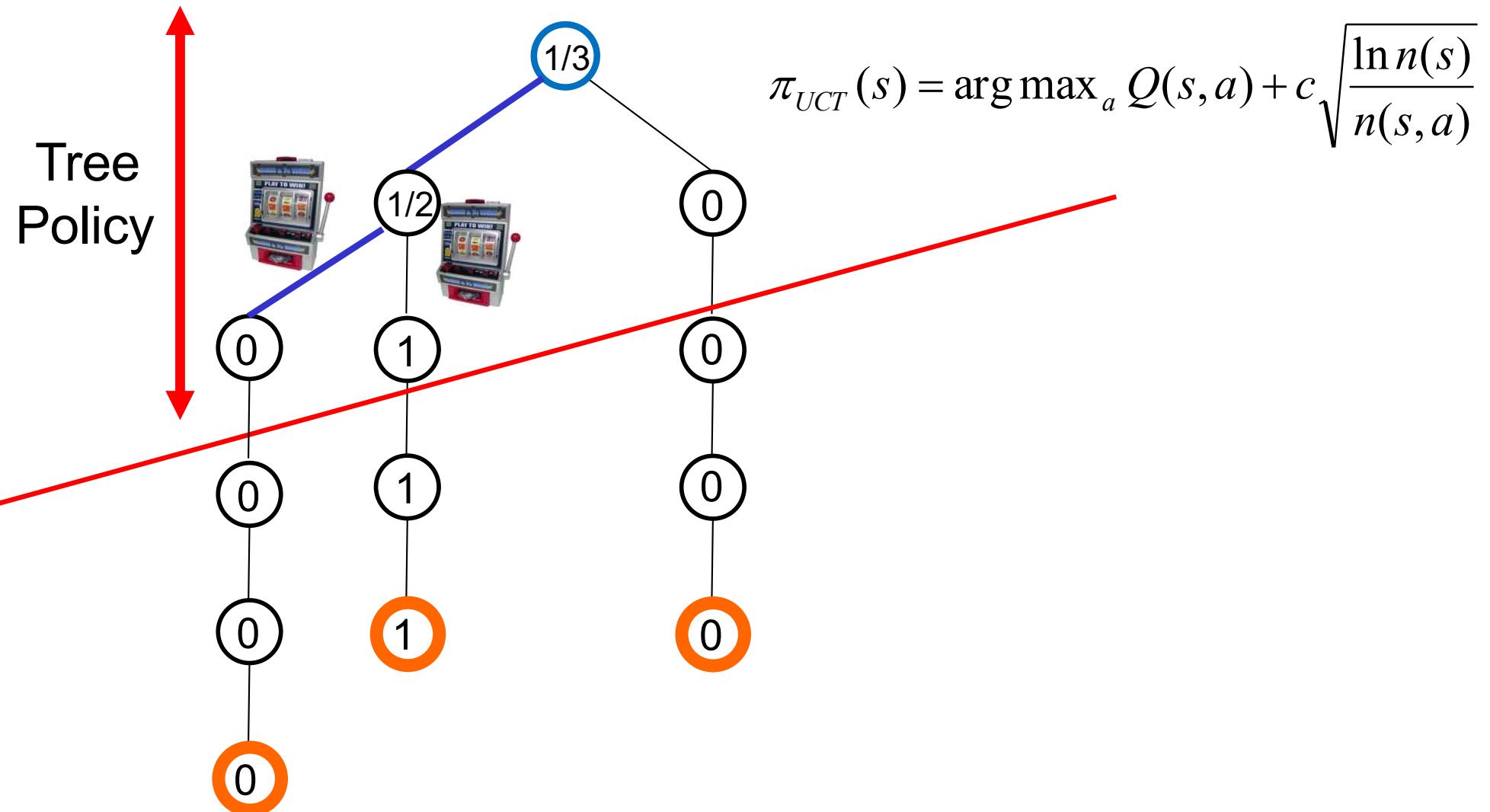
Theoretical constant that must
be selected empirically in practice

When all node actions tried once, select action according to tree policy



When all node actions tried once, select action according to tree policy

Current World State



UCT Recap

- To select an action at a state s
 - ▲ Build a tree using N iterations of Monte-Carlo tree search
 - Default policy is uniform random
 - Tree policy is based on UCB rule
 - ▲ Select action that maximizes $Q(s,a)$
(note that this final action selection does not take the exploration term into account, just the Q-value estimate)
- The more simulations the more accurate

Recap

- Markov Decision Processes are a mathematical model for abstractly representing the perception-cognition-action process in agents.
- Monte Carlo methods such as UCT are one way to identify good policies that agents can take in an uncertain domain
- The same ideas will be reused in our discussion of reinforcement learning.
- Let's look at an example of using UCT in a game system.

Paper Overview

- Lavers and Sukthankar: A Monte Carlo Approach for Football Play Generation (AIIDE 2010)
- Use UCT in an offline fashion to improve on the football plays from a database of existing plays.
- To make the problem easier:
 - Use information from existing plays
 - Select subset of players to examine
- Video describing system:
 - https://youtu.be/6bN221LS-_U



Multi-Agent Adversarial Games

- ▶ Problems
 - Search space is large
 - Adding agents causes an exponential computational requirement
 - Continuous player placement (x, y)
 - In Rush, only 30% successful passing plays will result in a positive reward

- ▶ Why use football?
 - ▶ American football has a “playbook”
 - ▶ Highly structured plays
 - ▶ Domain complexity
 - ▶ Similarity to military operations
 - ▶ Ground maneuvers
 - ▶ Air battles
 - ▶ Naval encounters
 - ▶ This work should generalize well to these domains





Dynamic Adaptation–Single Action

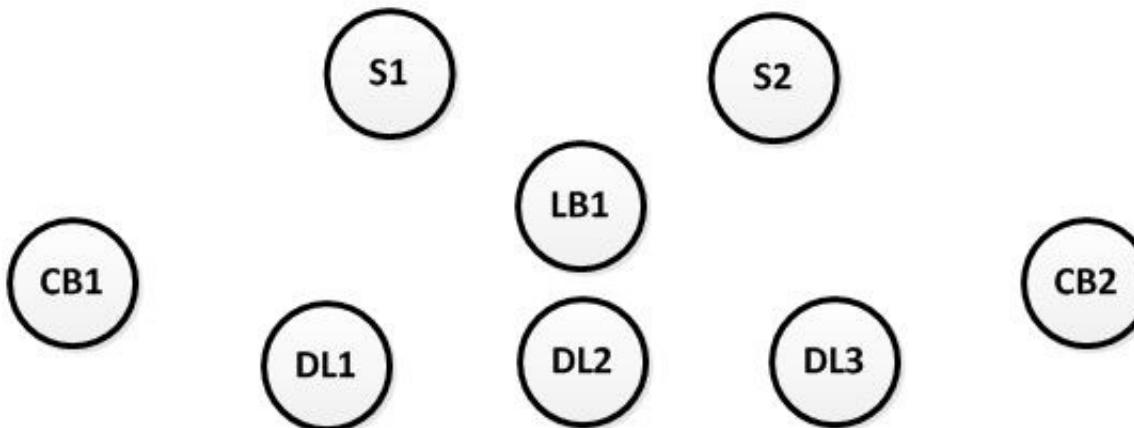
- ▶ Offline: Train SVM Models and determine the offensive key players
- ▶ Start play using built-in playbook
- ▶ As soon as defensive strategy is known, change our strategy for key players

	Static	Dynamic
Single-Action Search	–Use play we know is best	–Start with random play --Perform intention recognition --Switch to best play
Multi-Action Search	–UCT to learn new plays	–Start with random play --Perform intention recognition --Use UCT to guide key players



Opponent Modeling

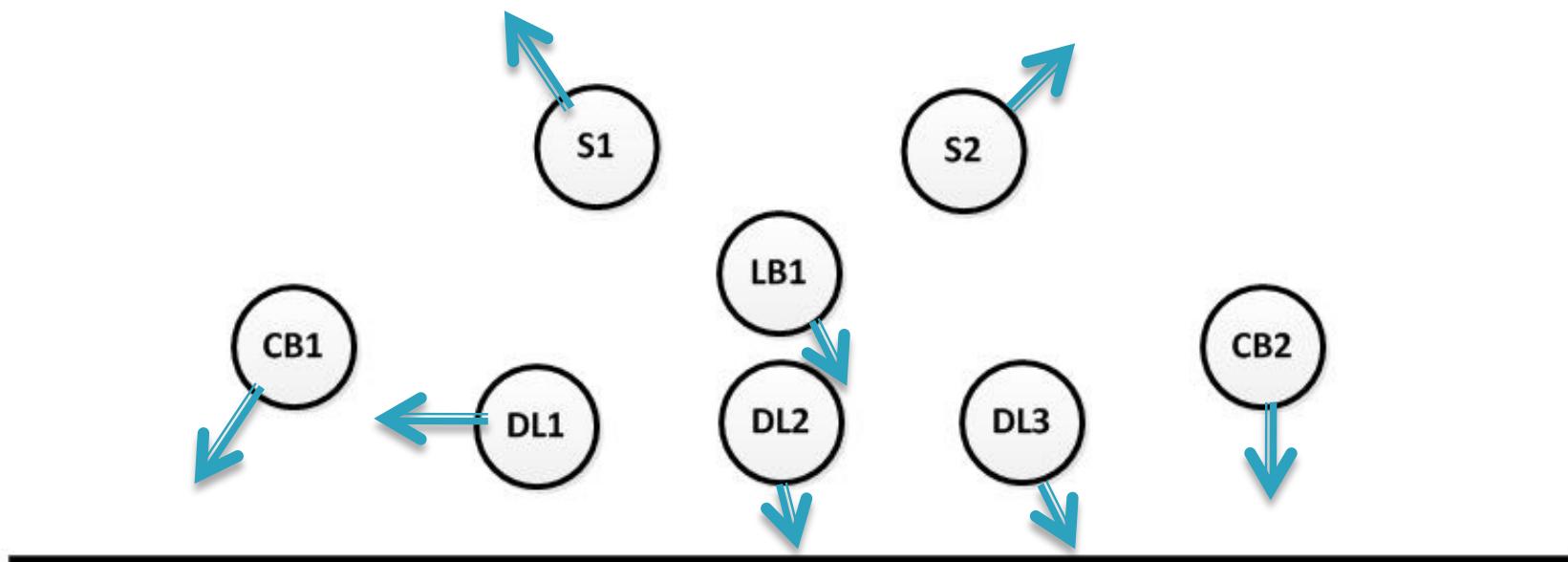
- ▶ Support Vector Machines (SVM)
 - Models trained offline
 - Multi-class classification problem
 - In Rush, the offense is consistently identified at time-step 3.





Opponent Modeling

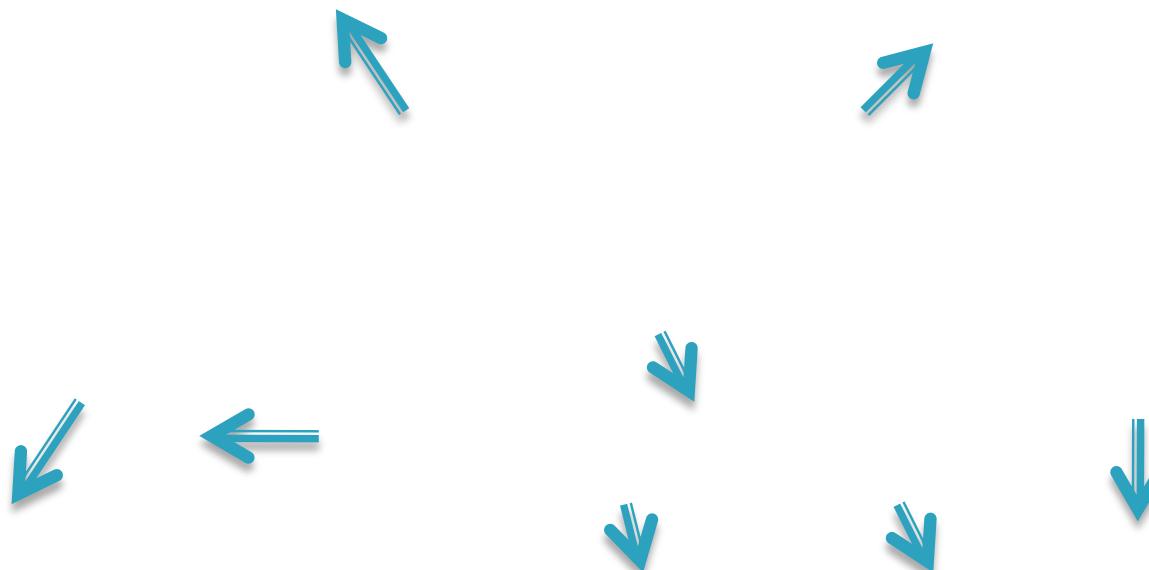
- ▶ Support Vector Machines (SVM)
 - Models trained offline
 - Multi-class classification problem
 - In Rush, the offense is consistently identified at time-step 3.





Opponent Modeling

- ▶ Support Vector Machines (SVM)
 - Models trained offline
 - Multi-class classification problem
 - In Rush, the offense is consistently identified at time-step 3.





Grouping Players

- ▶ How to select group of players
 1. Random
 2. Domain knowledge
 3. Use historical data
- ▶ Identify candidate groups
 - Use movement patterns (Kullback and Leibler Divergence)
 - KL divergence: calculate similarity between 2 distributions
 - Used to identify whether players move together

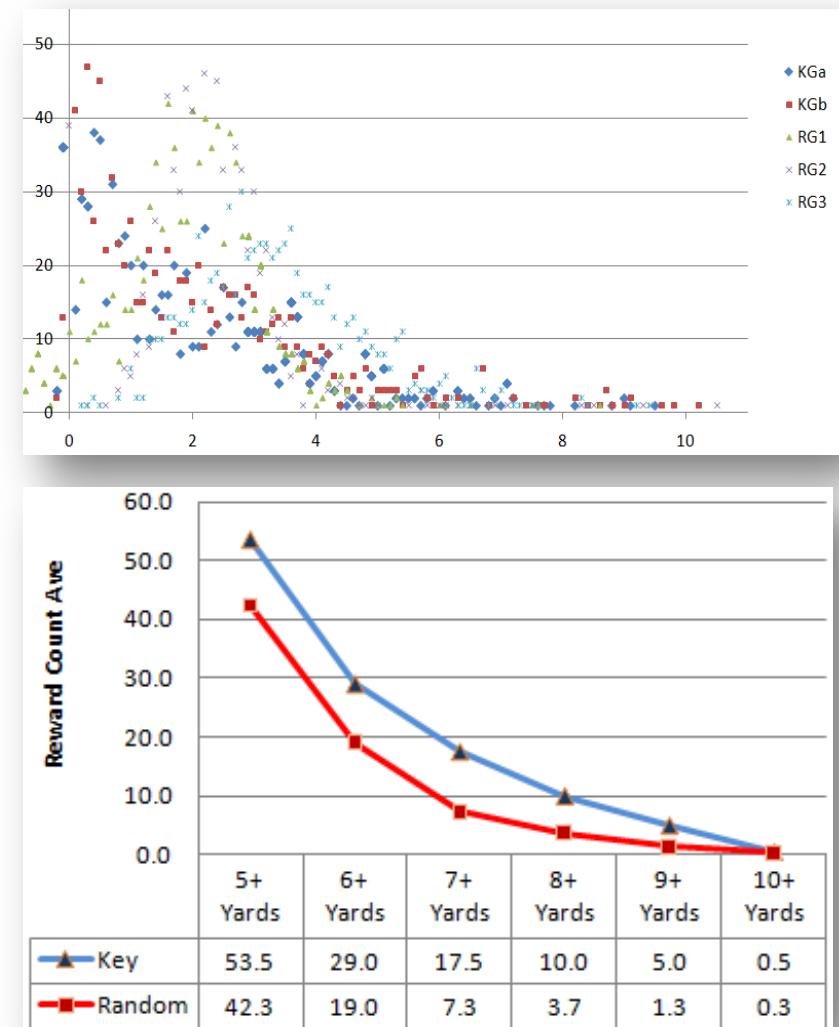
$$S(o_i, o_j) = D_{KL}(F_i || F_j) + D_{KL}(F_j || F_i),$$

- $D_{KL}(F_i || F_j) = \sum_k F_i(k) \log \left(\frac{F_i(k)}{F_j(k)} \right).$
- Use workflow
 - Used to identify whether players pass to each other



Using Key Group

- ▶ Rewards counted for the different groups
- ▶ Random groups tend to cluster at the baseline yardage of 2.8
- ▶ Key group is more evenly distributed
- ▶ At the higher yardage rewards, the key group more than doubles random groups





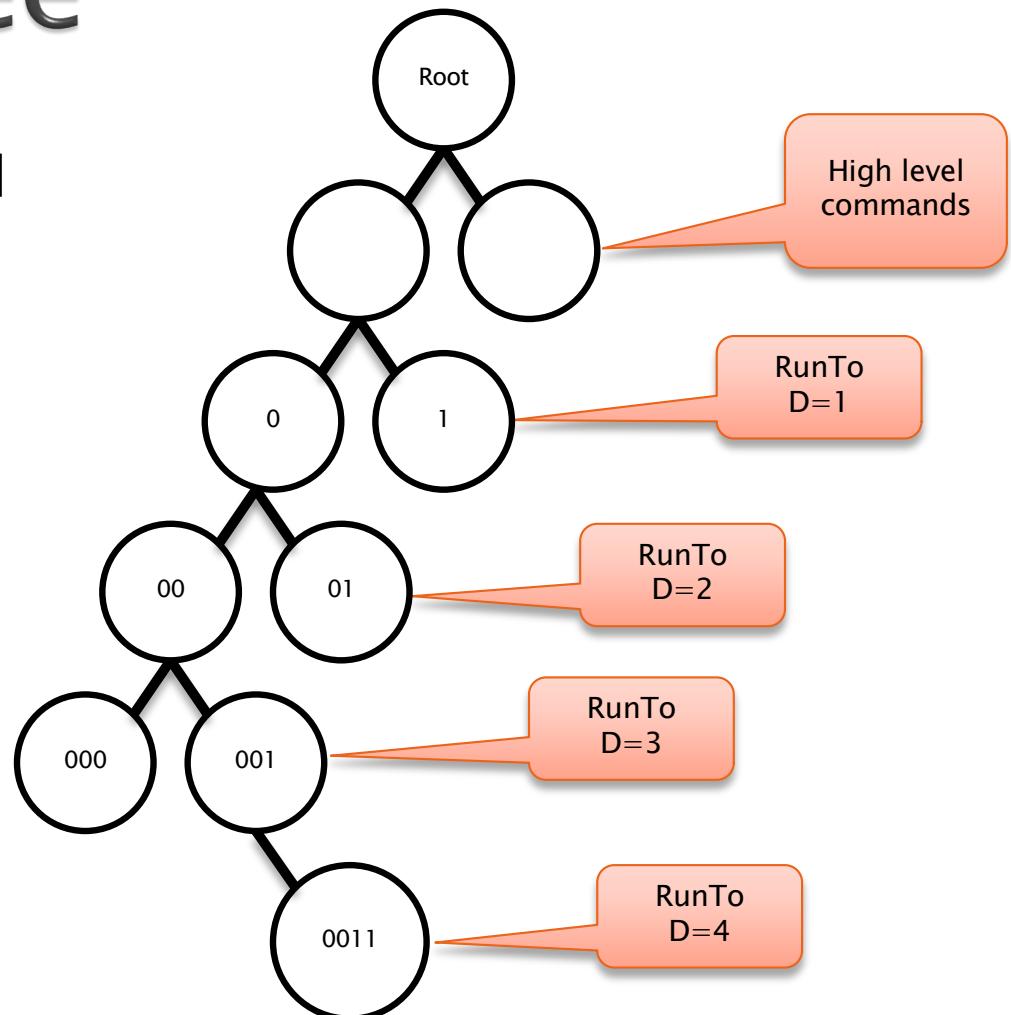
UCT for Football

- ▶ We use UCT to search for optimal plays
- ▶ Leaves are the only nodes sampled
- ▶ Most important decision closest to the root
- ▶ Two commands
 - Strategic movement
 - Execute high-level command
- ▶ Example
 - LWR (StartAt Sx Sy) (RunTo Rx Ry)(Do Something)



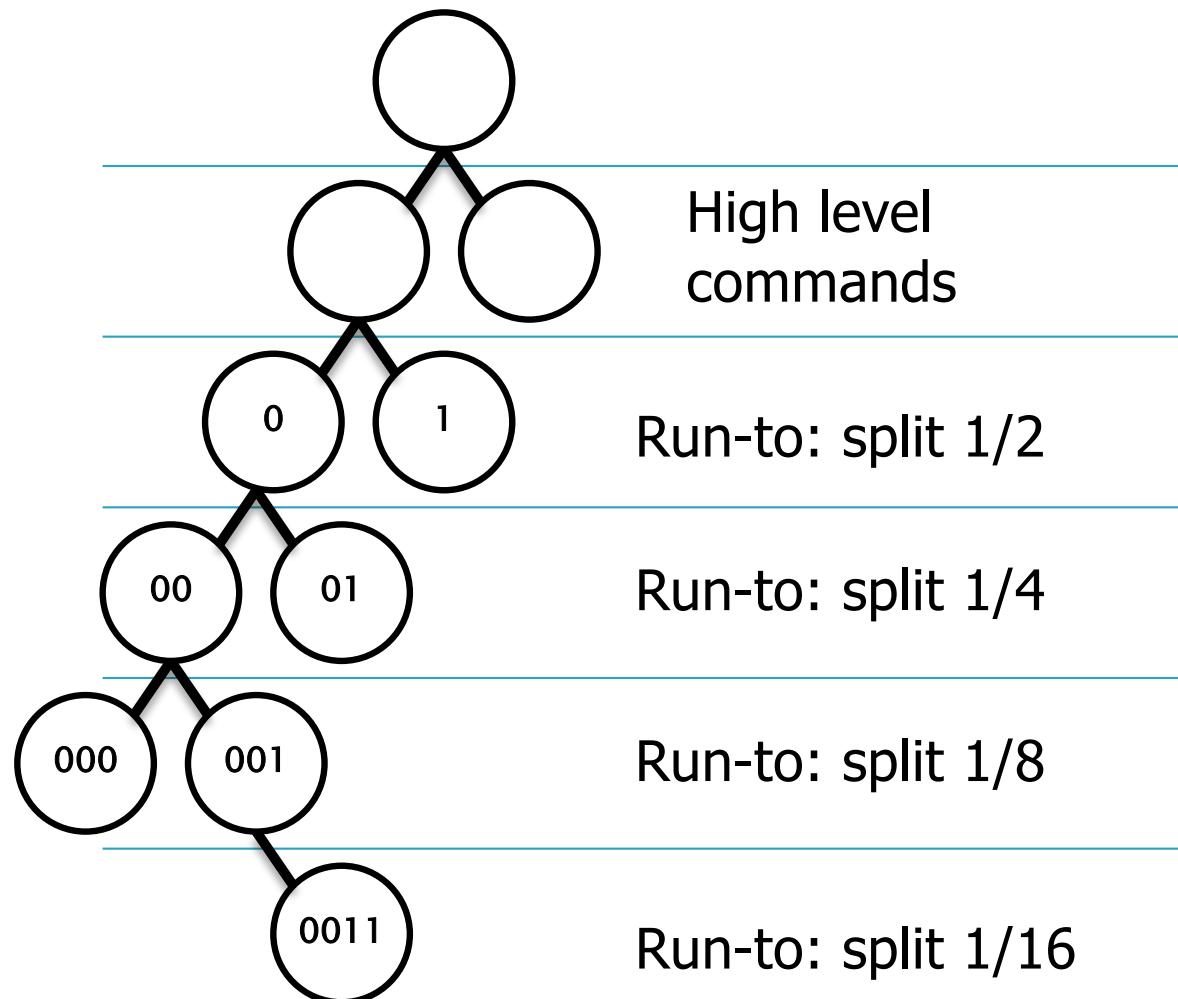
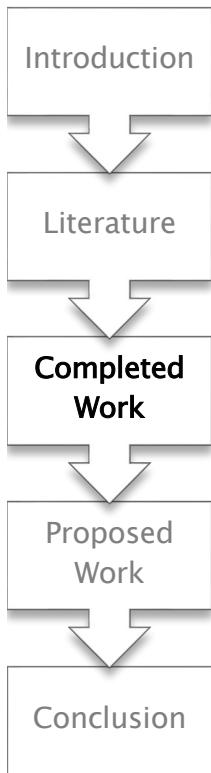
Building the Tree

- ▶ Find high-level command for selected players
- ▶ Find best runTo location
- ▶ Tree shown is only one player, in reality all key players are considered



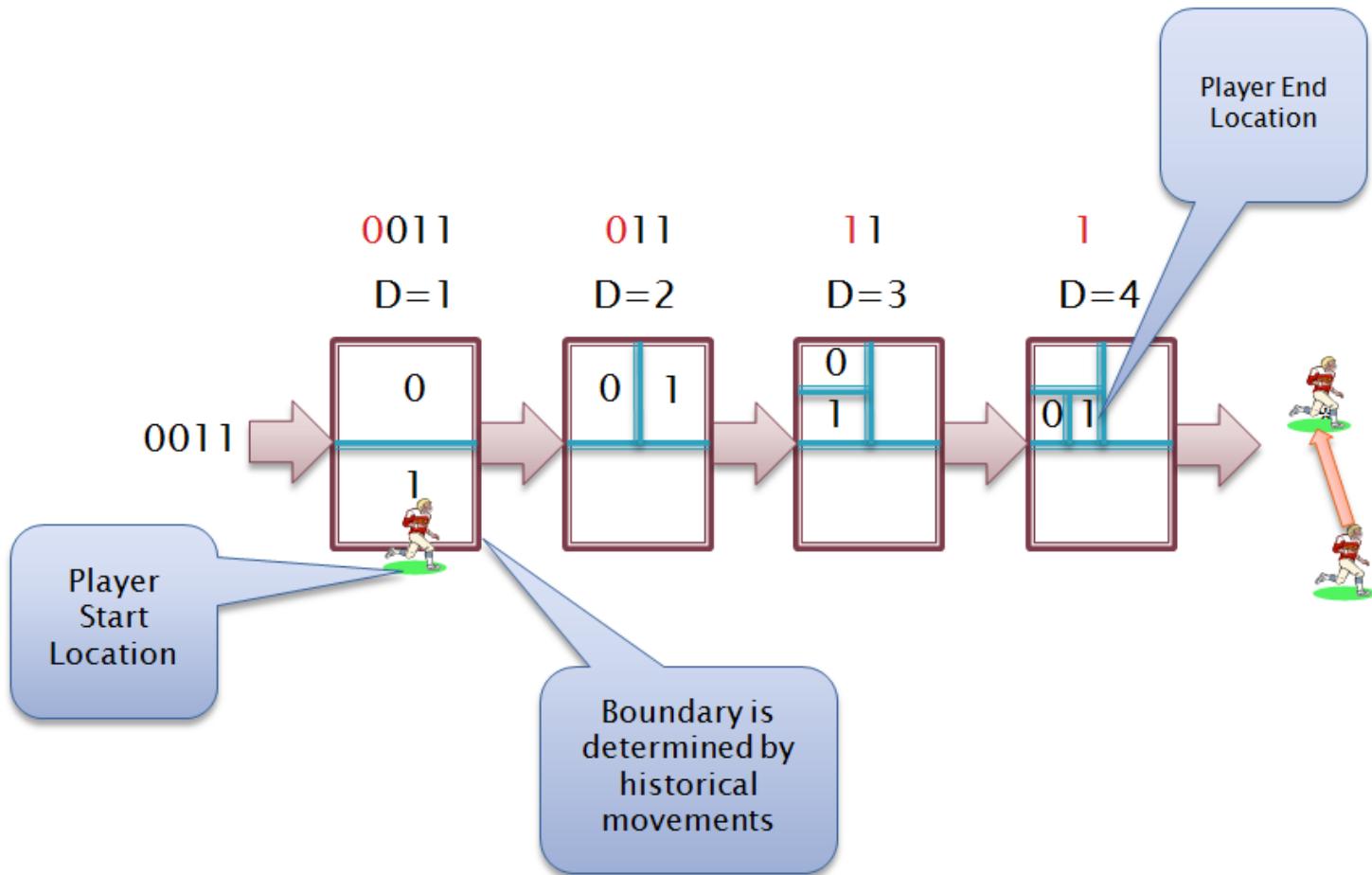
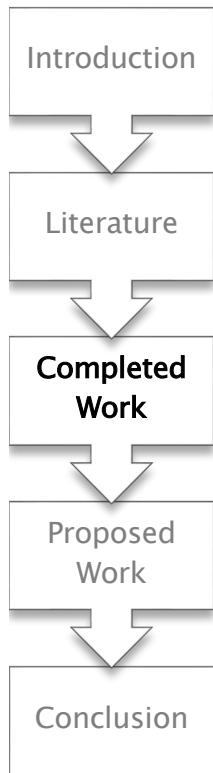


Space Sampling



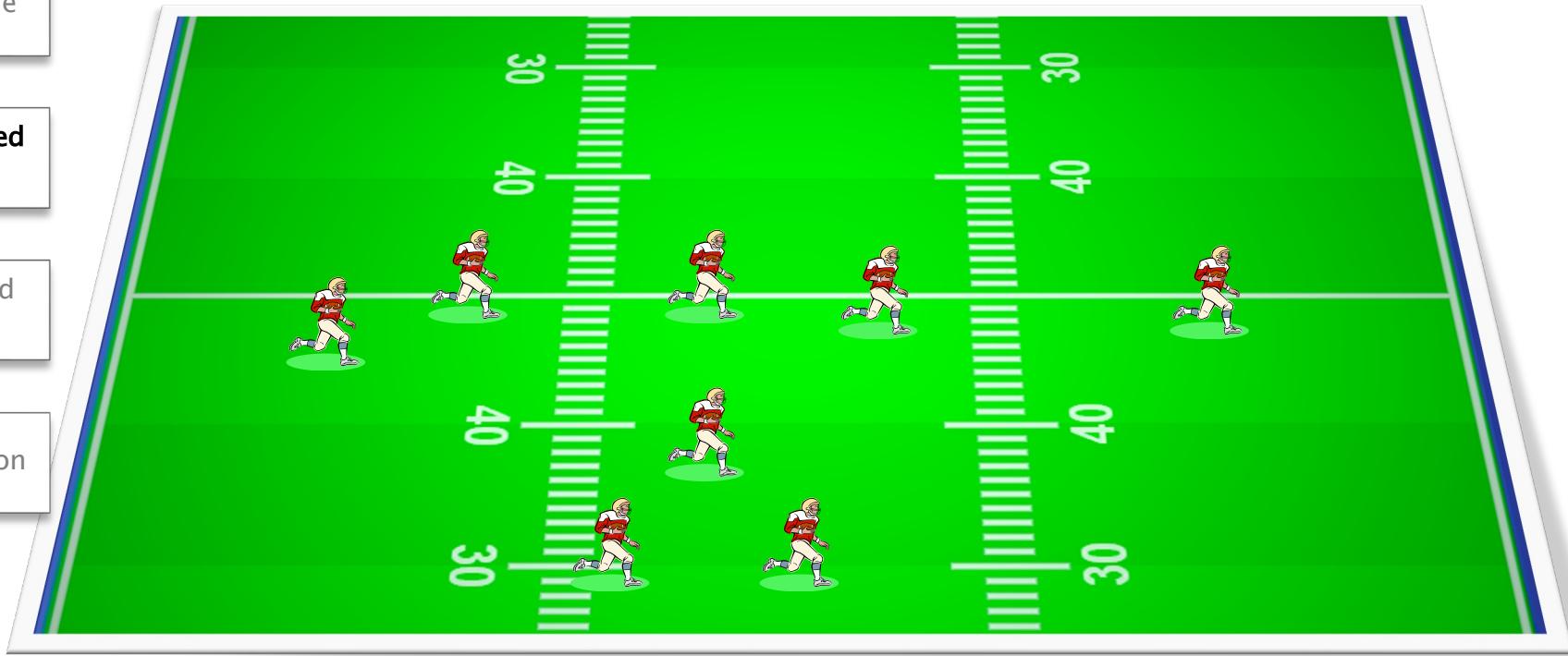


How it Works





Example

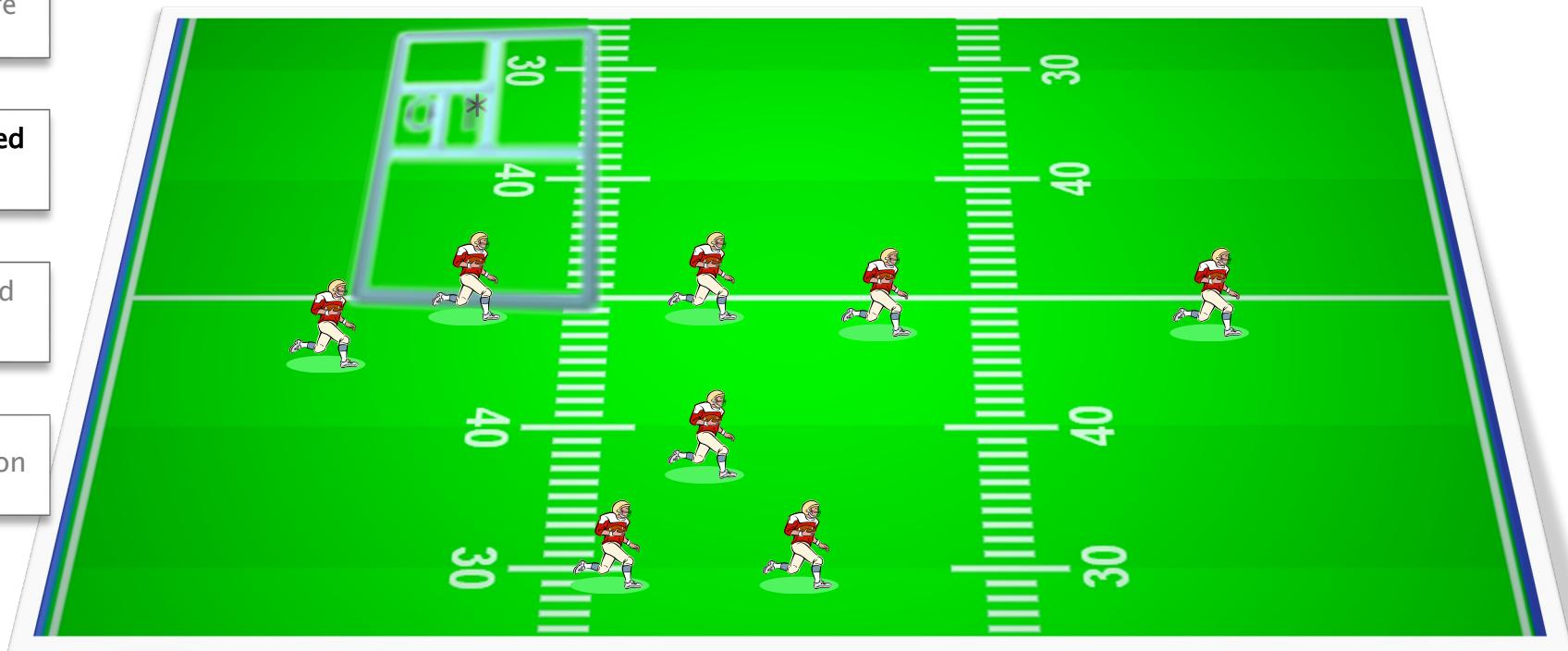
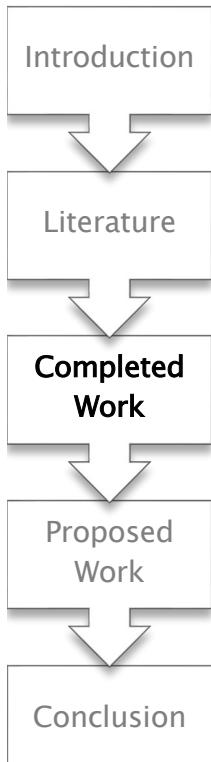


Single-Action
Static

Single-Action
Dynamic

Multi-Action
Static

Multi-Action
Dynamic

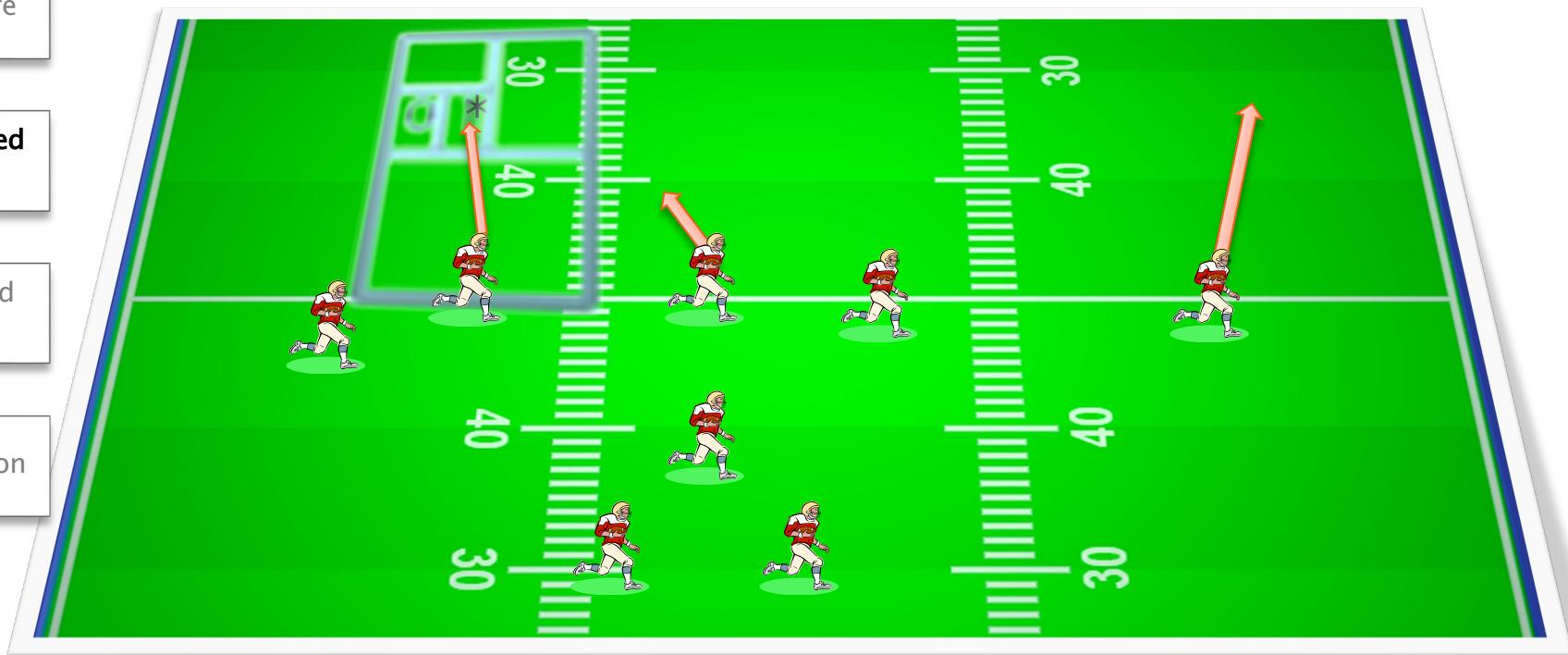
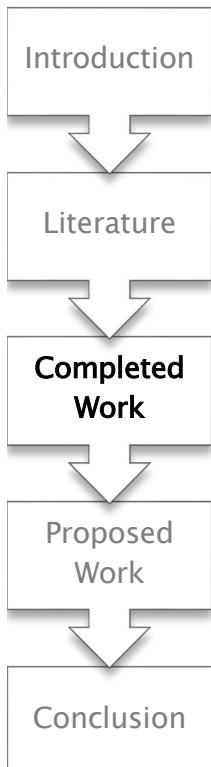


Single-Action
Static

Single-Action
Dynamic

Multi-Action
Static

Multi-Action
Dynamic

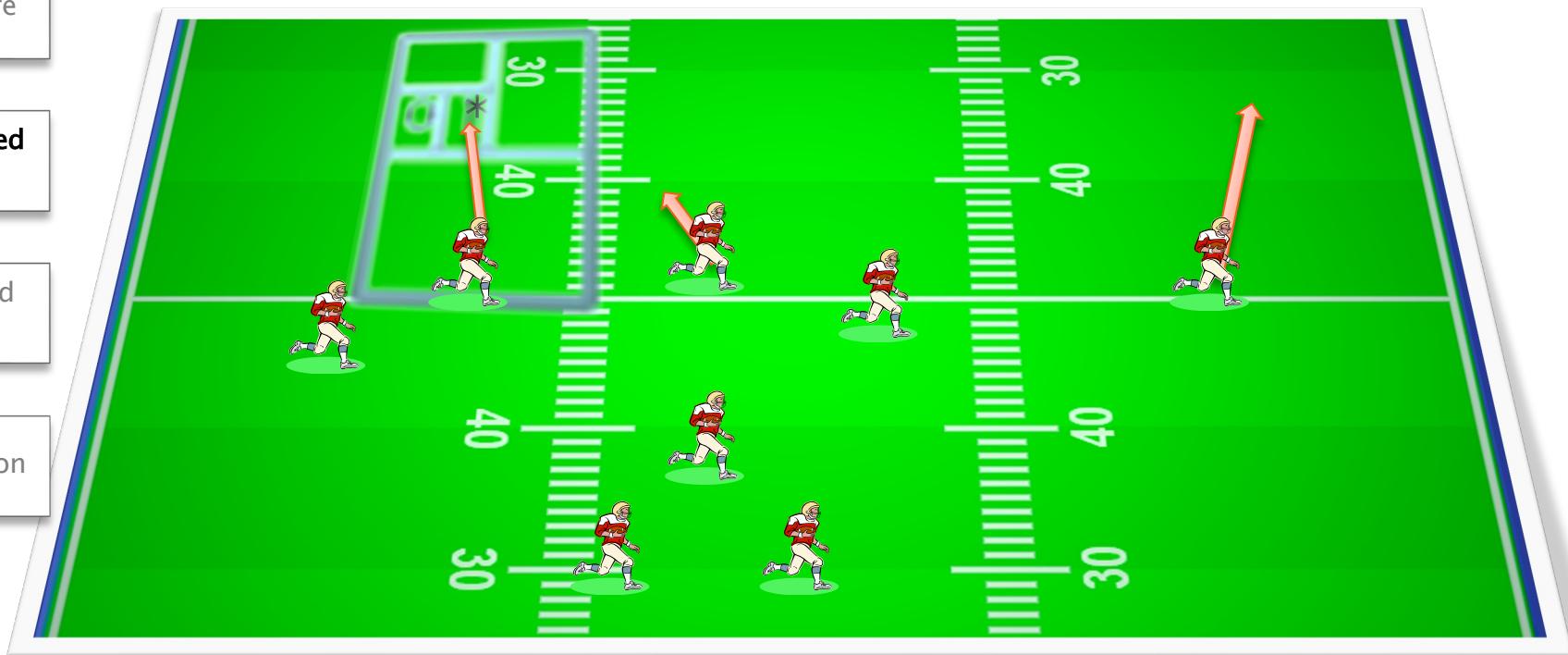
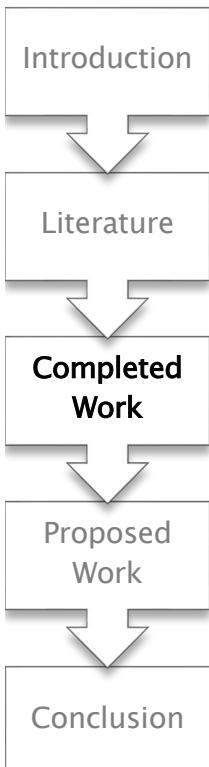


Single-Action
Static

Single-Action
Dynamic

Multi-Action
Static

Multi-Action
Dynamic

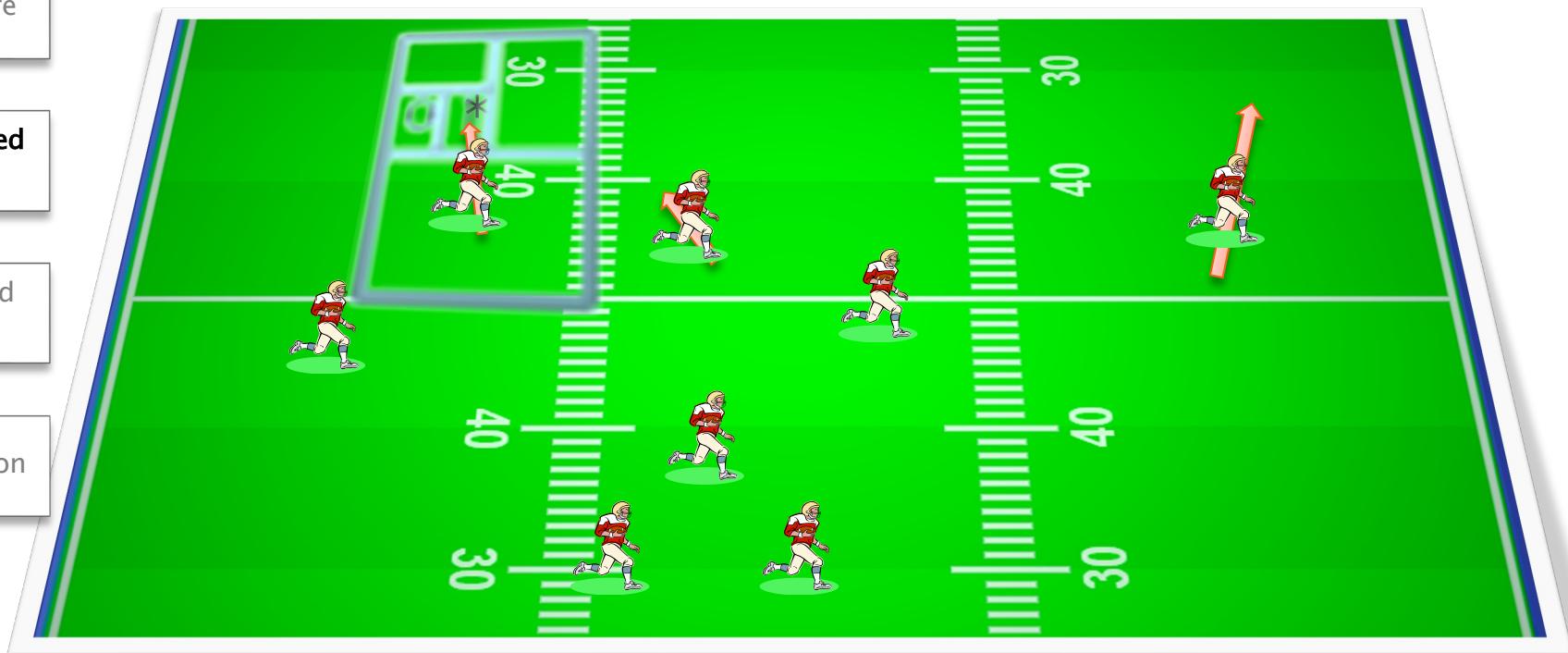
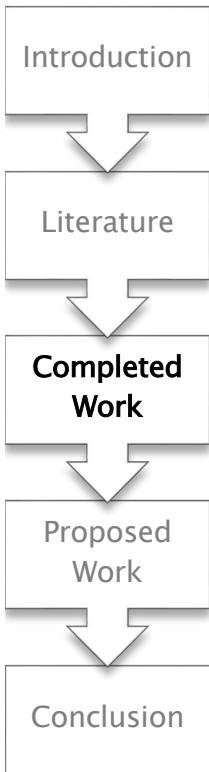


Single-Action
Static

Single-Action
Dynamic

Multi-Action
Static

Multi-Action
Dynamic

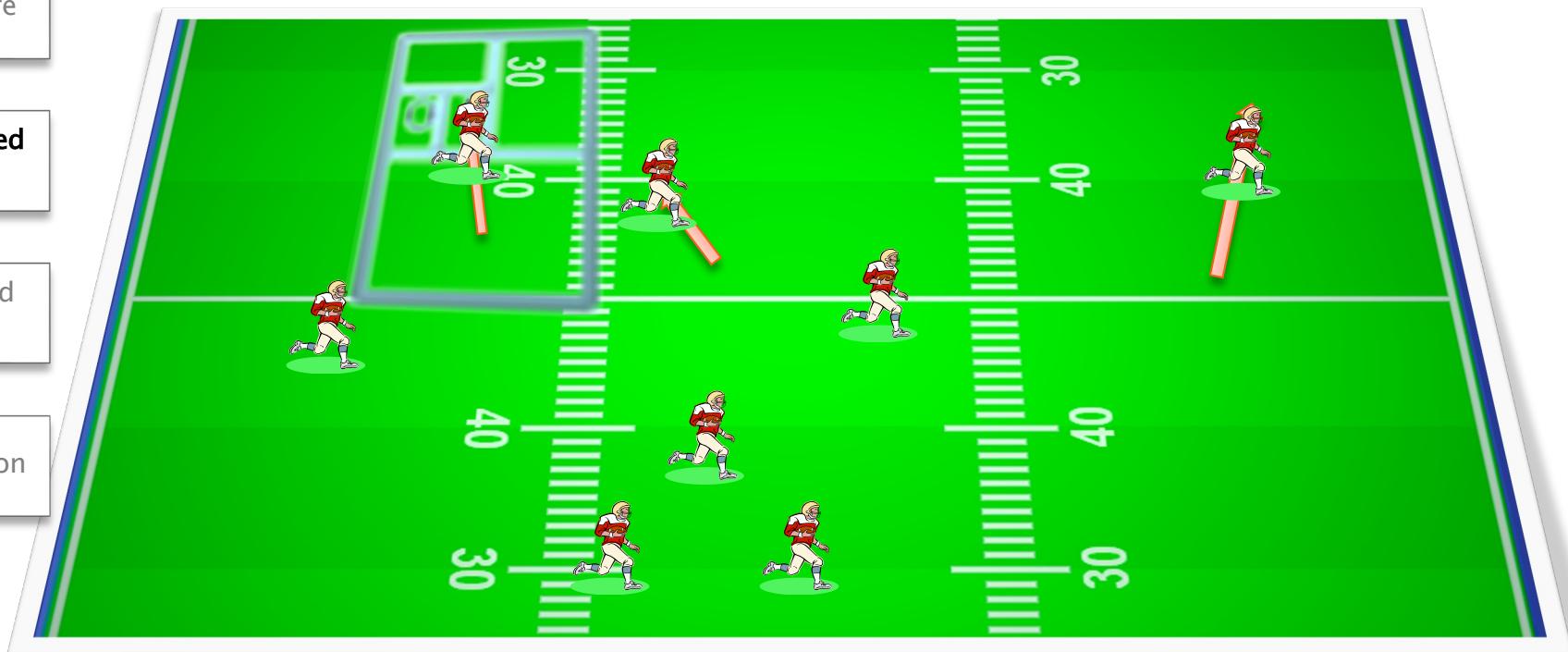
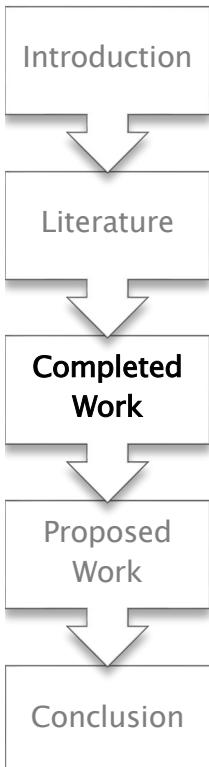


Single-Action
Static

Single-Action
Dynamic

Multi-Action
Static

Multi-Action
Dynamic

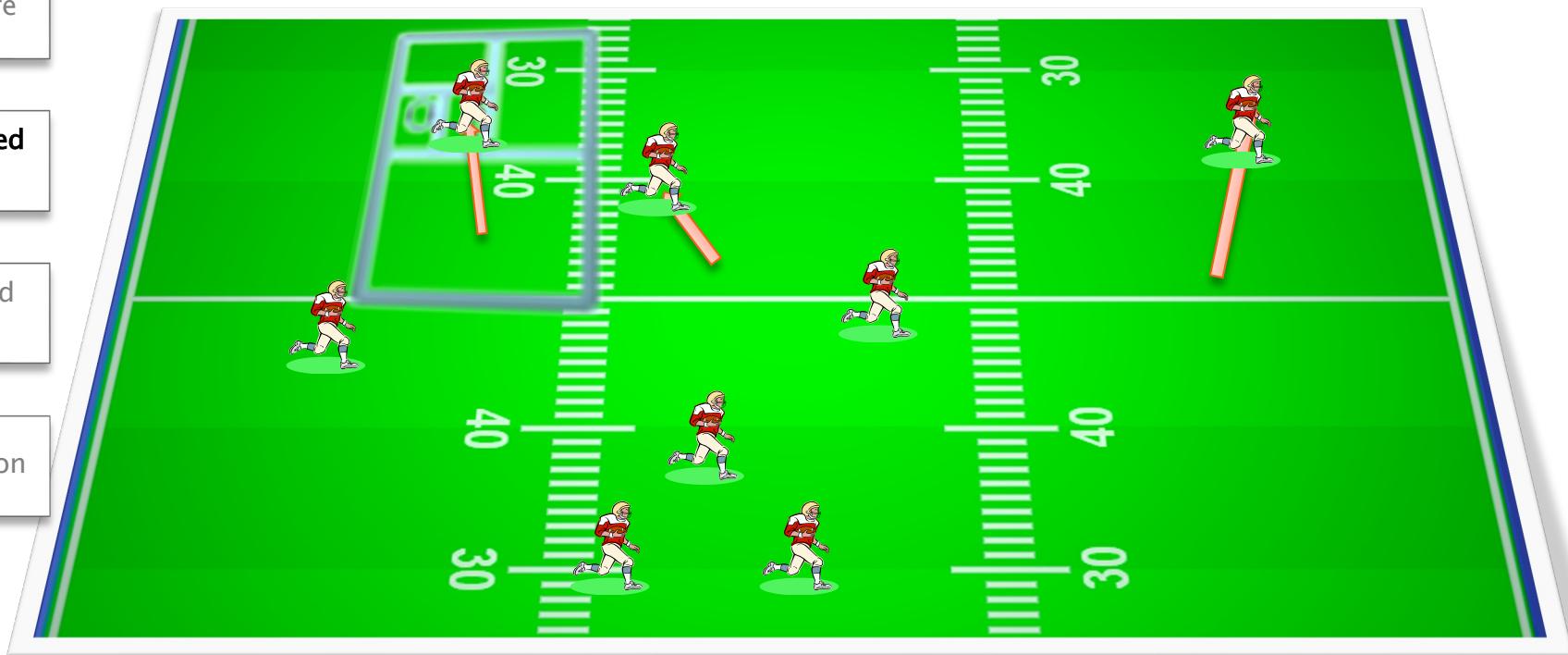


Single-Action
Static

Single-Action
Dynamic

Multi-Action
Static

Multi-Action
Dynamic

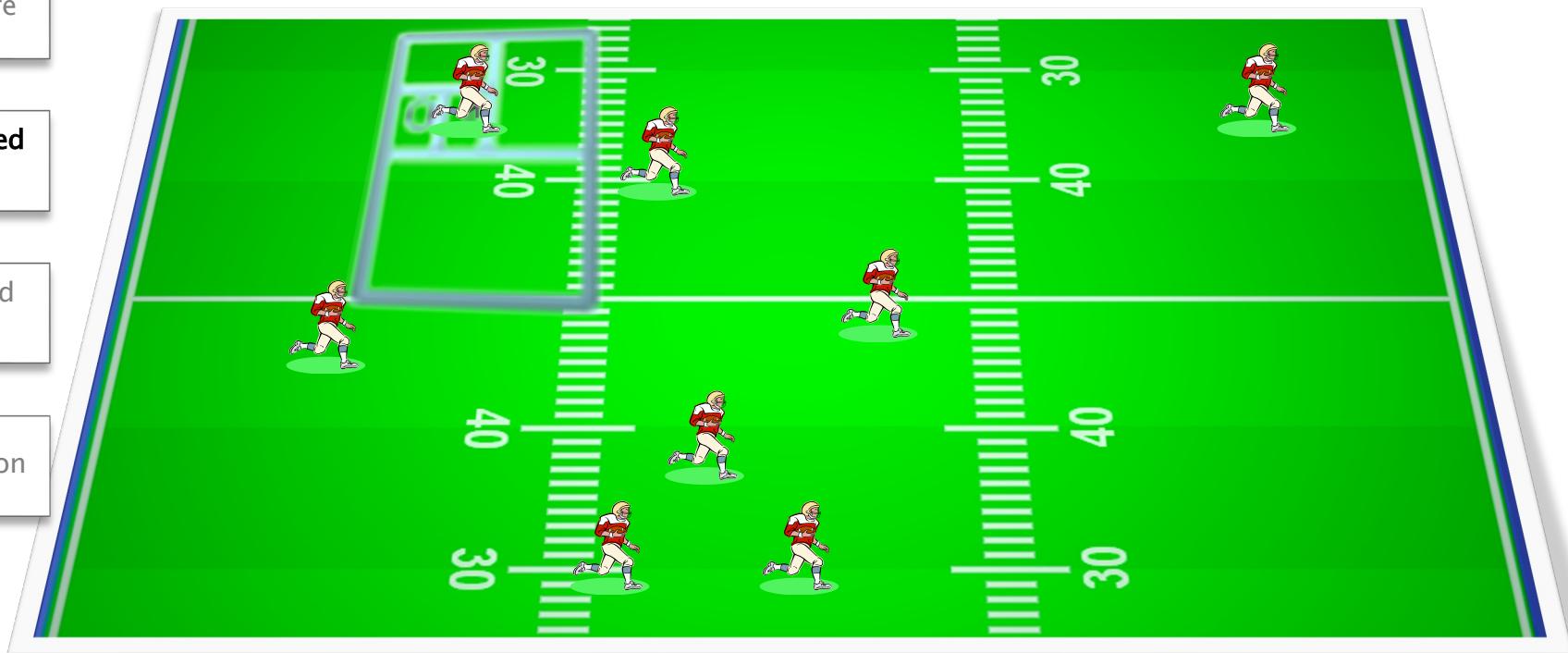
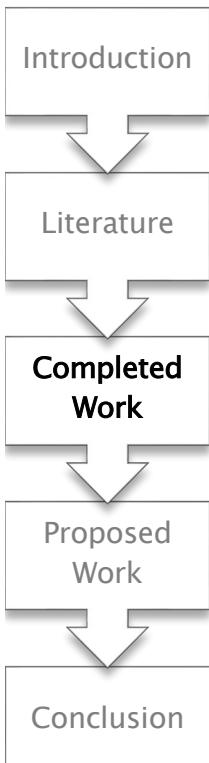


Single-Action
Static

Single-Action
Dynamic

Multi-Action
Static

Multi-Action
Dynamic



Single-Action
Static

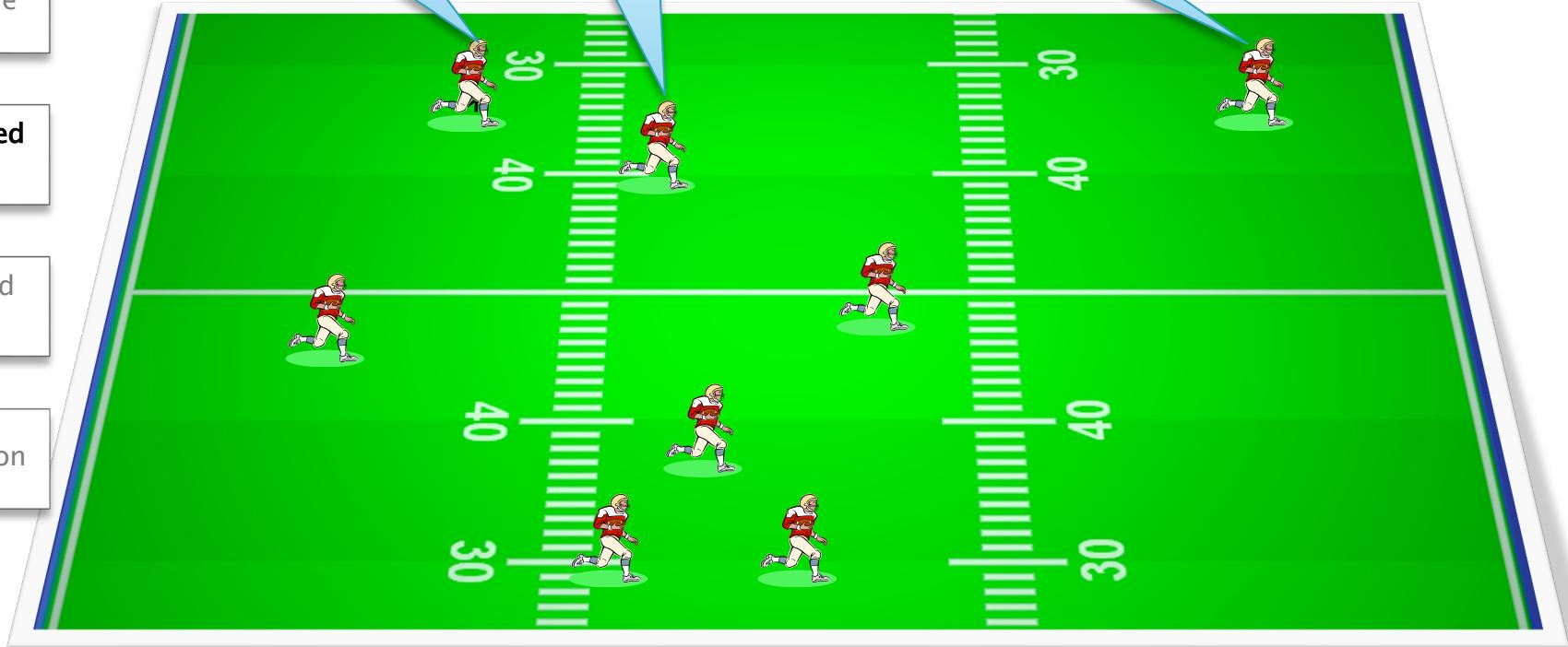
Single-Action
Dynamic

Multi-Action
Static

Multi-Action
Dynamic



Sweep Charge PassBlock



Single-Action
Static

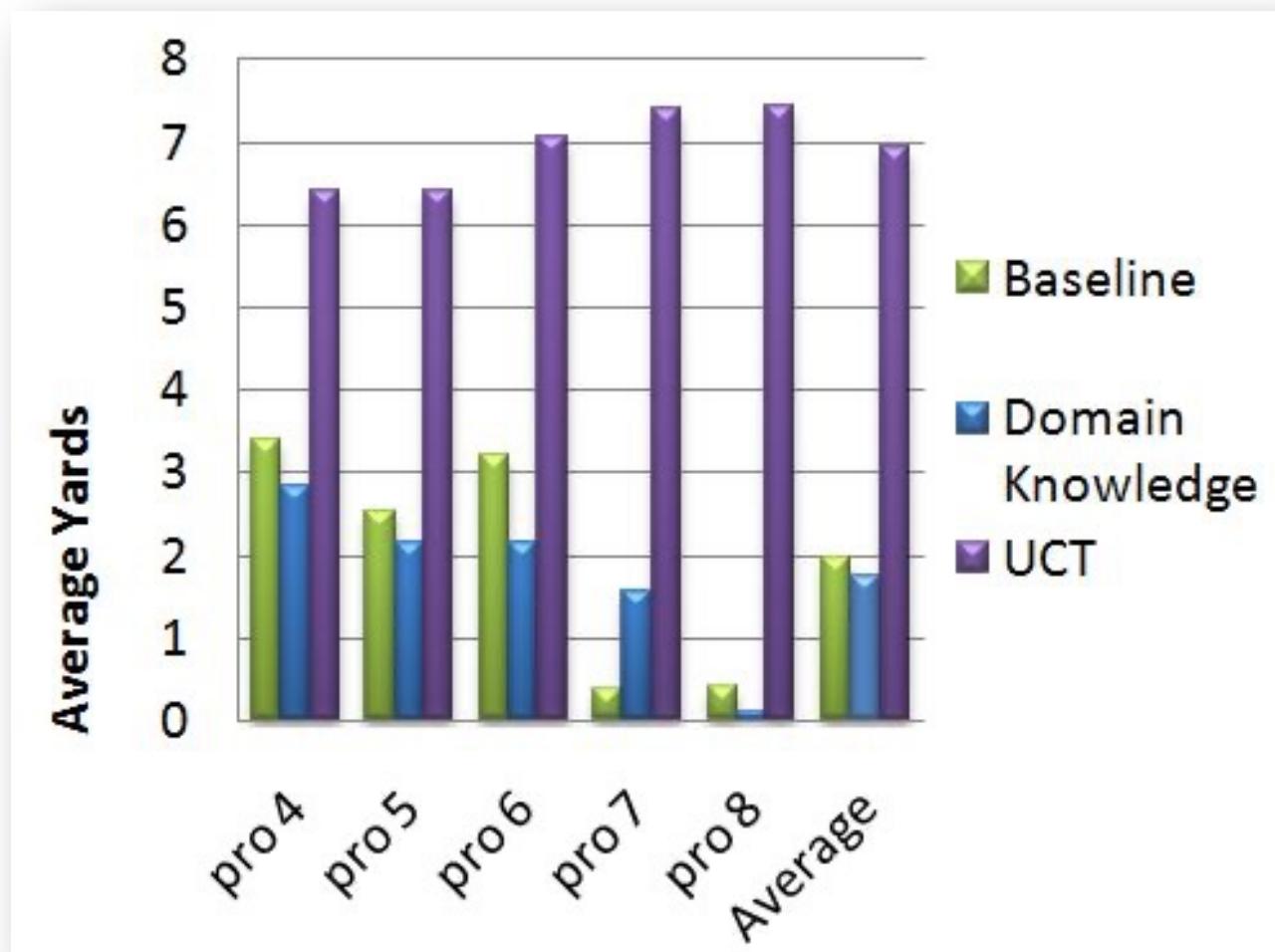
Single-Action
Dynamic

Multi-Action
Static

Multi-Action
Dynamic



Results



Acknowledgments

- Slides from Alan Fern's ICAPS tutorial on Monte Carlo Planning and Ken Lavers dissertation

Next Time

- Now you're all experts on planning!
- Now we'll move into a discussion of multi-agent problems that remain challenging for planners
 - Dynamic world
 - Multiple (adversarial) agents
 - Imperfect/uncertain info
- Reading: A. Greenwald and P. Stone, Autonomous Bidding Agents in the Trading Agent Competition IEEE Internet Computing, 5(2), March/April 2001
- Reading: A. Greenwald and J. Boyan Bidding Under Uncertainty: Theory and Experiments In Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (Jul 2004), pp. 209-216.
- First homework assignment to be released next week

CAP6671 Intelligent Systems

Lecture 6: Trading Agent Competitions: Bidding under Uncertainty

Instructor: Dr. Gita Sukthankar

Email: gitars@eeecs.ucf.edu

Planning Overview

Even when classical planning assumptions are not valid, planning can still be used as a useful tool to produce a starting plan/policy for future action.

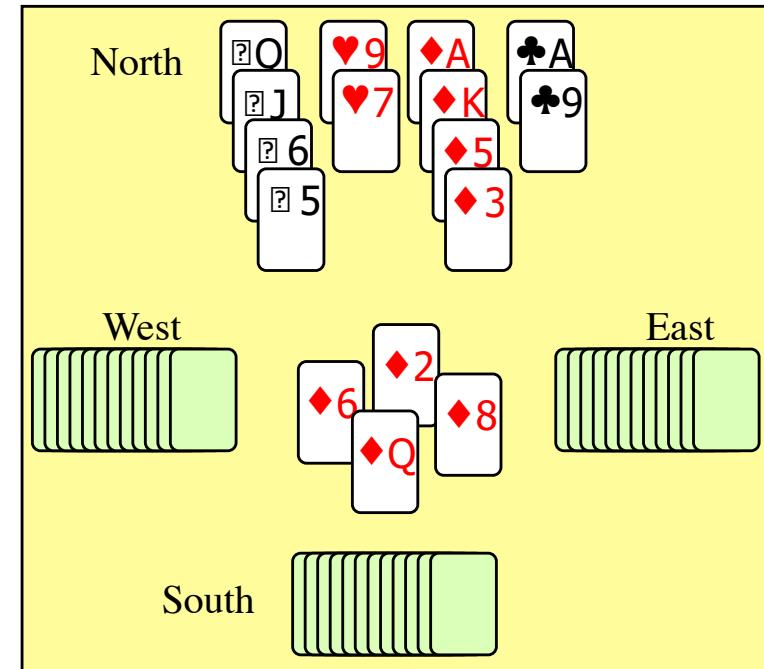
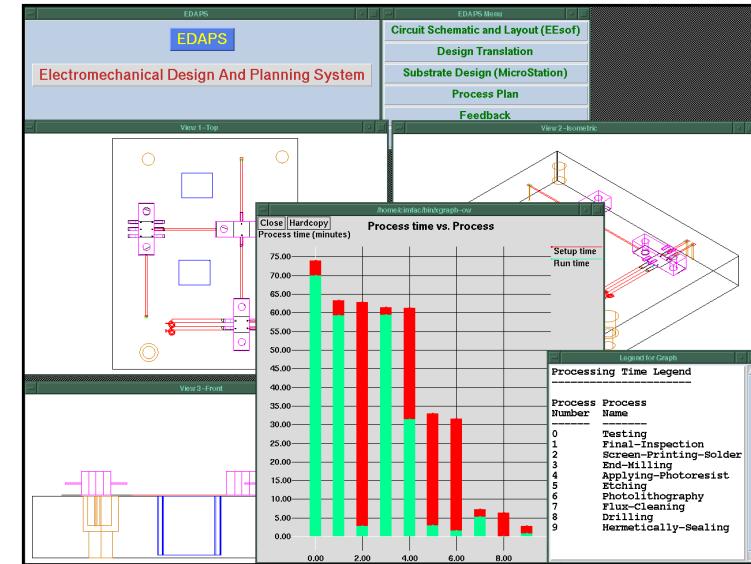
After all, people still make plans about their future even though they know that many things might change.

Examples:

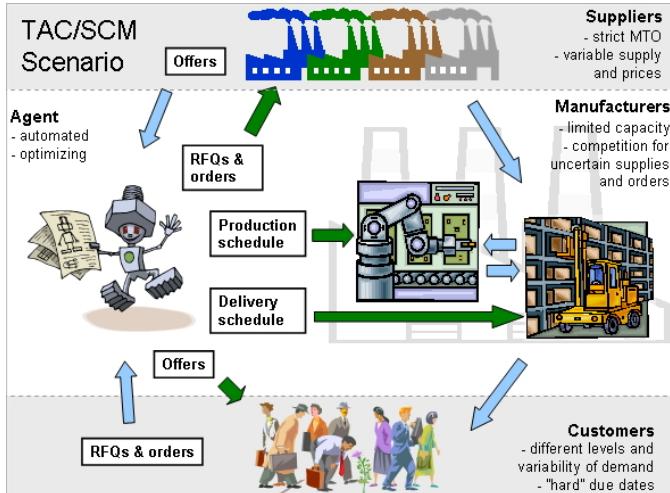
- Create a plan using static environment assumptions and repair it when things go wrong
- Create a plan for many time steps out, use it to optimize a single action choice, and replan at every time step
- Model action uncertainty using an Markov Decision Process; use domain knowledge to come up with the model of world dynamics.
- Create an approximate simulator of the world and use Monte Carlo search techniques (e.g. UCT) to identify a policy
- Use reinforcement learning to learn about the world dynamics from data (will be discussing in two weeks)

Example Agent Problems

- ?
- Typical characteristics of application domains
 - ?
 - Dynamic world
 - ?
 - Imperfect/uncertain info
 - ?
 - External info sources
 - » users, sensors, databases
 - ?
 - Durations, time constraints, asynchronous actions
 - ?
 - Limited resources
 - ?
 - Multiple agents/adversaries/group dynamics
 - ?
 - These problems can become hard to model with existing techniques.*



AI Challenge Problems



Trading Agent Competition



Starcraft AIIDE/CIG



Robocup Soccer

Why have agent competitions?

Why have agent competitions?

- Drive research toward important problems that have been identified by the general AI community
- Create standardized testbed to allow for head-to-head comparison between different approaches
- Problems are more complicated than many of the toy problems commonly used by researchers
- Promote code sharing by releasing winning code from previous years competitions
- Competitions change every year to become more difficult
- What are the drawbacks?

Drawbacks

- Often winning solutions ignore certain aspects of the domain that made the problem challenging.
- Winning solutions that dominate in scaled-down worlds might not be applicable for larger real-world problems.
- Promising research paths identified by losers can be ignored
- But still a great way to drive innovation in a certain area!

Reading

- To be discussed:
 - A. Greenwald and P. Stone, Autonomous Bidding Agents in the Trading Agent Competition IEEE Internet Computing, 5(2), March/April 2001
 - A. Greenwald and J. Boyan Bidding Under Uncertainty: Theory and Experiments In Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (Jul 2004), pp. 209-216.
 - A. Greenwald, S. Lee, V. Narioditskiy RoxyBot06: Stochastic Prediction and Optimization in TAC Travel Journal of Artificial Intelligence Research (2009), pp. 513-546

Why care about trading agents?

Why care about trading agents?

- Growing importance of web based auctions such as eBay
- “Shopping agents” are a useful class of personal assistant agents
- Trading is becoming an important aspect of MMORGs

What are the research problems?

What are the research problems?

- Rapid optimization/search
- Computing valuations of items
- Robust bidding strategies
- Prediction of future prices (e.g. hotel auction)
- Planning under uncertainty (possibility of not acquiring resources)
- Learning models from past data
- **Adversarial strategies**

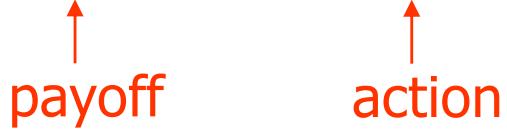
What Is Game Theory About?

- To understand how decision-makers interact
- A brief history
 - **1920s**: study on strict competitions
 - **1944**: Von Neumann and Morgenstern's book
Theory of Games and Economic Behavior
 - **After 1950s**: widely used in economics, politics, biology...
 - Competition between firms
 - **Auction design**
 - Role of punishment in law enforcement
 - International policies
 - Evolution of species

Elements of a Game: Strategies

- Decision-maker's choice(s) in any given situation
- Fully known to the decision-maker
- Examples
 - Price set by a firm
 - Bids in an auction
 - Routing decision by a routing algorithm
- Strategy space: set of all possible actions
 - Finite *vs* infinite strategy space
- Pure *vs* mixed strategies
 - Pure: deterministic actions
 - Mixed: randomized actions

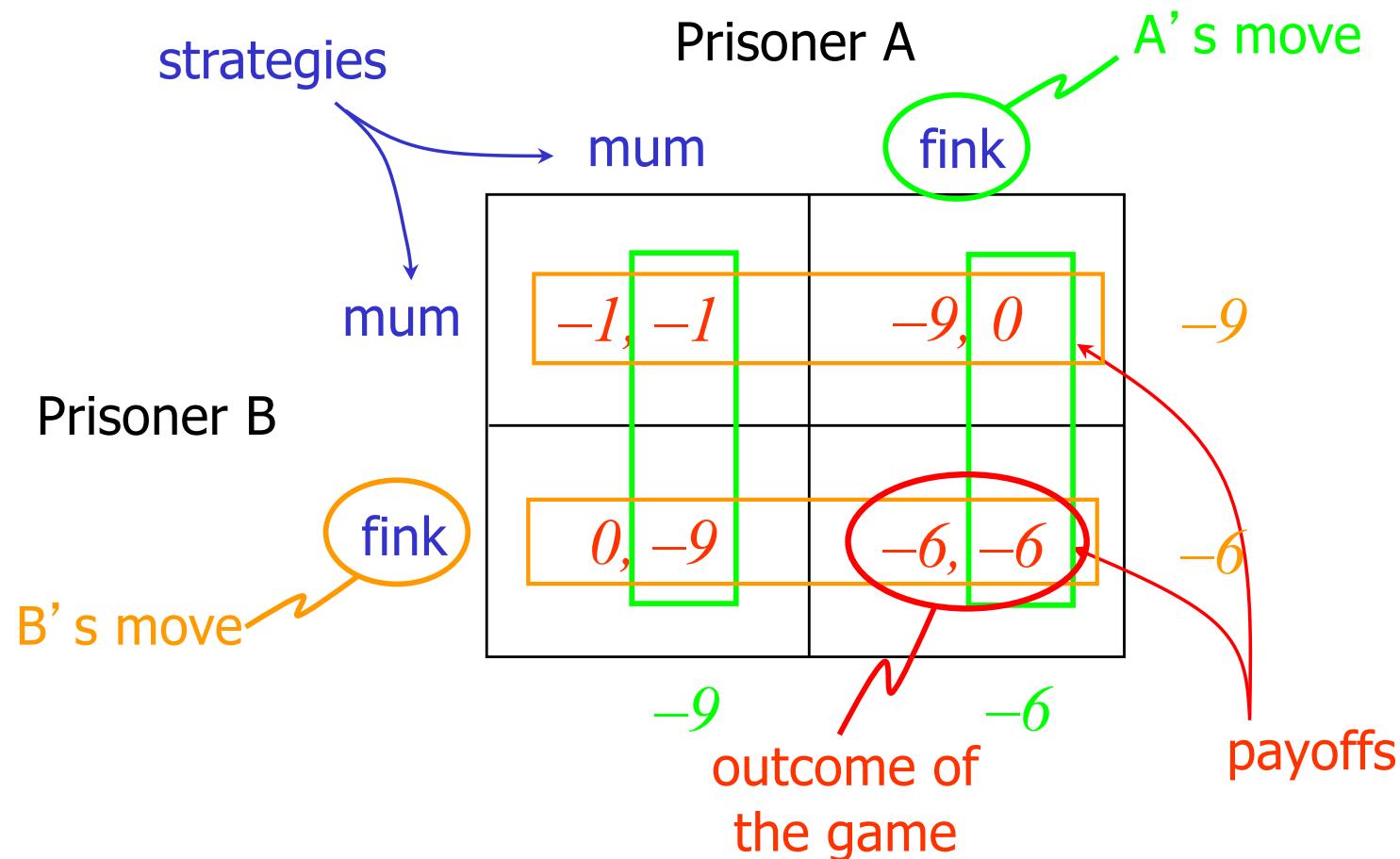
Elements of a Game: Preference and Payoff

- Preference
 - Transitive ordering among strategies
if $a \gg b$, $b \gg c$, then $a \gg c$
- Payoff
 - An order-preserving mapping from preference to \mathbf{R}^+
 - Example: in flow control, $U(x) = \log(1+x) - px$


Rational Choice

- Two axiomatic assumptions on games
 1. *In any given situation a decision-maker always chooses the action which is the best according to his/her preferences (a.k.a. rational play).*
 2. *Rational play is common knowledge among all players in the game.*

Example: Prisoners' Dilemma

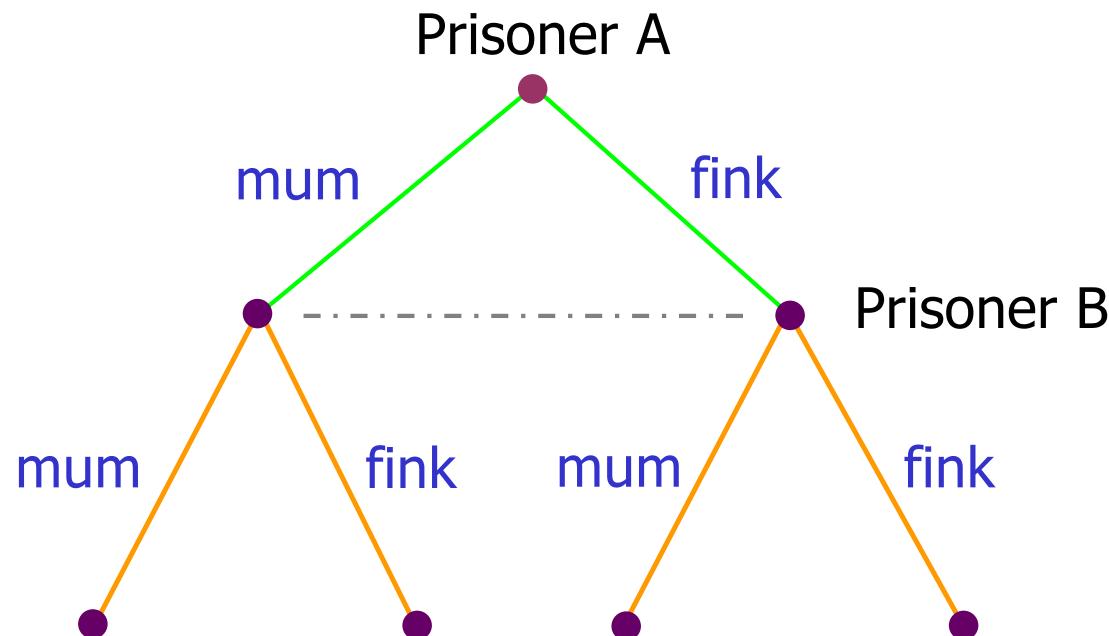


Different Types of Games

- Static *vs* multi-stage
 - Static: game is played only once
 - Prisoners' dilemma
 - Multi-stage: game is played in multiple rounds
 - Multi-round auctions, chess games
- Complete *vs* incomplete information
 - Complete info: players know each others' payoffs
 - Prisoners' dilemma
 - Incomplete info: other players' payoffs are not known
 - Sealed auctions

Representations of a Game

- Normal- *vs* extensive-form representation
 - Normal-form
 - like the one used in previous example
 - Extensive-form



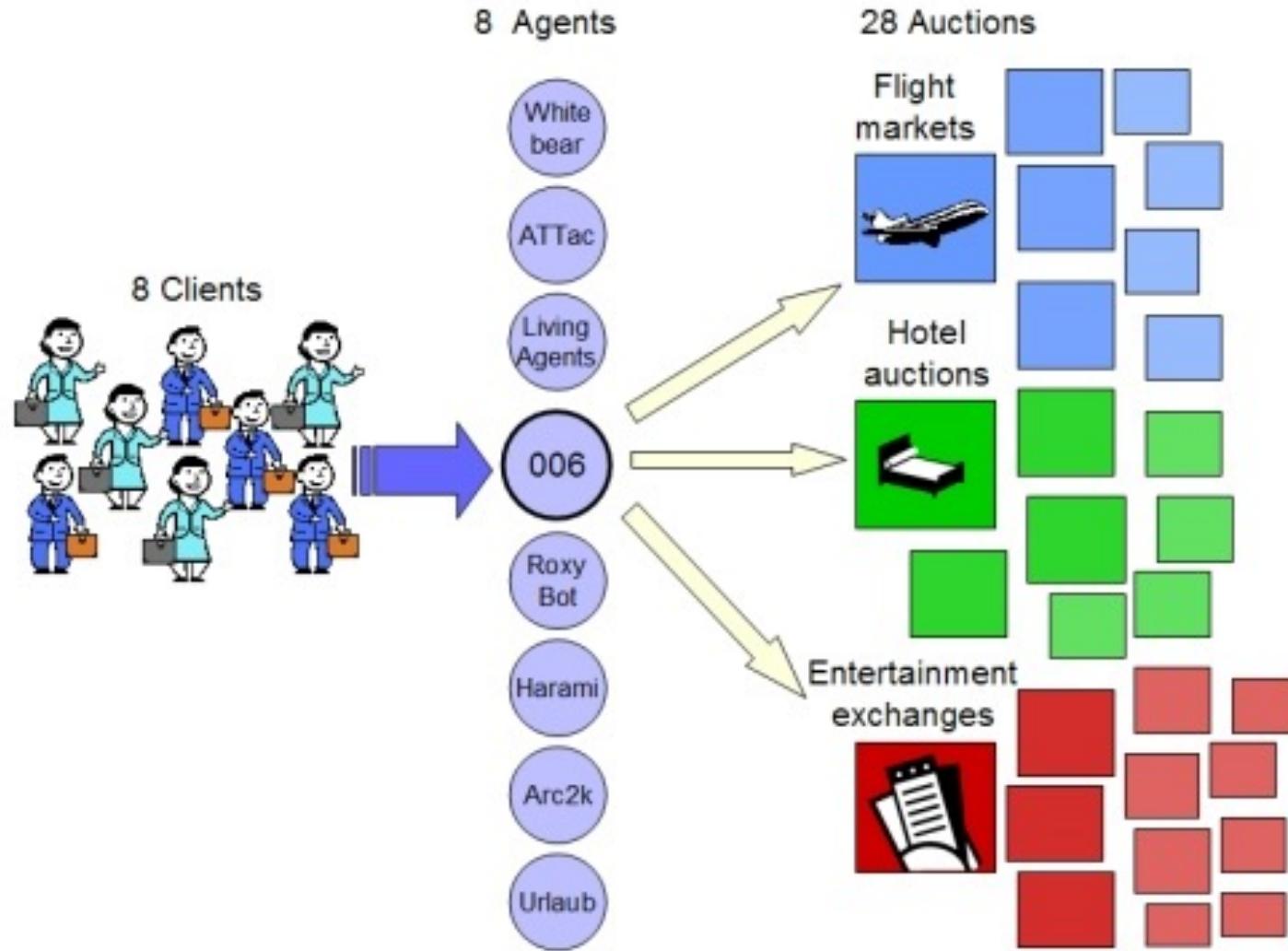
History of TAC

- First competition was in 2001, organized by Michael Wellman
- Competitions are often held at AAAI or AAMAS
- Variants on competition that we'll be discussing.
 - TAC Classic: agents act as personal travel agents for a group of clients and attempt to maximize their clients utilities
 - TAC Supply Chain Management: agents manufacture PCs, win customer orders, procure components
 - TAC Ad Auctions: compete to sell products by bidding to have ads shown in sponsored search engine results

New Variants

- Power TAC: bidding strategies for smart power grid economies (electricity trading)
- TAC Market Design: Organizers provide the agents and the entrants set the market rules to win the game according to a certain set of assessment criteria. Traders can move between markets to explore and select the best market to trade in to maximize their own profits.

TAC Classic Competition Rules



Game Design

- Game:
 - 8 agents competing in ~15 minute games
 - Agents are simulated travel agents with 8 clients
 - Client needs to travel from TACtown to Boston and home again in a 5 day period
- Auctions for flight, hotels, and entertainment ticket
 - Server maintains markets, sends prices to agents
 - Agents sends bids to server over network (must be able to cope with network issues)

28 Simultaneous Auctions

- Flights: Inflight days 1-4, Outflight days 2-5 (8)
 - Separate auction for each type of plane ticket
 - Ask price set by server periodically increases/decreases randomly
 - Obtain ticket by bidding at or above ask price
- Hotels: Expensive hotel/cheap hotel for days 1-4 (8)
 - 16 rooms per auction; 16th price ascending English auction; no resale
 - Ask price published by server is 16th highest price; no information about other bids
 - No bid withdrawal; no resale
 - Hotel auctions can close after a specified period of inactivity
- Entertainment: 3 different types of tickets, 4 nights (12)
 - Agents start with a set of entertainment tickets
 - Server publishes bid-ask spreads (highest bid price, lowest ask price)
 - Continuous double auction (no trading phases, prices to buy and sell may be submitted at any time)
 - Resale allowed

Auction Definitions

- **Reservation price:** seller will not sell below the price
- **Clearing mechanism:** process of matching buyers and sellers
- **English auction:** 1st price open-cry ascending auction
 - Dominant strategy: bid at current price plus small increment until the price goes over your valuation
- **Dutch auction:** 1st price open-cry descending auctions
 - No dominant strategy; highly real-time efficient
- First price sealed bid
 - No dominant strategy (spying?)
- **Vickery auction:** sealed bid but price paid is the second highest bid
- **Double auction:** buyers place buy and sell order for instances of items; orders are periodically cleared by matching up existing orders

Client Preferences

- Preferences: randomly generated per client
 - Ideal arrival, departure days
 - Good hotel value
 - Entertainment values

Agent Design

- Bidding: offering payments for goods to gain utility
- Allocating: Constructing travel packages for each of the 8 clients
- After auctions close, agents have 4 minutes to report allocations of goods to clients.
- Score: difference between summed clients utilities and agents expenditure

Game Structure

1. Get market prices from server
2. Decide on what goods to bid
3. Decide at prices
4. Decide for how many to bid
5. Decide at what time to bid

UNTIL game over

Allocate goods to clients

Bidding Strategies

- Hotel auction
 - Hotel rooms are a limited resource
 - Refrain from bidding early unless the auction seems near to closing
 - One strategy:
 - Treat current holdings as sunk costs and calculate the utility of an unsecured hotel room reservation as the utility of the package (marginal utility)
 - Bid this utility—due to the structure of the m-unit auction the agent will pay less than the closing price

Marginal Utility

- Marginal utility is a mechanism for determining whether it's worth bidding on a new object
- Definition:
 - Difference between the utility of owning the set plus the new object vs. owning the set without the new object minus purchase costs
- Utility of the set of objects is not always the sum of the utilities of owning individual items.
- Example:
 - MU(camera+flash) vs MU(flash alone)

Bidding Strategies

- Flight auction
 - Delay bidding on flights until end of game
 - In other versions of the competition the prices increased towards end of the game which made bidding earlier more advantageous
 - Account for unpredictable network and server delays to make sure bid is received before the game is over
 - Bid at maximum price to make sure that bids were not rejected because of information delays resulting from network asynchrony

Bidding Strategies

- Entertainment auction:
 - Focus on obtaining complete packages
 - Resell sub-optimal tickets
 - Separate the problem into calculating travel packages and entertainment packages separately and solving greedily
 - Greedy strategy has problems:
 - If client doesn't have ticket to event then better to extend client stay when utility gain exceeds cost of ticket plus hotel plus travel penalties
 - Similarly it can be better to shorten stay and sell off tickets.

Problem Formulation

- Acquire a certain number of items within a period of time
- Can we write a plan to handle item acquisition?
 - Input: domain information
 - State: current ask/sell prices
 - Output: list of bidding actions
 - Bid(Item, Price, Time, Number)

Problems with Planning for TAC

- Continuous state, action space makes it difficult to consider all alternatives
- High probability of failure at every decision point
- **Simultaneous decision on multiple items**
- Irrecoverable errors
- **Dependencies between goals**
- Parallel decision-making (bidding in simultaneous auctions)

How to build an agent

- What to buy?
- How much are items worth (valuation)?
- How much are items likely to cost (prediction)?
- How to win a bid?
- How to allocate items between customers?

Short Cuts

- Factor the problem into two separate subproblems?
 - Optimize complete package vs. optimize travel package and entertainment package separately
- Greedy client allocation strategies?
 - Allocate tickets to clients sequentially
 - 2 tickets for an evening event (theater and symphony each for \$50)
 - A values symphony at \$90, theater at \$80, baseball at \$70
 - B values symphony at \$175, theater at \$150, baseball at \$125
 - Allocate A before B yields \$140 vs. \$155 utility

Results from 2000 TAC Competition

1. ATTac (P. Stone)
2. Roxybot (J. Boyan and A. Greenwald)
3. Aster (InterTrust Research Lab)
4. UMBCTAC (UMBC)

Differences between Bots

- Search process used to decide which items to acquire
- Price prediction model
- How to model already acquired items: “sunk” costs
- “Sunk” cost: a cost that has already been incurred and can’t be recovered
- What to bid and when to bid: note that bid is **not** equal to acquisition price in the hotel auction

ATTac

- Bidding:
 - Calculate G^* (most profitable allocation of goods to clients based on current holdings and predicted future prices) for use in bidding marginal utility
 - Buy/sell bids for entertainment based on a sliding price strategy (dependent on time till end of game)
- Allocation:
 - Uses mixed-integer linear programming to find optimal allocation
- Online adaptation to game conditions:
 - Passive/active bidding modes based on server latency
 - Hotel bidding based on closing prices in previous games
 - Calculate accurately; bid aggressively

Linear Programming

- Optimize a linear objective function subject to linear equality and inequality constraints

$$\begin{aligned} & \text{maximize} && \mathbf{c}^T \mathbf{x} \\ & \text{subject to} && A\mathbf{x} \leq \mathbf{b} \\ & \text{and} && \mathbf{x} \geq \mathbf{0} \end{aligned}$$

- Used extensively in business, operations research, and military planning
- Can be solved using simplex or interior-point methods
- If some of the unknown variables are required to be integers, it becomes a mixed integer linear programming problem
- MILP problems can be solved by using the LP solution as the starting point for a branch and bound search.

Example: ILP

Objective Function

$$\max_{\Gamma, \Phi, M, N, Y, Z} \sum_S \left(\underbrace{\sum_{C,T} \mathcal{U}_{ct} \gamma_{cts}}_{trip value} - \underbrace{\sum_{A_f} \left(\overbrace{\mathcal{M}_a \mu_a}^{current} + \overbrace{\mathcal{Y}_{as} v_{as}}^{future} \right)}_{flight cost} - \underbrace{\sum_{A_h, Q, p \geq \mathcal{Y}_{as}} \mathcal{Y}_{as} \phi_{apq}}_{hotel cost} + \right.$$

Constraints

$$\sum_{A_e} \left(\underbrace{\mathcal{N}_a \nu_a + \mathcal{Z}_{as} \zeta_{as}}_{event revenue} - \underbrace{\mathcal{M}_a \mu_a - \mathcal{Y}_{as} v_{as}}_{event cost} \right)$$

$$\sum_T \gamma_{cst} \leq 1 \quad \forall c \in C, s \in S$$

$$\sum_{C,T} \underbrace{\gamma_{cst} \mathcal{G}_{at}}_{allocation} \leq \underbrace{\mathcal{O}_a}_{own} + \underbrace{(\mu_a + v_{as})}_{buy} \quad \forall a \in A_f, s \in S$$

$$\sum_{C,T} \underbrace{\gamma_{cst} \mathcal{G}_{at}}_{allocation} \leq \underbrace{\mathcal{O}_a}_{own} + \underbrace{\sum_{Q,p \geq \mathcal{Y}_{as}} \phi_{apq}}_{buy} \quad \forall a \in A_h, s \in S$$

$$\sum_{C,T} \underbrace{\gamma_{cst} \mathcal{G}_{at}}_{allocation} \leq \underbrace{\mathcal{O}_a}_{own} + \left(\underbrace{\mu_a + v_{as}}_{buy} \right) - \left(\underbrace{\nu_a + \zeta_{as}}_{sell} \right) \quad \forall a \in A_e, s \in S$$

$$\sum_{P,Q} \phi_{apq} \geq \mathcal{H}_a \quad \forall a \in A_h$$

$$\sum_P \phi_{apq} \leq 1 \quad \forall a \in A_h, q \in Q$$

RoxyBot

- Allocation
 - Using an A* search with admissible heuristic
 - Narrow the search from 10^{30} to 10^3
 - First levels of the search allocate hotels and flights; later levels allocate entertainment prices
- Completer
 - Variable-width beam search
 - Optimal quantity of resources to buy and sell using priceline structure to forecast future costs
 - Pricelines are learned using ML techniques (whereas ATTac uses heuristics to estimate future prices)
 - Bidding uses sample average approximation instead of MU

Aster

- Heuristic bidding and locally optimal search for final allocation
- Each iteration it estimates costs of resources, computes a tentative allocation, and bids for some resources
- Bidding:
 - Delay precommit phase
 - Bid for consecutive nights
 - Calculate utility of other agents when doing entertainment bids

UMBCTAC

- Allocation:
 - Consider agent itineraries individually rather than solving 8 client optimization problem
 - Relies on latest market quotes for price estimation
 - Switch itineraries often early on and then avoid switching itineraries late in game
- Bidding
 - Flights: bid maximum price
 - Hotels: bid current price plus a price increment based on past transactions
 - Entertainment:
 - Buy tickets if client is in town that night at market value
 - Sell tickets at average of preference values

Bidding Strategies

- Flights: delay commitment
- Entertainment tickets: resell sub-optimal decisions
- Hotels?
 - Irrevocable resource commitment
 - Simultaneous auctions
 - Combinatorial valuations

We only need to own one hotel room per night per client; owning anything more than that loses money.

Combinatorial Valuations

- Complements

$$v(X\bar{Y}) + v(\bar{X}Y) \leq v(XY)$$

- Camera, flash, and tripod

- Substitutes

$$v(X\bar{Y}) + v(\bar{X}Y) \geq v(XY)$$

- Canon AE-1 and A-1

Marginal Utility

- Marginal utility is a mechanism for determining whether it is worth bidding on a new object
- Definition:
 - Difference between the utility of owning the set plus the new object vs. owning the set without the new object minus purchase costs
- Formally:

the *marginal utility* of good $x \notin X \cup Y$ is defined as follows: $\mu(x, X, Y, \vec{p}) = \alpha(X \cup \{x\}, Y, \vec{p}) - \alpha(X, Y, \vec{p}).$
- Example:
 - MU(camera+flash) vs MU(flash alone)

Problems with Marginal Utility

- Simultaneous auctions
 - Marginal utility does not account for the possibility of obtaining other substitute valuations through simultaneous auctions.
 - Example:
 - A set of N items is auctioned simultaneously, and the combinatorial valuation of one or more of these items is 2.
 - Imagine an auction where the prices are set deterministically and each of the items costs 1.
 - Bidding the marginal utility in each auction (1) will lose us money since we will win all of N auctions, pay N , yet only obtain a utility of 2 ($2-N$)
 - We would have done better by bidding 1 for one good ($2-1$)
 - Failed to take into account our expected chance of winning auction

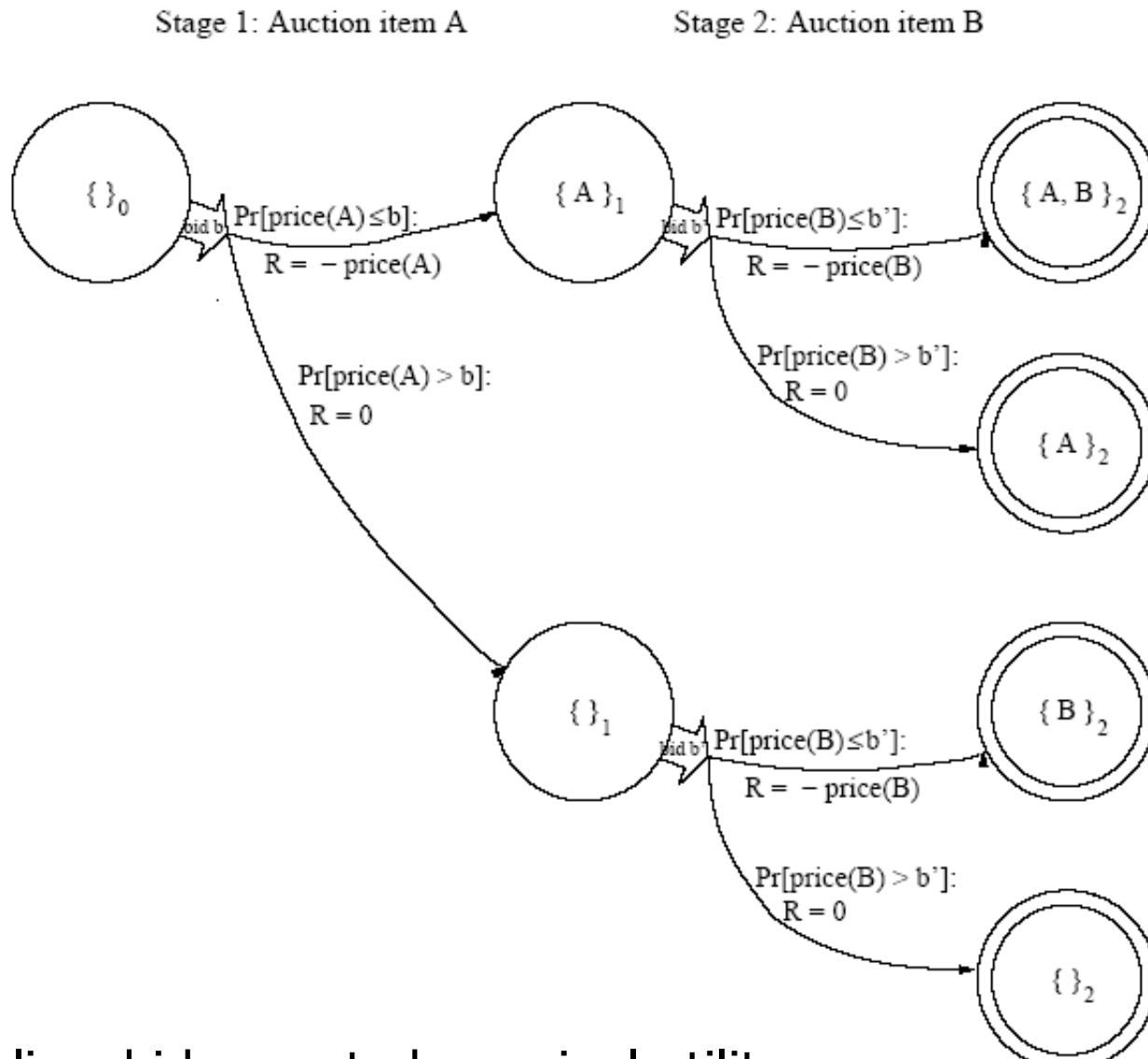
Research Problem

- Comparison of marginal utility bidding (ATTac) and policy search (RoxyBOT)
- Results:
 - Marginal utility is not optimal in simultaneous auction (as shown by example)
 - Optimal in sequential option
 - Empirical results demonstrate that MU bidding is a reasonable heuristic for TAC Classic hotel auctions

Markov Decision Processes

- Sequential auctions can be modeled as an MDP.
- Classical planning models:
 - logical representation of transition systems
 - goal-based objectives
 - plans as sequences
- Markov decision processes generalize this view
 - controllable, stochastic transition system
 - general objective functions (rewards) that allow tradeoffs with transition probabilities to be made
 - more general solution concepts (policies)

MDP for Bidding



Optimal policy: bid expected marginal utility

Simultaneous Auctions

- How are simultaneous auctions different?
- Camera and flash scenario:
 - $U(\text{camera, flash})=750$
 - $U(\text{camera})=0$
 - $U(\text{flash})=0$
 - $\text{Price}(\text{flash})=50$
 - $\text{Price}(\text{camera})$ either \$500 (0.5) or \$1000 (0.5)
 - What to bid?
 - Bid of (0,0) yields an expected utility of \$0
 - Bid of (500,50) yields an expected utility of \$75 (\$200 half the time, -\$50 half the time)

Problem Formulation

- Best solution: expectation over all possible outcomes (win Item 1, win Item 2)
- Problem: exactly computing this expectation is exponential in number of goods
- Approaches:
 - Expected MU bidding
 - Expected value with MU bidding
 - Stochastic sampling technique
- Interactions between auction outcomes are what make the problem tricky

Expected MU bidding (ATTac)

Scenario: We would like to own one of two goods (x, y) (utility=1 for owning either). Prices of goods follow a bimodal distribution (probability 0.5 price=1, probability 0.5 price=101)

Method: expected MU (as calculated by an MDP) considers each the result of each auction separately

Result: expected marginal utility bidding (bid 1 on both goods) performs worse than bidding 0 on both of the goods

x	y	$\mu(x)$	$\mu(y)$	Evaluation
1	1	1	1	-1
1	101	1	1	0
101	1	1	1	0
101	101	1	1	0
Average		1	1	$-\frac{1}{4}$

Expected Value Method

- Expected value: solve deterministic version of problem using prices calculated by expected values
 - $U(\text{camera, flash})=750$
 - $\text{Price(flash)}=50$
 - Price(camera) either \$500 (0.5) or \$1000 (0.5)
 - Calculate policy using expected price
 $\$750+\$50=\$800$
 - Since expected price is higher than utility, expected value recommends no bid
 - But that isn't quite right either...

Expected Value Method/MU Bid

- Marginal utility bidding can do better than using the expected value.
- Example:
 - Consider an item A with value \$100
 - A costs \$1 with probability .9 but \$1 million with probability .1
 - The expected price of the good is $\sim \$1000010$
 - Since the expected price is higher than our value we bid 0
 - However the policy “bid \$100” (expected marginal utility) scores roughly \$89 (.9 we win \$99)
 - EVMU: use mean to estimate price, determine whether item is desired, if so use marginal utility bidding

Sample Average Approximation

- Used by RoxyBot-02 (TAC-02)
- Solve stochastic program using a subset of scenarios
- Similar to UCT, SAA is a Monte Carlo search approach where we are taking samples over possible outcomes.
- Policies can be generated using MU, EVMU or expected MU rather than just selecting random policies.
- Sampling the outcomes is faster than exhaustive search

Method

- Use training data from other games to learn distributions for the clearing prices of each good
- Represent distributions as a lookup table indexed by different domain features
- Generate seed set of candidate policies using different bidding variants (including MU, EVMU)
- Evaluate the performance of the candidacy policies on 50 scenarios sampled from distribution

Results

- MU outperforms expected MU
 - RoxyBot-00 (EVMU) outperforms MU
 - RoxyBot-02 (using SAA) outperforms-00
-
- Problems:
 - SAA is very slow if it just uses brute force search
 - Needs a good heuristic to direct the search
 - Computing policies to direct the search is in itself computationally expensive

RoxyBot-06

- Most in depth description of how to construct a bot
- Predict: build stochastic models
 - Flights: Bayesian updating
 - Hotels: simultaneous ascending auction
 - Entertainment: sample historical data
- Optimize: sample average approximation
- Maintain three separate threads uploading bids to server

Take-Home Message

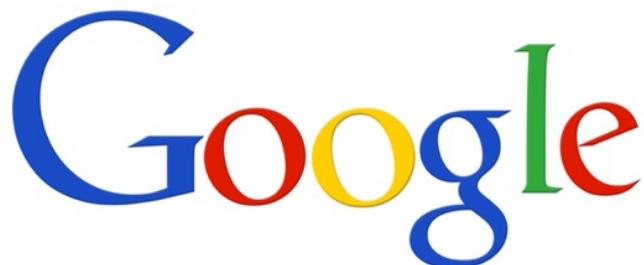
- Combinatorial valuation and simultaneous auctions make the TAC problem more difficult.
- Sequential auctions can be expressed as an MDP and solved with expected marginal utility bidding strategy.
- Bidding the expected marginal utility can lead to suboptimal results in problems.
- The paper presents a Sample Average Approximation algorithm in which all of these strategies are used to generate and evaluate possible bids.
- Key insight: in some cases modeling a distribution by its mean doesn't work well

Summary of RoxyBot

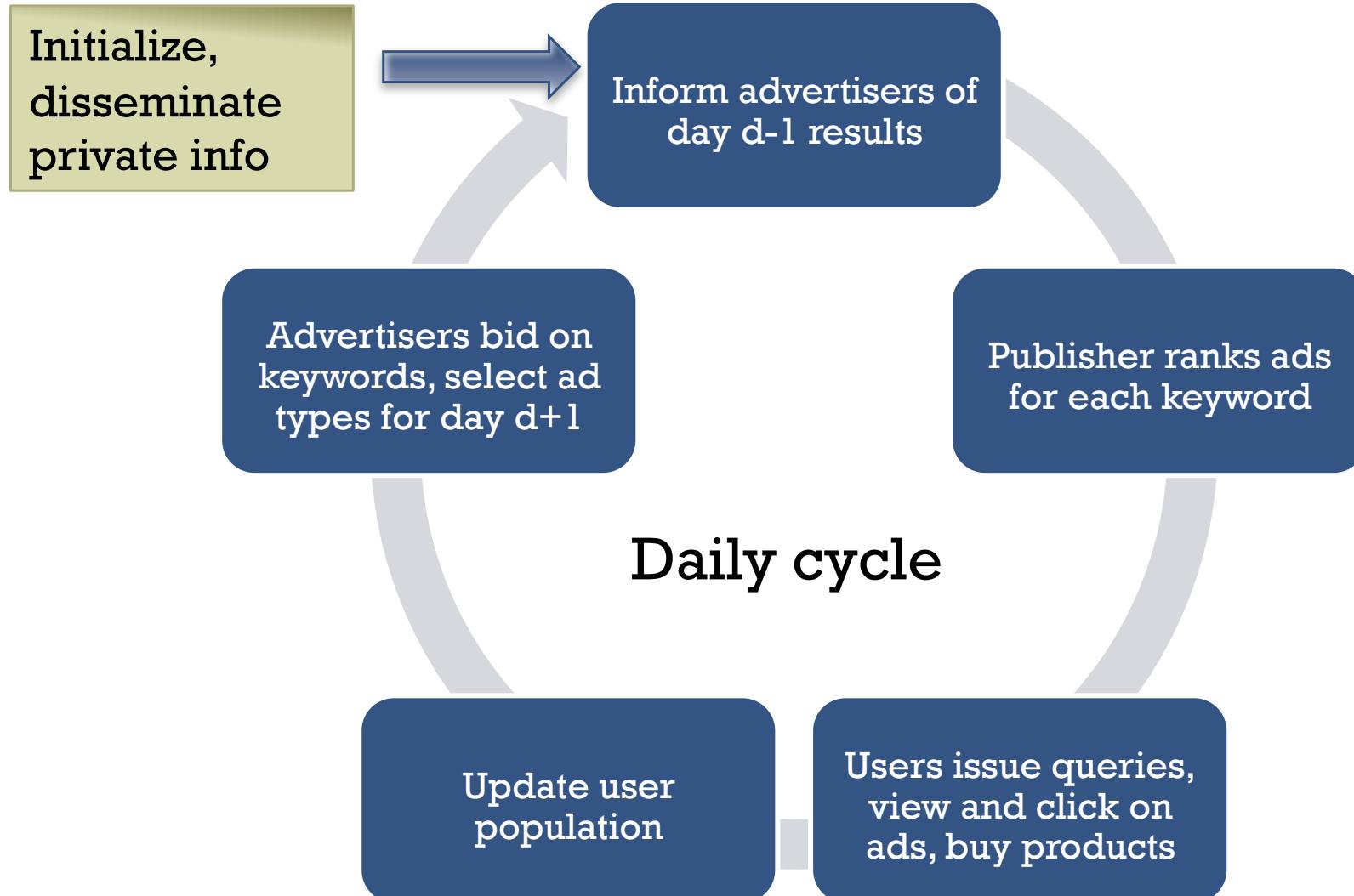
- Marginal utility is a reasonable upper bound for bidding since you never bid more than the set is worth.
- Calculating MU is exponential in the number of possible items in the set.
- Bidding strategies:
 - Marginal utility: does not account for probability of winning or losing other items
 - Expected marginal utility: optimal for sequential auction, can be calculated with MDP, takes into account probability of winning the set
 - Expected value/MU: use the expected amount that the item is expected to sell for to determine whether to bid, then bid marginal utility

Real World Applications

- Online advertisement is a billion dollar business now conducted using mini-auctions!



Agent Design

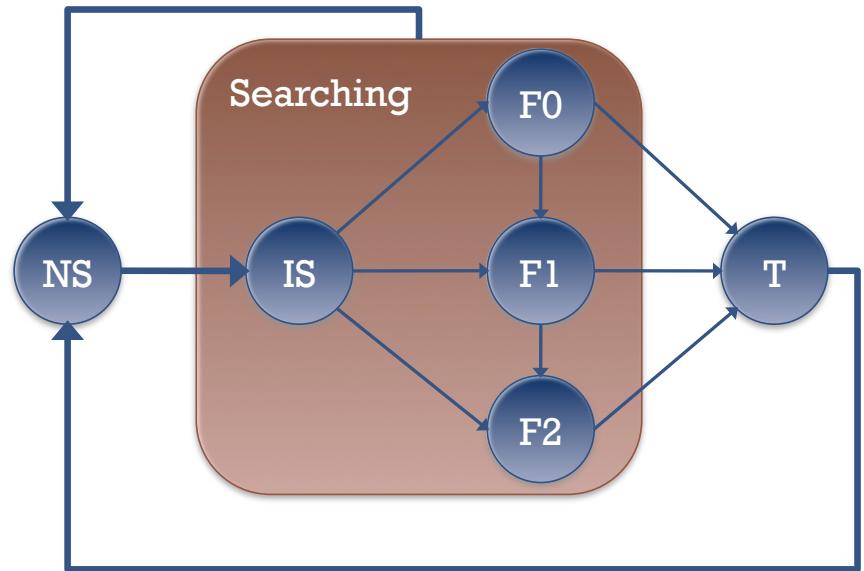


Competition Details

- **Users:**
 - Fixed pool who remain interested in one specific product
 - Transition through different interest states
- **Click model:**
 - Users shown ads in order of advertiser ranking
 - Click based on a hidden probability
 - Purchase with a probability that increases with focus
 - Advertisers have component and manufacturer specialties which modify revenue and conversion
- **Auction**
 - Rankings are done by combination of bid and click probability (squashed bid)
- **Capacity**
 - Manufacturers have capacity limitations
- **Information**
 - Daily report on impressions, clicks, and conversions

Domain

Manufacturer	Lioneer	PG	Flat
Type	TV	Audio	DVD



Available goods

User model

F0, F1, F2: type of queries

F0: null

F1: one null, one specified

F2: both specified

TacTex-09: Ad Auction

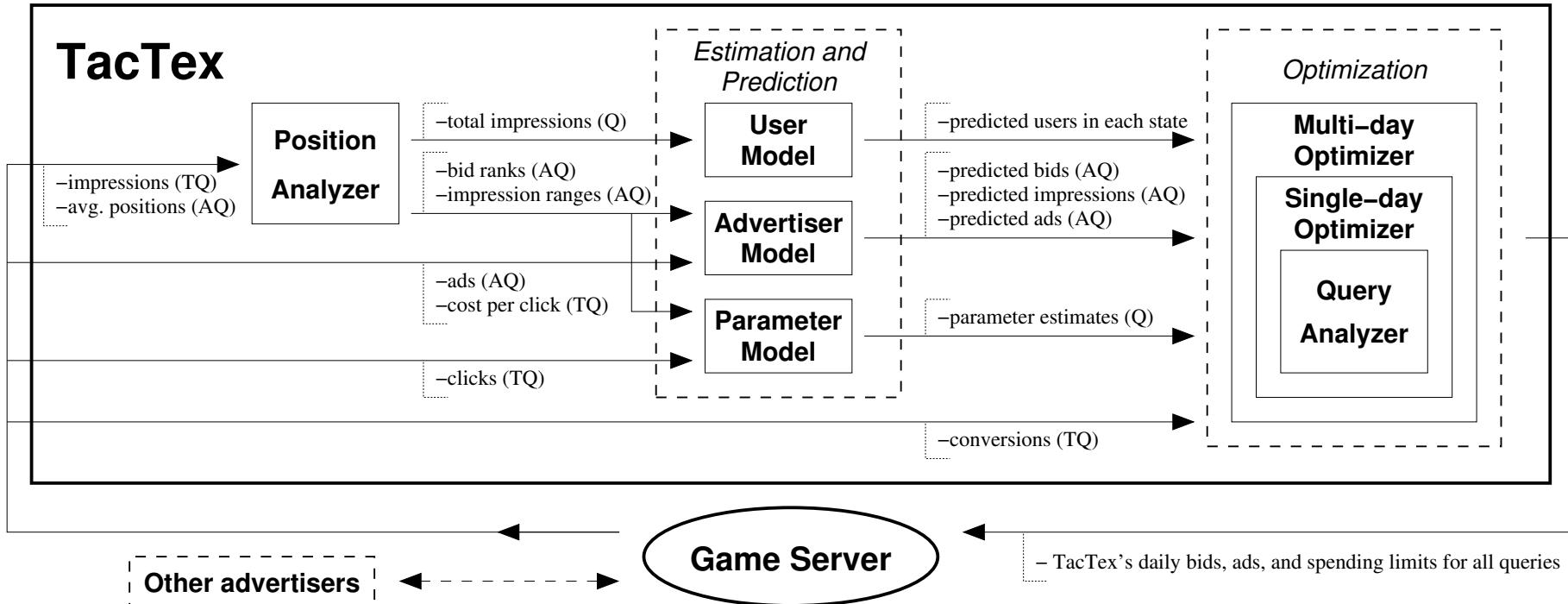


Figure 1: Flow of information in TacTex. T = TacTex only, A = all advertisers, Q = all queries.

General structures:

- use particle filters for estimation process
- optimization uses greedy search and hill climbing

Query Results

Advertiser	Agent actions				Results					Avg pos
	Bid	Sq. bid	Ad	Sp. limit	CPC	Imps	Clicks	Convs	Impression range	
MetroClick	0.315	0.109	generic	50.93	0.310	426	164	16		1.000
QuakTAC	0.266	0.107	lioneer:dvd	-	0.194	718	156	6		1.593
TacTex	0.235	0.091	generic	0.236	0.201	77	1	0		3.000
UMTac09	0.216	0.078	generic	7.583	0.209	700	36	6		2.719
munsey	0.190	0.075	generic	-	0.174	718	16	2		3.675
epflagent	0.214	0.068	generic	-	0.184	641	3	0		4.510
AstonTAC	0.158	0.059	generic	500.0	0.133	292	1	0		4.938
Schlemazl	0.062	0.020	flat:dvd	5.617	-	0	0	0		-

Table 1: Results for the query *null:dvd* from one game day of the 2009 TAC/AA finals

CAP6671: Intelligent Systems

Lecture 7: Robocup Challenge Problem

Instructor: Dr. Gita Sukthankar
Email: gitars@eeecs.ucf.edu

Assignment 1: Related Work

- Due Feb 27
- Write the introduction, related work, conclusion, and references sections of a research paper related to one of the agent competitions
- Paper should be 2-3 pages in double column AAAI conference format; latex formatting is preferred.
- An extension of a week is possible but please do not wait until the last minute to request one!
- Choose one of three topics:
 - Trading Agent Competition
 - Robocup
 - Starcraft (AIIDE or CIG) competition
 - Any variant of the competitions is possible
 - Paper should not cover multiple topics

Assignment

Writeup should include:

- Description of the competition
- What makes the problem difficult?
- What real-world problems will solving this problem facilitate?
- What is the related work on this problem?
 - Looking for at least 5 papers beyond the ones we read in class
 - Comparison of the different approaches used by the different authors
- Can include 1-2 small figures (should be cited if borrowed)
- Conclusion/future-work: what are unsolved problems, limitations with current approaches

Notes

- Related work description should go beyond a series of concatenations describing each paper
- Focus should be on the AI methods used by the authors and not just the heuristics
- Good: “Using the assumption that we can obtain all items at the final prices from the last competition, the system identifies the best package using a modified depth-first search”.
- Bad: “The authors bid for the cheapest hotel first, then bid for the cheapest flight, then cheapest ticket.”
- References should be complete and correct
- The newer competitions tend to have fewer references than the older ones so check to make sure enough references exist.

General Structure

- Introduction: stating the broad problem, convincing the readers that the problem is of interest
- Related Work: giving the background on the current work that does exist
- Conclusion: reminding the reader of the main points of the paper, hinting at future work to be done

After reading the Introduction/Related Work, the readers should be convinced that your problem is interesting, should be informed about the state of the art in that research area, and aware of any gaps in the current research.

Grade

- Based on presentation, clarity of writing style, research content, and quality of references cited.
- This is meant to be practice for writing your own research papers.
- Please do not plagiarize original reference papers (or move everything to within quotation marks).
- You need to understand the original paper well enough to summarize it in your own words.

ChatGPT

- If you want to use ChatGPT, you must do the following:
 - Mark the text generated by ChatGPT in a different color.
 - Submit a separate document describing
 - How you used ChatGPT
 - Describe the strengths/weaknesses
 - Include a reference list that you generate yourself for the competition.

ChatGPT Example

- Prompt: by year 2050 develop a team that can win against the human world soccer champion team
- ChatCPT: It's not possible to predict with certainty what the state of soccer or technology will be in the year 2050. However, the development of a team capable of defeating the human world champion team would require significant advancements in the field of robotics and artificial intelligence. The team would need to be equipped with advanced sensors, actuators, and decision-making algorithms, and be trained through extensive simulations and real-world scenarios. Additionally, the rules and regulations regarding the use of AI in sports may also play a role in determining the feasibility of such a project.

Other Considerations

- Unifying theme
- Same league or different leagues?
- Multiple papers from same group of authors
- Valid references: not web pages or technical documentation
- How to find papers:
 - Competition websites
 - Research group publication websites
 - Google scholar
 - Specialized distribution channels such as Robocup Springer series

Robocup

- Goal: by year 2050 develop a team that can win against the human world soccer champion team
- Research platform: various
- Interesting research findings:
 - New research: how to program intelligent systems with coordinated behaviors and adversarial modeling
 - Has been a useful testbed to demonstrate techniques in vision and robot locomotion
- Potential applications:
 - Building robots for rapid moving dynamic environments
 - Urban rescue robots (special league)
 - Household environments (special league)

Research Challenges?

Research Challenges

- Real-time, dynamic environment
- Adversarial reasoning
- Incomplete information
- Rapid processing of visual data
- Distributed control/teamwork
- Mapping and localization (Rescue)
- Legged locomotion (Aibo, Humanoid, Rescue)
- Human-robot interaction (Rescue, Robocup@Home)

Why Robocup?

- Set ambitious goal to drive innovation
- A platform for project-oriented education in science and technology
- Challenge problems have worked before in areas like computer chess
- Dream: to create a computer to beat the world chess champion
- Result: Deep Blue (defeated Kasparov in 1997)
- Produced: improvements in search algorithms, parallel computing, parallel machine architectures

Computer Chess vs Robocup

- Static
- Turn-taking
- Complete information
- Symbolic
- Central control
- Dynamic
- Real-time
- Incomplete info
- Non-symbolic
- Distributed control

2D Simulation League



- Simulated robots have identical physical capabilities (speed, stamina)
- Emphasis is on intelligent action selection, prediction, strategy, and coordination
- Simulation allows greater scope for techniques like machine learning that require lots of data

Simulation League

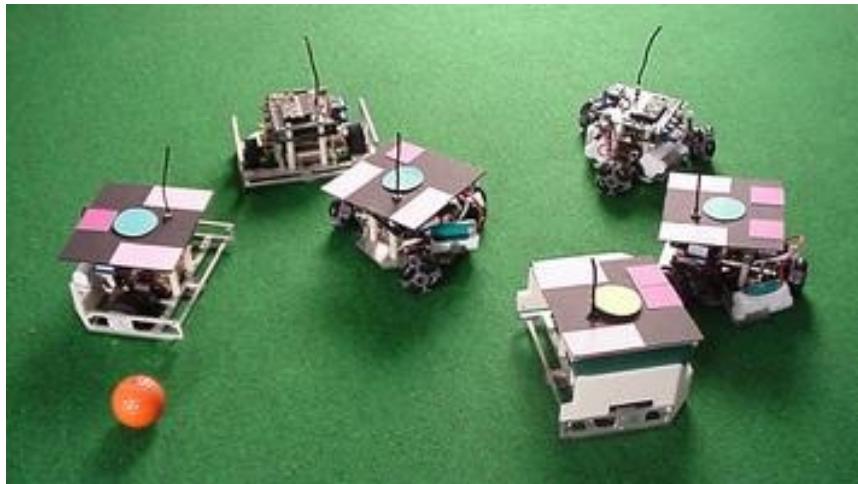
Robocup 2018 2D Final

<https://youtu.be/o-iYdQjXmjk>

Robocup 2018 3D Final

<https://youtu.be/7T1ElDs5eSQ>

Small Size League



- Robots usually view the playing field through overhead cameras
- Teams build their own robots
- Emphasis is on making the robots turn faster, move quicker, and kick powerfully
- To win you need good hardware design

Small Size League

- Robocup Small Size Finals
 - <https://youtu.be/sZI2DS-OK4s>

Mid Size League



- All sensors and computers are on board the robots
- Contestants often start from a commercial robot platform
- Unlike the small size league the robots don't have a centralized view of the playing field
- Camera position can be challenging

Mid Size League

- Robocup 2013 Mid Size Final Introduction
 - <http://www.youtube.com/watch?v=aO1rcoMEBsA>

Standard Platform League



- Aldebaran Nao robot used as the platform---no hardware modifications allowed.
- Camera motion is less smooth than on a wheeled robot.
- Robots need to move fast and kick without falling!
- Robots also need to be able to stand up as well.

Standard Platform League

- Robocup 2019 Standard Platform Finals
 - B-Human vs. HTWK
 - https://youtu.be/4_BWQI91p9Y
- Contrast this to the Humanoid league:
 - https://youtu.be/IwWt8CS_oBg

Industrial Leagues

These leagues focus on practical applications of robotics.

- Logistics
- @Work
- Rescue Robot
- Rescue Simulation
- @Home

Robocup Rescue



- Spinoff from Robocup
- Teams of robots compete to map an unknown urban environment and find “victims” buried in rubble
- Robots can either be autonomous or teleoperated
- Challenges: mapping, locomotion over rubble, good teleoperation interfaces

Robocup Rescue Competitor

- <http://www.youtube.com/watch?v=PKI378kadp8>

Robocup @Home

- Robots compete to complete domestic tasks.
 - <https://youtu.be/h3AIw1gM6rM>

Case Study: CMUnited-98

- Champion simulation team 20 years ago
- Code from every champion team is released as open source so many of the modern teams started from this code base
- Reference: Peter Stone, Manuela Veloso, and Patrick Riley. [The CMUnited 98 Champion Simulator Team](#). In M. Asada and H.Kitano, editors, RoboCup-98: Robot Soccer World Cup II Lecture Notes in Artificial Intelligence, pp. 61.76, Springer Verlag 1999
- Implementation details
 - Action loop
 - Updating world model/managing uncertainty
 - Low-level soccer skills
 - Coordination models
 - Potential field models

Agent Control Loop

- Assume robot has consistent information about state of world ($t-2$), sends action at $t-1$
- Upon receipt of sensory information from $t-1$, store new information in temporary buffer
- At time t , use state of the world, action, and sensory information to update the world model (for $t-1$)
- Choose action for time t
- Repeat

Note that the robot also looks at predicted effects of its own actions and not just the current sensed information

Updating World Model

- Global vs. local coordinate frame
- Objects are assigned confidence values
- Simulation domain is relatively easy compared to the leagues that use vision systems
- Vision systems need:
 - Rapid processing (usually color histogram based)
 - Robust ball tracking
 - To correctly orient the camera (for mid size/legged)

World Model

Object :

- Global (x, y) position coordinates
- Confidence within $[0,1]$ of the coordinates' accuracy

Stationary Object : nothing additional

Mobile Object :

- Global (dx, dy) velocity coordinates
- Confidence within $[0,1]$ of the coordinates' accuracy

Ball : nothing additional

Player :

- Team
- Uniform number
- Global θ facing angle
- Confidence within $[0,1]$ of the angle's accuracy

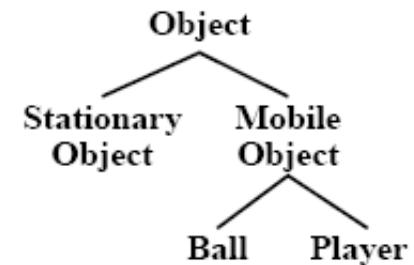
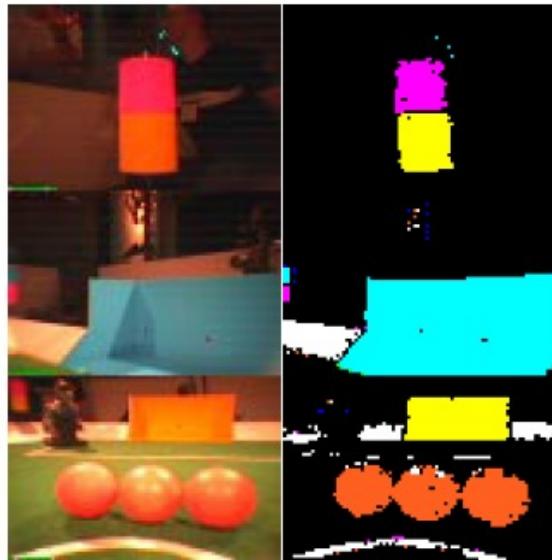


Image Segmentation



Focus is on rapid, primarily color-based segmentation techniques

Field is of a known dimension with set landmarks

Robots are allowed a vision calibration period

Robot's Perspective

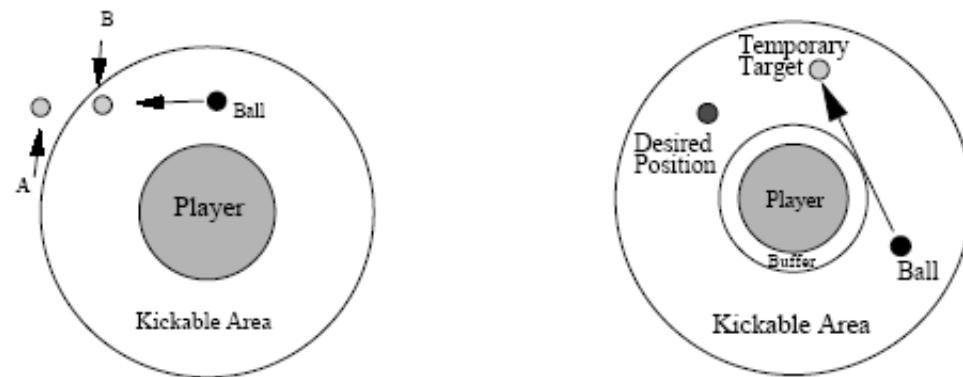
https://www.youtube.com/watch?v=Ka1_aHj8leg&feature=em-upload_owner

Low-level Skills

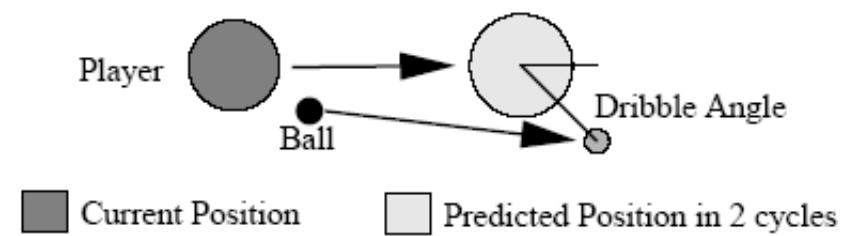
- Includes things: kicking, dribbling, interception, defending, clearing
- Principal challenge: not losing control of ball
- Note: simulator league models stamina to keep agents from running at full speed all the time
- PLOS model (Predictive Locally Optimal)
 - Use some form of prediction to decide on action
 - Execute first step of “plan”
 - Assume that you must replan at next time step

Kicking and Dribbling

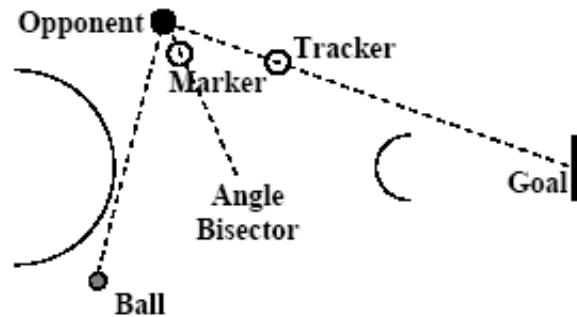
Kick and Turnball



Dribble



Adversarial Reasoning



Tracking: agent stays between opponent and goal to block shots on goal

Marking: agent stays on angle bisector line to block incoming passes to agent

Example Behavior Modes

- Goaltend: goalie only
- Localize
- Face Ball
- Handle Ball
- Active Offense: move toward ball
- Auxiliary Offense: get open for pass
- Passive Offense: move to a useful position
- Active Defense: go to ball
- Auxiliary Defense: mark opponent
- Passive Defense: move to a useful position

Locker-Room Agreement

- Designed for low-communication, time-critical adversarial environments
- Common problem: due to errors in the world model the robots misjudge the situation (e.g. 2 robots deciding that they are closest defender to the ball)
- Agents shift into different modes based on time, score, and game status which are guaranteed to be accurately known to all agents

Attraction/Repulsion Models

- Used to determine where the agent should be on the field
- Represent agent's policy as a potential field
- Objects in the environment exert attraction and repulsion forces
- Weighting forces creates different policies
- Drawback:
 - Potential minima possible (often solved by introducing randomness)

Summary

- Many interesting problems
- Different leagues highlight different research problems
- Every year new leagues are created and older leagues become more competitive.
- Criticism:
 - Robocup is a toy domain
 - Color-based landmarks
 - Not cluttered environment
 - Specific tricks (fast kick speeds) often can do very well.
 - Approaches are very “Robocup” specific.

CAP6671 Intelligent Systems

Lecture 8: Robocup Soccer Robocup: Urban Rescue Coordination and HRI

**Instructor: Dr. Gita Sukthankar
Email: gitars@eeecs.ucf.edu**

Robocup Humor

- Funniest Falls from Robocup

<https://youtu.be/1h5147KLikU>

Overview

- Finish discussion of Robocup league
 - Multi-robot coordination
 - Human-robot interaction

Robocup Research Questions

- Game understanding
 - Recognizing plays from motion trajectories
 - Robocup Coach league
 - Camera-based sports analysis/commentator
- Machine learning
 - Transferring learning from one task to another
 - Reinforcement learning/evolutionary methods for learning single agent/multi-agent policies
- Team coordination
 - Role allocation
- Human-robot interaction
 - Issue in Robocup Rescue and Robocup@Home

Single Bots: HMM Approach

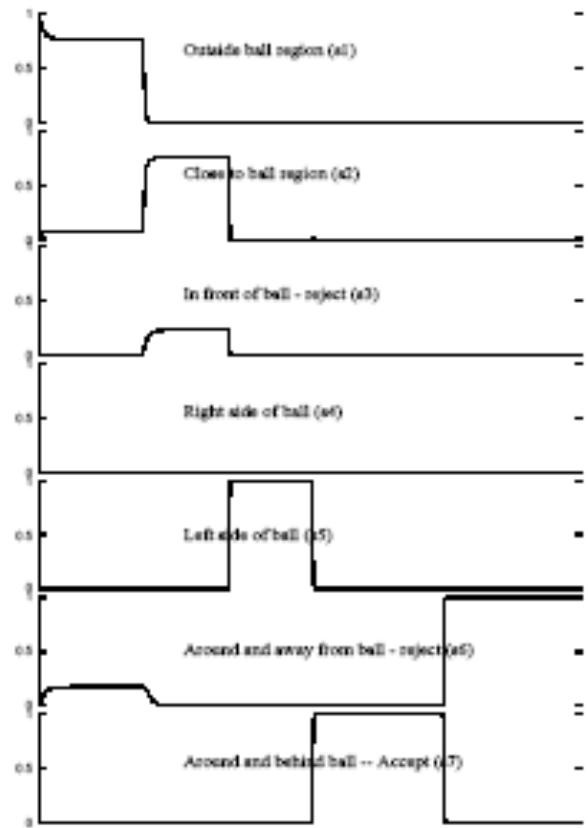


Figure 5: Go-Behind-Ball Behavior HMM probabilities

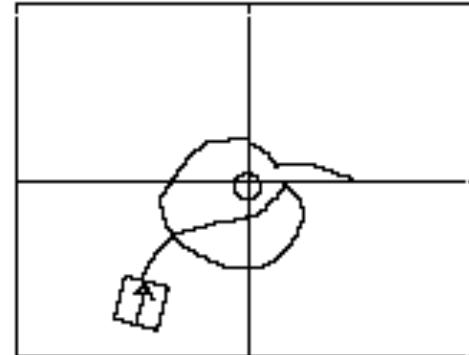
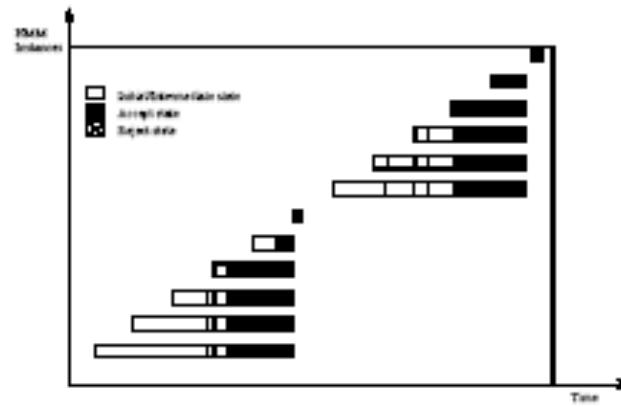


Figure 6: Robot Trajectory



(Han and Veloso, Automated Robot Behavior Recognition Applied to Robotic Soccer, 1999)

Coach League: Parametric Models

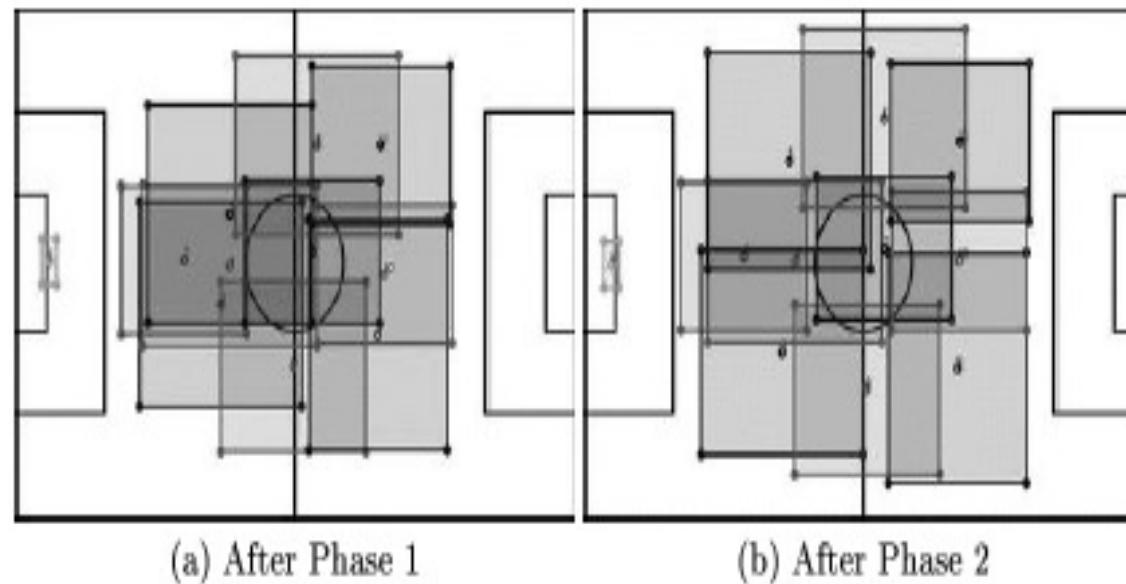


Fig. 1. The learning of the CMUnited99 formation from RoboCup2000 games.

(Riley, Veloso, Kaminka, An Empirical Study of Coaching, 2002)

(Kuhlmann, Knox, Stone, Know Thine Enemy: A Champion Robocup Coach Agent, 2006)

Detecting Team Failure with MBD

Model-based Diagnosis

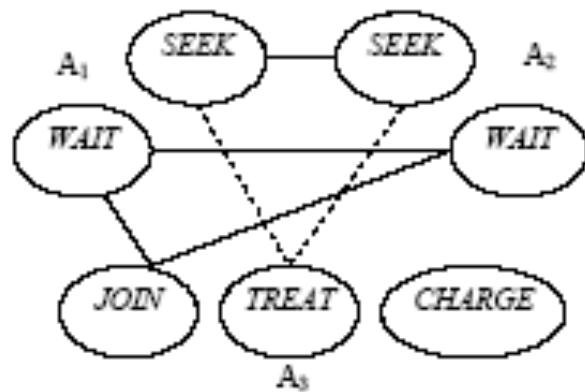


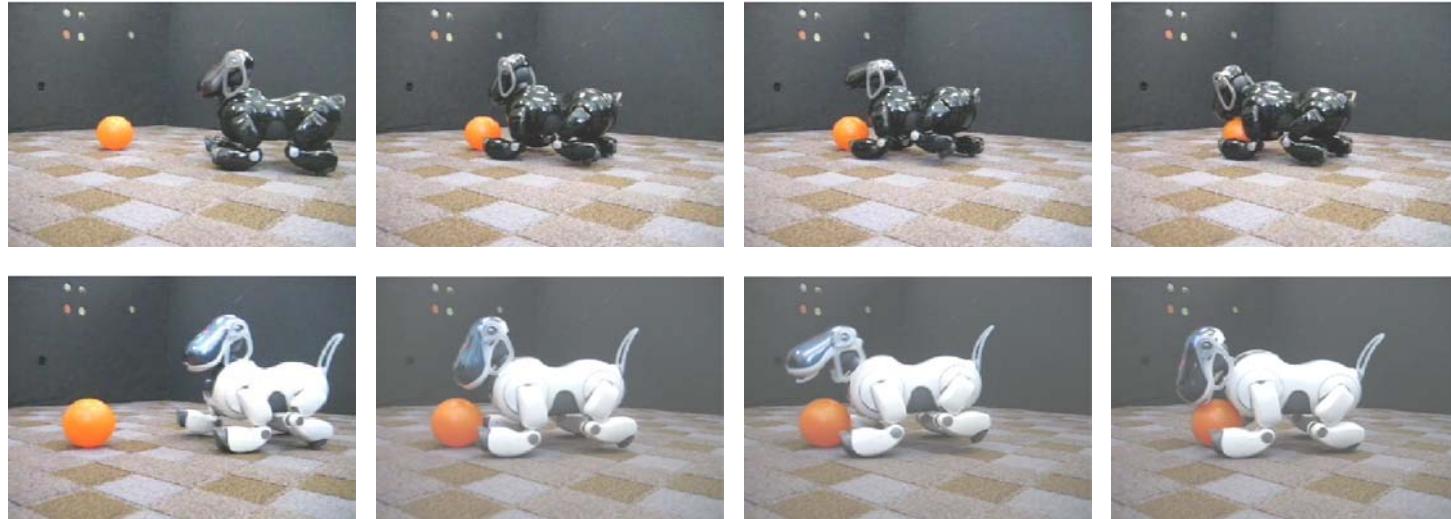
Figure 1: The coordination graph for team $\{A_1, A_2, A_3\}$.

(Kaminka and Tambe, Robust Agent Teams via Socially Attentive Monitoring, 2002)

(Kalech and Kaminka, Towards Model-Based Diagnosis of Coordination Failures, 2005)

Learning from Demonstration

Ball Grasping



(Grollman and Jenkins, Learning Robot Soccer from Demonstration:
Ball Grasping, 2007)

Camera-Based Systems



Figure 5: Estimating the camera parameters by taking the current parameters, projecting the field model using the current parameters onto the image (left), finding the correspondences between model points and the imagepoints (middle), and adjusting the camera parameters to achieve minimal errors (right).

(Beetz et al., Computerized real-time analysis of football (soccer) games, 2005)

Multi-robot Coordination

- How do you figure out what the individual robots should be doing (roles)?
- How do you figure out where the robots should go? (spatial position)

Role Allocation

<https://youtu.be/7mV4bCSxI3A?list=PLBADDCC903C448F39>

P. MacAlpine, F. Barrera, and P. Stone: *Positioning to Win: A Dynamic Role Assignment and Formation Positioning System*, Robocup-2012, Lecture Notes in AI, 2013

Compute the best allocation of team members to roles on the team out of $n!$ possible mappings

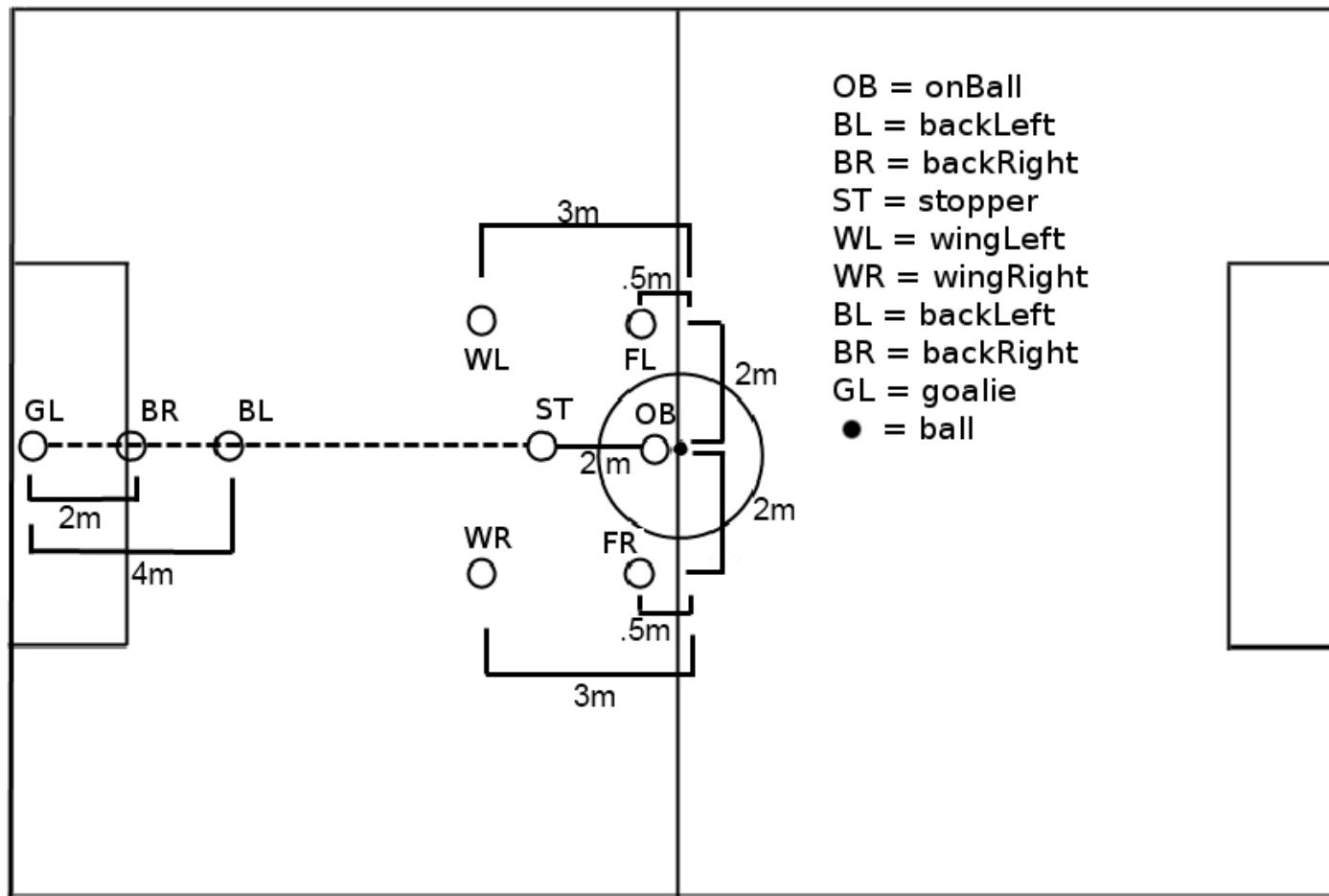
Good allocations:

1. Minimize the distance robots have to move
2. Do not change before the robots reaches the target position

2011 3D Simulation Domain

- 9 vs. 9 autonomous agents play soccer
- Realistic physics using Open Dynamics Engine
- Modeled after Aldebaran Nao robot
- Agents receive noisy visual information
- Communicate with each other over limited bandwidth

Formation Role Positions



Role Assignment Mapping

- One to one mapping of agents to positions
- Assumptions:
 - No two agents or roles occupy the same position
 - Agents move at a constant speed along a straight line
- Brute force mapping requires evaluating $n!$ mappings ($n=8, 40,320$)

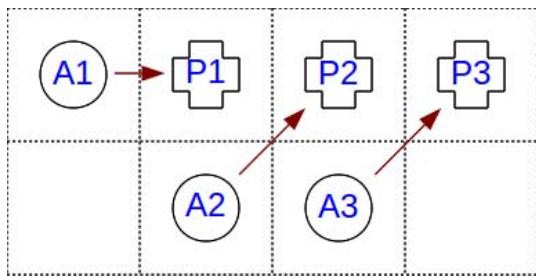
Recursive Property of Role Assignment

Theorem

Let A and P be sets of n agents and positions respectively. Denote the mapping $m := f_v(A, P)$. Let m_0 be a subset of m that maps a subset of agents $A_0 \subset A$ to a subset of positions $P_0 \subset P$. Then m_0 is also the mapping returned by $f_v(A_0, P_0)$.

- Any subset of a lowest cost mapping is itself a lowest cost mapping
- If within any subset of a mapping a lower cost mapping is found, then the cost of the complete mapping can be reduced by augmenting the complete mapping with that of the subset's lower cost mapping

Lexicographical Cost



- 1: $\sqrt{2}$ (A2→P2), $\sqrt{2}$ (A3→P3), 1 (A1→P1)
- 2: 2 (A1→P2), $\sqrt{2}$ (A3→P3), 1 (A2→P1)
- 3: $\sqrt{5}$ (A2→P3), 1 (A1→P1), 1 (A3→P2)
- 4: $\sqrt{5}$ (A2→P3), 2 (A1→P2), $\sqrt{2}$ (A3→P1)
- 5: 3 (A1→P3), 1 (A2→P1), 1 (A3→P2)
- 6: 3 (A1→P3), $\sqrt{2}$ (A2→P2), $\sqrt{2}$ (A3→P1)

Rather than summing the total cost, order the rankings like sorting words in an alphabet (each allocation has highest cost to lowest cost)

Like sorting a list of words:

Cat

Dog

Dot

Role Allocation

Aims of the process

- Minimize longest distance
- Avoid collisions
- Dynamically consistent (doesn't change)

$\{P_1\}$	$\{P_2, P_1\}$	$\{P_3, P_2, P_1\}$
$A_1 \rightarrow P_1$	$A_1 \rightarrow P_2, f_v(A_2 \rightarrow P_1)$	$A_1 \rightarrow P_3, f_v(\{A_2, A_3\} \rightarrow \{P_1, P_2\})$
$A_2 \rightarrow P_1$	$A_1 \rightarrow P_2, f_v(A_3 \rightarrow P_1)$	$A_2 \rightarrow P_3, f_v(\{A_1, A_3\} \rightarrow \{P_1, P_2\})$
$A_3 \rightarrow P_1$	$A_2 \rightarrow P_2, f_v(A_1 \rightarrow P_1)$ $A_2 \rightarrow P_2, f_v(A_3 \rightarrow P_1)$ $A_3 \rightarrow P_2, f_v(A_1 \rightarrow P_1)$ $A_3 \rightarrow P_2, f_v(A_2 \rightarrow P_1)$	$A_3 \rightarrow P_3, f_v(\{A_1, A_2\} \rightarrow \{P_1, P_2\})$

Dynamic Programming

$\{P_1\}$	$\{P_2, P_1\}$	$\{P_3, P_2, P_1\}$
$A_1 \rightarrow P_1$	$A_1 \rightarrow P_2, f_v(A_2 \rightarrow P_1)$	$A_1 \rightarrow P_3, f_v(\{A_2, A_3\} \rightarrow \{P_1, P_2\})$
$A_2 \rightarrow P_1$	$A_1 \rightarrow P_2, f_v(A_3 \rightarrow P_1)$	$A_2 \rightarrow P_3, f_v(\{A_1, A_3\} \rightarrow \{P_1, P_2\})$
$A_3 \rightarrow P_1$	$A_2 \rightarrow P_2, f_v(A_1 \rightarrow P_1)$ $A_2 \rightarrow P_2, f_v(A_3 \rightarrow P_1)$ $A_3 \rightarrow P_2, f_v(A_1 \rightarrow P_1)$ $A_3 \rightarrow P_2, f_v(A_2 \rightarrow P_1)$	$A_3 \rightarrow P_3, f_v(\{A_1, A_2\} \rightarrow \{P_1, P_2\})$

Begin evaluating mappings of 1 agent and build up to n agents

Only evaluate mappings built from subset mappings returned by f_v

Evaluates $n2^{n-1}$ mappings, for $n = 8$ is 1024 (brute force = 40,320)

Dynamic Programming

Algorithm 1 Dynamic programming implementation

```
1: HashMap bestRoleMap =  $\emptyset$ 
2: Agents =  $\{a_1, \dots, a_n\}$ 
3: Positions =  $\{p_1, \dots, p_n\}$ 
4: for  $k = 1$  to  $n$  do
5:   for each  $a$  in Agents do
6:      $S = \binom{n-1}{k-1}$  sets of  $k - 1$  agents from Agents –  $\{a\}$ 
7:     for each  $s$  in  $S$  do
8:       Mapping  $m_o = \text{bestRoleMap}[s]$ 
9:       Mapping  $m = (a \rightarrow p_k) \cup m_o$ 
10:       $\text{bestRoleMap}[\{a\} \cup s] = \text{mincost}(m, \text{bestRoleMap}[\{a\} \cup s])$ 
11: return bestRoleMap[Agents]
```

After each agent computes optimal allocation, communicate it, its position, and the ball position to other players for vote

Voting Coordination

- Each agent broadcasts ball position, own position, and suggested role mapping during allotted time slot
- Sliding window stored of mappings received over last n time slots evaluated and mapping with the most number of votes is chosen
- If two mappings both have greatest number of votes then tie breaker goes to mapping with most recent vote received
- Synchronization: With voting system = 100%, without = 36%

Benchmarks

- **AllBall** No formations and every agent except for the goalie goes to the ball.
- **NoCommunication** Agents do not communicate with each other.
- **Static** Role statically assigned to agents based on uniform number.
- **Offensive** Offensive formation in which all agents except for the goalie are positioned in a close symmetric formation behind the ball.
NearestStopper The *stopper* role position is mapped to nearest agent.
- **PathCost** Agents add in the cost of needing to walk around known obstacles (using collision avoidance), such as the ball and agent assuming the *onBall* role, when computing distances of agents to role positions.
- **PositiveCombo** *Offensive + PathCost + NearestStopper* agents

Results

	UTAustinVilla	Apollo3D	CIT3D
PositiveCombo	0.33 (.07)	2.16 (.11)	4.09 (.12)
Offensive	0.21 (.09)	1.80 (.12)	3.89 (.12)
AllBall	0.09 (.08)	1.69 (.13)	3.56 (.13)
PathCost	0.07 (.07)	1.27 (.11)	3.25 (.11)
NearestStopper	0.01 (.07)	1.26 (.11)	3.21 (.11)
UTAustinVilla	—	1.05 (.12)	3.10 (.12)
Defensive	-0.05 (.05)	0.42 (.10)	1.71 (.11)
Static	-0.19 (.07)	0.81 (.13)	2.87 (.11)
NoCollAvoid	-0.21 (.08)	0.82 (.12)	2.84 (.12)
NoCommunication	-0.30 (.06)	0.41 (.11)	1.94 (.10)
NoTeamwork	-1.10 (.11)	0.33 (.15)	2.43 (.12)
Boxes	-1.38 (.11)	-0.82 (.13)	1.52 (.11)

Play team vs. simpler versions of itself.

Dynamic assignment is definitely better than static assignment.

Summary

- Minimizing longest distance any agent travels is effective function
- Dynamic programming provides considerable increase in computational efficiency
- Dynamic roles with an aggressive formation does the best
- Communication and path planning are important
- Future Work
 - Learn better formation with machine learning
 - Explore other role assignment functions
 - Extensions for heterogeneous agents

Reactive Coordination

<https://youtu.be/YIWJbFjSOe8>

Juan Pablo Mendoza, Joydeep Biswas, Philip Cooksey, Richard Wang, Steven Klee, Danny Zhu and Manuela Veloso, Selectively Reactive Coordination for a Team of Robot Soccer Champions, AAAI 2016

CMDragons 2015 won all 6 games, scoring 48 goals and losing 0!

2 levels of planning:

Coordinated opponent-agnostic layer: compute its own plans

Individual opponent-reactive active selection layer: react to opponents

CMDragons



Action choices:

Action	Effect	Type
$\text{move}(p)$	Move to location p	Passive
getBall	Move to intercept ball	Active
shoot	Shoot ball to opponent's goal	Active
$\text{pass}(p)$	Pass ball to location p	Active
dribble	Dribble ball to hold possession	Active

2 offense roles: primary attacker and support attacker

Algorithm

Algorithm 1 Selective Reactive Coordination for Offense.

Input: State of the world \mathbf{x} .

Output: Individual robot actions.

```
function PlanAction( $\mathbf{x}$ )
    Instantiate roles  $r_i$  with zones  $\mathbf{z}_i$  (Section 3)
     $\{(\mathbf{z}_i, \mathbf{p}_i^0)\}_{i=1}^{n-1} \leftarrow \text{ComputeZones}(\mathbf{x})$ 
     $\{r_i\}_{i=1}^n \leftarrow [\text{SA}(\mathbf{z}_1, \mathbf{p}_1^0), \dots, \text{SA}(\mathbf{z}_{n-1}, \mathbf{p}_{n-1}^0), \text{PA}]$ 
    Optimally assign roles (Section 3)
     $\{(\rho_i, r_i)\}_{i=1}^n \leftarrow \text{OptAssign}(\{r_i\}_{i=1}^n, \mathbf{x})$ 
    Choose actions individually (Section 4)
    for  $i$  in  $[1, 2, \dots, n]$  do
         $a_i^* \leftarrow \text{IndividualAction}(\rho_i, r_i, \mathbf{x})$ 
    end for
end function
```

This layered joint-individual algorithm maintains tractability: ComputeZones is $O(n)$, OptAssign is $O(n^3)$, and IndividualAction is $O(n + m)$ for each robot, where m is the number of opponents. As the size of the team grows, the OptAssign step might need to be modified to maintain real-time planning.

Zones



(a) Coverage-zones for 3 SAs



(b) Dynamic-zone assignment, step i



(c) Dynamic-zone assignment, step $i + 1$

Probabilities

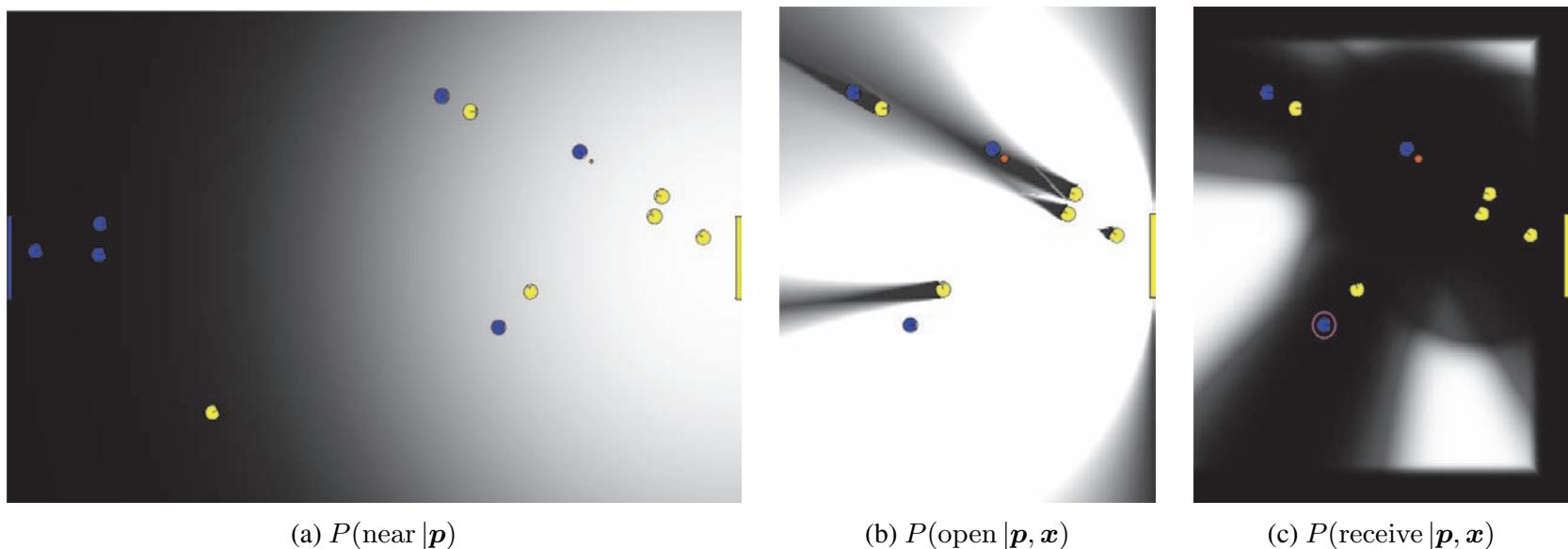


Figure 3: Estimated probabilities for individual decision-making of the blue robots. Lighter gray indicates higher probability points. Probability that \mathbf{p} (a) is near enough to the yellow goal and (b) has a wide enough angle on it, to shoot and score, and (c) Probability that the highlighted SA can receive a pass at different locations \mathbf{p} from the PA holding the orange ball.

Successful Pass



(a) Pass initial configuration



(b) Pass final configuration, immediately preceding a goal

Figure 5: Pass-ahead maneuver leading to a goal in RoboCup 2015. The figure shows the initial and final world configurations, and the motion of the ball and pass receiver.

Robocup Rescue: Research Issues

- Autonomy
- Coordinating multiple robots
- Mobility (climbing ramps, obstacles)
- Perception
 - Fusing multiple types of sensors
- Heterogeneous capabilities
- Human-robot interaction

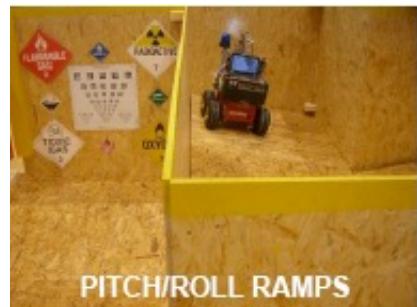
Competition Environment



Zones



Obstacles



PITCH/ROLL RAMPS



ORANGE STEPFIELDS

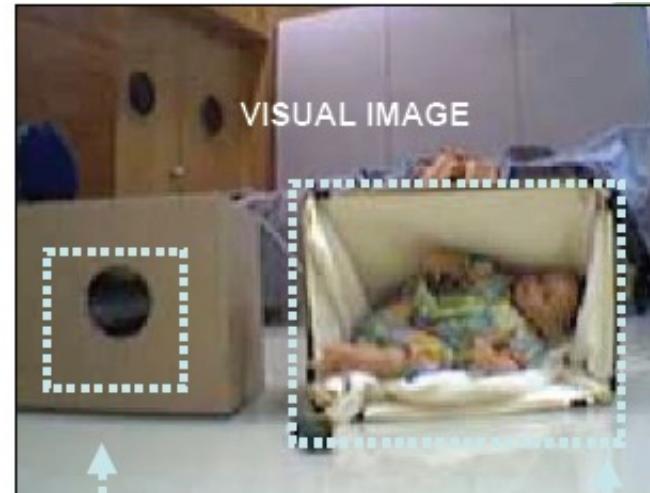


RED INCLINED PLANE

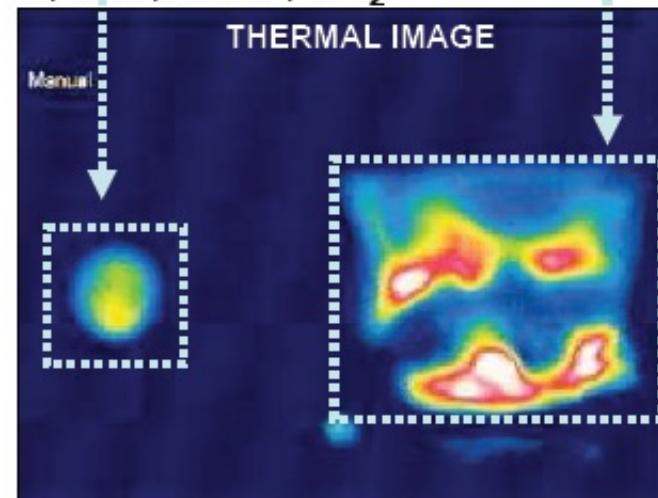


RED STEPFIELDS

Victims



Signs of life: form, motion, heat, sound, CO₂



Details

- 15/20/25 minute missions
- Teams are allowed one operator during the mission
- Start point is in the yellow arena with all robots facing same direction
- Yellow arena victims can be scored only by robots with autonomous navigation and victim ID
- Teleoperated robots can only score orange or red arena victims
- Robots can be fixed at start point with loss of victims, maps, and time
- Awards:
 - Autonomy: find most yellow arena victims
 - Mobility: find most red arena victims

Competition Directions

- Encourage autonomous behaviors for all robots
- Manipulation tasks (last victim in each arena)
 - Door opening (push/pull, assorted knobs)
 - Placing items (radio, water)
 - Picking items (sample, ID badge)
- Continue integration with Virtual Robot Competition
- Autonomous behaviors in complex environments
- A bracket of common robot platforms similar to legged league: Rhex, Volksbot, Kenaf
- Centralize repository of data, algorithms, etc.

Robocup Rescue 2013

<http://www.youtube.com/watch?v=IAAZwQVFYRk>

Coordination Mechanisms

- How to decide which robot does what....

Locker-room Agreement

- Robots agree on coordination triggers in advance
- When trigger occurs robots shift between pre-arranged plans
- Useful for situations in which you have periods of unlimited communication alternating with low communication
- Developed for Robocup (Stone, Veloso)

Dynamic Programming

- Discussed earlier!

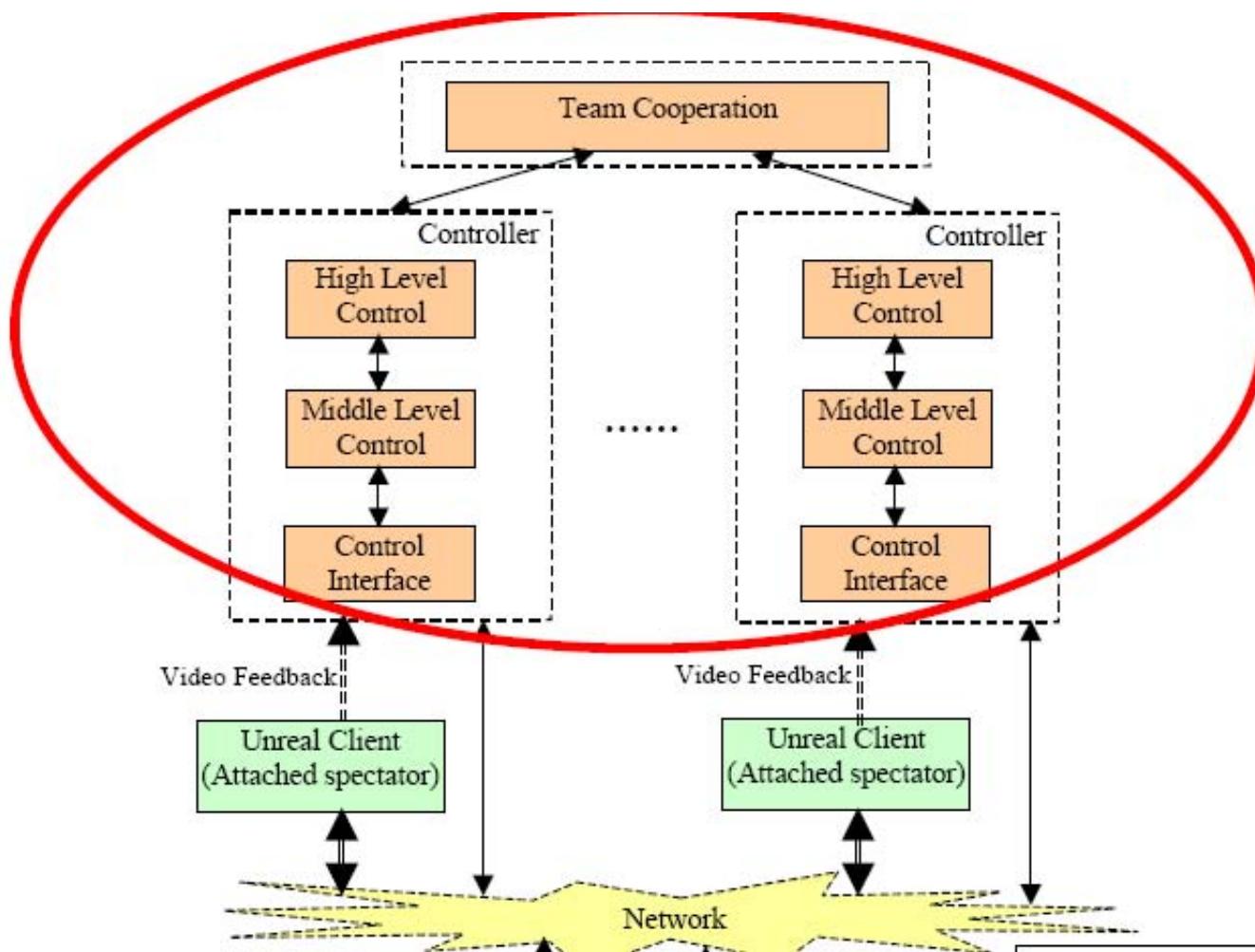
Team-Oriented Planning

- TEAMCORE (Tambe, USC)
- Extension of ideas from deliberative planning to multi-agent planning
- Robots reason about their actions based on a formal model of task completion requirements
- Draw upon models of human teamwork to decide when to communicate with other robots
- Different from locker-room coordination because robots use the planner to decide on actions
- Plans are constructed with roles for different robots
- Allocation of robot to role can change

Market-based Approaches

- Formulate the robot task allocation problem as a group auction
- Robots bid their cost for completing a task
- Central auctioneer injects tasks into the market
- Robots can re-auction tasks to other robots
- Market can be made increasingly robust through:
 - Distributed auctions
 - Monitoring awarded tasks and reauctioning them if not completed within a certain period of time

Example Controller



HCI

- Human-computer interaction (HCI)
 - HCI: How people interact with computers
 - Many sub-areas
 - Ergonomics
 - Human Factors
 - Usability
 - Designers of computer interfaces have to have a model of what the user wants to do and their preferences/expectations in how to do it

HRI: Human-robot interaction

- Human-robot *interaction* (HRI)
 - How humans and robots interact with each other
 - The space in which the agent system works including the task, agents and skills, environment and conditions, social informatics, and communication.
 - Bi-directional HCI plus industrial organization theory
- *Communication*
 - How do they communicate (verbal, signals, etc.)? What do they say to each other? When?
 - Same communication patterns can be used to communicate to other robots as well
 - Related to research on multi-robot coordination

Methodologies in HRI

Like in Psychology, it is difficult to agree on a methodology on how to carry out research.

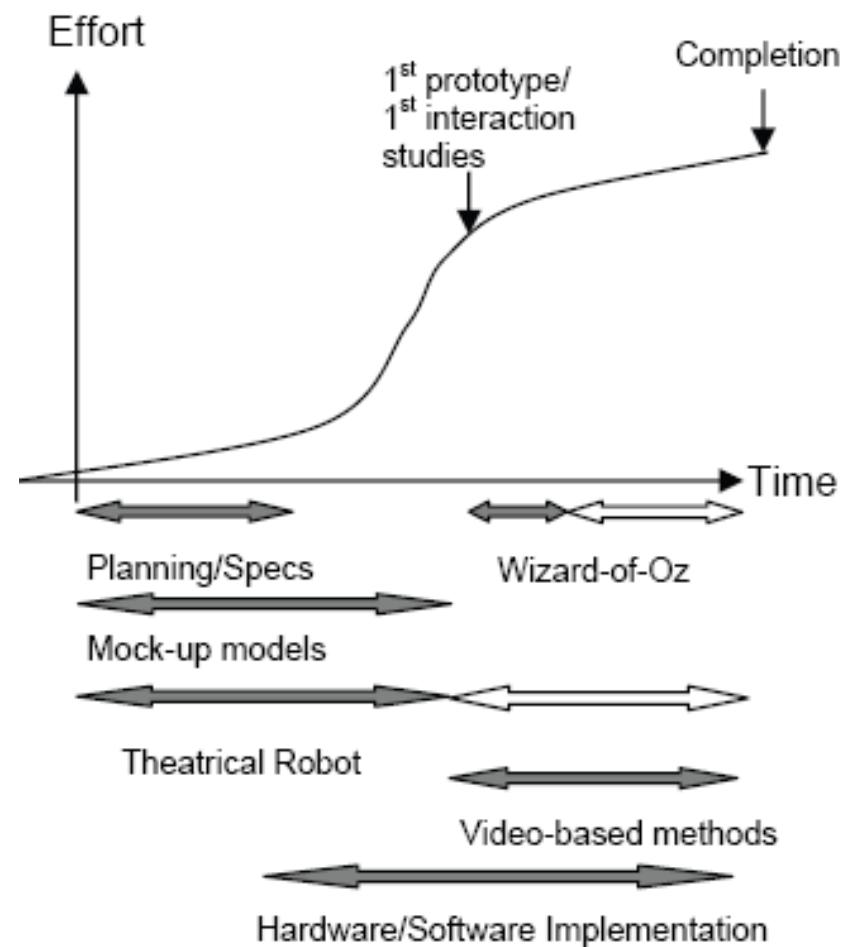
Should one

- apply quantitative, statistical methods requiring large-scale experiments, i.e. involving large sample sizes of participants,
- or pursue a qualitative approach focusing on case studies and in-depth analysis.

Methodologies in HRI

Sketch of a typical development time line of HRI robots.

- In an initial phase of planning and specification, mock-up models might be used before hardware and software development commences.
- Once a system's main components have been implemented, a WoZ study is applicable, and/or video based methods can be applied.
- As soon as working prototypes exist that also conform with safety requirements, interaction studies can be conducted.



Experimental Design

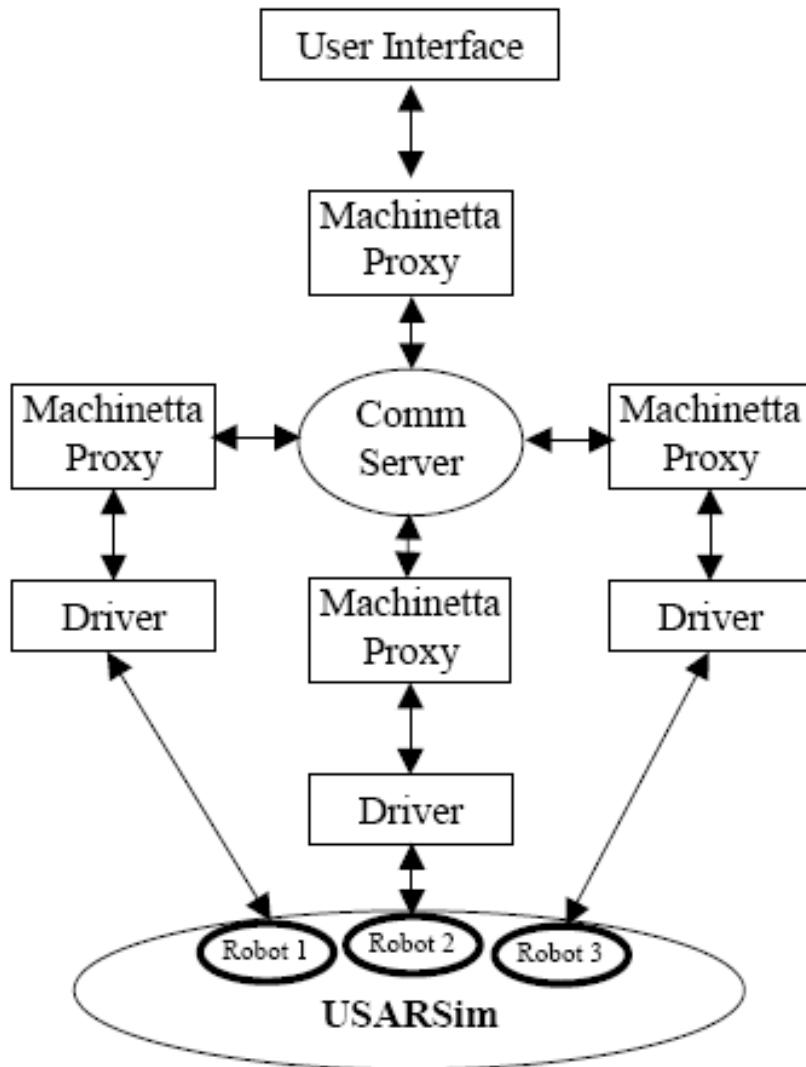
- What is the general premise that you are trying to prove?
- What are the metrics that can be measured to prove your premise?
- What statistical tests are appropriate to your study?
- What is the magnitude of the effect that you are looking for?
- Seek IRB approval for your study

Now onto this paper....

- What were the authors trying to prove?

J.,Wang, M. Lewis, and P. Scerri. Cooperating robots for search and rescue, Proceedings of the AAMAS 1st International Workshop on Agent Technology for Disaster Management, 2006, 92-99.

System Architecture



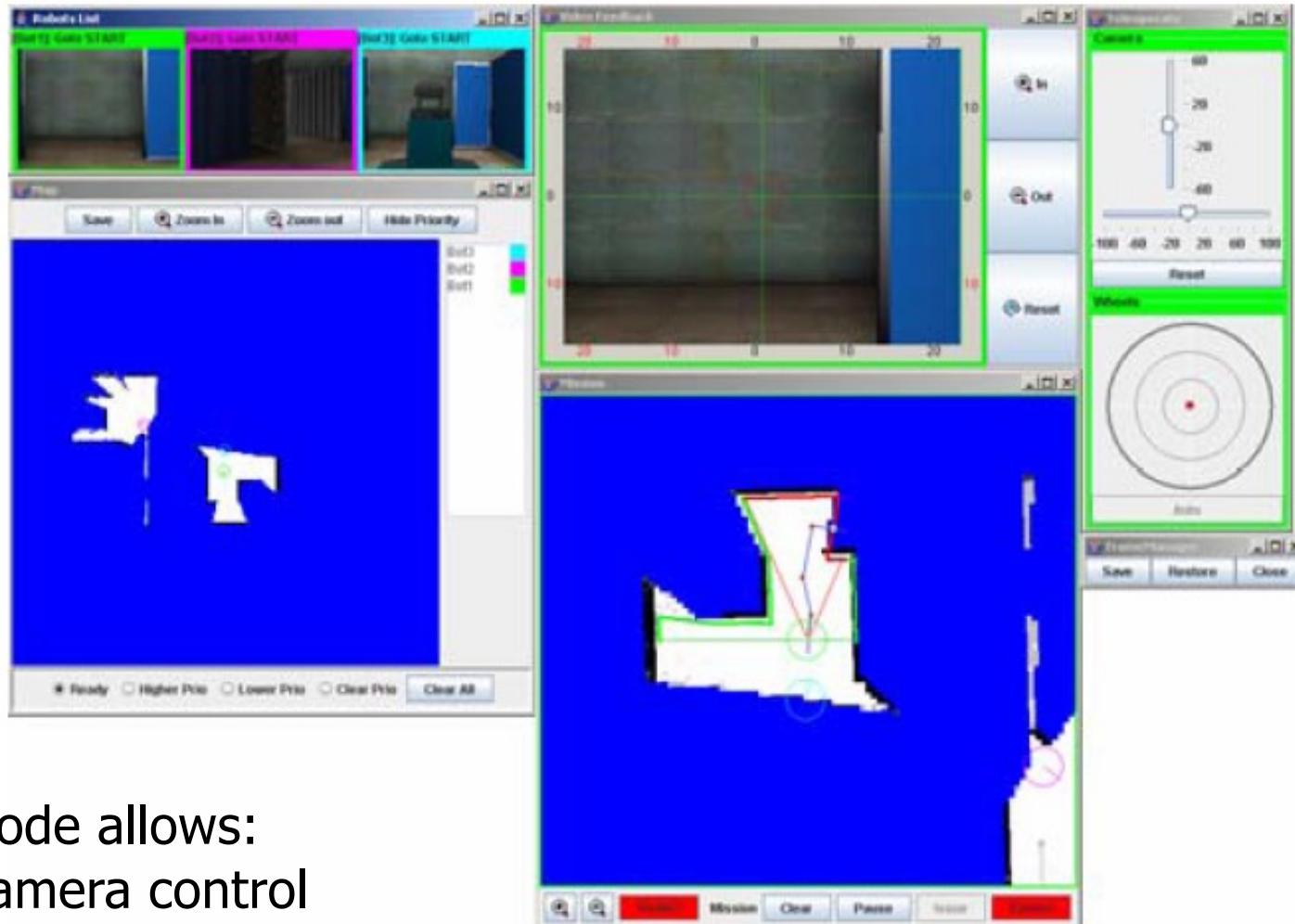
Machinetta provides:

- Coordination
- Adjustable autonomy
- Information fusing
- Role allocation
- Task deconfliction

Team-oriented plans
describe plans in terms
of:

- Roles
- Constraints between roles

User Interface



Mode allows:
Camera control
Waypoint selection
Teleoperation

Results

- **Evaluated:**
 - Subjects experience (via questionnaire)
 - Increases in number of victims found
 - Area explored
 - Distribution of attention
- **Results:**
 - Subject performance improves under increased autonomy
 - However improvements could be attributed to the path planning and autonomy mechanisms rather than the improved coordination.

Summary

- Many interesting problems
- Different leagues highlight different research problems
- Every year new leagues are created and older leagues become more competitive.
- Criticism:
 - Robocup is a toy domain
 - Color-based landmarks
 - Not cluttered environment
 - Specific tricks (fast kick speeds) often can do very well.
 - Approaches are very “Robocup” specific.

CAP6671 Intelligent Systems

**Lecture 9:
Starcraft CIG/AIIDE Competition;**

**Instructor: Dr. Gita Sukthankar
Email: gitars@eeecs.ucf.edu**

Reading

- Michal Certicky , David Churchill, Kyung-Joong Kim , Martin Certicky , and Richard Kelly, StarCraft AI Competitions, Bots, and Tournament Manager Software, IEEE Transactions on Games 2019
- D. Churchill and M. Buro, Incorporating Search Algorithms into RTS Game Agents, AIIDE Workshop on AI in RTSs, 2012
- Sparcraft tutorial: <https://code.google.com/p/sparcraft/>

AI in the News

DeepMind's StarCraft 2 AI is now better than 99.8 percent of all human players

16 ▾

AlphaStar is now grandmaster level in the real-time strategy game

By Nick Statt | @nickstatt | Oct 30, 2019, 2:00pm EDT

[https://www.theverge.com/2019/10/30/20939147/deepmin
d-google-alphastar-starcraft-2-research-grandmaster-level](https://www.theverge.com/2019/10/30/20939147/deepmind-google-alphastar-starcraft-2-research-grandmaster-level)

Real-time Strategy Games

- Complex strategic elements plus real time action
- Combine economic and military control with decision-making
- Compared to chess (10^{50}) or Go (10^{170})
 - Real-time, simultaneous moves
 - Playing field 256 times as large
 - Several hundred pieces per player; pieces created and destroyed
 - Games more difficult to simulate
 - Fog of war prevents players from having perfect information
- Has some similarities to military planning and logistics in the real world

Starcraft: BroodWar



- Three interstellar species: Terran, Protoss, and Zerg
- Different races have different abilities and troop types
- Players conquer territory, gather resources, build bases, and upgrade their militaries to defeat opponents

Research Challenges

.

Research Challenges

- Resource management
- Decision making under uncertainty
- Spatial/temporal reasoning
- Collaboration
- Opponent modeling and learning
- Adversarial real-time planning

Overview

AIIDE 2010 Competition

- <https://youtu.be/4V5aYSILkmA>
- Autonomous bots playing each other in a one vs. one game
- Round-robin format: bots play each 20 times on each of 10 maps
- Can store information from previously completed rounds
- Man-machine competition
- Code from previous year's winning bots are released

Real-time Strategy Games

- Strategy
- Production/Economy
- Recon/Terrain Analysis
- Tactics
- Reactive Control/Path Finding
- Both production and military strategy need to be integrated.
 - Rushing: build cheap troops quickly and surprise enemy with a quick attack
 - Turtling: built strong fortifications, attack late
 - Steamrolling: make expansionist outposts to create booming economy in order to purchase better tech

Implementation

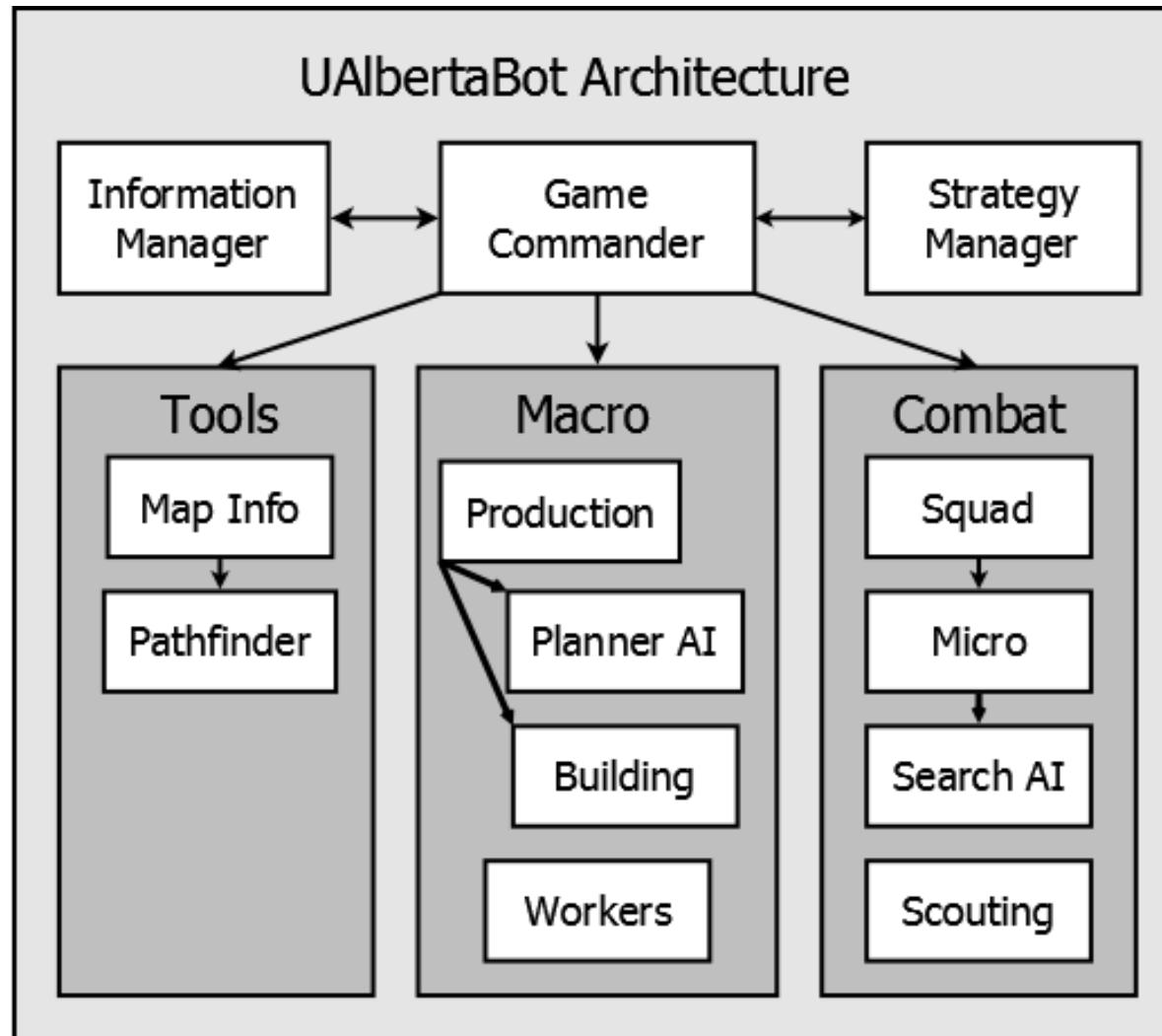
BroodWar API: <http://code.google.com/p/bwapi/>

- Perform unit actions: attack, move, build
- Obtain data about visible units: position, HP, mana, isIdle
- Obtain data about unit types: MaxSpeed, Damage, MaxHP, Size, isFlyer
- In competition, bots were given 55 ms per frame to perform calculations
- Starcraft will cancel an attack command if another command is given before it is completed so bots can't just keep pushing commands without waiting for outcomes

UAlbertaBot

- AIIDE2013 champion
- Modular and hierarchical architecture
- Original bot used scripts to do everything and then as years have passed authors replaced scripts with more sophisticated systems
- Focus on optimal build orders
 - Ordered sequence of actions that produces a goal set of build units

UAlbertaBot



Action Loop

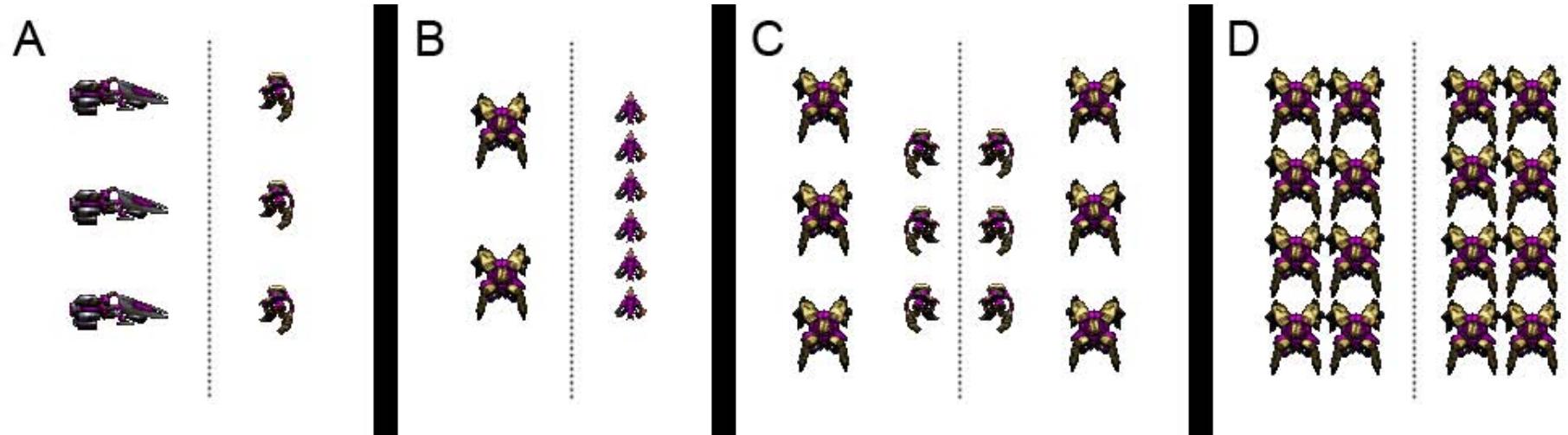
Algorithm 1 Game Commander onFrame

```
1: procedure ONFRAME ()
2:   Clock.start();
3:   Tools.update();
4:   ... Tools.Map.update();
5:   ... Tools.Map.Pathfinder.update();
6:   Macro.update();
7:   ... Macro.WorkerManager.update();
8:   ... Macro.ProductionManager.update();
9:   Combat.update();
10:  ... Combat.ScoutManager.update();
11:  ... Combat.SquadManager.update();
12:  ... Combat.MicroManager.update();
13:  BuildPlannerAI.run(TimeLimit-Clock.elapsed());
14:  MicroAISystem.run(TimeLimit-Clock.elapsed());
```

Time-Constrained Search

- Effective search can enable developers to produce bots that don't have to rely on hard coded expert knowledge
- However, exponential in depth of tree.
- Bots have 55 ms to perform calculations before the next frame
- Managing time:
 - Resources given to the AI system may have to be reduced to produce an answer.
 - Computations may have to be staggered between frames or performed in a separate thread.
 - Estimating future states may be helpful for the search process

Experiments: Micro Search



Four different scenarios with different mixes of melee and ranged units

Search: search-based allocation strategy

AttackWeakest: attack lowest health

Kiting: keep using mobility to move and shoot from outside opponent's range

Evaluation

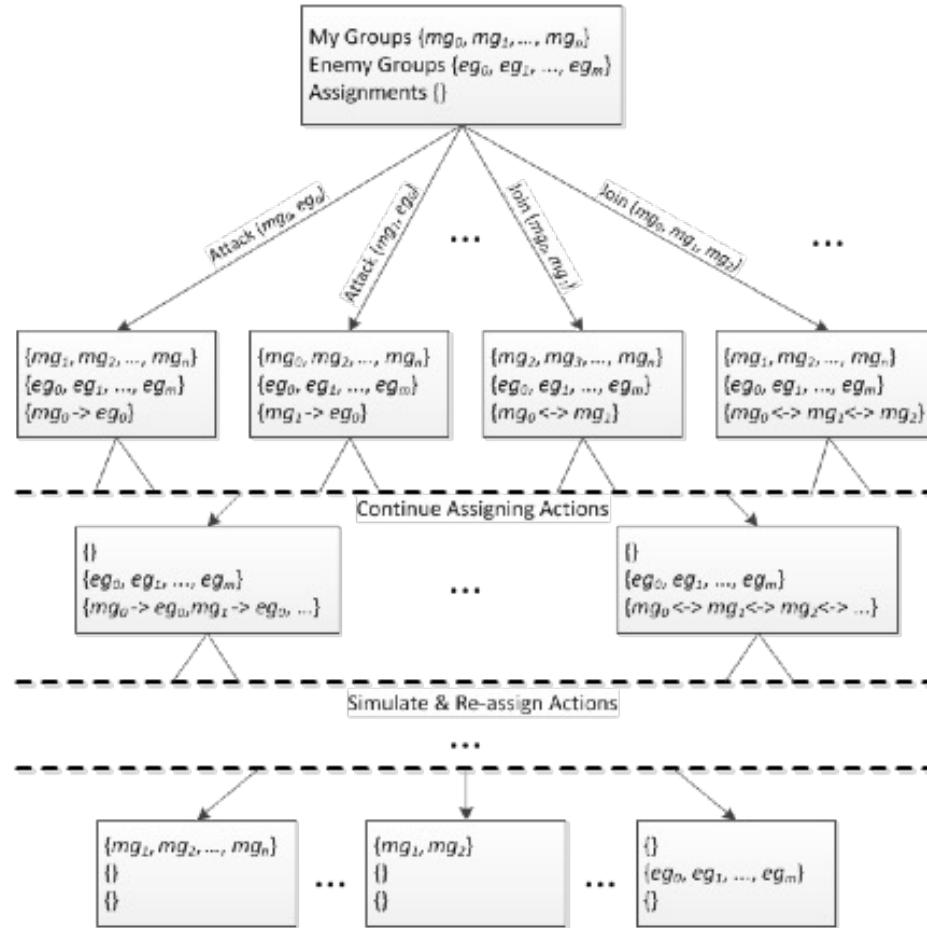
Combat Decision Settings						
	Search (5 ms)		AtkWeakest		Kiter	
	Sim	Game	Sim	Game	Sim	Game
A	1.00	0.81	0	0	1.00	0.99
A'	1.00	0.78	0	0	1.00	0.99
B	1.00	0.65	0	0	1.00	0.94
B'	1.00	0.68	0	0	1.00	0.89
C	1.00	0.95	0.50	0.56	0	0.14
C'	1.00	0.94	0.50	0.61	0	0.09
D	1.00	0.96	0.50	0.58	0	0.11
D'	1.00	0.97	0.50	0.55	0	0.08
Avg	1.00	0.84	0.25	0.29	0.50	0.53

Search wins more consistently than the hand-coded strategies.

Adjutant Bot

- Best newcomer in CIG 2012 competition, placing 4th
- Final project for this class
 - <https://youtu.be/PfeB--1qaww>

UCT for Micromanagement

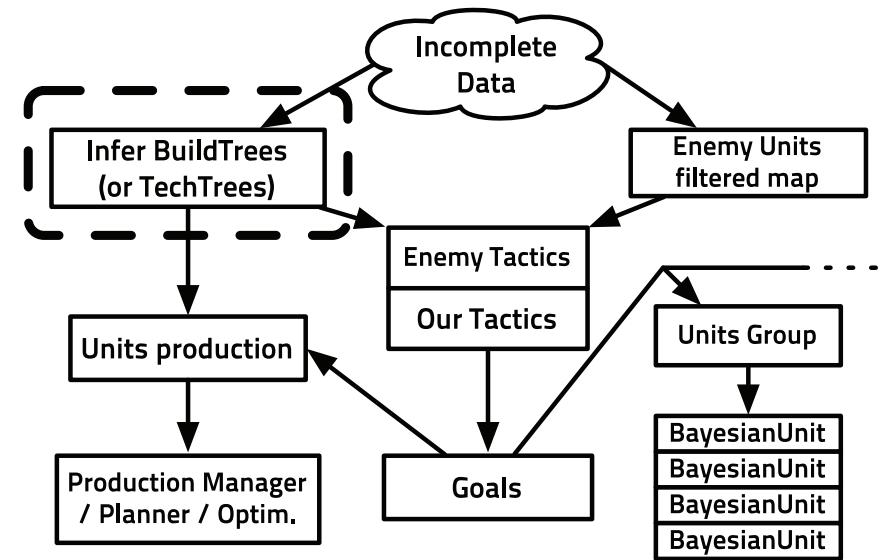


Described in "Adjutant Bot: An Evaluation of Unit Micromanagement Tactics". Nicholas Bowen, Jonathan Todd, and Gita Sukthankar. IEEE Conference on Computational Intelligence in Games (Competition). 2013.

Research Areas

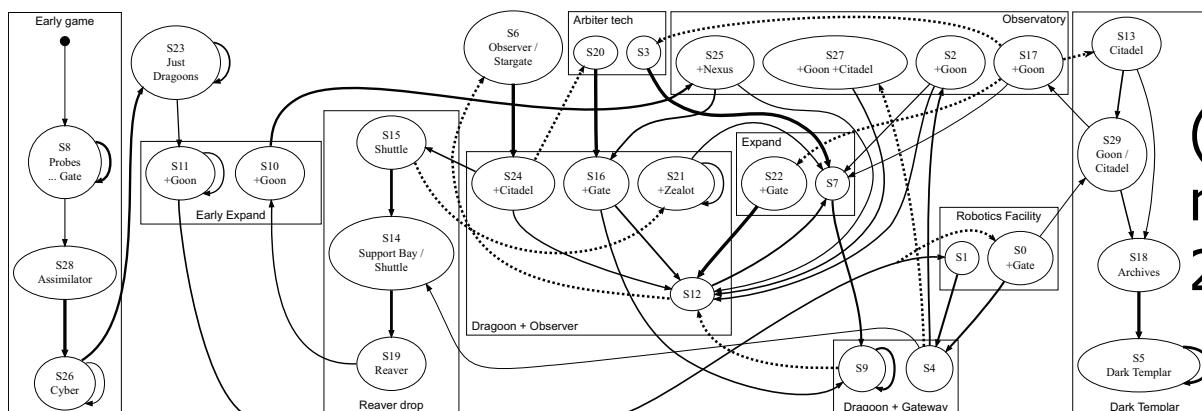
- Build order optimization
 - Identify the fastest production/resource gathering strategy for building units (branch and bound search)
- Tactics:
 - Micro-management (UCT)
 - Kiting: using mobility to avoid troop loss
- Opponent modeling and recognizing games from log trace
 - Identify if players are trying to rush you
 - Predict their build orders

Log Analysis and Action Prediction



(Weber, Mateas, and Jhala, AIIDE 2011)

(BroodWarBotQ
Synnaeve and Bessiere, 2011)



(Fern, Dietterich, and many others, AIIDE 2011)

Sparcraft

- Simplified Starcraft combat simulation
- Useful for both:
 - Calculating the reward when using Monte Carlo Tree search algorithms (e.g. UCT) within the Starcraft competition
 - Doing a simplified project on Starcraft bots
 - <https://code.google.com/p/sparcraft/>
- Recommended platform for final project

Bot Summary 2015-2017

Bot	Created	Rules	ML	HS	IO	SIM
CherryPi	2017	Yes	No	No	Yes	Yes
cpac	2017	Yes	No	Yes	No	Yes
ForceBot	2017	Yes	No	No	No	No
Iron	2016	Yes	No	No	No	No
KillAll	2017	Yes	Yes	Yes	No	Yes
Krasi0bot	2010	Yes	Yes	Yes	Yes	Yes
LetaBot	2014	Yes	No	Yes	Yes	Yes
McRave	2017	Yes	No	No	No	Yes
MegaBot	2016	Yes	No	No	Yes	Yes
PurpleWave	2017	Yes	No	Yes	Yes	Yes
StarcraftGP	2015	Yes	Yes	No	No	No
Steamhammer	2016	Yes	No	Yes	Yes	Yes
tscmoo	2015	Yes	Yes	Yes	Yes	Yes
UAlbertaBot	2010	Yes	No	Yes	Yes	Yes
ZZZKbot	2015	Yes	Yes	No	Yes	No

Shown are bots' creation year, and whether the bot uses any: "Rules" = rule-based systems, "ML" = machine learning, "HS" = heuristic search, "IO" = file I/O for learning during competitions, and "SIM" = simulations.

Where are Starcraft bots weak?

- Detecting that they are being exploited
 - Humans are usually able to exploit weaknesses in bot play during man-machine matches
- Opponent modeling
- Strategy switching
- Base construction
 - Human players create intricate maze structures to defend their production facilities while they construct units.
 - Bots are also weak at attacking these type of structures

Deepmind AlphaStar

- <https://youtu.be/jtlrWbI0yP4>

Next Time

Reinforcement learning---but on more tractable problems

Starcraft II Learning Environment

<https://youtu.be/6L448yg0Sm0>

Conclusion

- Real-time strategy games are helping drive the start of the art in real-time search and opponent modeling.
- Learning was a factor in 2013 competition, allowing the best bots to improve their win rates by 10% during the course of the competition.
- 2015-2017: about half the bots using it
- In 2022, learning is essential and the best systems (done by Deepmind) rely on a lot of self-play.

CAP6671 Intelligent Systems

Lecture 10: Reinforcement Learning; Multi-agent RL

**Instructor: Dr. Gita Sukthankar
Email: gitars@eeecs.ucf.edu**

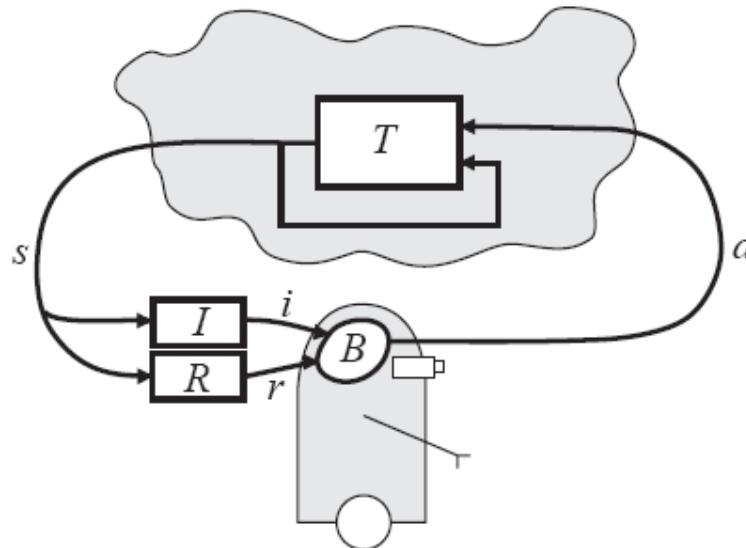
Reading

- Recommended background reading:
 - Reinforcement Learning (Sutton and Barto)
 - Chapters 1, 4, and 6
 - Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. [Reinforcement learning: a survey](#).
- Peter Stone and Manuela Veloso. [Team-Partitioned, Opaque-Transition Reinforcement Learning](#) (TPOT-RL)

A Humorous Introduction

<https://www.youtube.com/watch?v=99yzUs57m2c&feature=youtu.be>

Reinforcement Learning



Environment: You are in state 65. You have 4 possible actions.

Agent: I'll take action 2.

Environment: You received a reinforcement of 7 units. You are now in state 15. You have 2 possible actions.

Agent: I'll take action 1.

Environment: You received a reinforcement of -4 units. You are now in state 65. You have 4 possible actions.

Agent: I'll take action 2.

Environment: You received a reinforcement of 5 units. You are now in state 44. You have 5 possible actions.

:

:

Reinforcement Learning

- Methods that allow an agent to learn good sequences of actions (policies) by exploring the environment and receiving reward signals
- No labeled training data needed
- What kind of problems can we solve?

Problems

- Robot control
- Mazes
- Games
- Many of the problems you can use evolutionary computing techniques you can also use reinforcement learning
- Example competitions:
 - Acrobat, Adversarial Tetris, Helicopter, Infinite Mario, Octopus Arm, Polyathon

Markov Decision Processes

- Agents actions can affect the state of the world
- Most popular model is the Markov Decision Process
- An MDP has four components, S , A , R , Pr :
 - (finite) state set S ($|S| = n$)
 - (finite) action set A ($|A| = m$)
 - transition function $\text{Pr}(s,a,t)$
 - each $\text{Pr}(s,a,-)$ is a distribution over S
 - represented by set of $n \times n$ stochastic matrices
 - bounded, real-valued reward function $R(s)$
 - represented by an n -vector
 - can be generalized to include action costs: $R(s,a)$
 - can be stochastic (but replacable by expectation)
- Model easily generalizable to countable or continuous state and action spaces

Assumptions

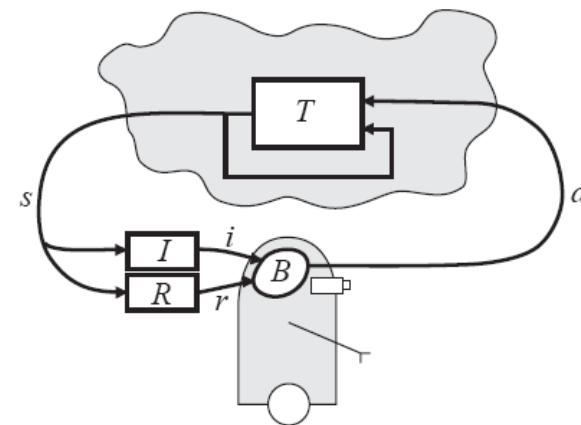
- Markovian dynamics (history independence)
 - $\Pr(S^{t+1}|A^t, S^t, A^{t-1}, S^{t-1}, \dots, S^0) = \Pr(S^{t+1}|A^t, S^t)$
- Markovian reward process
 - $\Pr(R^t|A^t, S^t, A^{t-1}, S^{t-1}, \dots, S^0) = \Pr(R^t|A^t, S^t)$
- Stationary dynamics and reward
 - $\Pr(S^{t+1}|A^t, S^t) = \Pr(S^{t'+1}|A^{t'}, S^{t'})$ for all t, t'
- **Full observability**
 - though we can't predict what state we will reach when we execute an action, once it is realized, we know what it is

Issues

- Does the agent know the world dynamics or not?
- How does the agent decide on an action with only incomplete information?
- How to estimate the value of the state?
- Create a model of world dynamics or just learn a model of values and actions?

Known World Dynamics

- Let's say that the agent knows:
- T (transition matrix)
- R (rewards per state)
- How to calculate the best policy?



Value Iteration (Bellman 1957)

- Markov property allows exploitation of DP principle for optimal policy construction
 - no need to enumerate $|A|^T$ possible policies
- Value Iteration

$$V^0(s) = R(s), \quad \forall s$$

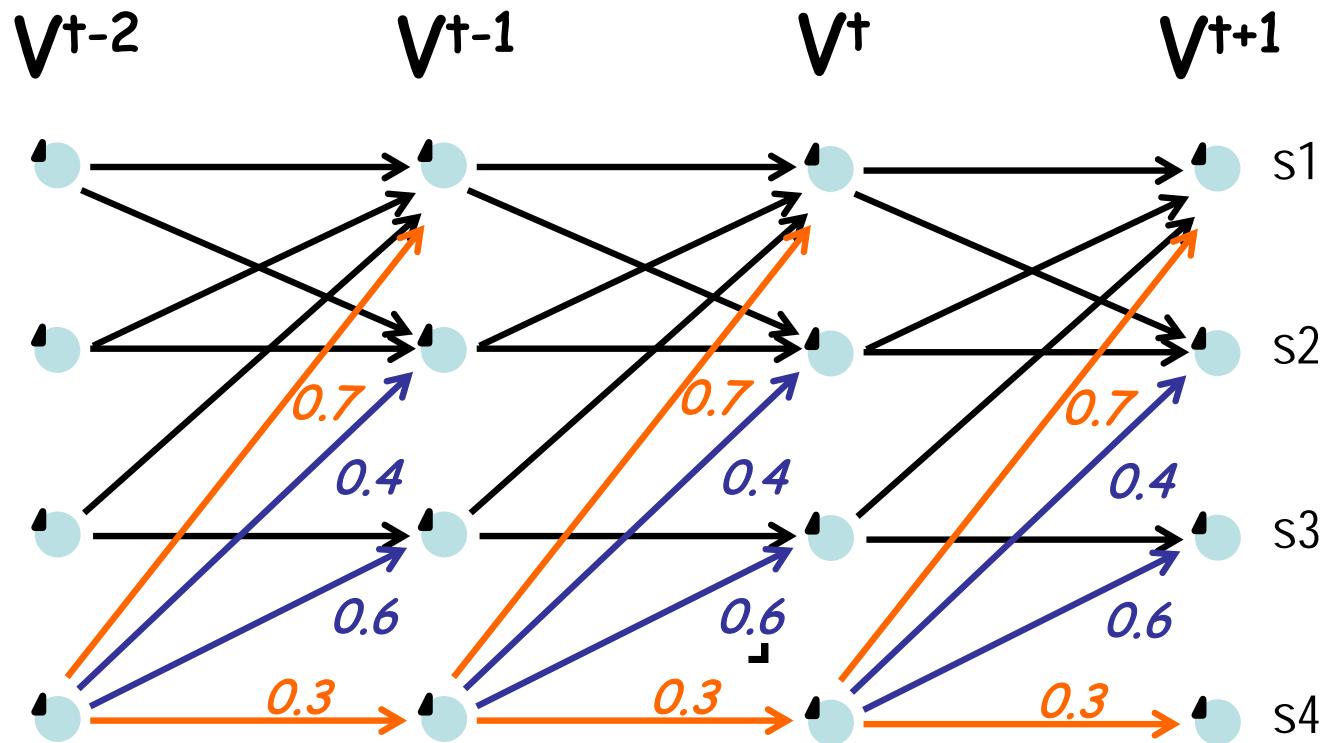
Bellman backup

$$V^k(s) = R(s) + \max_a \sum_{s'} \Pr(s, a, s') \cdot V^{k-1}(s')$$

$$\pi^*(s, k) = \arg \max_a \sum_{s'} \Pr(s, a, s') \cdot V^{k-1}(s')$$

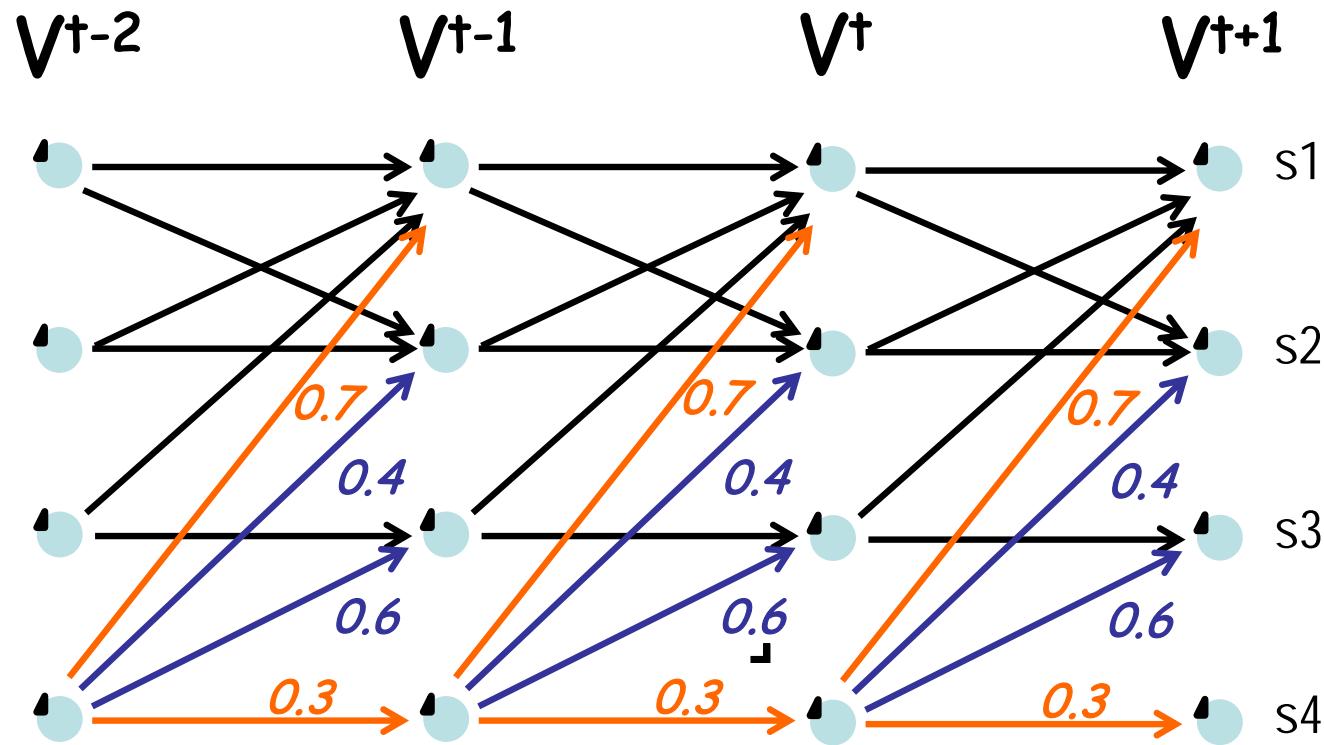
V^k is optimal k-stage-to-go value function

Value Iteration



$$V^t(s_4) = R(s_4) + \max \{ 0.7 V^{t+1}(s_1) + 0.3 V^{t+1}(s_4), 0.4 V^{t+1}(s_2) + 0.6 V^{t+1}(s_3) \}$$

Value Iteration



$$\Pi^*(s_4) = \max \{ \text{orange arrow value}, \text{blue arrow value} \}$$

Value Iteration

$V_1(s) := 0$ for all s

$t := 1$

loop

$t := t + 1$

loop for all $s \in \mathcal{S}$ and for all $a \in \mathcal{A}$

$Q_t^a(s) := R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V_{t-1}(s')$

$V_t(s) := \max_a Q_t^a(s)$

end loop

until $|V_t(s) - V_{t-1}(s)| < \epsilon$ for all $s \in \mathcal{S}$

Assumes that the transition model (T) is known or has been estimated from data

Policy Iteration

- Manipulate the policy directly rather than finding it indirectly

choose an arbitrary policy π'

loop

$$\pi := \pi'$$

compute the value function of policy π :

solve the linear equations

$$V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') V_\pi(s')$$

improve the policy at each state:

$$\pi'(s) := \arg \max_a (R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V_\pi(s'))$$

until $\pi = \pi'$

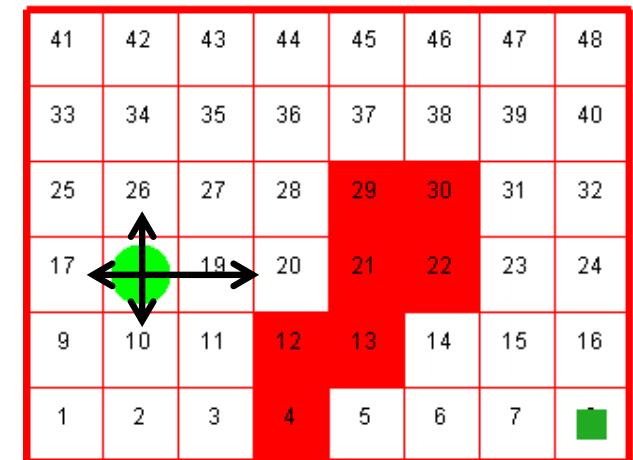
Fewer iterations required but each iteration is more computationally expensive

Unknown World Dynamics

- But most of the time the agent doesn't know how the world operates.
- The agent learns by experiencing events in the world; this is the traditional setting for reinforcement learning.
- What can the agent learn?
 - Best action to take in a given state (policy)
 - Value of a specific state/action pair
 - World model:
 - Reward for being a given state
 - Likely outcomes of taking an action (successor state)
 - Different RL algorithms attempt to learn different things.

Maze Simulation

- Select an action
- Stochastically move in that direction
 - chance = rand.nextDouble();
 - Action=(1,2,3,4)
 - if chance < 0.6
 - go in the direction of action
 - else if chance < 0.7
 - execute action+1 % 4
 - else if chance < 0.8
 - execute action+2 % 4
 - else
 - stay in place



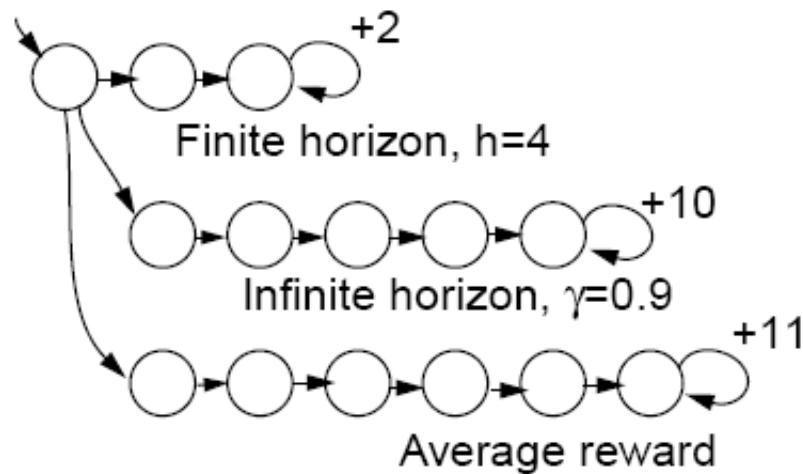
Reward Horizon

- Different reward models
 - Finite-horizon
 - Infinite-horizon
 - Average reward

$$E\left(\sum_{t=0}^h r_t\right)$$

$$E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right)$$

$$\lim_{h \rightarrow \infty} E\left(\frac{1}{h} \sum_{t=0}^h r_t\right)$$



Case where optimal policy depends on reward model

Discounted infinite horizon reward is most commonly used because of convergence properties

Determining Action

- How does the robot select an action before it has learned what to do?
 - Greedy
 - Max observed reward (generally employed after the robot has learned)
 - Exploration bonuses
 - Randomized
 - Purely random ($P(a) = 1/\text{total available actions}$)
 - Boltzmann exploration:
 - T constant that increases the probability of random action
 - Simple Combination
 - $\frac{1}{2}$ time greedy
 - $\frac{1}{2}$ time randomized

$$P(a) = \frac{e^{R(a)/T}}{\sum_{a' \in A} e^{R(a')/T}}$$

Approaches

- Temporal-difference learning
 - TD error: update is based on the error in our estimate of the state value
- Model free vs. model-based learning
 - Maintain a model of the world dynamics vs. no model
- On policy vs. off policy
 - While learning the policy the agent follows the policy
- Examples:
 - Q-learning: model-free, TD learner, off policy
 - SARSA: model-free, TD learner, on-policy
 - R-max: model-based

Q-Learning

- Learns an action-utility function
- Q-function

Alpha=learning rate
Gamma=discount reward

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') \max_{a'} Q^*(s', a') .$$

- Q-learning rule

$$Q(s, a) := Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

Q-function can be stored as a table or can be replaced by a function-approximator

Q-learning is exploration insensitive and will converge to the optimal values as long as S/A pairs are tried frequently enough

Q-Table

- Construct the Q-table
 - Initialized with 0s everywhere
 - Each row: state
 - Each column: action
- Mapping between map coordinates and Q-table coordinates
 - $\text{map}=(\text{roboti}, \text{robotj})$
 - $s=(\text{roboti}*\text{colNum})+\text{robotj}$
 - $\text{roboti}=(\text{int})(s/\text{colNum})$
 - $\text{robotj}=s\%\text{colNum}$

		N	E	S	W
	Q (s,a)	s ₁	0	0	0
		s ₂	0	0	0
		s ₃	0	0	0
		s ₄	0	0	0
		...	0	0	0
		s _n	0	0	0

Learning

Algorithm 1 Q-Learning Algorithm

Define $\alpha = 0.65, \gamma = 0.99$

repeat

 Initialize s to robot or random location

repeat

 Choose an action a from state s using random exploration policy

 Take the action a , observe reward $r(s')$ and next state s'

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r(s') + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

$s \leftarrow s'$

until s reaches goal or max episode length

until number of episodes reached (e.g. 150 steps)

} Single Episode

Episode=a sequence of exploration actions that terminates when you reach the goal or the max episode length

Max Episode Length=max length of path you want the robot to explore
(depends on size of map)

Total Episodes=learning time

Learning

repeat

 Choose an action a from state s using random exploration policy

 Take the action a , observe reward $r(s')$ and next state s'

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r(s') + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

$$s \leftarrow s'$$

until s reaches goal or max episode length

- Random exploration policy: $a = \text{rand.nextInt}(4);$
 - Current state: s_2
 - Choose an action: W
 - Simulate the robot and observe the next state
 - Next state: i.e. s_1
-
- $Q(s_2, W) \leftarrow Q(s_2, W) + 0.65 * [r(s_1) + 0.99 * \max_{a'} Q(s_1, a') - Q(s_2, W)]$
 - Q-value for best action= \max over row s_1

Q-Learning

- <https://youtu.be/j5C2LIZ5O7M>

SARSA

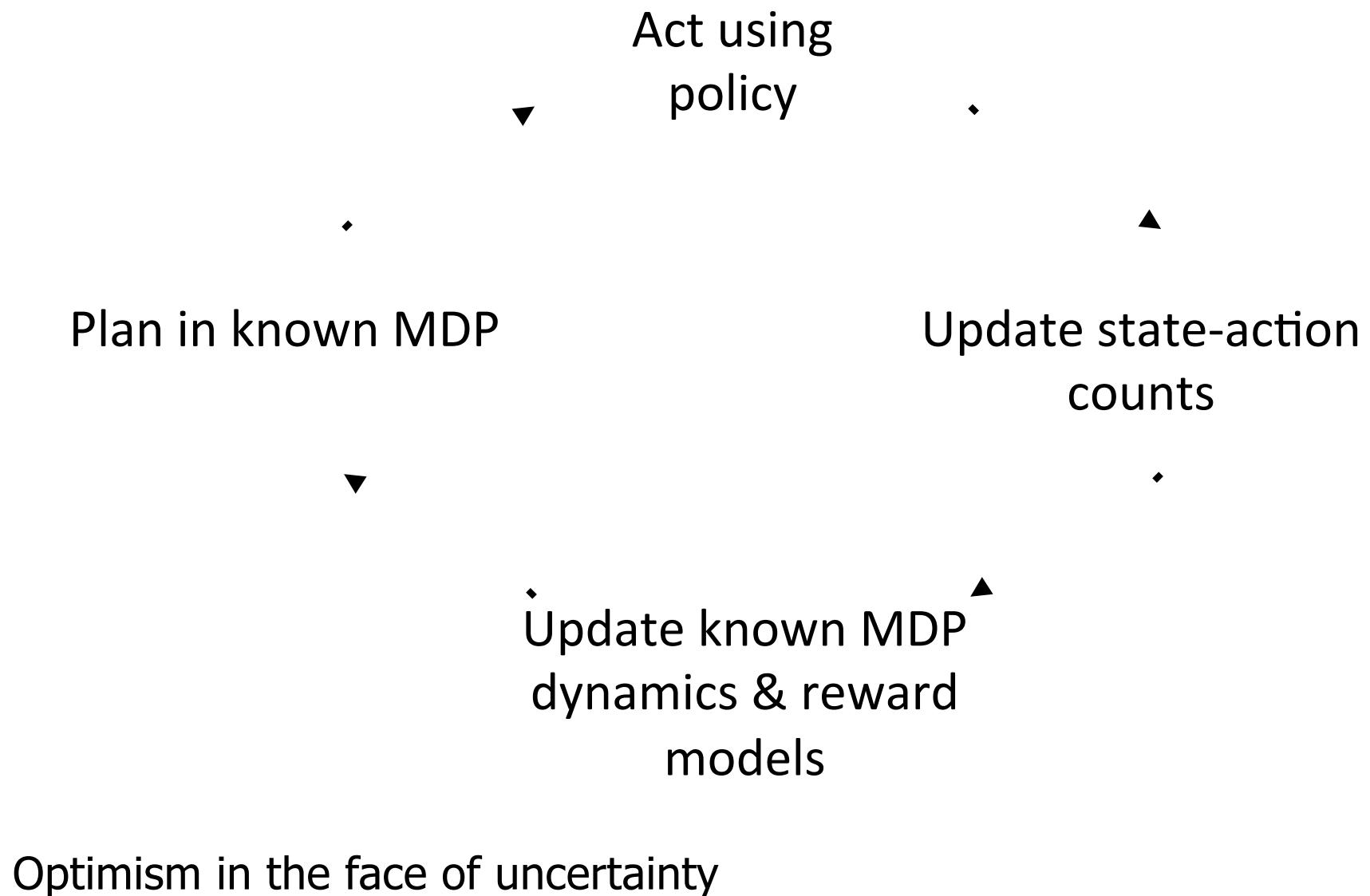
State Action Reward State Action

Algorithm 1 SARSA

- 1: Initialise Q arbitrarily, $Q(\text{terminal}, \cdot) = 0$
 - 2: **repeat**
 - 3: Initialize s
 - 4: Choose a ϵ -greedily
 - 5: **repeat**
 - 6: Take action a , observe r, s'
 - 7: Choose a' ϵ -greedily
 - 8: $Q(s, a) \leftarrow Q(s, a) + \alpha (r + \gamma Q(s', a') - Q(s, a))$
 - 9: $s \leftarrow s', a \leftarrow a'$
 - 10: **until** s is terminal
 - 11: **until** convergence
-

Unlike Q-learning, not taking the maximum over possible actions

Model-based: R-max



R-max

		Reward					
		S1	S2	S3	S4	...	
Known/ Unknown	▲	U	U	U	U	...	
	►	U	U	U	U	...	
	▼	U	U	U	U	...	
	◀	U	U	U	U	...	
	●	R _{max}	R _{max}	R _{max}	R _{max}	...	
Transition Counts	▲	0	0	0	0	...	In the “known” MDP, any unknown (s,a) pair has its dynamics set as a self loop & reward = Rmax
	►	0	0	0	0	...	
	▼	0	0	0	0	...	
	◀	0	0	0	0	...	
	●	0	0	0	0	...	

R-max

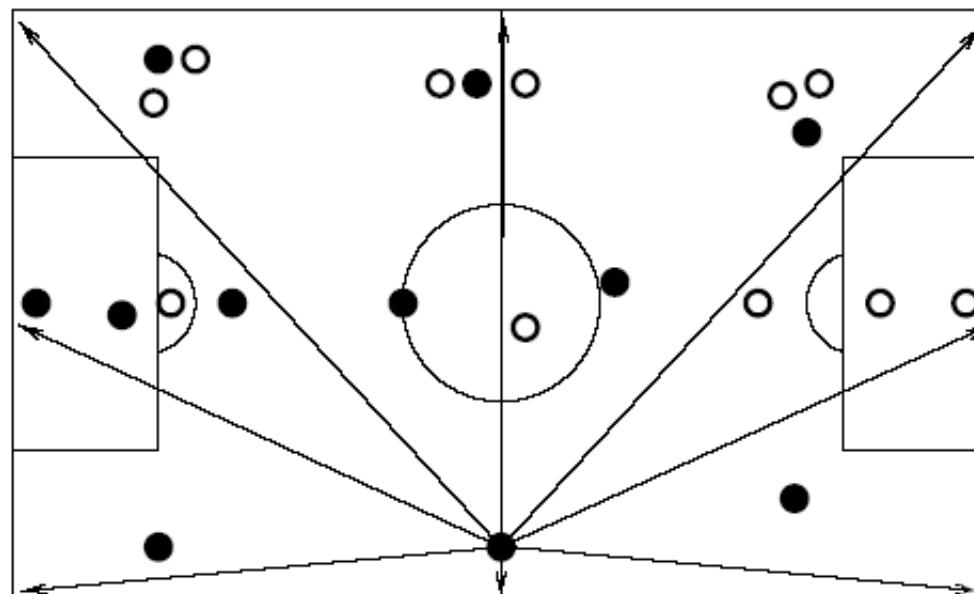
		Reward					
		S2	S2	S3	S4	...	
Known/ Unknown	↑	U	U	U	U		↑
	►	U	U	K	U		►
	▼	U	U	U	U		▼
	◀	U	U	U	U		◀
	·	R _{max}	R _{max}	R _{max}	R _{max}		·
Transition Counts	↑	3	3	4	3		If counts for (s,a) > N, (s,a) becomes known:
	►	2	4	5	0		use observed data to estimate transition & reward model for (s,a)
	▼	4	0	4	4		
	◀	2	2	4	1		when planning
	·	R _{max}	R _{max}	R _{max}	R _{max}		

Research Issues

- Combining deep learning and RL
- Sample reduction: update as much of the model as possible on the basis of one sample
- Exploration/exploitation: is it better to explore new options or to gain rewards based on already known information
- Credit assignment: in a long policy (possibly involving many agents actions) how to correctly assign value to the right state-action transitions
- Transfer learning: how to generalize to a new model or situation

Example: Multi-agent RL

- Robocup simulation league
 - Learn a policy of where to kick the ball
 - Shows how clever representation can make the problem significantly easier



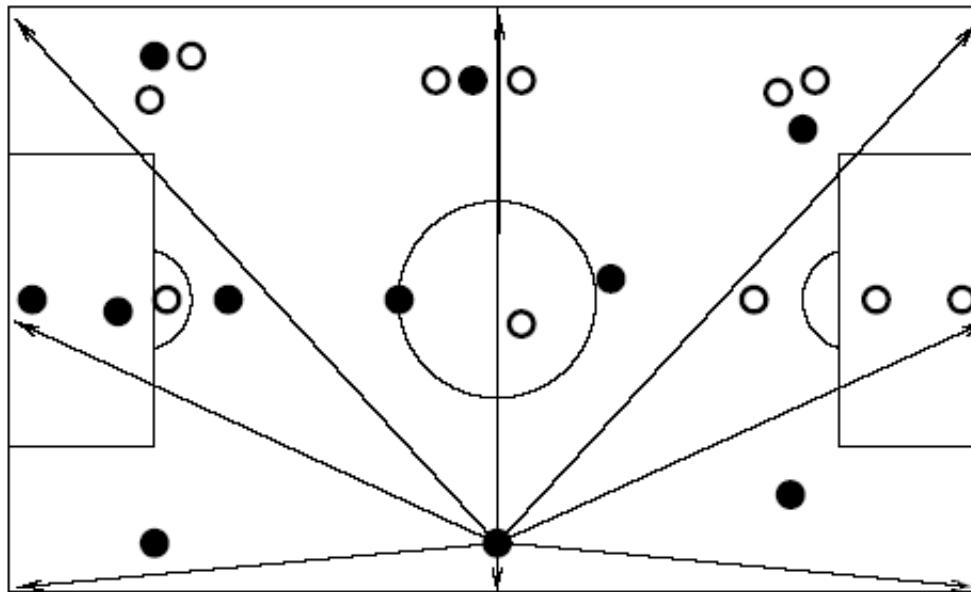
Characteristics

- Opaque-transition
 - No simple function which governs the transition from one state to another
- Chained agents
 - No single agent's actions can achieve the goal
- Team-partitioned
 - Each teammate learns and uses a separate part of the table based on its role in the team

Tricks

- Divide and conquer---learn different aspects of the problem separately and then combine
- Focus on reducing the state-space representation
- Select reward function that rewards for intermediate actions
- Do not bother using learning for behaviors that are easy to program anyways

Action Representation



Actions=possible kicks towards all the corners of the field

Agent is trying which of 8 possible kicks it should make when it has possession of the ball

How many Q-values are there?

- Q-values is normally equal to $S \times A$
- Naïve field representation would generate large number of states (23 moving entities, absolute field coordinates)
- Q-table used in this system only has 88 values; each player only learns 8 Q-values per game
- How is this done?
 - Answer: by the use of a single feature (kick success)

Algorithm

- State generalization
 - Move from raw representation of state space through the outputs ($f: S \rightarrow V$)
- Value function learning
 - Determine whether each possible action is likely to succeed (represented by Q-values)
- Action selection
 - Update Q-table based on action chosen and reward received

Learning Rule

- Modified version of the Q-learning which only takes into account immediate reward and not future transitions

$$Q(v, a) = Q(v, a) + \alpha(r - Q(v, a))$$

- Standard Q-learning

$$Q(s, a) := Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

Value Function Learning

- Insight: use high level action-dependent features to describe the state space
- Only need to consider the features relevant to the action being performed

The diagram illustrates the reduction of feature dimensions. On the left, there is a 3x4 grid of cells. The columns are labeled $e(s, a_0)$, $e(s, a_1)$, $Q(v, a_0)$, and $Q(v, a_1)$. The rows are labeled u_0 , u_1 , and u_2 . The values in the grid are: $e(s, a_0)$ contains u_0, u_1, u_2 ; $e(s, a_1)$ contains u_0, u_1, u_2 ; $Q(v, a_0)$ contains $q_{0,0}, q_{0,1}, q_{0,2}$; and $Q(v, a_1)$ contains $q_{1,0}, q_{1,1}, q_{1,2}$. An arrow points from this grid to a smaller 3x3 grid on the right. This smaller grid has columns labeled $e(s, a_i)$, $Q(v, a_0)$, and $Q(v, a_1)$. The rows are labeled u_0, u_1, u_2 . The values in the grid are: $e(s, a_i)$ contains u_0, u_1, u_2 ; $Q(v, a_0)$ contains $q_{0,0}, q_{0,1}, q_{0,2}$; and $Q(v, a_1)$ contains $q_{1,0}, q_{1,1}, q_{1,2}$.

$e(s, a_0)$	$e(s, a_1)$	$Q(v, a_0)$	$Q(v, a_1)$
u_0	u_0	$q_{0,0}$	$q_{1,0}$
u_0	u_1	$q_{0,0}$	$q_{1,1}$
u_0	u_2	$q_{0,0}$	$q_{1,2}$
u_1	u_0	$q_{0,1}$	$q_{1,0}$
u_1	u_1	$q_{0,1}$	$q_{1,1}$
u_1	u_2	$q_{0,1}$	$q_{1,2}$
u_2	u_0	$q_{0,2}$	$q_{1,0}$
u_2	u_1	$q_{0,2}$	$q_{1,1}$
u_2	u_2	$q_{0,2}$	$q_{1,2}$

⇒

$e(s, a_i)$	$Q(v, a_0)$	$Q(v, a_1)$
u_0	$q_{0,0}$	$q_{1,0}$
u_1	$q_{0,1}$	$q_{1,1}$
u_2	$q_{0,2}$	$q_{1,2}$

- Normally features are independent of actions

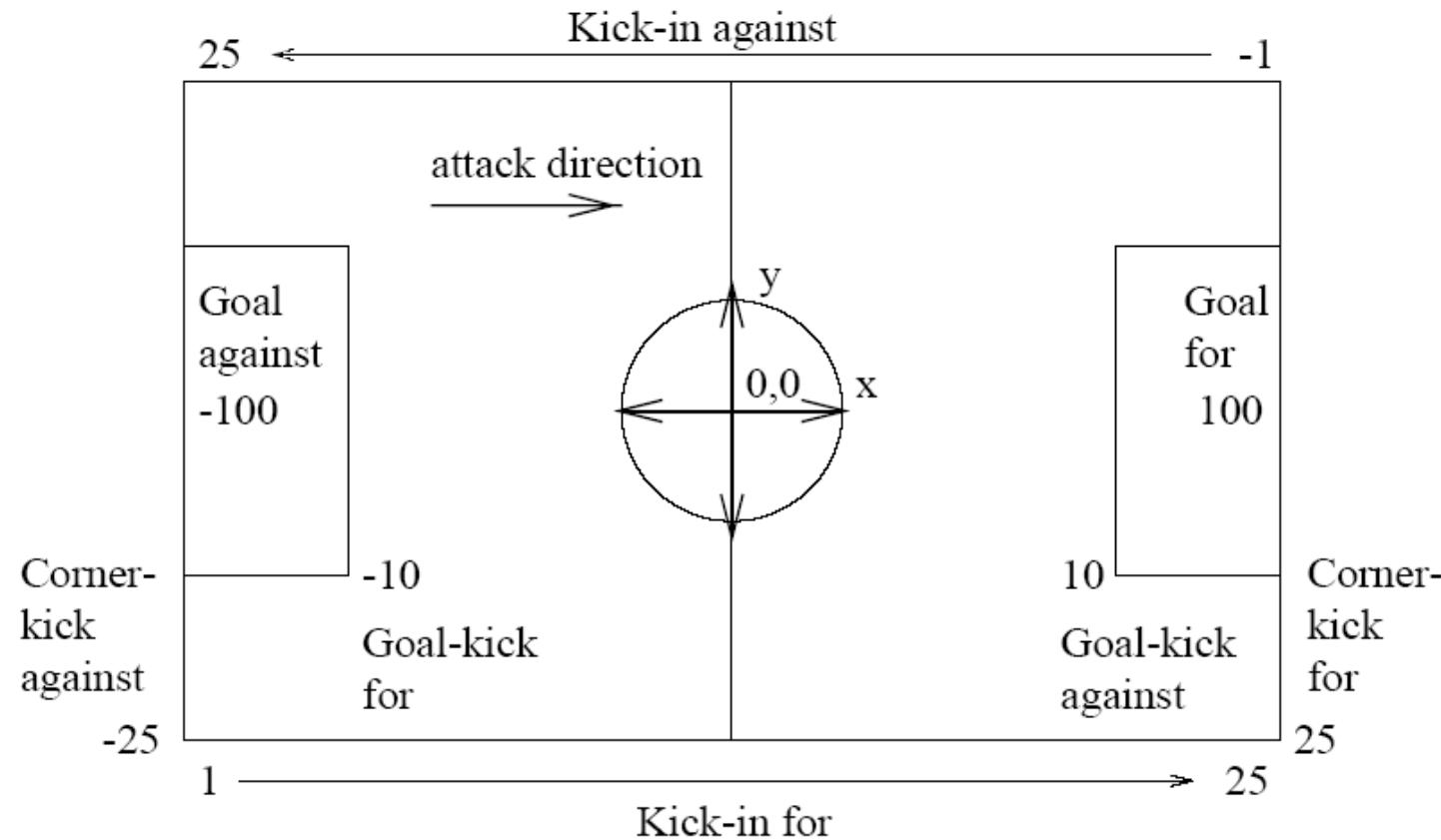
Action Selection

- Features themselves are discriminative of whether the action should be used
- Hence never consider actions when the feature falls outside a set of values
- If only successful kicks are considered then effectively there is only one state (SuccessfulKick) and 8 actions
- Number of Q-values learned by each agent is 8!
- During the game the actions can be selected using any standard exploration policy (e.g. Boltzmann, greedy)

Layered Approach

- Divide and conquer the learning task
- Low level modules make a big difference to success
- Predicting kick success
 - Use 200 continuous value attributes describing teammate/opponent positions
 - C4.5 decision tree algorithm
- Intercepting ball
 - Trained a neural network specifically for the interception behavior

Reward Representation

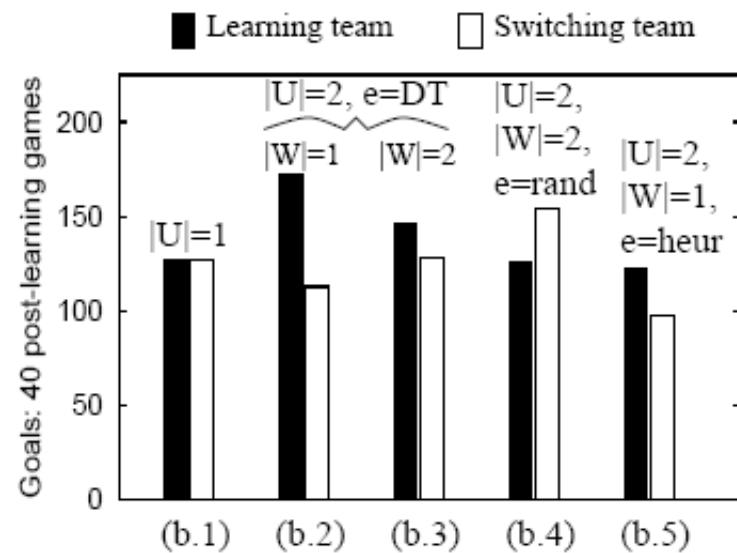
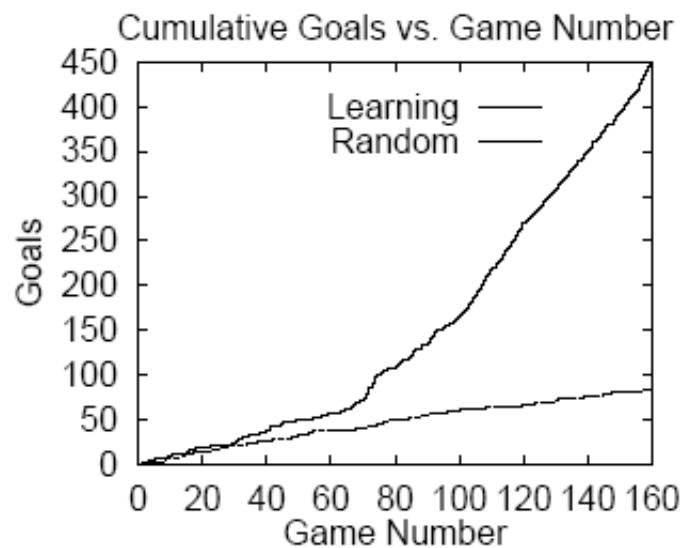


- Use intermediate rewards rather than just rewarding at goals
- Base reward functions are scaled by time events occur
- Must be observed by agent doing the learning

Results

- Evaluated vs. random team and switching team
- Random team just makes random passes
- Switching team uses hand-coded policy in which the agents all stay towards one half of field
- Interesting notes:
 - 11 players got an average of 1490 action reinforcement pairs per game
 - Each action is only tried on average 186 times over 160 games

Results



RL for Humanoid Robots

<http://www.youtube.com/watch?v=mRpX9DFCdwI>

Summary

- TPOT-RL tells the agent where to kick when it has possession of the ball
- Does not tell the agent whether to kick or not to kick
- Does not tell agents where to move
- Does not tell agent which agent will receive the ball
- Separate decision-tree used to calculate value of interception
- However, it is very useful for picking a good kick that helps the team

CAP 6671 Intelligent Systems

Lecture 11

Neural Networks

Deep Learning

Convolutional NN

Instructor: Dr. Gita Sukthankar
Email: gitars@eeecs.ucf.edu

Neural Network Learning

- Learning approach based on modeling adaptation in biological neural systems.
- **Perceptron**: Initial algorithm for learning simple neural networks (single layer) developed in the 1950's.
- **Backpropagation**: More complex algorithm for learning multi-layer neural networks developed in the 1980's.

Artificial Neuron Model

- Model network as a graph with cells as nodes and synaptic connections as weighted edges from node i to node j , w_{ji}

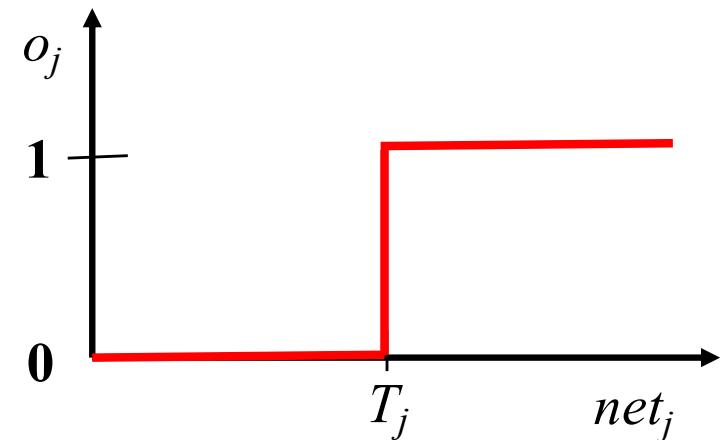
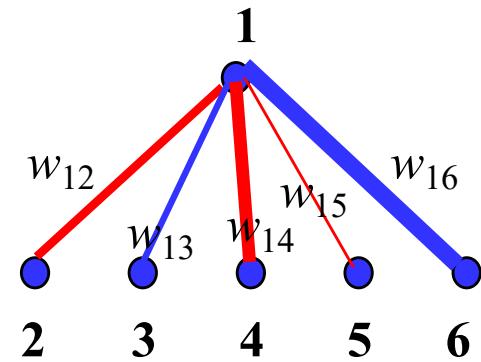
- Model net input to cell as

$$net_j = \sum_i w_{ji} o_i$$

- Cell output is:

$$o_j = \begin{cases} 0 & \text{if } net_j < T_j \\ 1 & \text{if } net_j \geq T_j \end{cases}$$

(T_j is threshold for unit j)



Neural Computation

- McCollough and Pitts (1943) showed how such model neurons could compute logical functions and be used to construct finite-state machines.
- Can be used to simulate logic gates:
 - AND: Let all w_{ji} be T_j/n , where n is the number of inputs.
 - OR: Let all w_{ji} be T_j
 - NOT: Let threshold be 0, single input with a negative weight.
- Can build arbitrary logic circuits, sequential machines, and computers with such gates.

Perceptron Training

- Assume supervised training examples giving the desired output for a unit given a set of known input activations.
- Learn synaptic weights so that unit produces the correct output for each example.
- Perceptron uses iterative update algorithm to learn a correct set of weights.

Perceptron Learning Rule

- Update weights by:

$$w_{ji} = w_{ji} + \eta(t_j - o_j)o_i$$

where η is the “learning rate”

t_j is the teacher specified output for unit j .

- Equivalent to rules:

- If output is correct do nothing.
- If output is high, lower weights on active inputs
- If output is low, increase weights on active inputs

- Also adjust threshold to compensate:

$$T_j = T_j - \eta(t_j - o_j)$$

Perceptron Learning Algorithm

- Iteratively update weights until convergence.

Initialize weights to random values

Until outputs of all training examples are correct

For each training pair, E , do:

 Compute current output o_j for E given its inputs

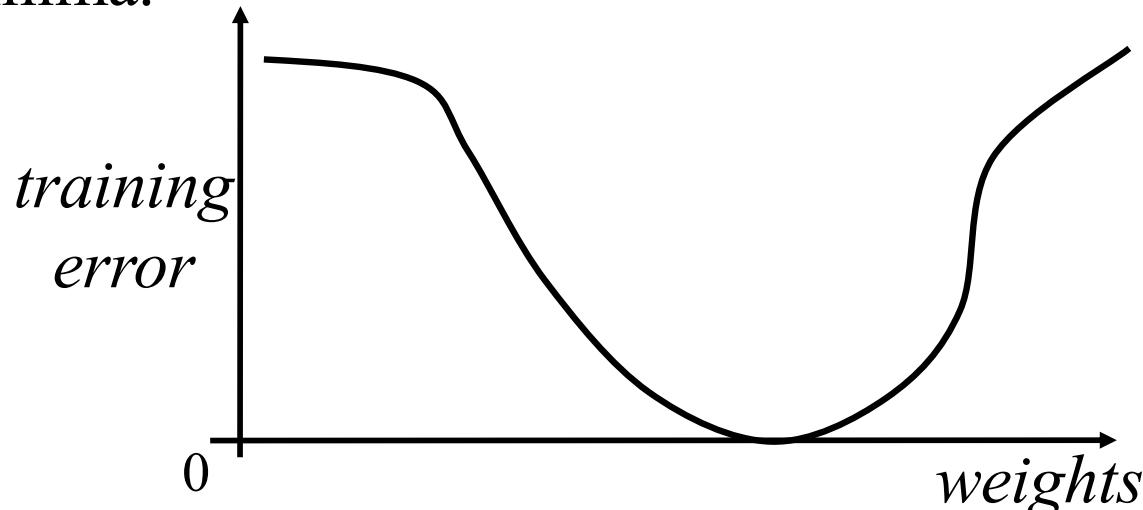
 Compare current output to target value, t_j , for E

 Update synaptic weights and threshold using learning rule

- Each execution of the outer loop is typically called an *epoch*.

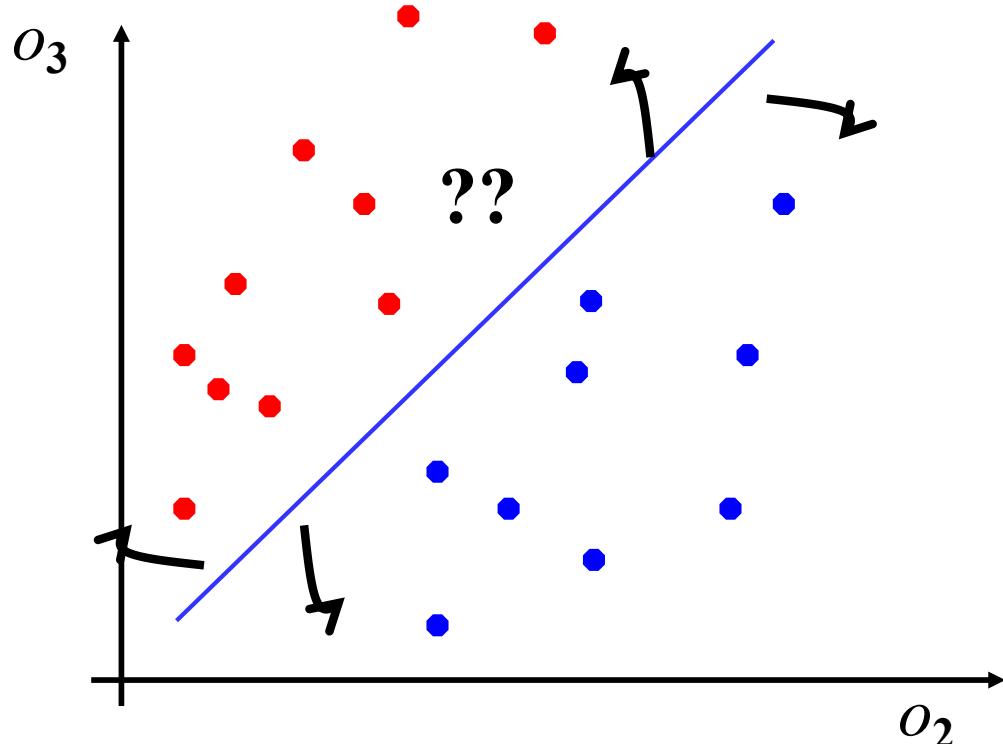
Perceptron as Hill Climbing

- The hypothesis space being search is a set of weights and a threshold.
- Objective is to minimize classification error on the training set.
- Perceptron effectively does hill-climbing (gradient descent) in this space, changing the weights a small amount at each point to decrease training set error.
- For a single model neuron, the space is well behaved with a single minima.



Perceptron as a Linear Separator

- Since perceptron uses linear threshold function, it is searching for a linear separator that discriminates the classes.



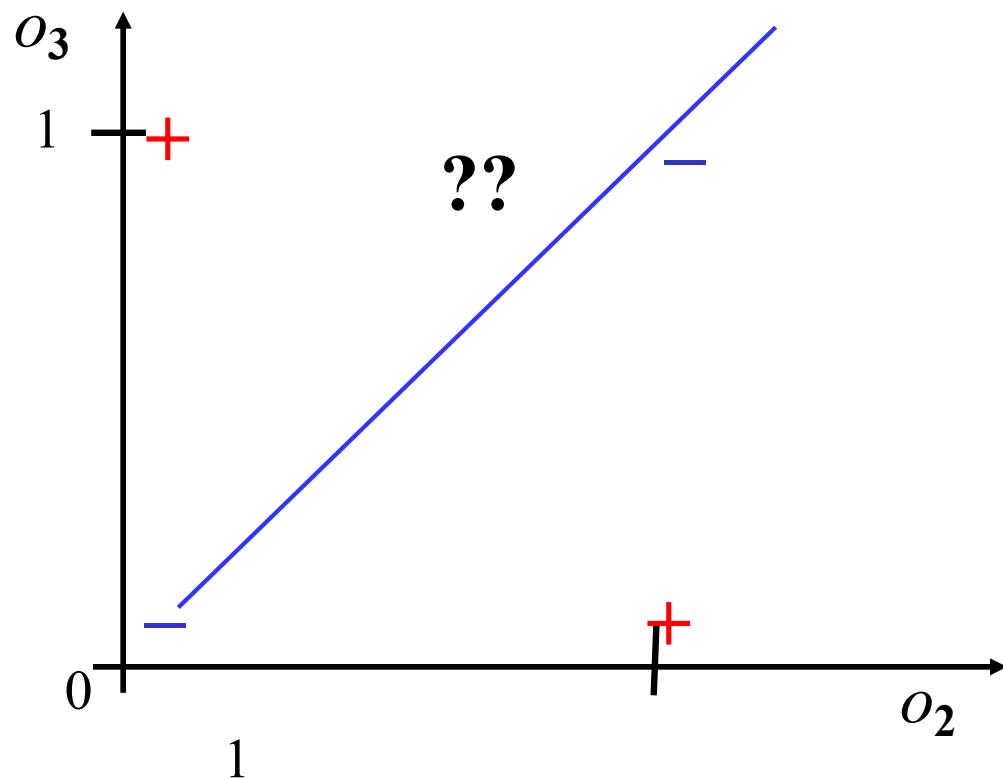
$$w_{12}o_2 + w_{13}o_3 > T_1$$

$$o_3 > -\frac{w_{12}}{w_{13}}o_2 + \frac{T_1}{w_{13}}$$

Or *hyperplane* in
n-dimensional space

Concept Perceptron Cannot Learn

- Cannot learn exclusive-or, or parity function in general.

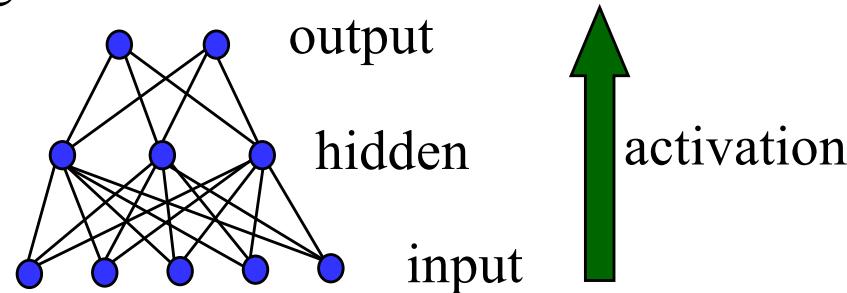


Perceptron Limits

- System obviously cannot learn concepts it cannot represent.
- Minsky and Papert (1969) wrote a book analyzing the perceptron and demonstrating many functions it could not learn.
- These results discouraged further research on neural nets; and symbolic AI became the dominate paradigm.

Multi-Layer Networks

- Multi-layer networks can represent arbitrary functions, but an effective learning algorithm for such networks was thought to be difficult.
- A typical multi-layer network consists of an input, hidden and output layer, each fully connected to the next, with activation feeding forward.

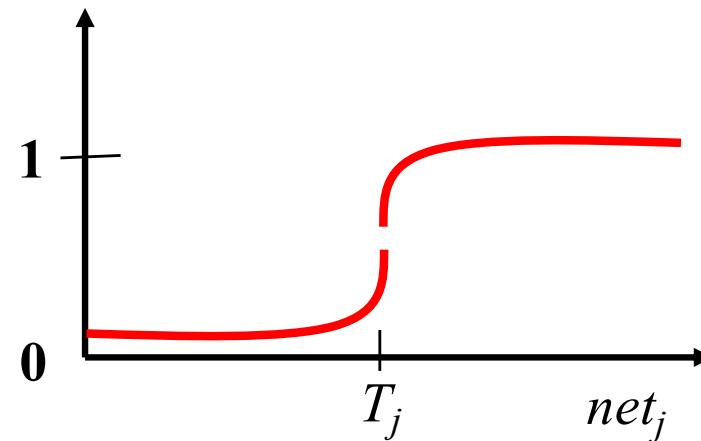


- The weights determine the function computed. Given an arbitrary number of hidden units, any boolean function can be computed with a single hidden layer.

Differentiable Output Function

- Need non-linear output function to move beyond linear functions.
 - A multi-layer linear network is still linear.
- Standard solution is to use the non-linear, differentiable sigmoidal “logistic” function:

$$O_j = \frac{1}{1 + e^{-(net_j - T_j)}}$$



Can also use tanh or Gaussian output function

Gradient Descent

- Define objective to minimize error:

$$E(W) = \sum_{d \in D} \sum_{k \in K} (t_{kd} - o_{kd})^2$$

where D is the set of training examples, K is the set of output units, t_{kd} and o_{kd} are, respectively, the teacher and current output for unit k for example d .

- The derivative of a sigmoid unit with respect to net input is:

$$\frac{\partial o_j}{\partial \text{net}_j} = o_j(1 - o_j)$$

- Learning rule to change weights to minimize error is:

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}$$

Backpropagation Learning Rule

- Each weight changed by:

$$\Delta w_{ji} = \eta \delta_j o_i$$

$$\delta_j = o_j(1 - o_j)(t_j - o_j) \quad \text{if } j \text{ is an output unit}$$

$$\delta_j = o_j(1 - o_j) \sum_k \delta_k w_{kj} \quad \text{if } j \text{ is a hidden unit}$$

where η is a constant called the learning rate

t_j is the correct teacher output for unit j

δ_j is the error measure for unit j

Backpropagation Training Algorithm

Create the 3-layer network with H hidden units with full connectivity between layers. Set weights to small random real values.

Until all training examples produce the correct value (within ε), or mean squared error ceases to decrease, or other termination criteria:

Begin epoch

For each training example, d , do:

Calculate network output for d 's input values

Compute error between current output and correct output for d

Update weights by backpropagating error and using learning rule

End epoch

Error Backpropagation

- First calculate error of output units and use this to change the top layer of weights.

Current output: $o_j=0.2$

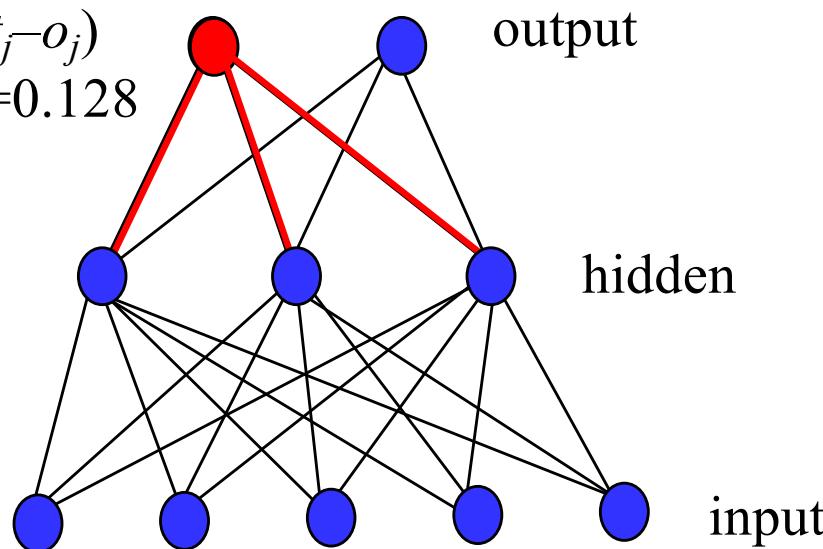
Correct output: $t_j=1.0$

$$\text{Error } \delta_j = o_j(1-o_j)(t_j-o_j)$$

$$0.2(1-0.2)(1-0.2)=0.128$$

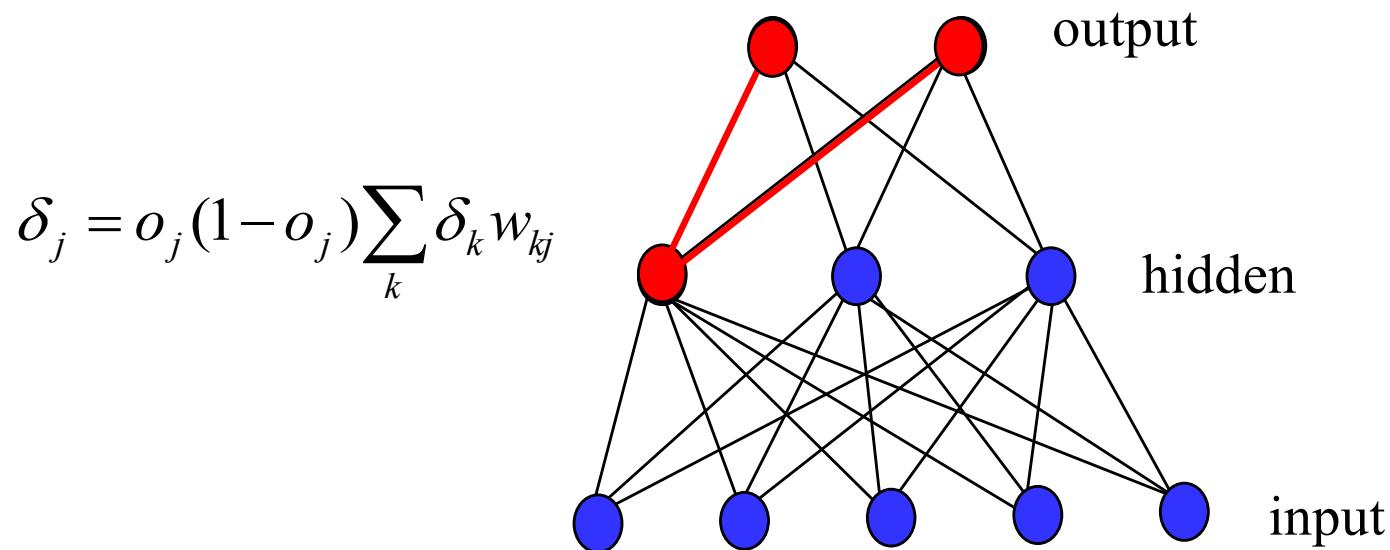
Update weights into j

$$\Delta w_{ji} = \eta \delta_j o_i$$



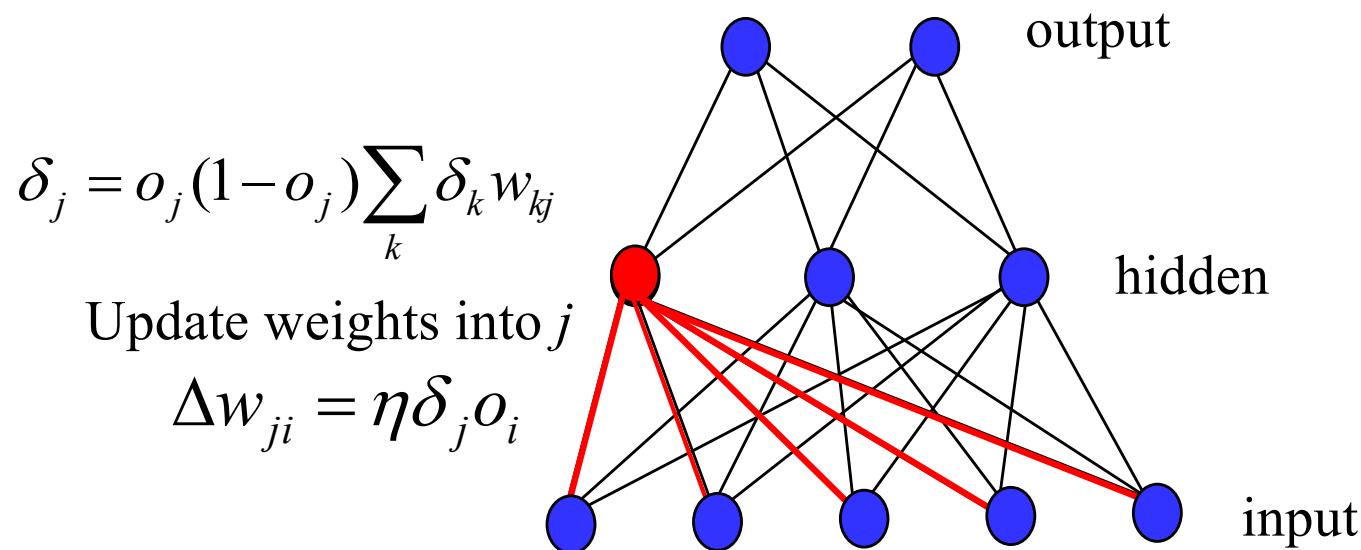
Error Backpropagation

- Next calculate error for hidden units based on errors on the output units it feeds into.



Error Backpropagation

- Finally update bottom layer of weights based on errors calculated for hidden units.

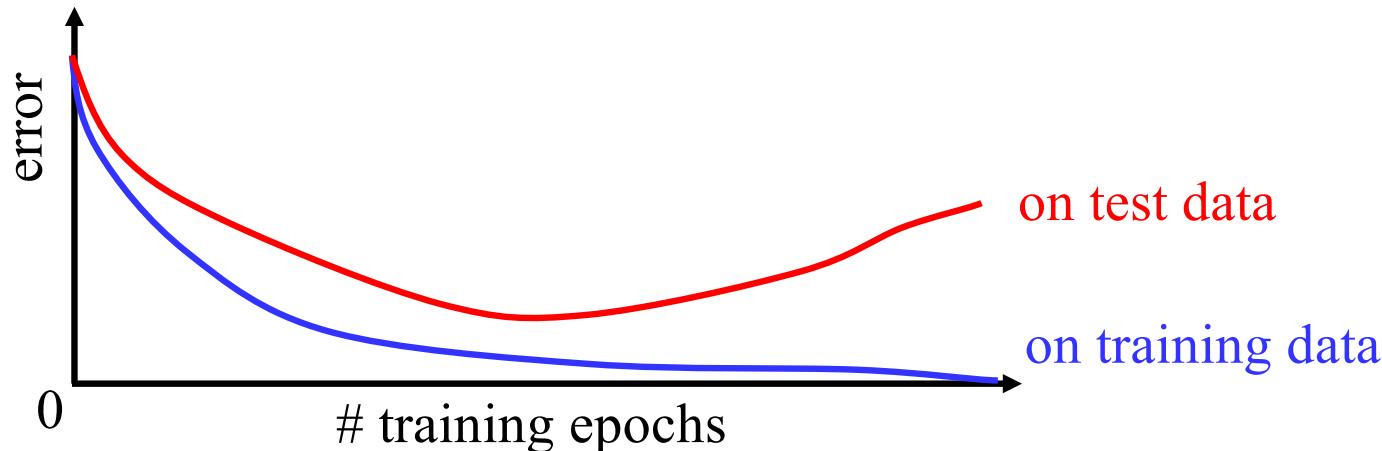


Comments on Training Algorithm

- Not guaranteed to converge to zero training error, may converge to local optima or oscillate indefinitely.
- However, in practice, does converge to low error for many large networks on real data.
- Many epochs (thousands) may be required, hours or days of training for large networks.
- To avoid local-minima problems, run several trials starting with different random weights (*random restarts*).
 - Take results of trial with lowest training set error.
 - Build a committee of results from multiple trials (possibly weighting votes by training set accuracy).

Over-Training Prevention

- Running too many epochs can result in over-fitting.



- Keep a hold-out validation set and test accuracy on it after every epoch. Stop training when additional epochs actually increase validation error.
- To avoid losing training data for validation:
 - Use internal 10-fold CV on the training set to compute the average number of epochs that maximizes generalization accuracy.
 - Train final network on complete training set for this many epochs.

Representational Power

- **Boolean functions:** Any boolean function can be represented by a two-layer network with sufficient hidden units.
- **Continuous functions:** Any bounded continuous function can be approximated with arbitrarily small error by a two-layer network.
 - Sigmoid functions can act as a set of basis functions for composing more complex functions, like sine waves in Fourier analysis.
- **Arbitrary function:** Any function can be approximated to arbitrary accuracy by a three-layer network.

Visualizing Learning

<https://youtu.be/Ilg3gGewQ5U>

So, 1. what exactly is deep learning ?

And, 2. why is it generally better than other methods on image, speech and certain other types of data?

The short answers

1. ‘Deep Learning’ means using a neural network with several layers of nodes between input and output
2. the series of layers between input & output do feature identification and processing in a series of stages, just as our brains seem to.

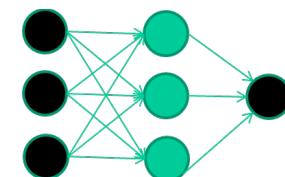
hmmm... OK, but:

**3. multilayer neural networks have been around for
25 years. What's actually new?**

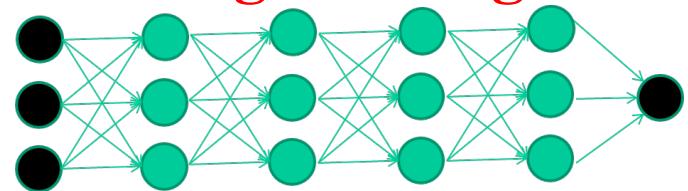
hmmm... OK, but:

3. multilayer neural networks have been around for 25 years. What's actually new?

we have always had good algorithms for learning the weights in networks with 1 hidden layer



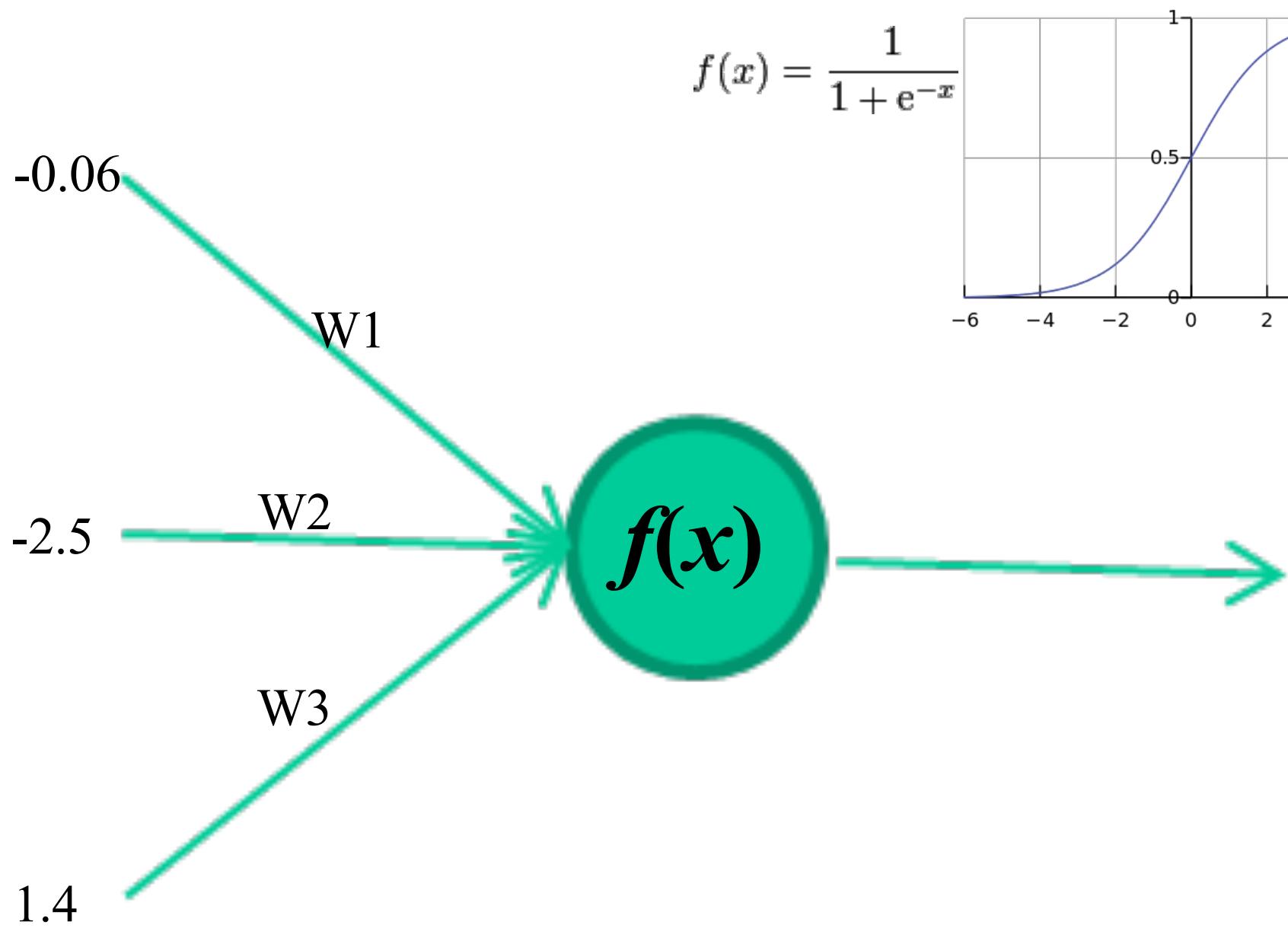
but these algorithms are not good at learning the weights for networks with more hidden layers



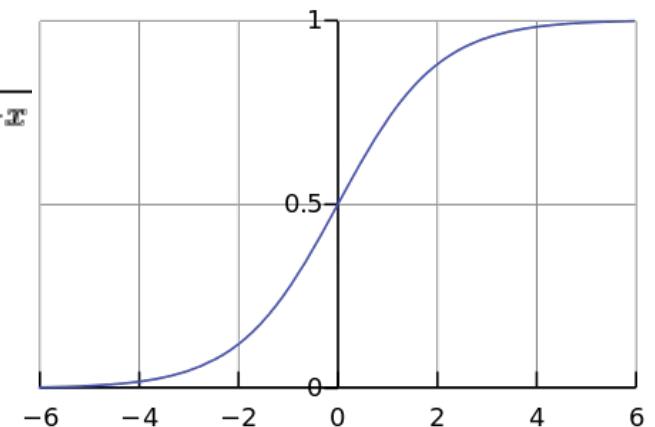
what's new is: algorithms for training many-layer networks

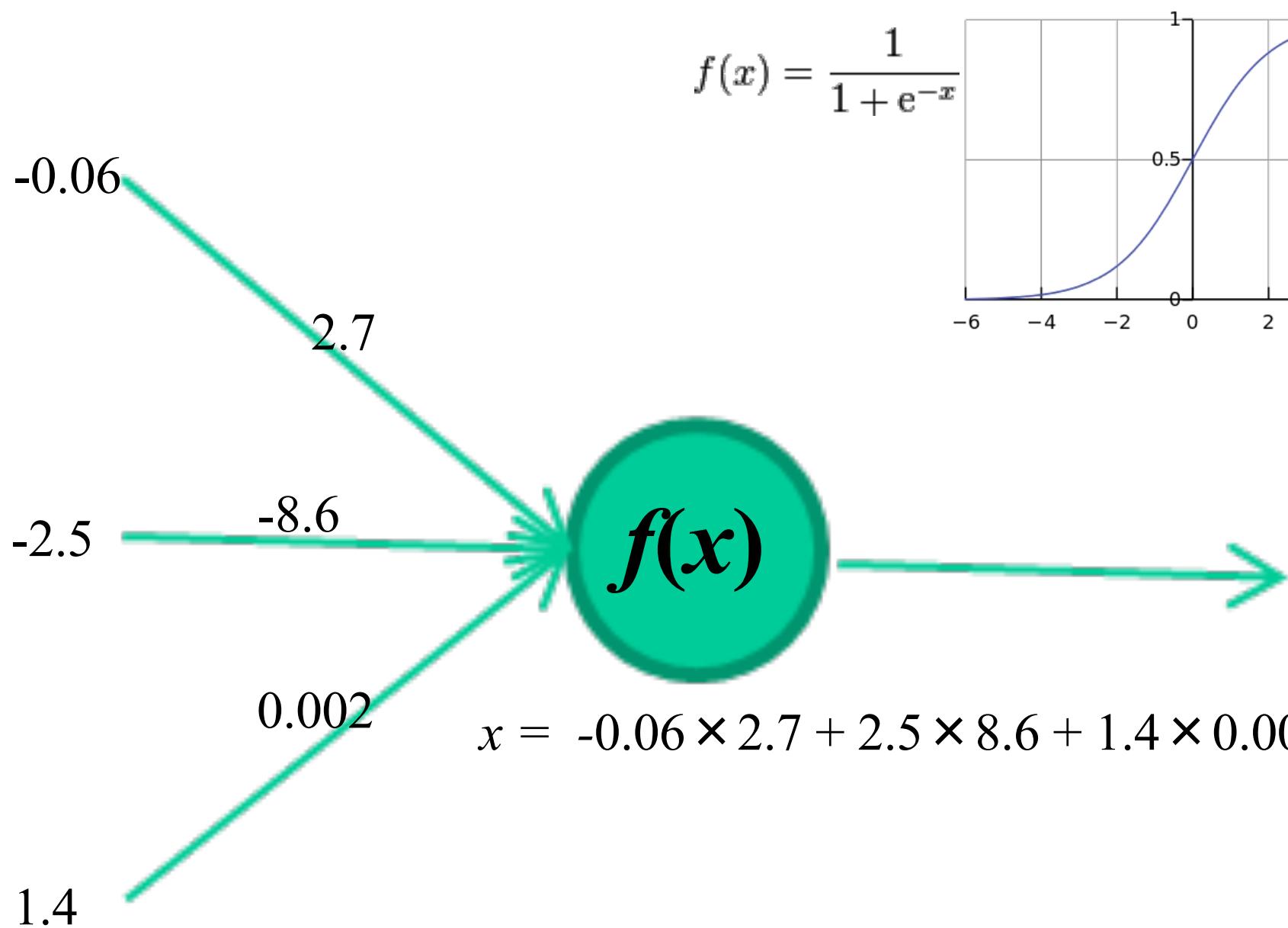
longer answers

1. reminder/quick-explanation of how neural network weights are learned;
2. the idea of **unsupervised feature learning** (why ‘intermediate features’ are important for difficult classification tasks, and how NNs seem to naturally learn them)
3. The ‘breakthrough’ – the simple trick for training Deep neural networks



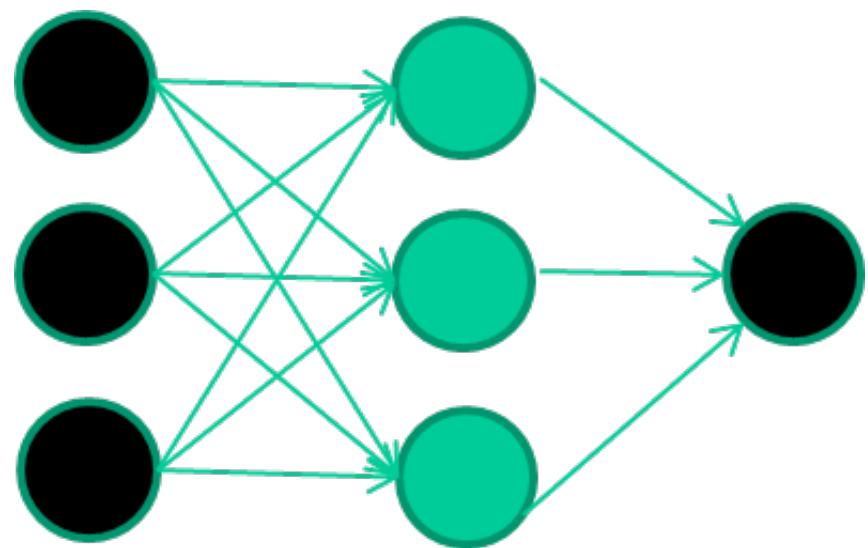
$$f(x) = \frac{1}{1 + e^{-x}}$$





A dataset

<i>Fields</i>	<i>class</i>
1.4 2.7 1.9	0
3.8 3.4 3.2	0
6.4 2.8 1.7	1
4.1 0.1 0.2	0
etc ...	

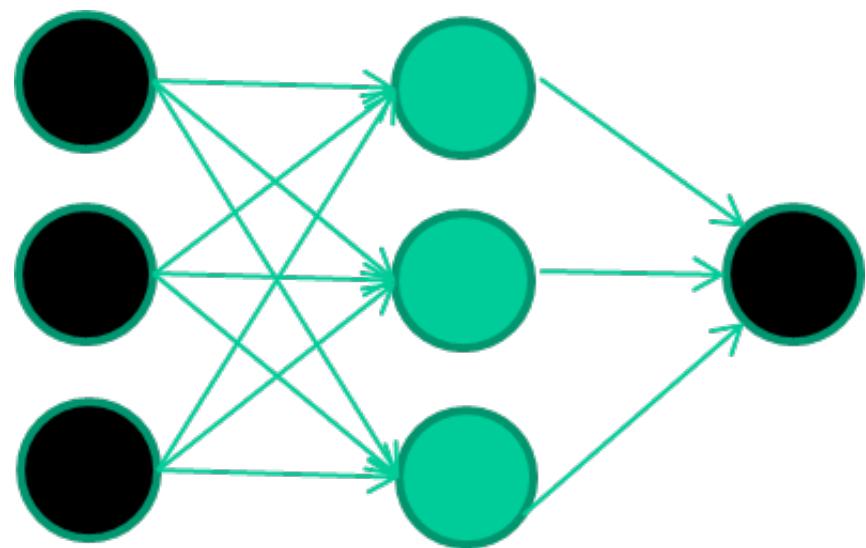


Training the neural network

Fields **class**

1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc ...



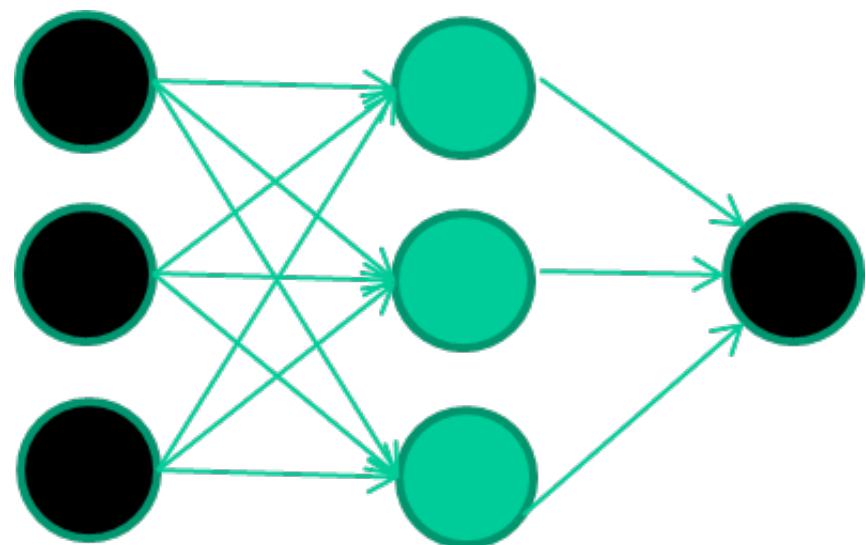
Training data

Fields *class*

1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc ...

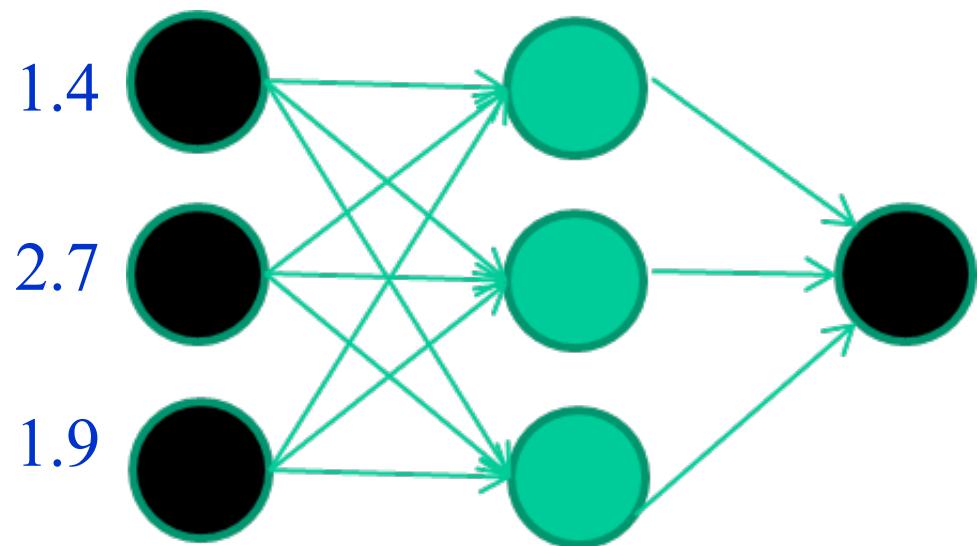
Initialise with random weights



Training data

<i>Fields</i>	<i>class</i>		
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc ...			

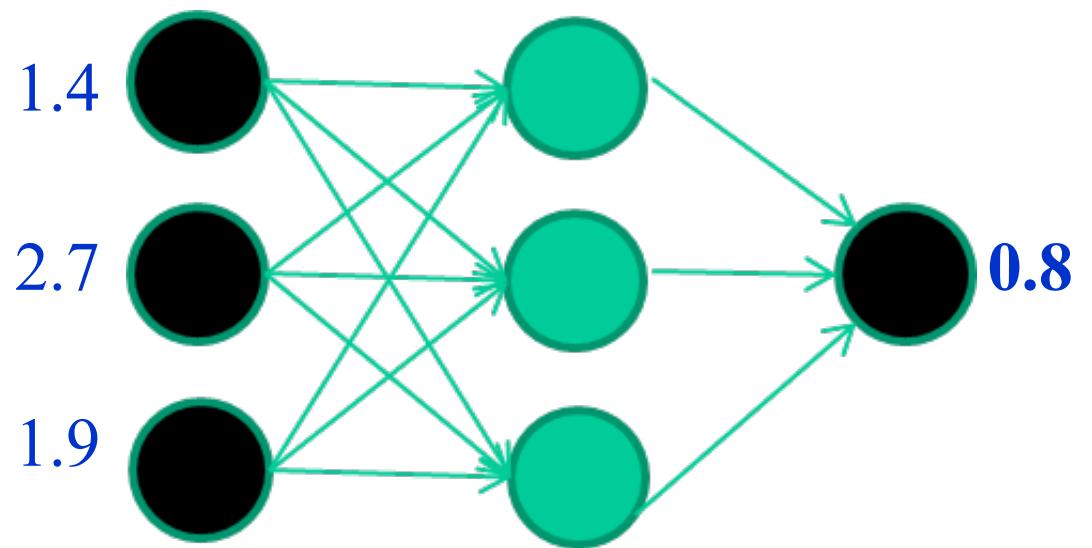
Present a training pattern



Training data

<i>Fields</i>	<i>class</i>		
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc ...			

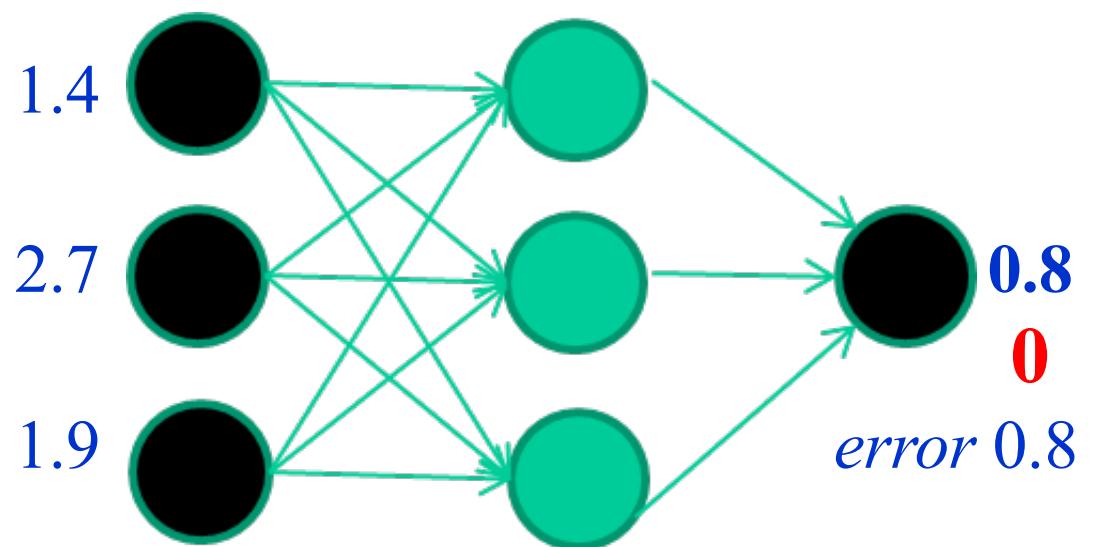
Feed it through to get output



Training data

<i>Fields</i>	<i>class</i>		
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc ...			

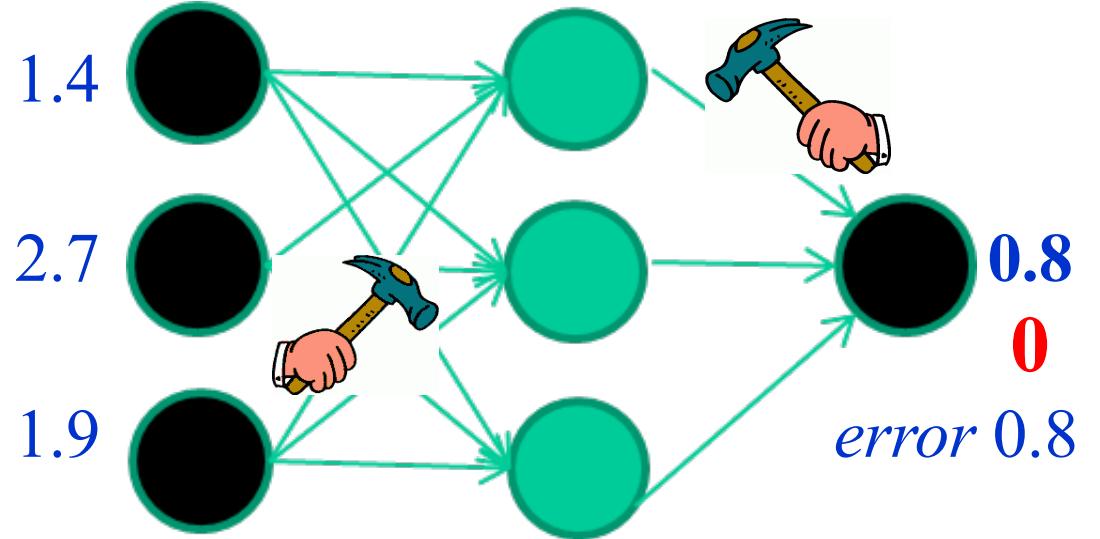
Compare with target output



Training data

<i>Fields</i>	<i>class</i>		
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc ...			

Adjust weights based on error



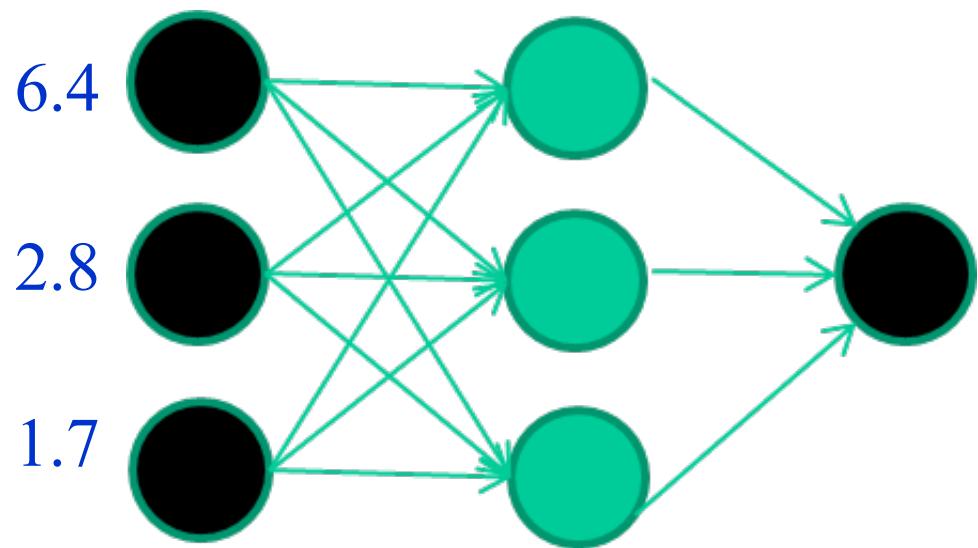
Training data

Fields *class*

1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc ...

Present a training pattern



Training data

Fields *class*

1.4 2.7 1.9 0

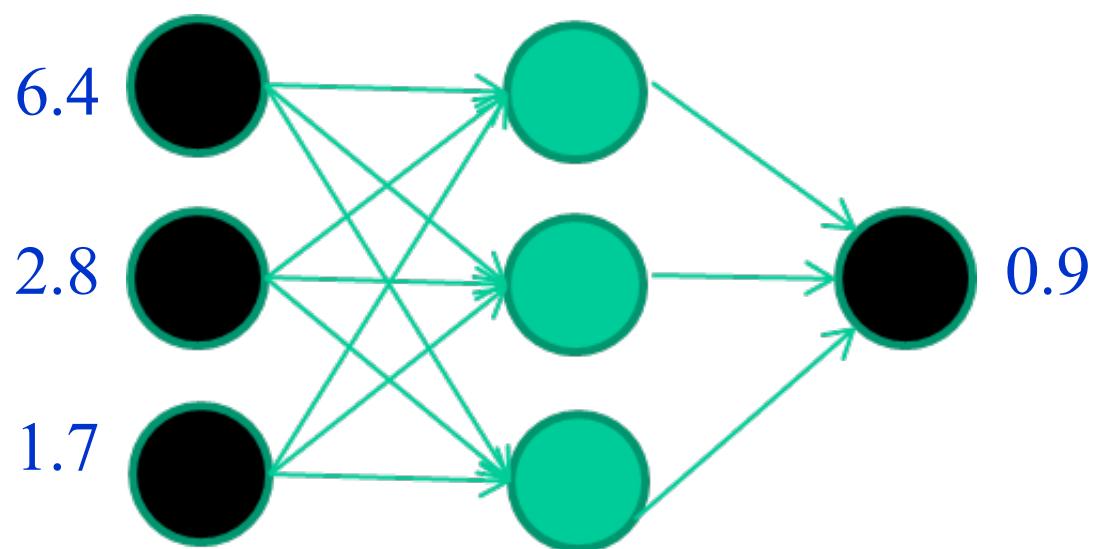
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

Feed it through to get output



Training data

Fields *class*

1.4 2.7 1.9 0

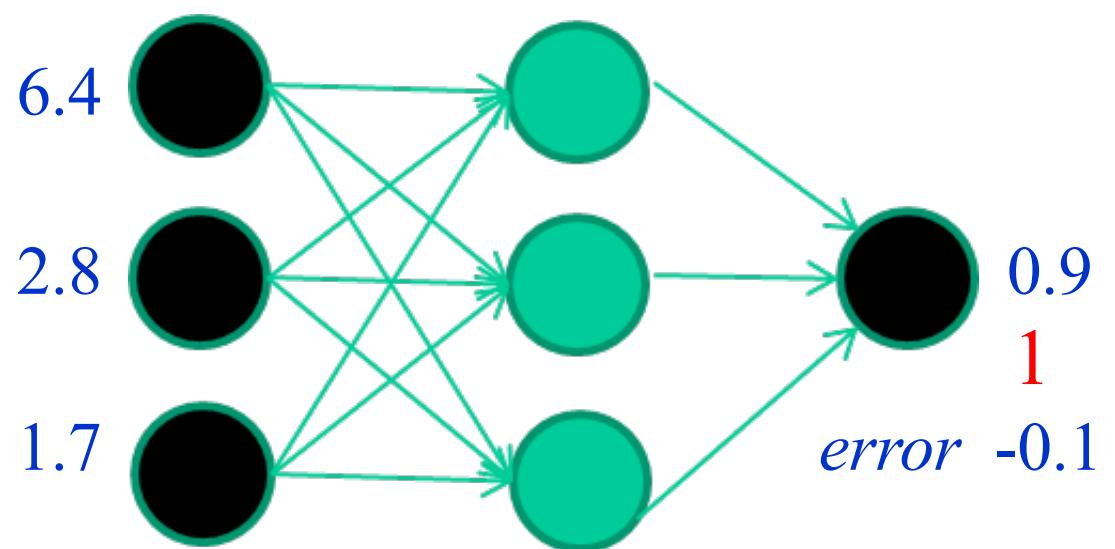
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

Compare with target output



Training data

Fields *class*

1.4 2.7 1.9 0

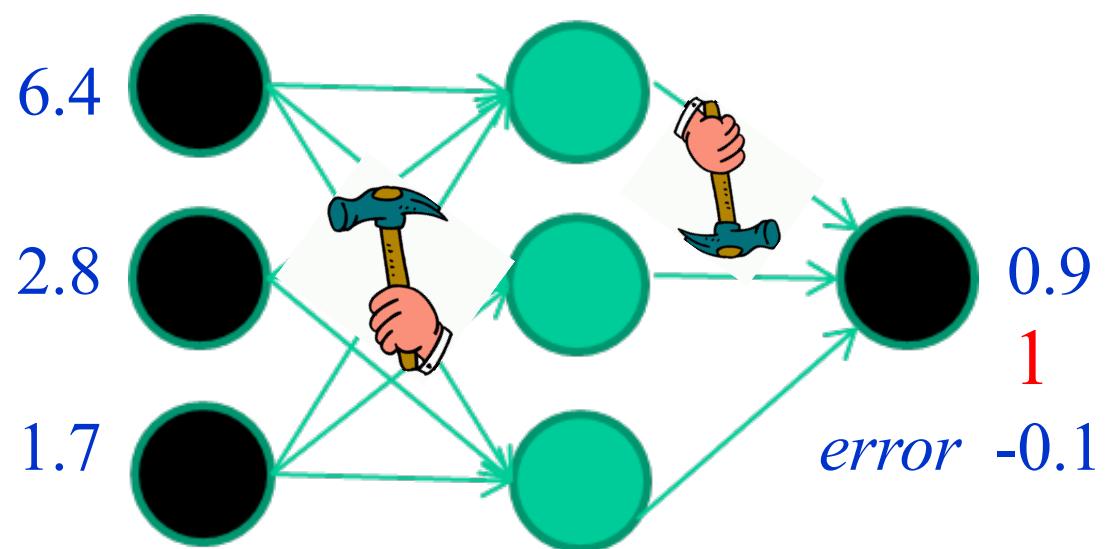
3.8 3.4 3.2 0

6.4 2.8 1.7 1

4.1 0.1 0.2 0

etc ...

Adjust weights based on error

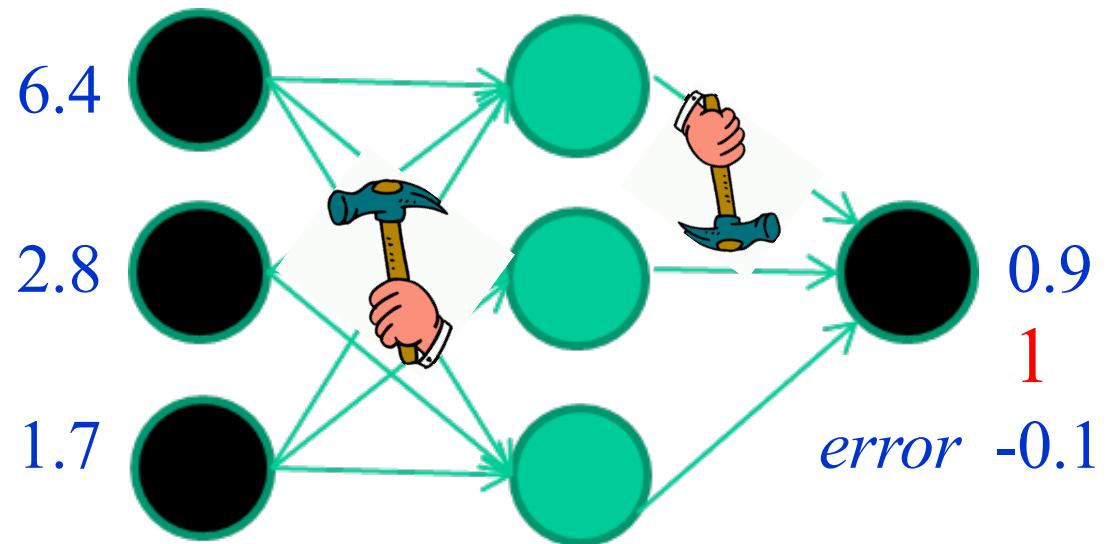


Training data

<i>Fields</i>	<i>class</i>		
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc ...

And so on

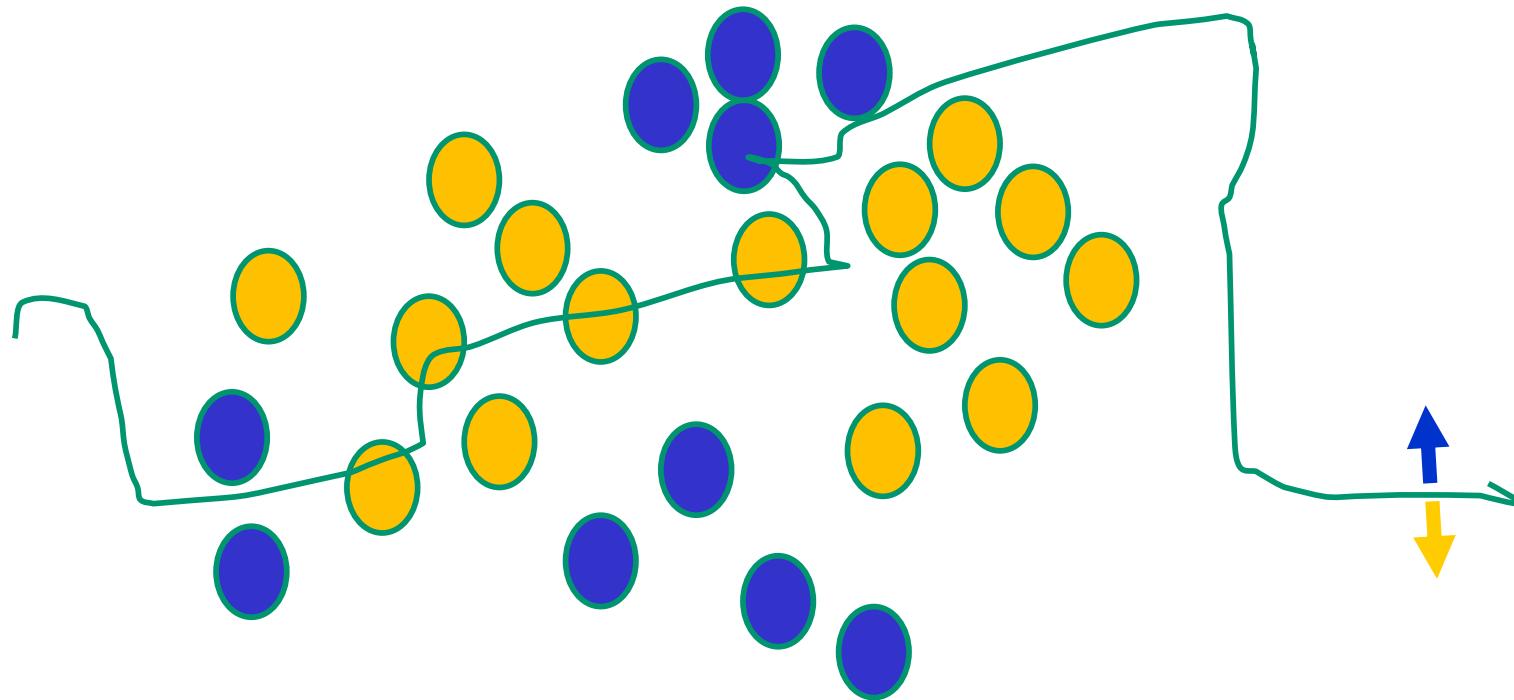


Repeat this thousands, maybe millions of times – each time taking a random training instance, and making slight weight adjustments

Algorithms for weight adjustment are designed to make changes that will reduce the error

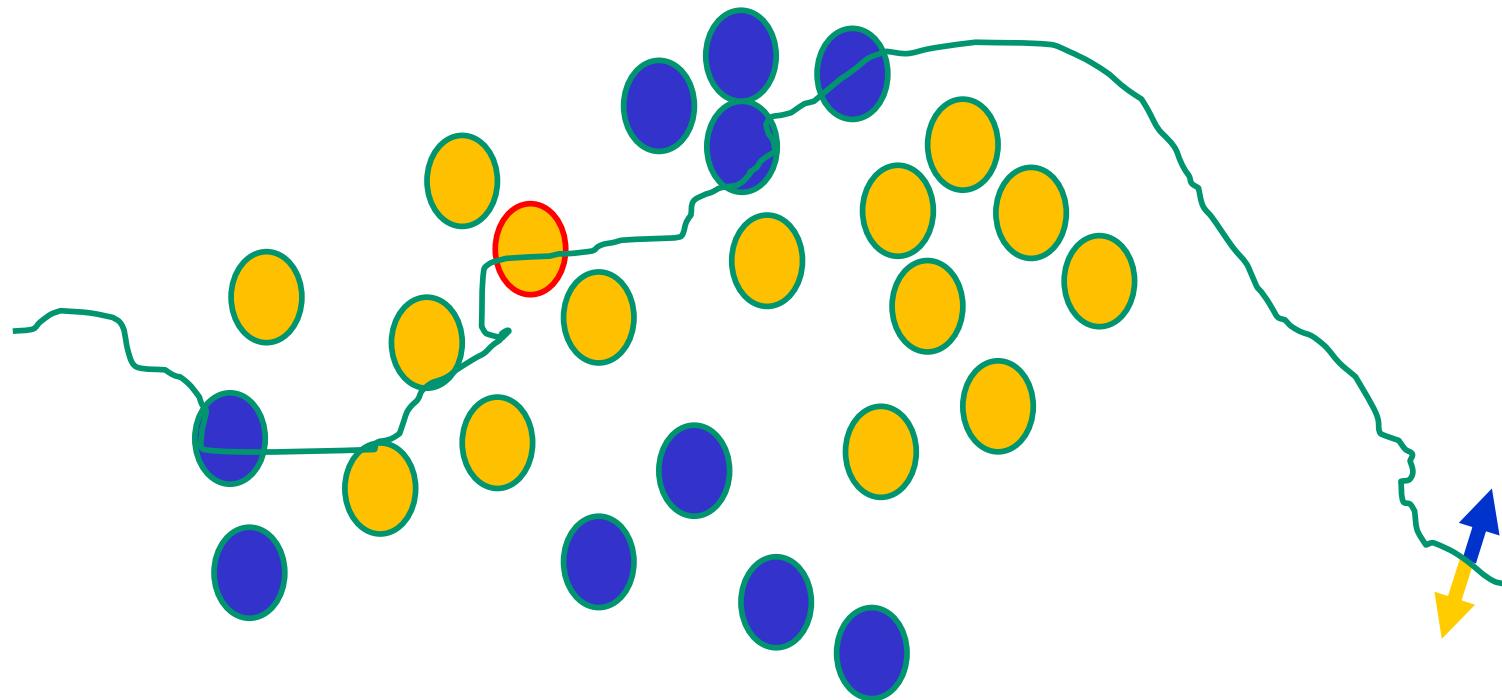
The decision boundary perspective...

Initial random weights



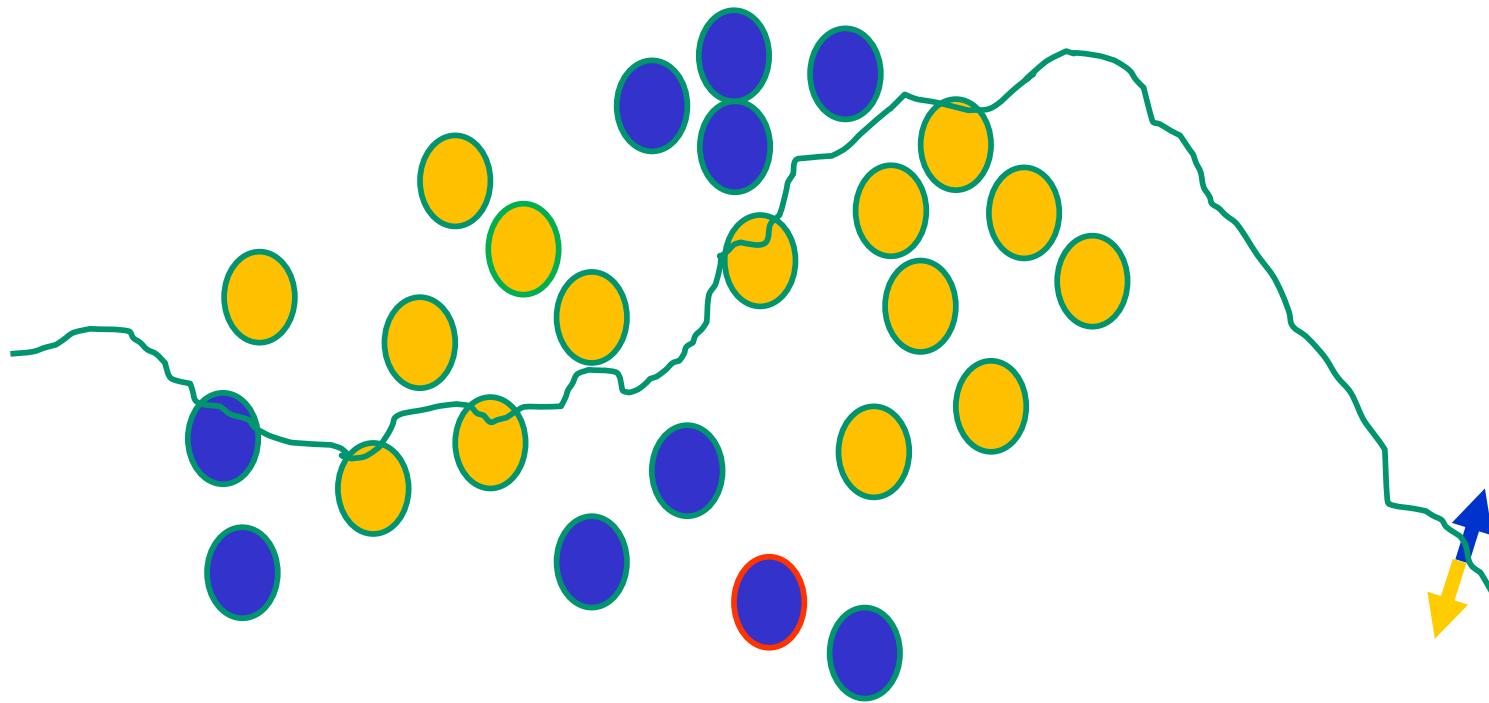
The decision boundary perspective...

Present a training instance / adjust the weights



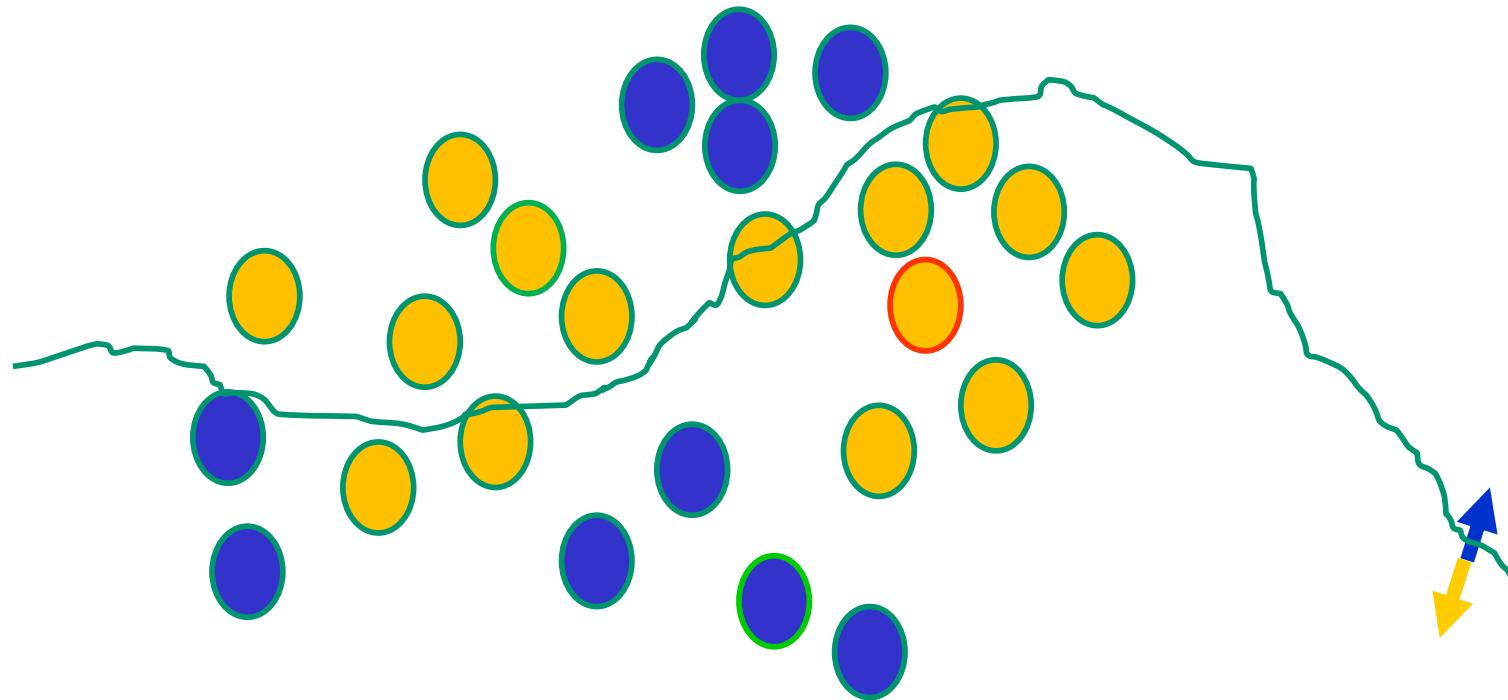
The decision boundary perspective...

Present a training instance / adjust the weights



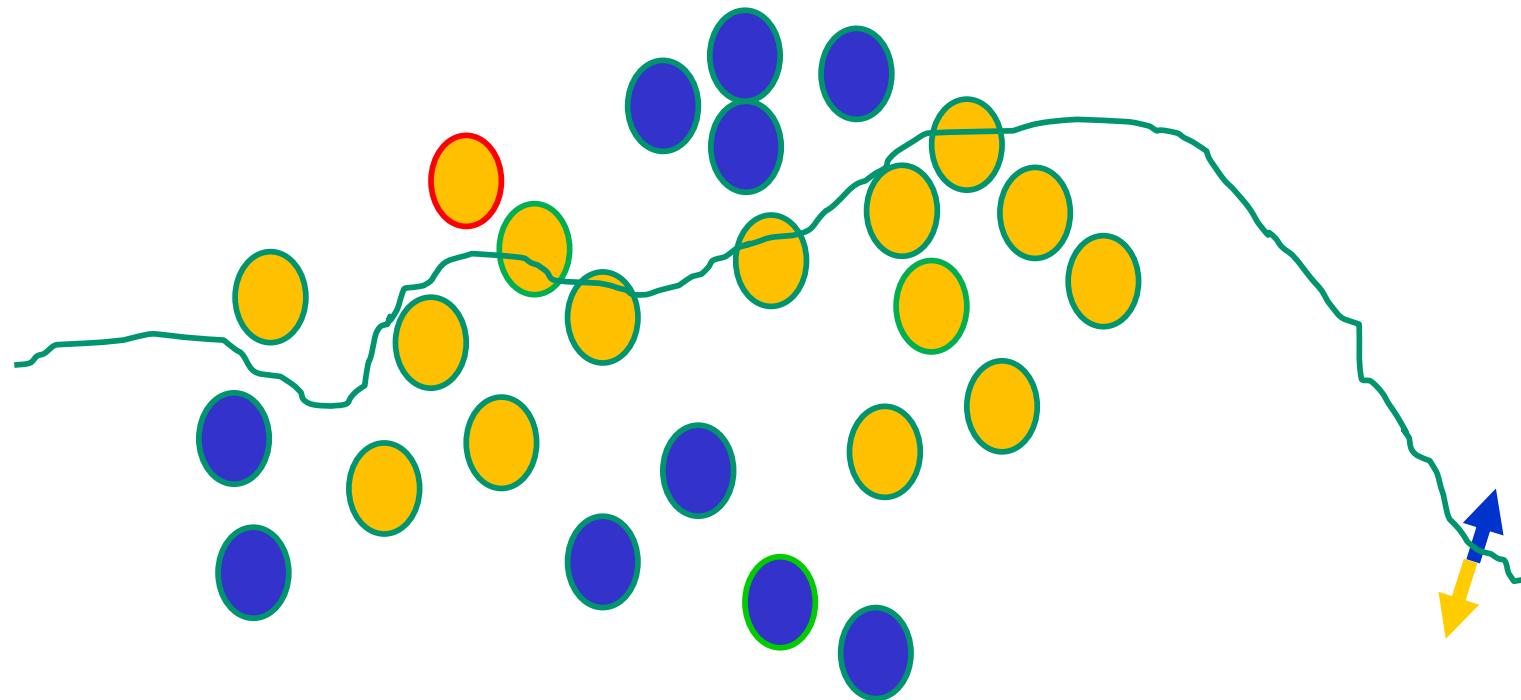
The decision boundary perspective...

Present a training instance / adjust the weights



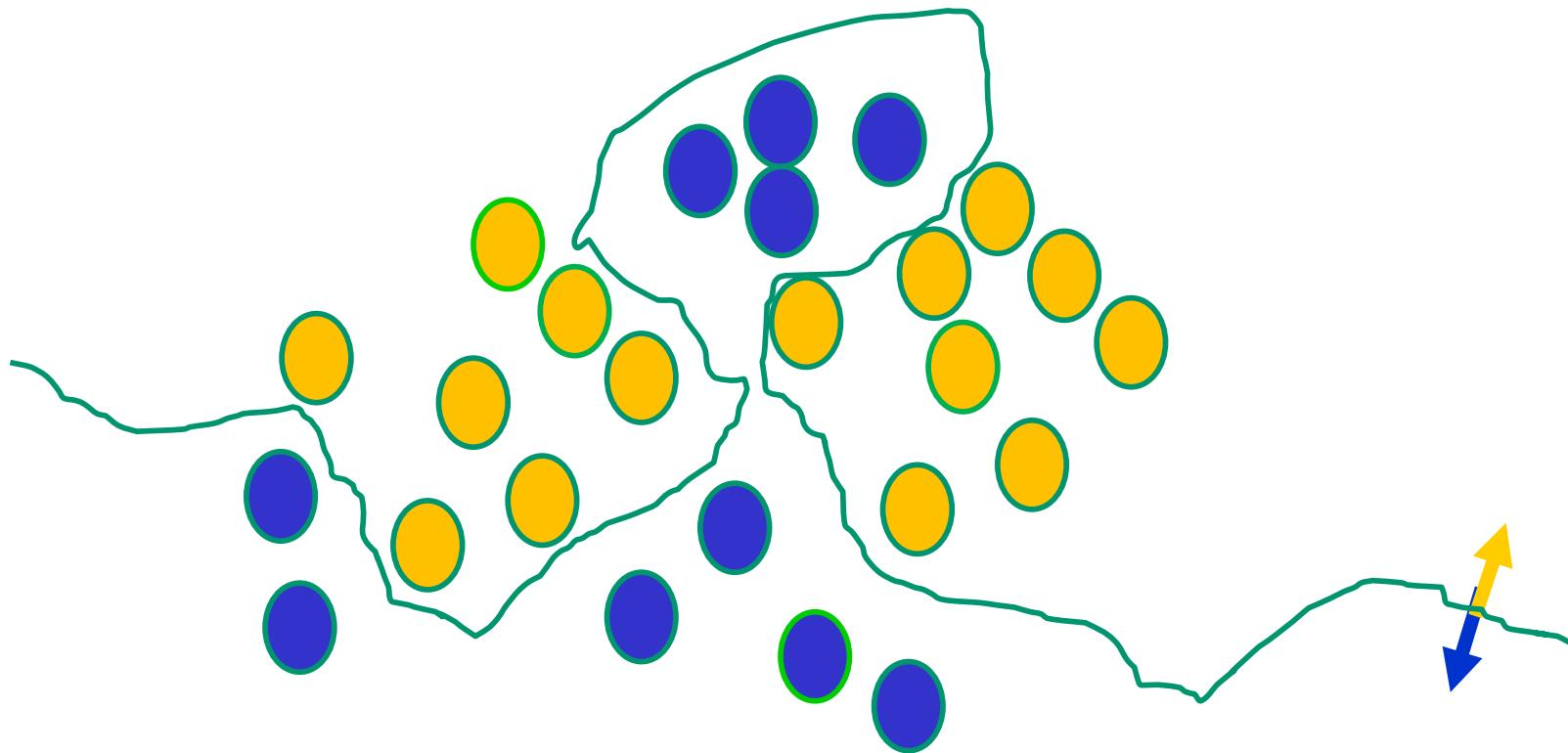
The decision boundary perspective...

Present a training instance / adjust the weights



The decision boundary perspective...

Eventually



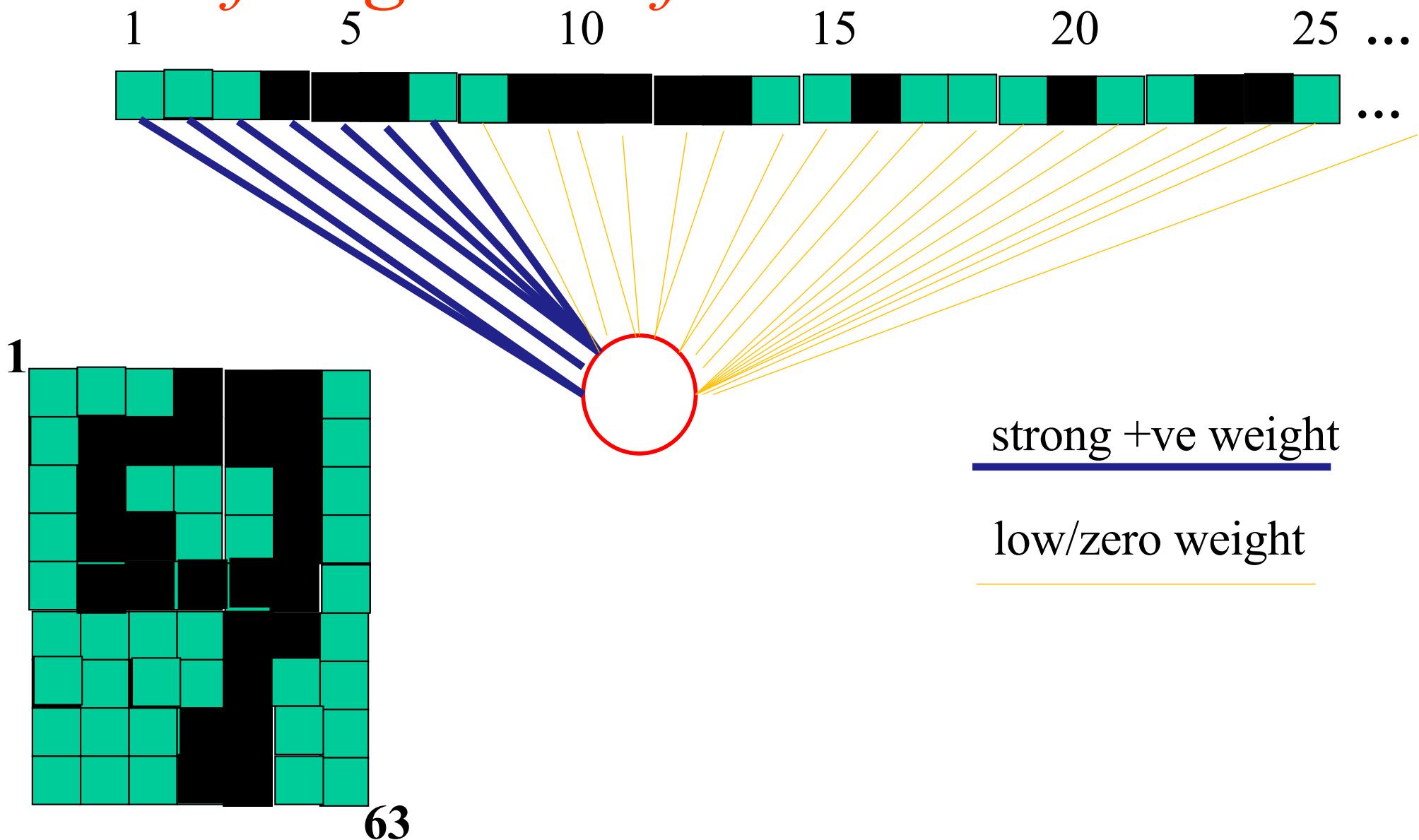
Summary

- weight-learning algorithms for NNs are dumb
- they work by making thousands and thousands of tiny adjustments, each making the network do better at the most recent pattern, but perhaps a little worse on many others
- but, by dumb luck, eventually this tends to be good enough to learn effective classifiers for many real applications
- If $f(x)$ is non-linear, a network with 1 hidden layer can, in theory, learn perfectly any classification problem. A set of weights exists that can produce the targets from the inputs. The problem is finding them.

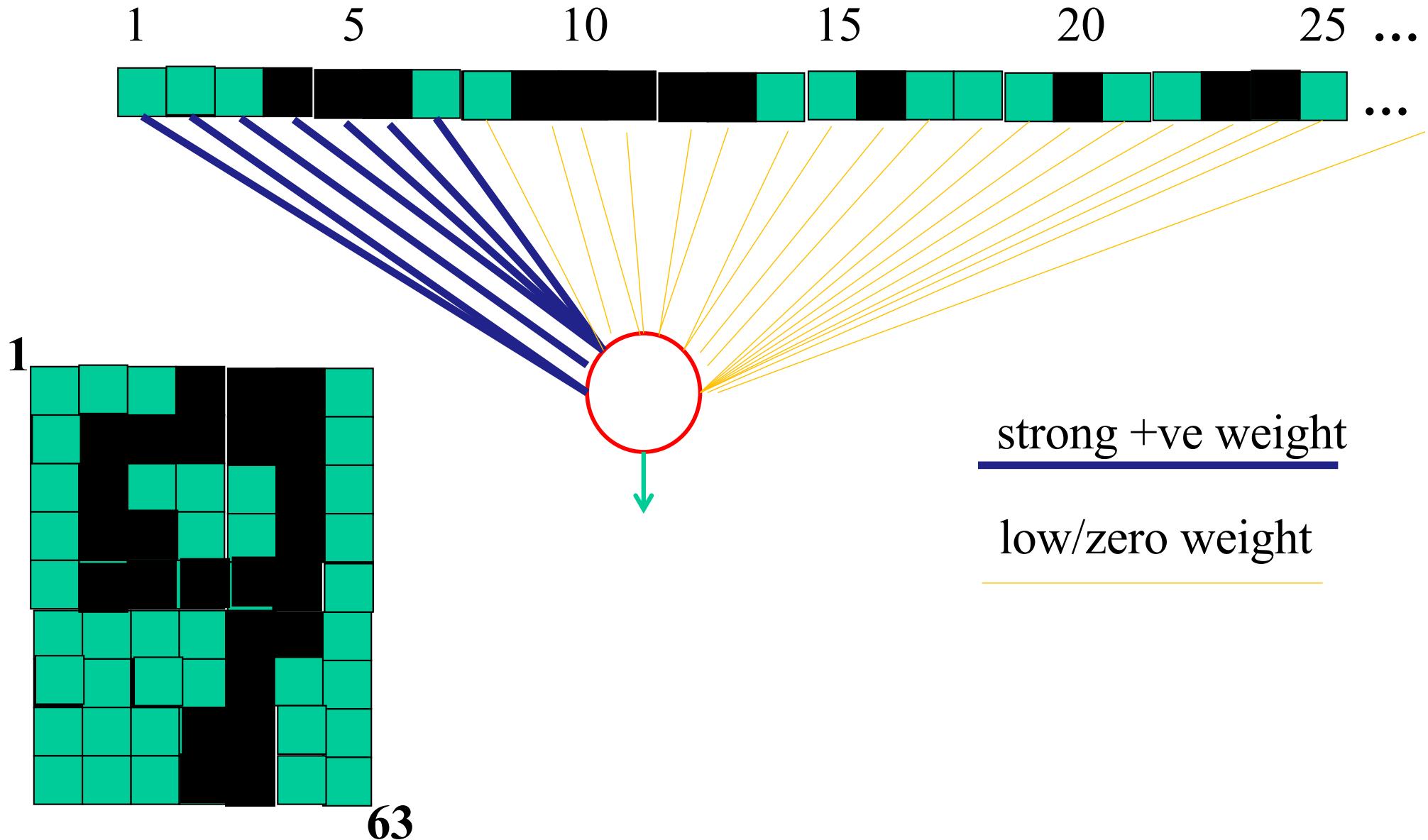
LeNet -5

- Digit recognition on MNIST dataset
- Famous deep learning system
- <http://yann.lecun.com/exdb/lenet/>

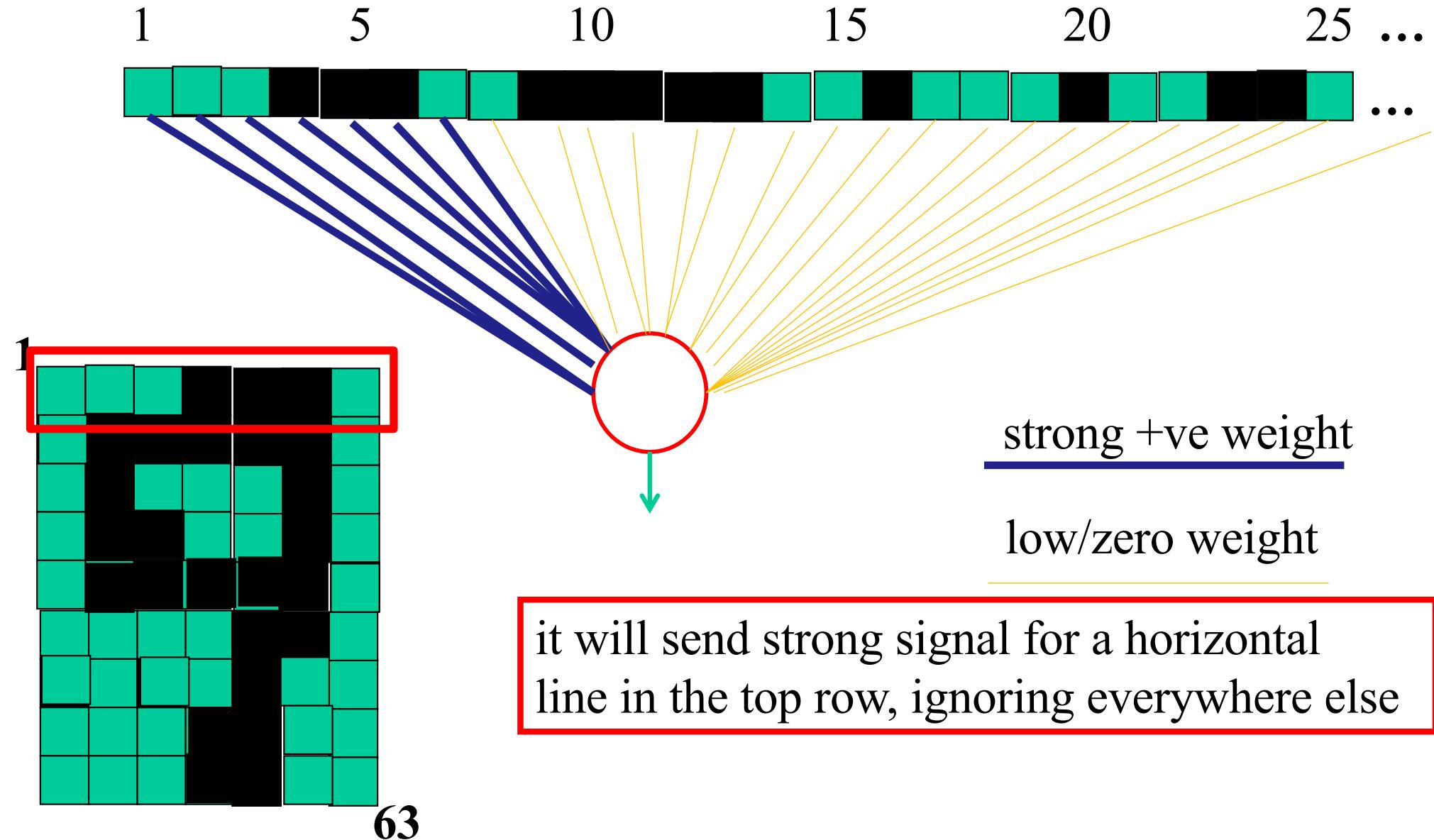
Hidden layer units become *self-organised feature detectors*



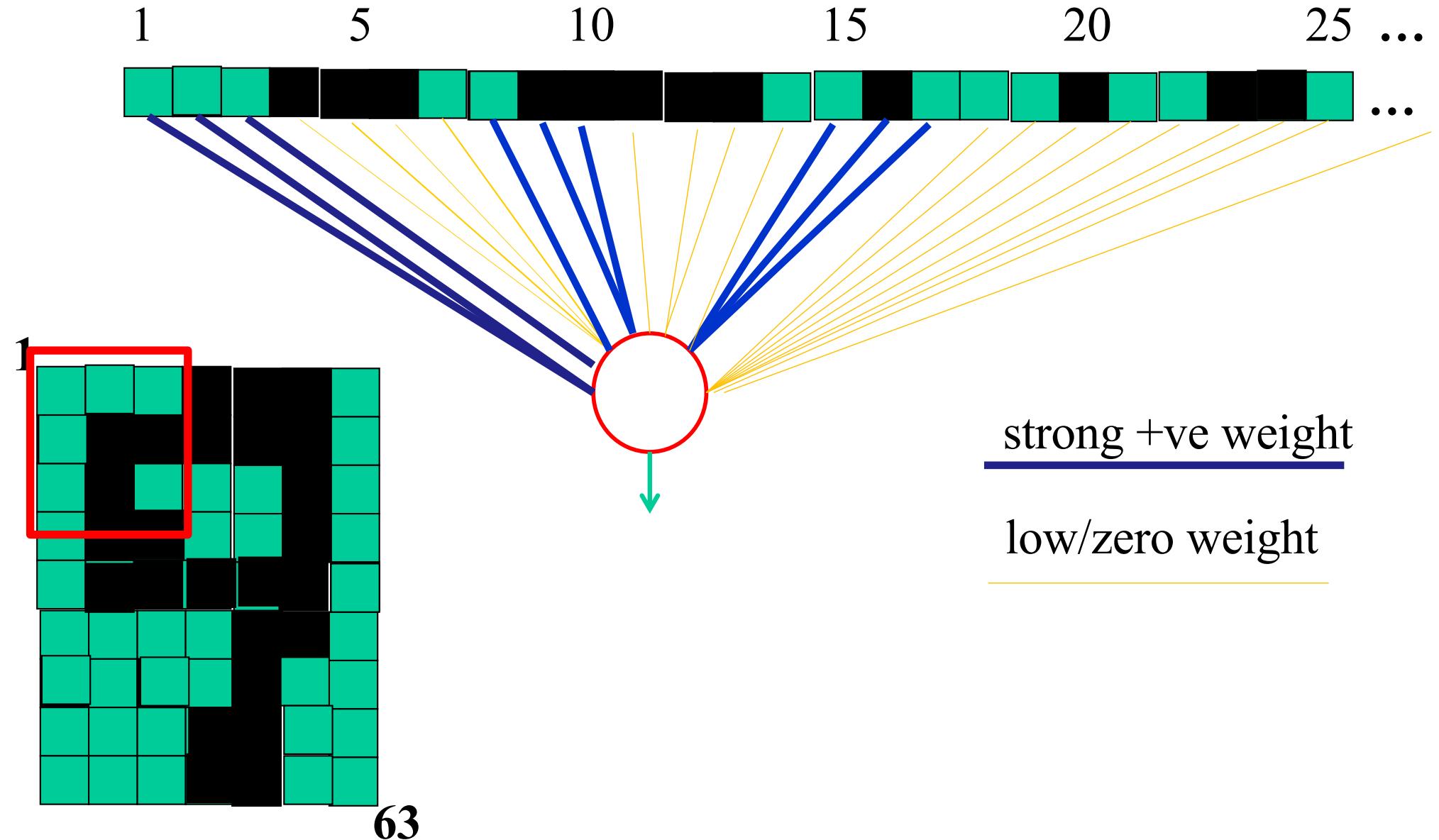
What does this unit detect?



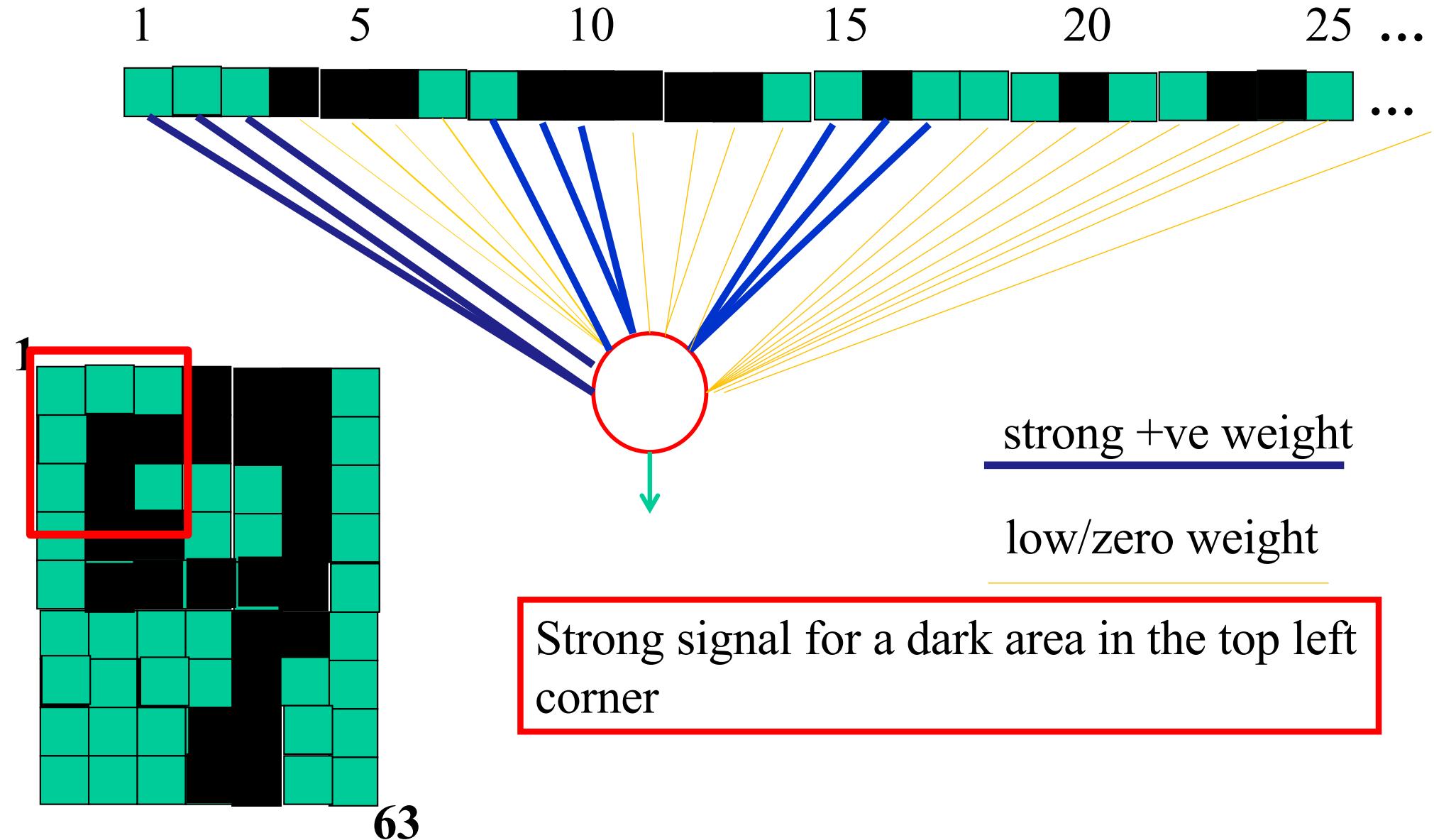
What does this unit detect?



What does this unit detect?



What does this unit detect?



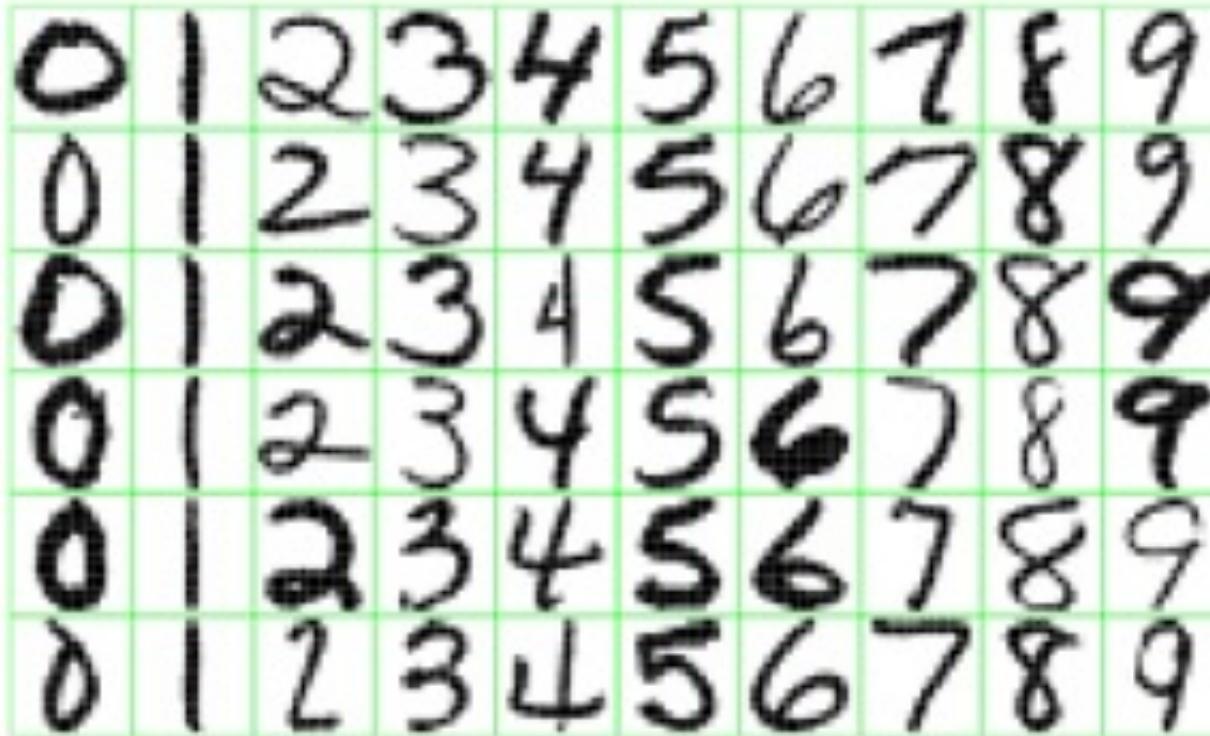


Figure 1.2: Examples of handwritten digits from U.S. postal envelopes.

What features might you expect a good NN to learn, when trained with data like this?

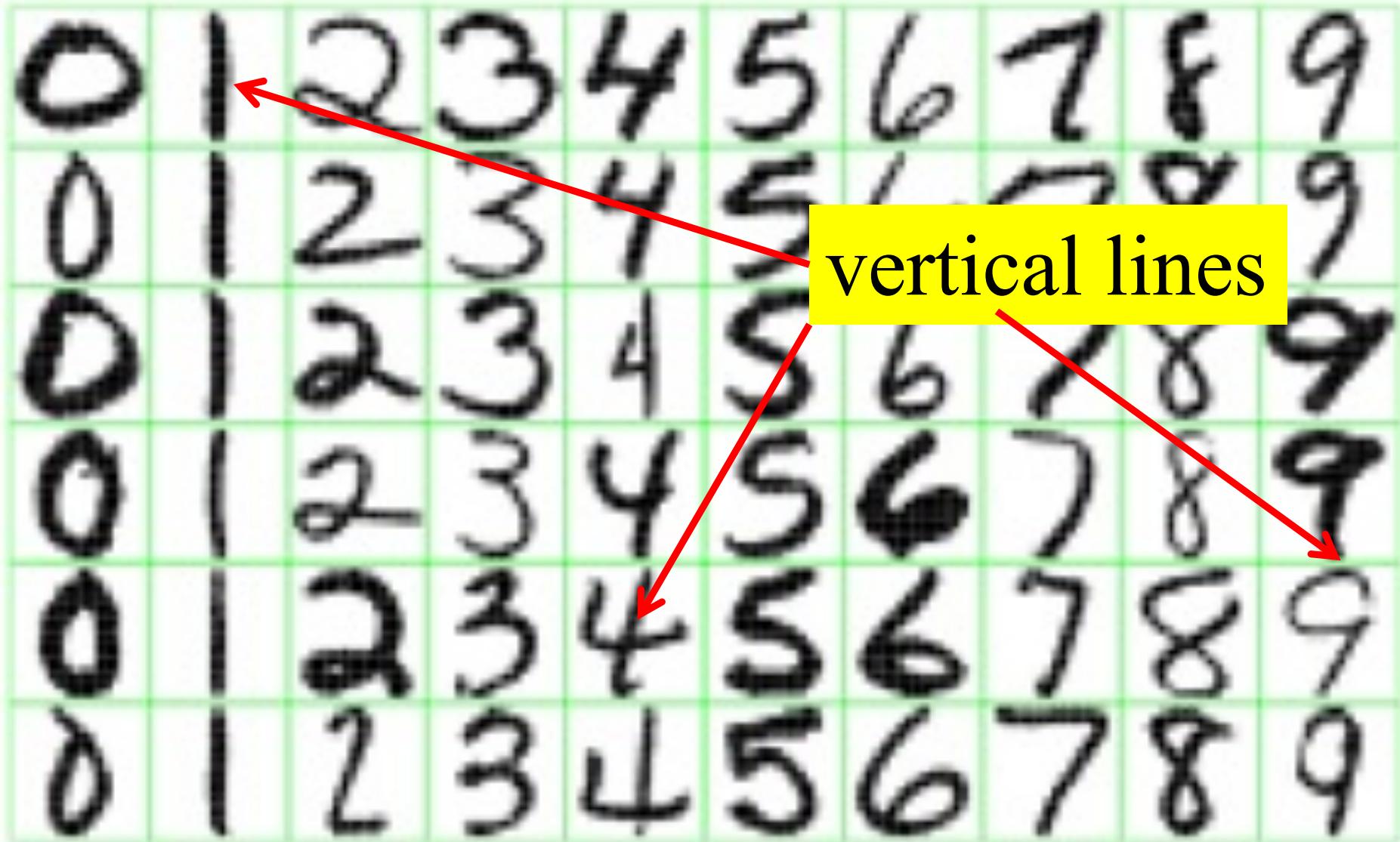


Figure 1.2: Examples of handwritten digits from U.S. postal envelopes.



Figure 1.2: Examples of handwritten digits from U.S. postal envelopes.

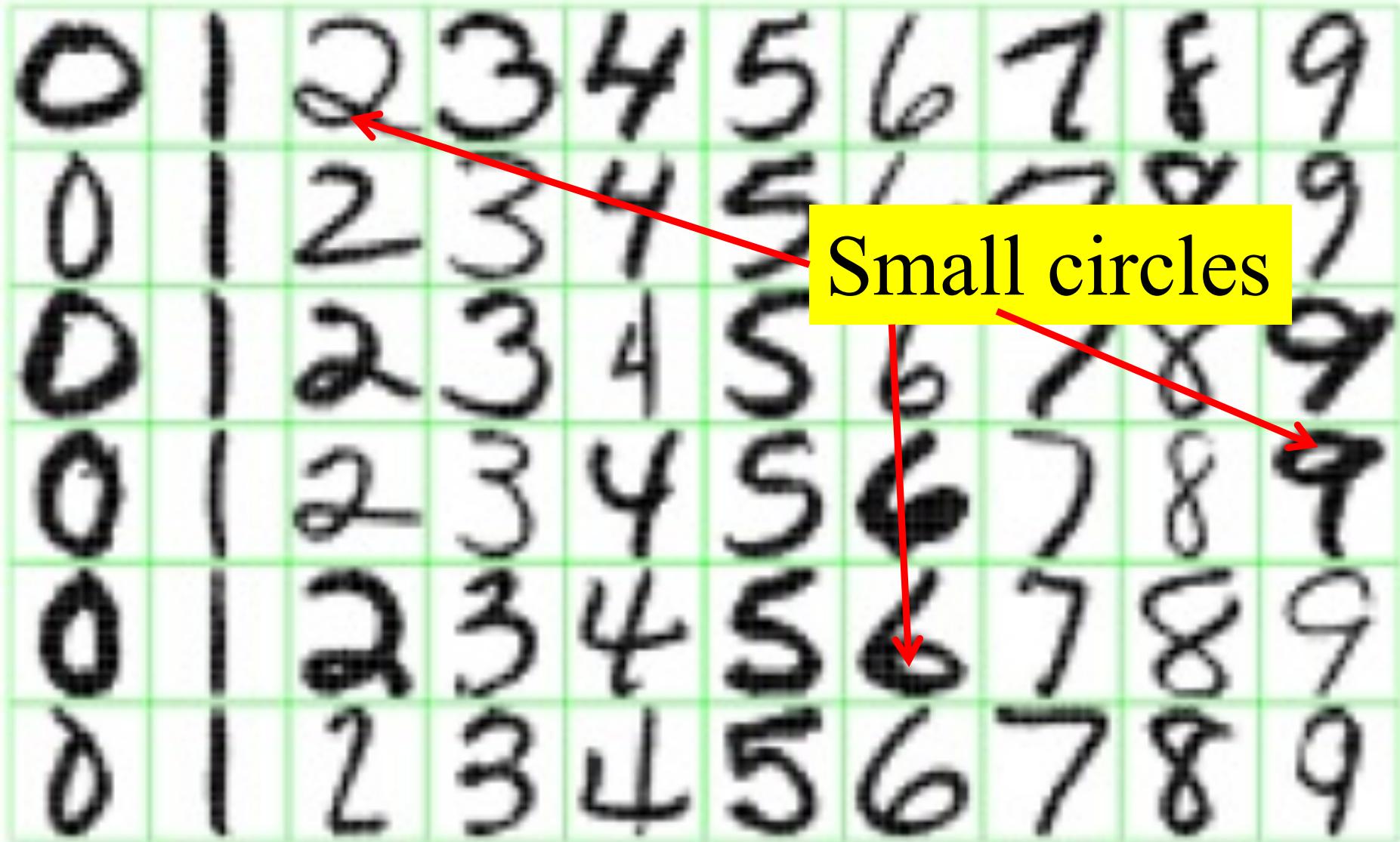


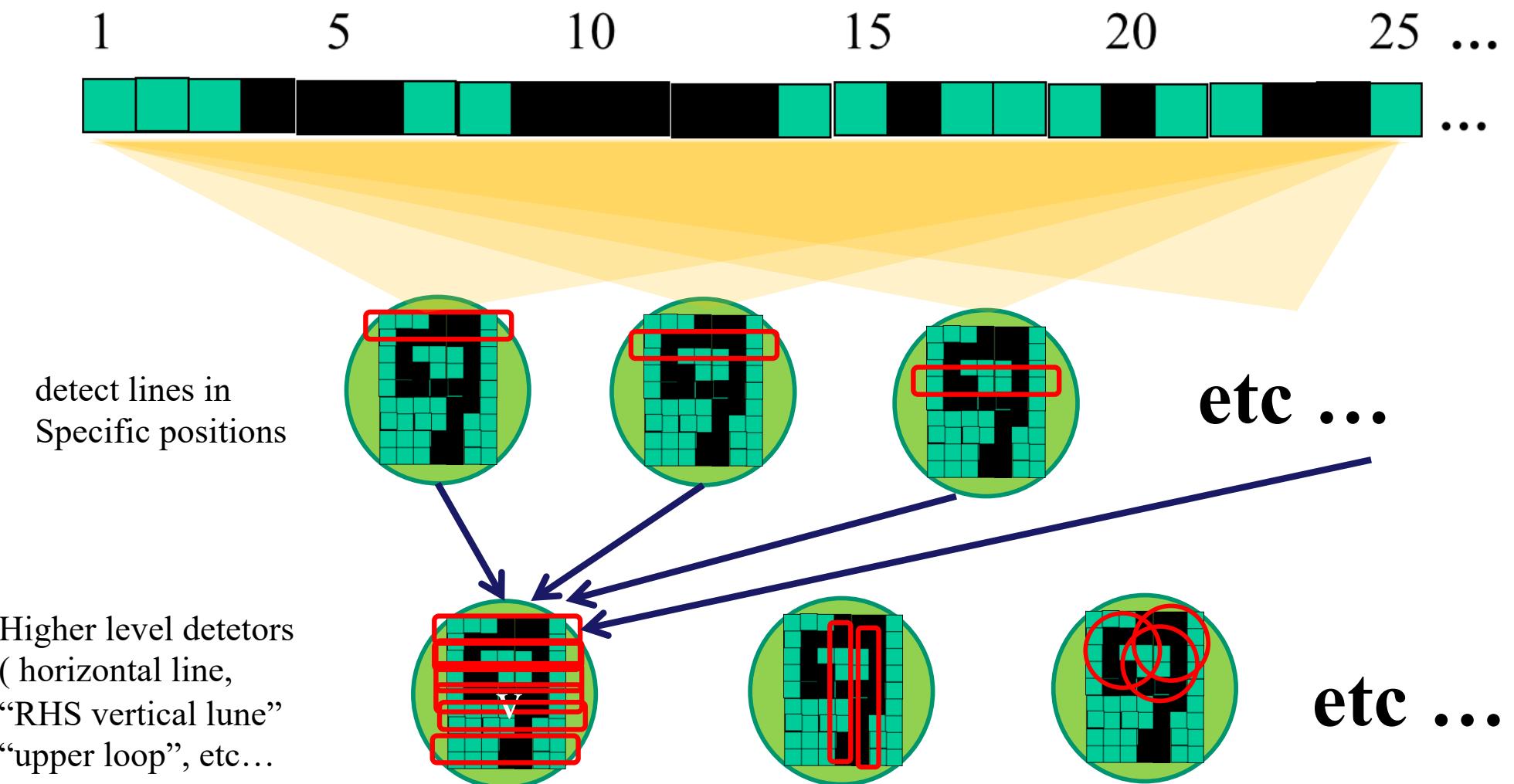
Figure 1.2: Examples of handwritten digits from U.S. postal envelopes.

1



But what about position invariance ???
our example unit detectors were tied to
specific parts of the image

successive layers can learn higher-level features ...



successive layers can learn higher-level features ...

1

5

10

15

20

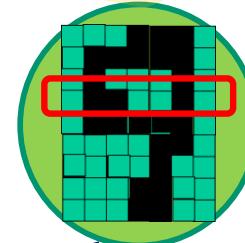
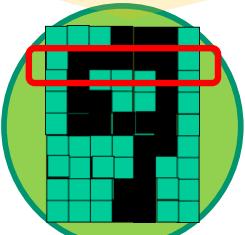
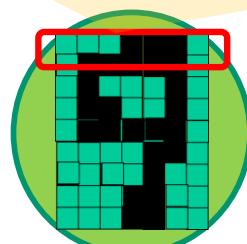
25

...



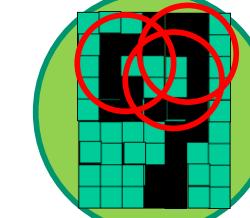
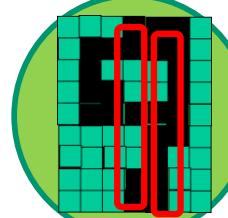
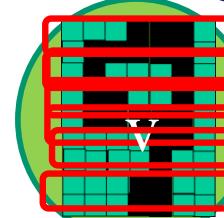
...

detect lines in
Specific positions



etc ...

Higher level detectors
(horizontal line,
“RHS vertical lune”
“upper loop”, etc...)

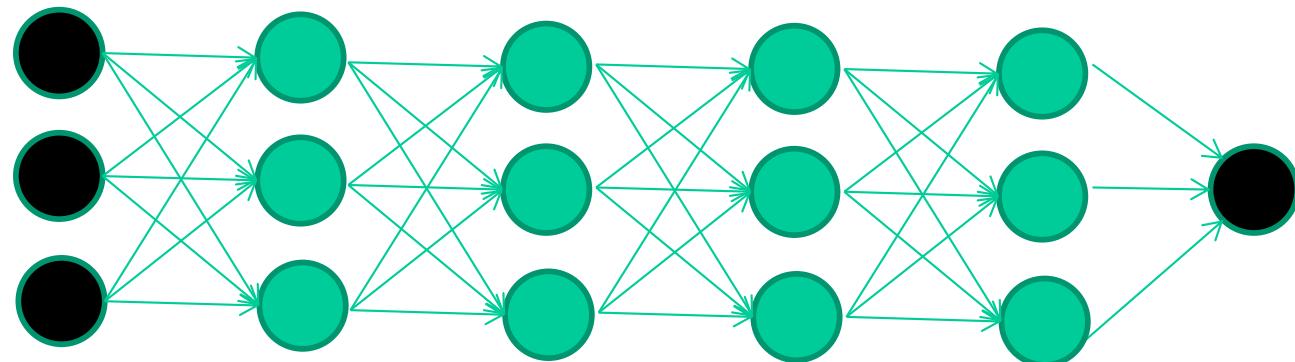


etc ...

What does this unit detect?

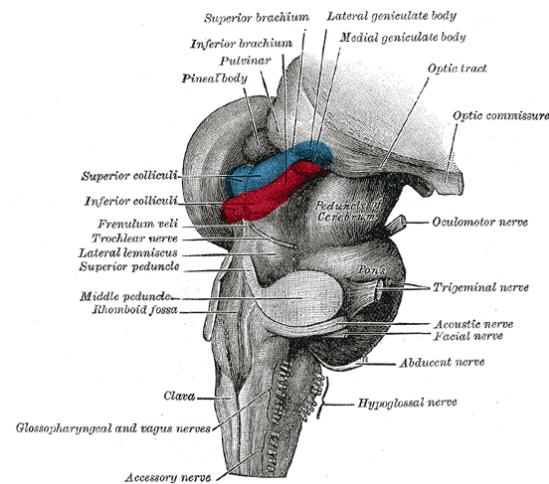
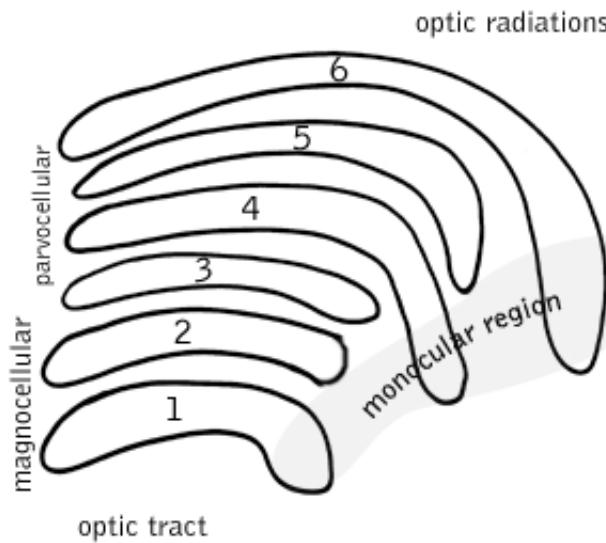


So: multiple layers make sense



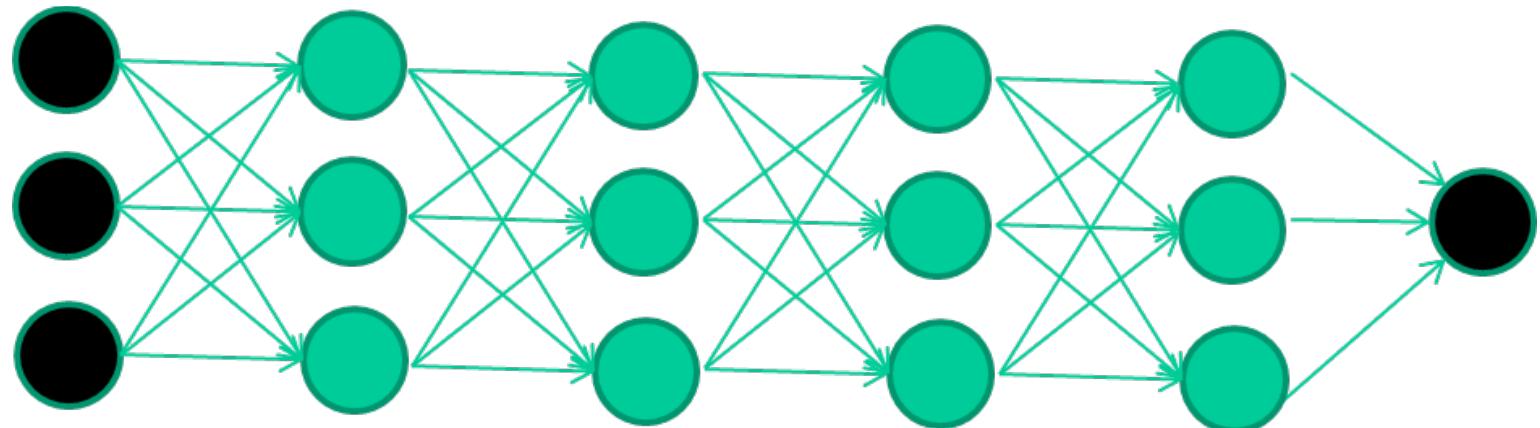
So: multiple layers make sense

Your brain works that way

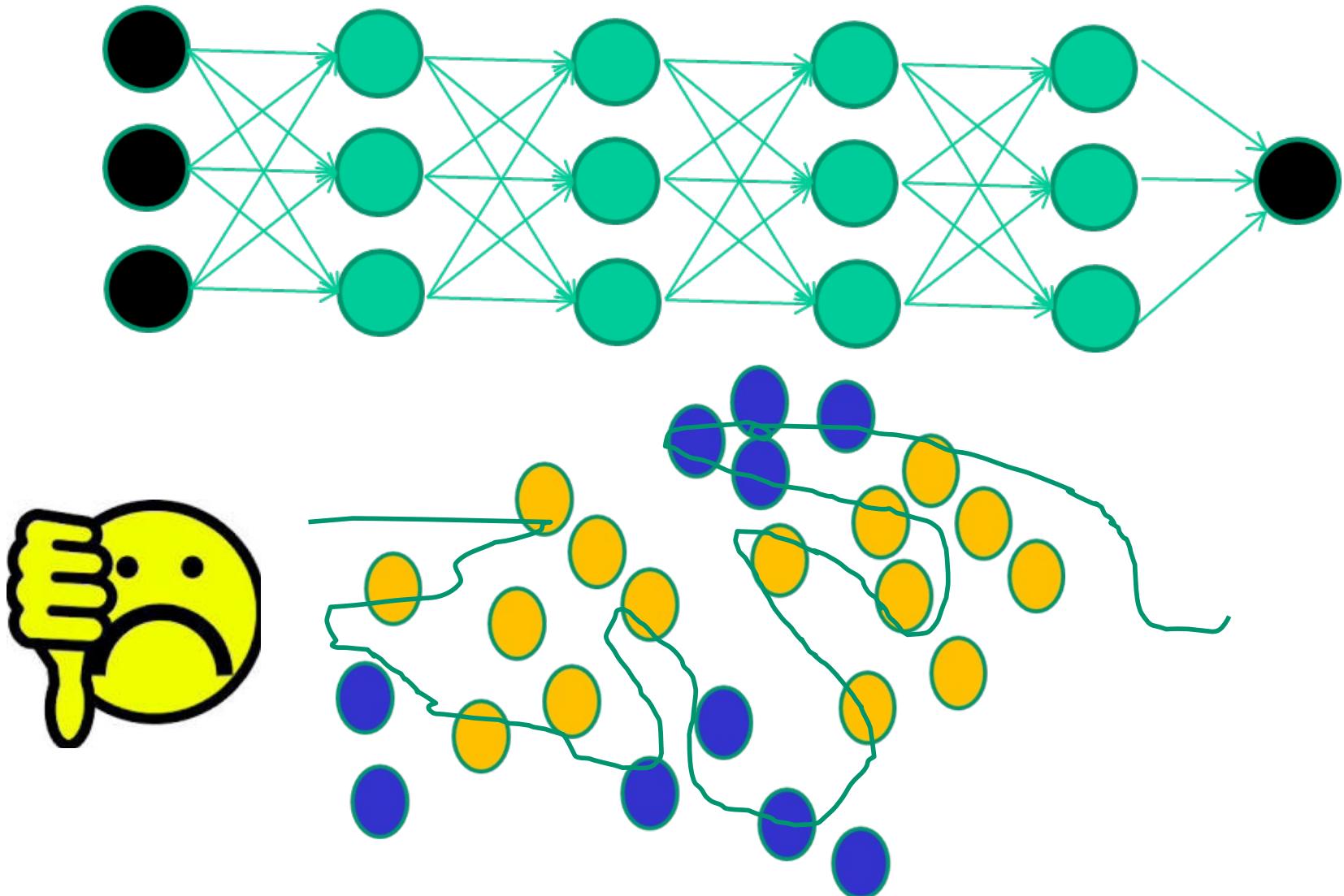


So: multiple layers make sense

Many-layer neural network architectures should be capable of learning the true underlying features and ‘feature logic’, and therefore generalise very well ...

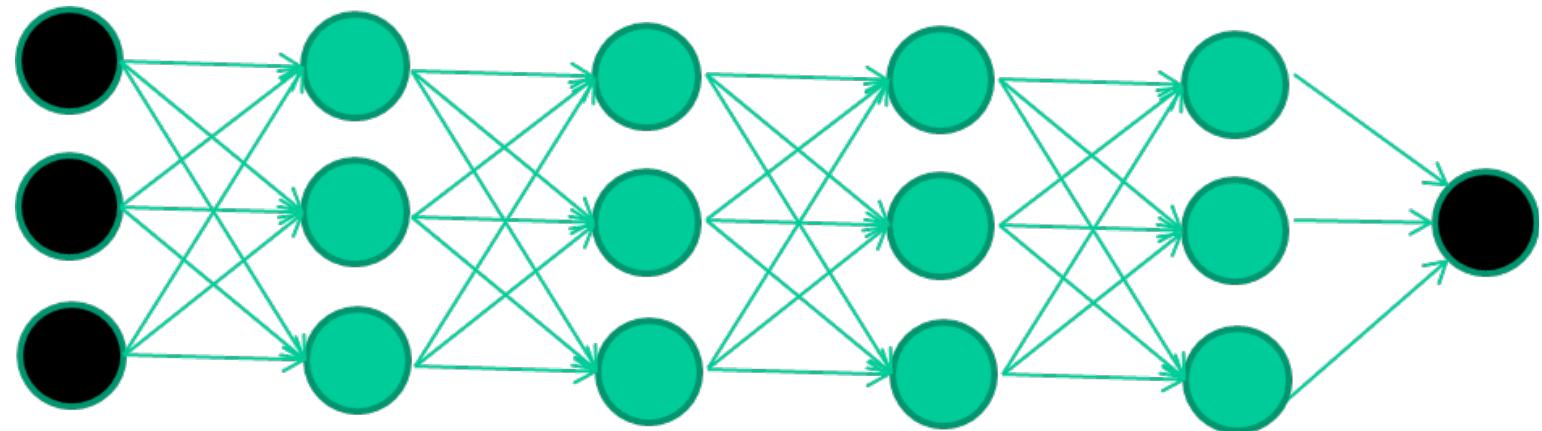


But, until very recently, our weight-learning algorithms simply did not work on multi-layer architectures

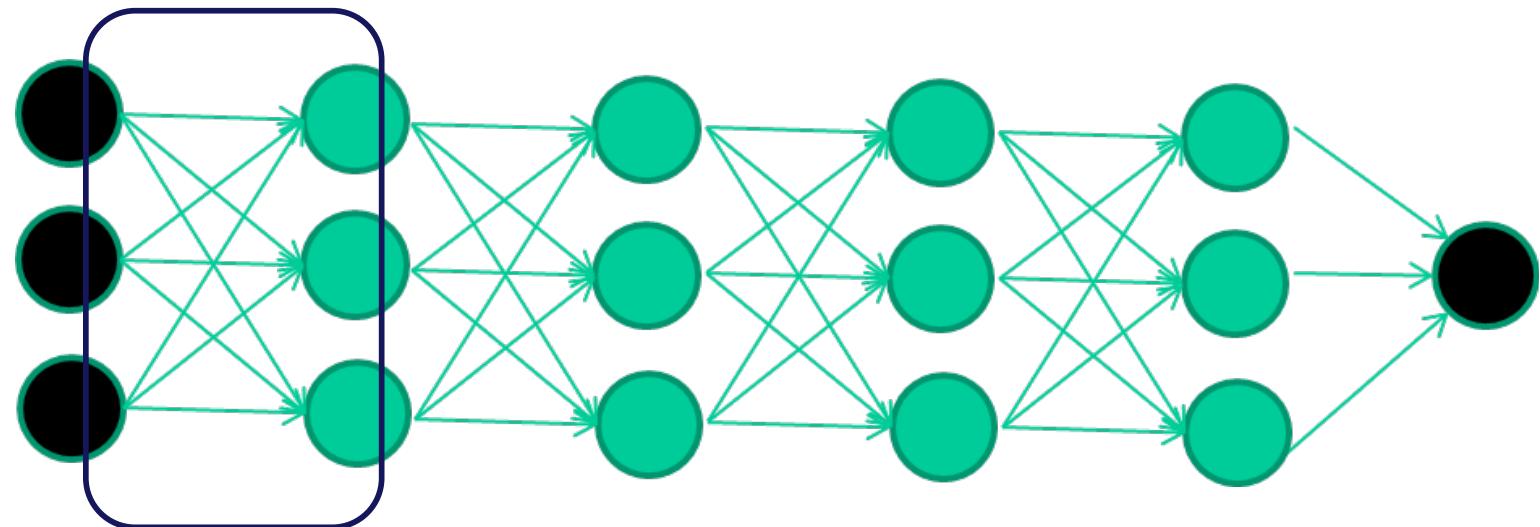


Along came deep learning ...

The new way to train multi-layer NNs...

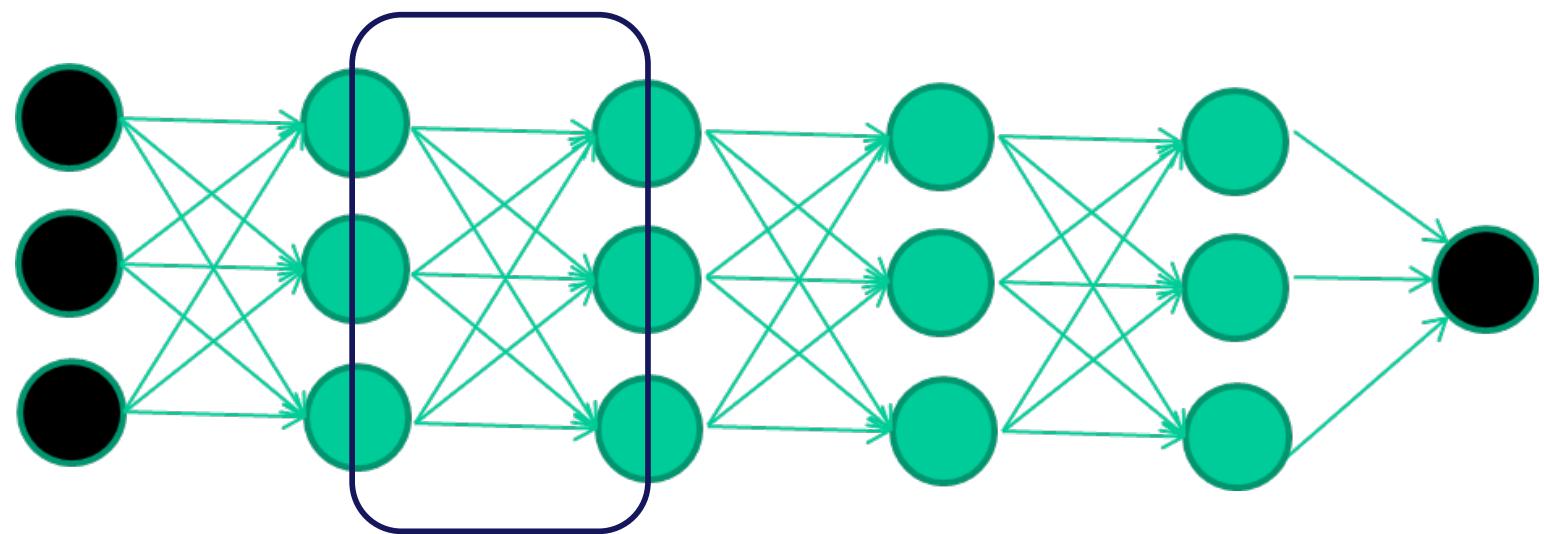


The new way to train multi-layer NNs...



Train **this** layer first

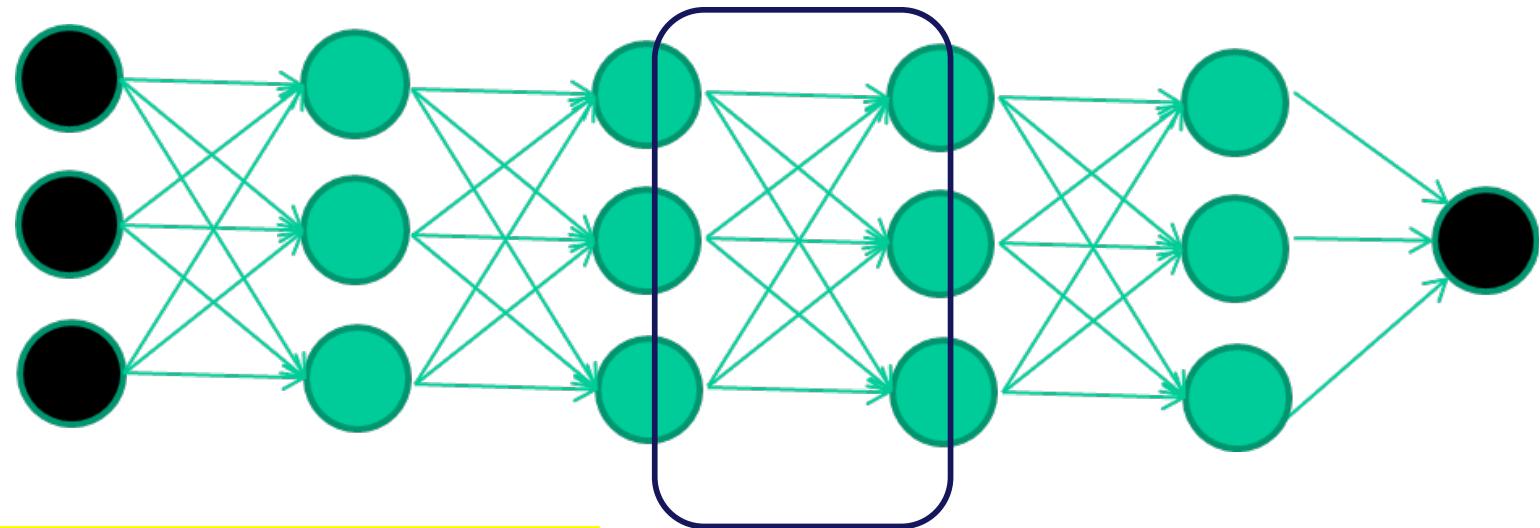
The new way to train multi-layer NNs...



Train **this** layer first

then **this** layer

The new way to train multi-layer NNs...

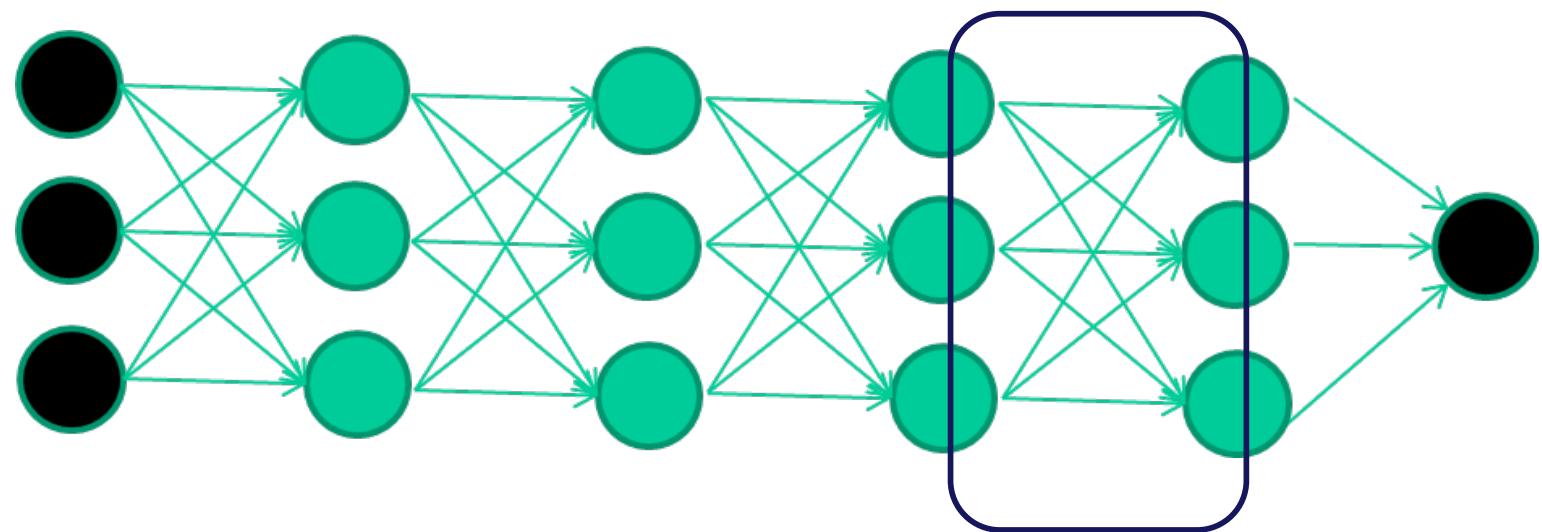


Train **this** layer first

then **this** layer

then **this** layer

The new way to train multi-layer NNs...



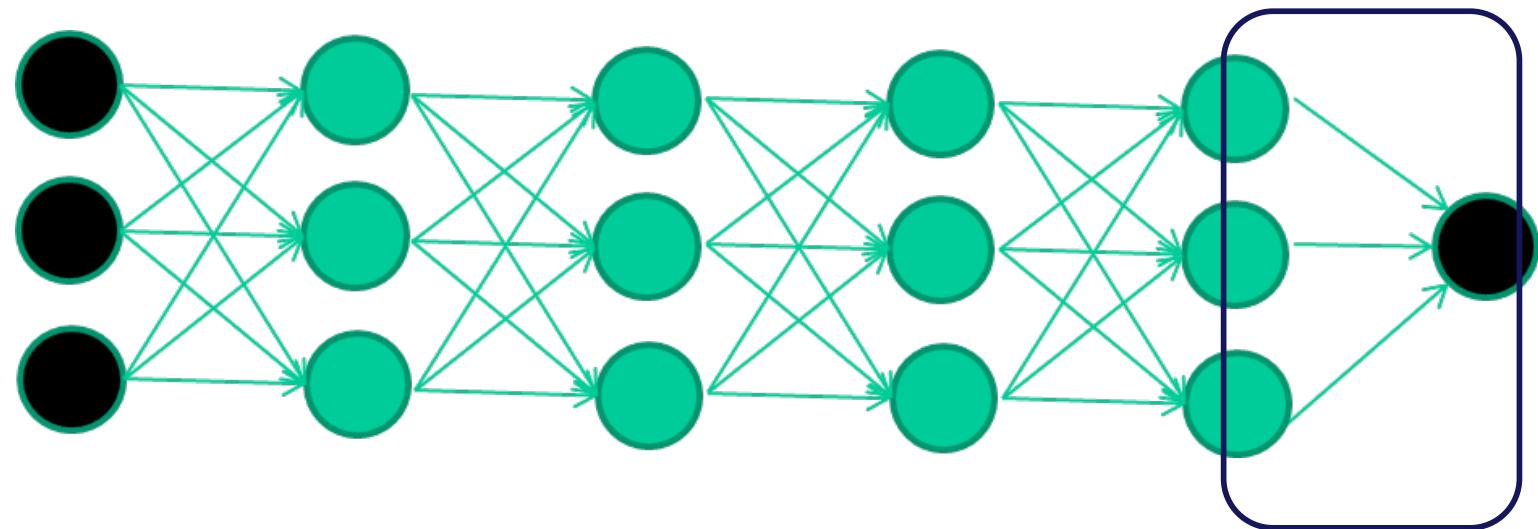
Train **this** layer first

then **this** layer

then **this** layer

then **this** layer

The new way to train multi-layer NNs...



Train **this** layer first

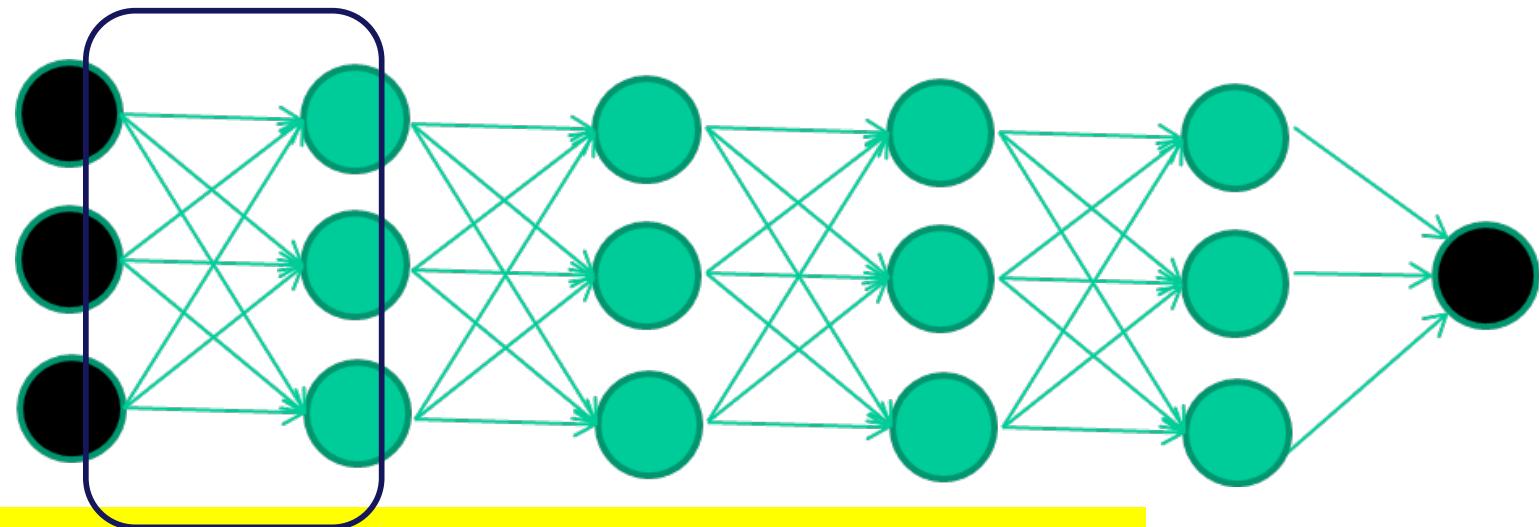
then **this** layer

then **this** layer

then **this** layer

finally **this** layer

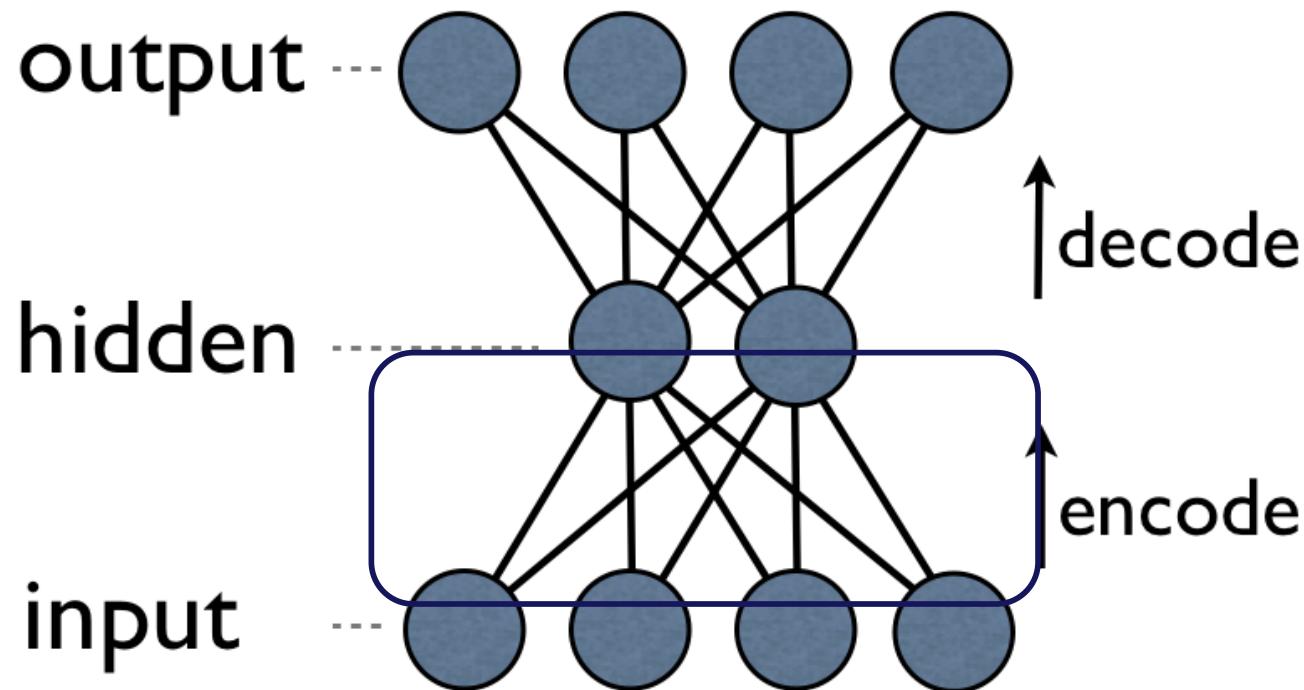
The new way to train multi-layer NNs...



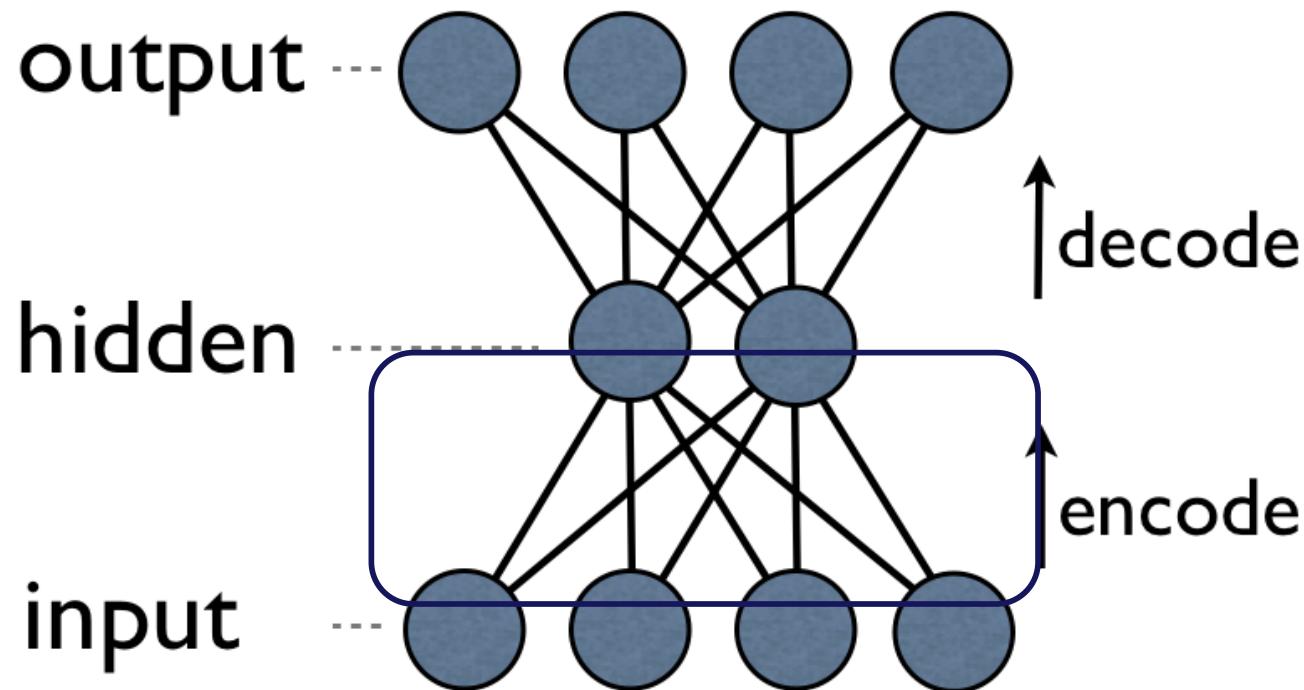
*EACH of the (non-output) layers is
trained to be an **auto-encoder***

*Basically, it is forced to learn good
features that describe what comes from
the previous layer*

an auto-encoder is trained, with an absolutely standard weight-adjustment algorithm to reproduce the input

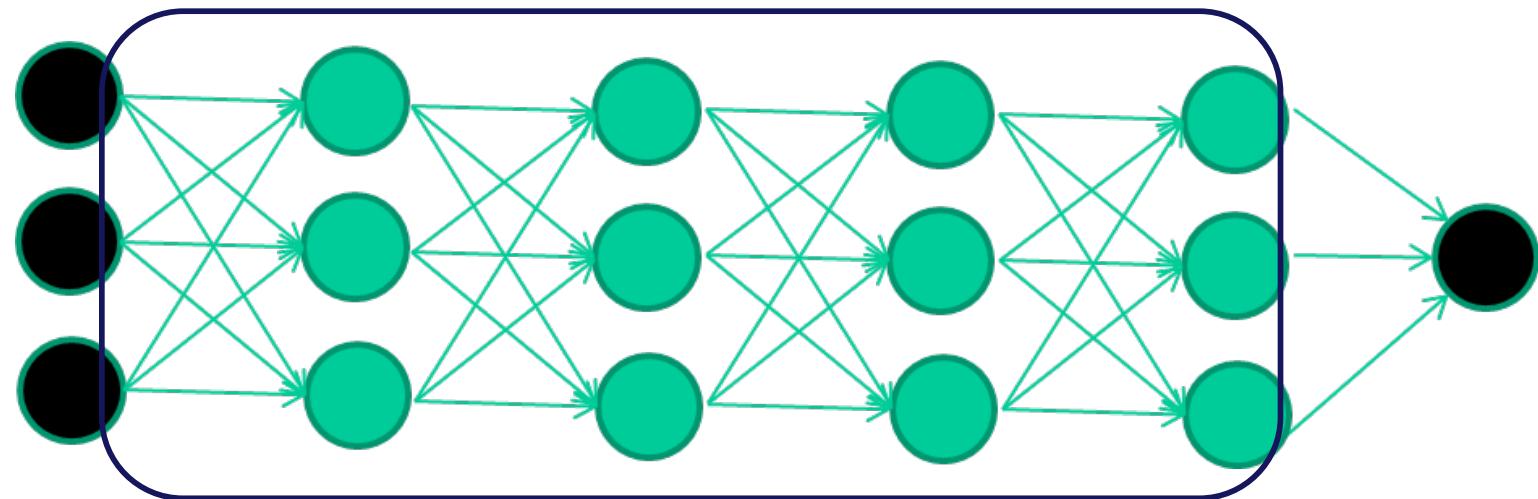


an auto-encoder is trained, with an absolutely standard weight-adjustment algorithm to reproduce the input

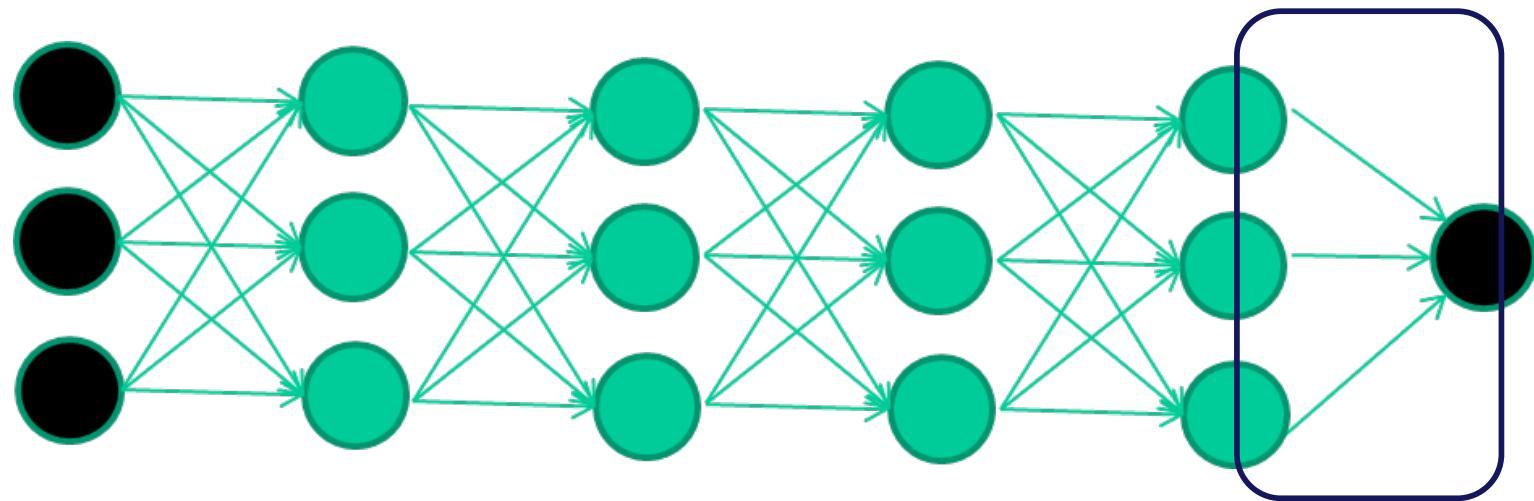


By making this happen with (many) fewer units than the inputs, this forces the ‘hidden layer’ units to become good feature detectors

intermediate layers are each trained to be
auto encoders (or similar)



Final layer trained to predict class based
on outputs from previous layers



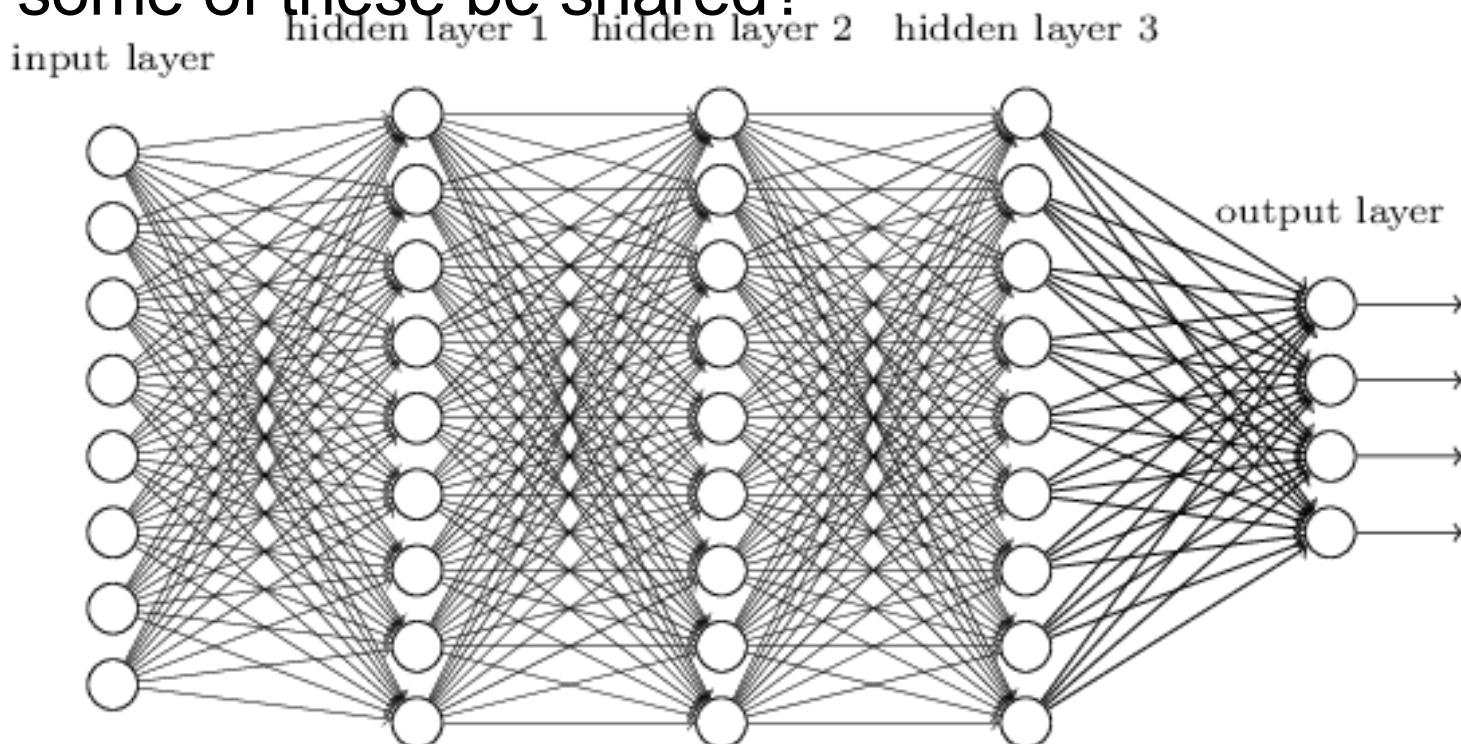
Why does deep learning work?

Back propagation was problematic on deeper networks due to the larger number of parameters and vanishing gradient.

- More computation (GPU) and bigger datasets
- Reduction in number of parameters from fully connected to convolutional
- Training intermediate layers to autoencode
- Solutions to vanishing/exploding gradient
 - Gated functions (LSTM)
 - Rectified linear units (ReLU): $\max(0, x)$
 - Residual networks: pass inputs directly to later layers

Convolutional Neural Networks

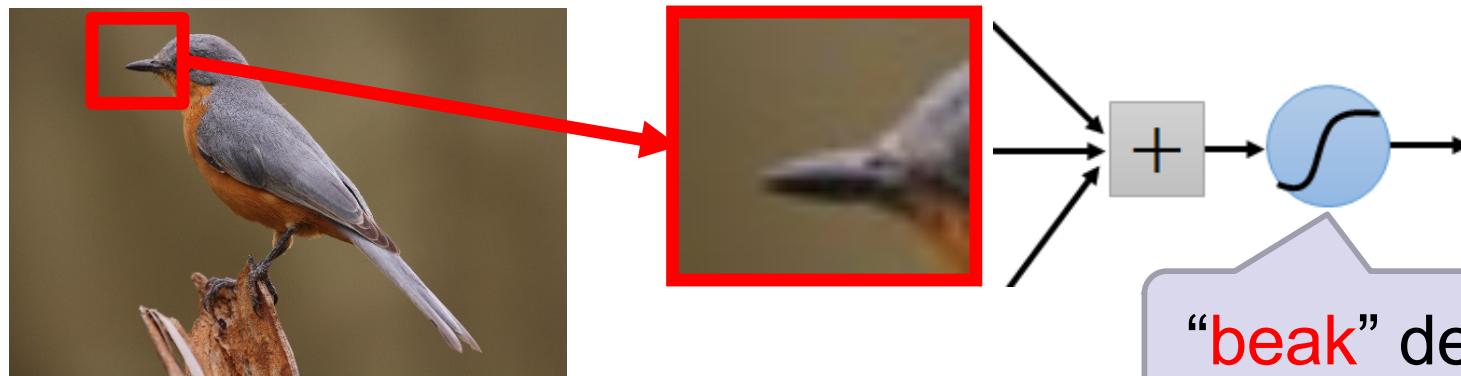
- This is a NN architecture that has performed well on images.
- From this fully connected model, do we really need all the edges?
- Can some of these be shared?



Consider learning an image:

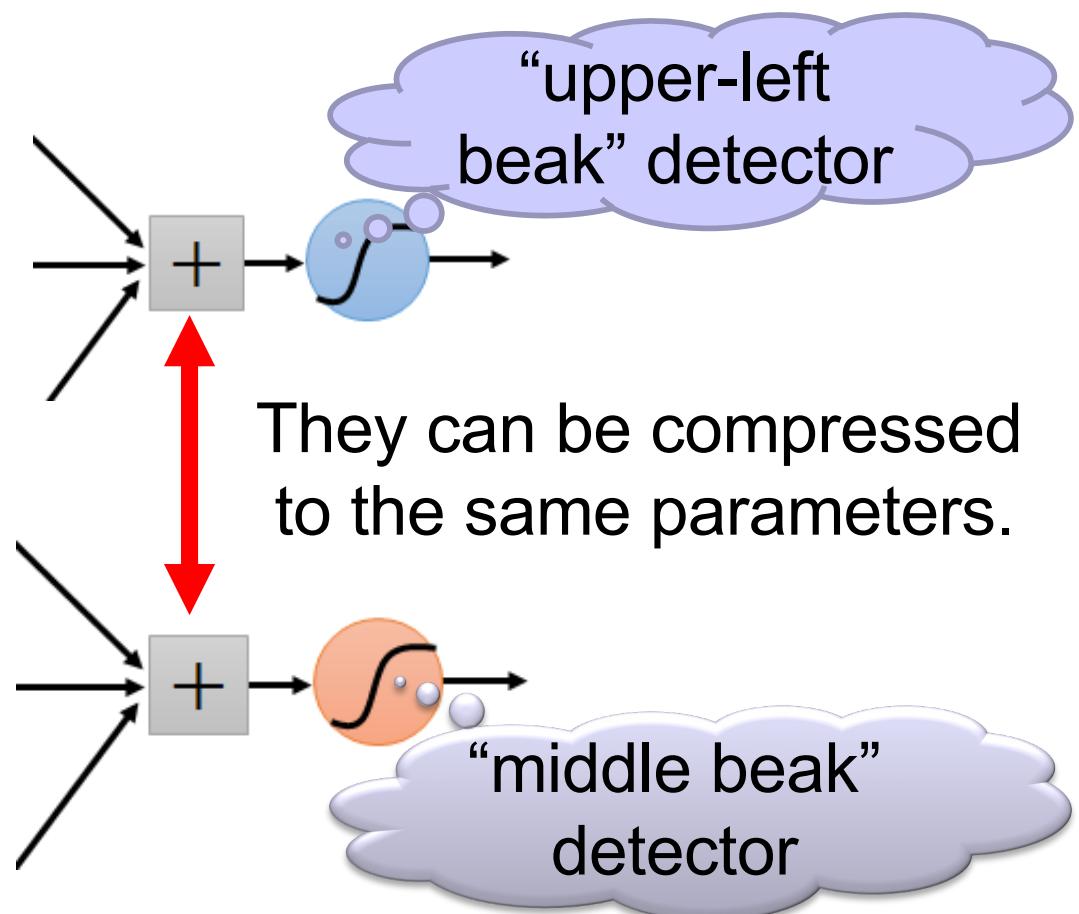
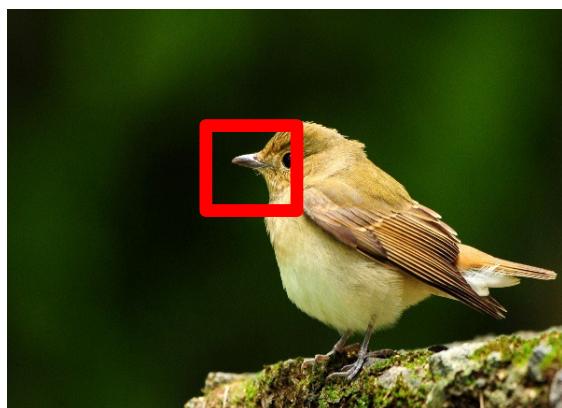
- Some patterns are much smaller than the whole image

Can represent a small region with fewer parameters



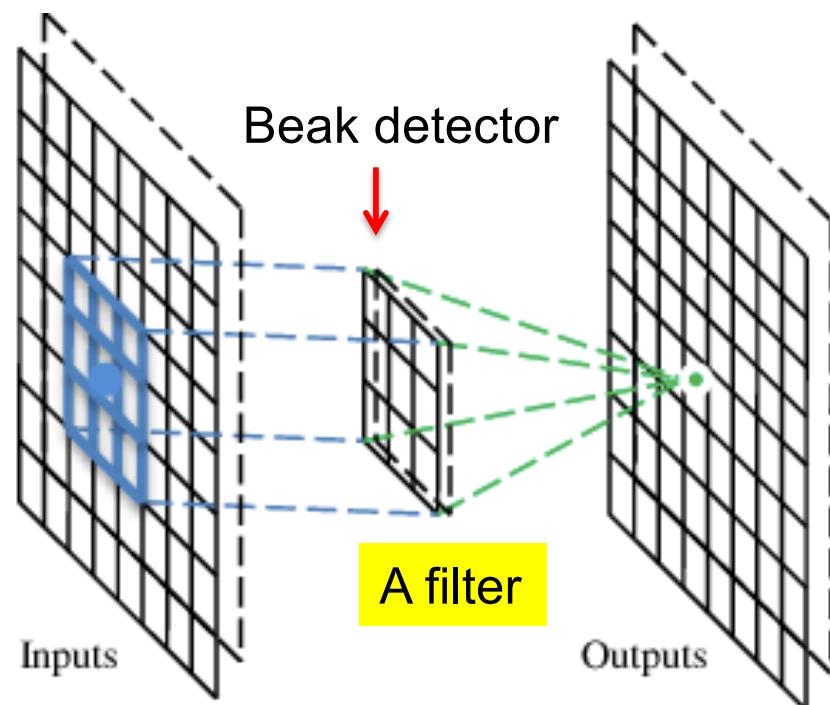
Same pattern appears in different places:
They can be compressed!

What about training a lot of such “small” detectors
and each detector must “move around”.



A convolutional layer

A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that does convolutional operation.



Convolution

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

These are the network parameters to be learned.

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

Each filter detects a small pattern (3 x 3).

Convolution

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Dot
product



1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

6 x 6 image

Convolution

If stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

3

-3

Convolution

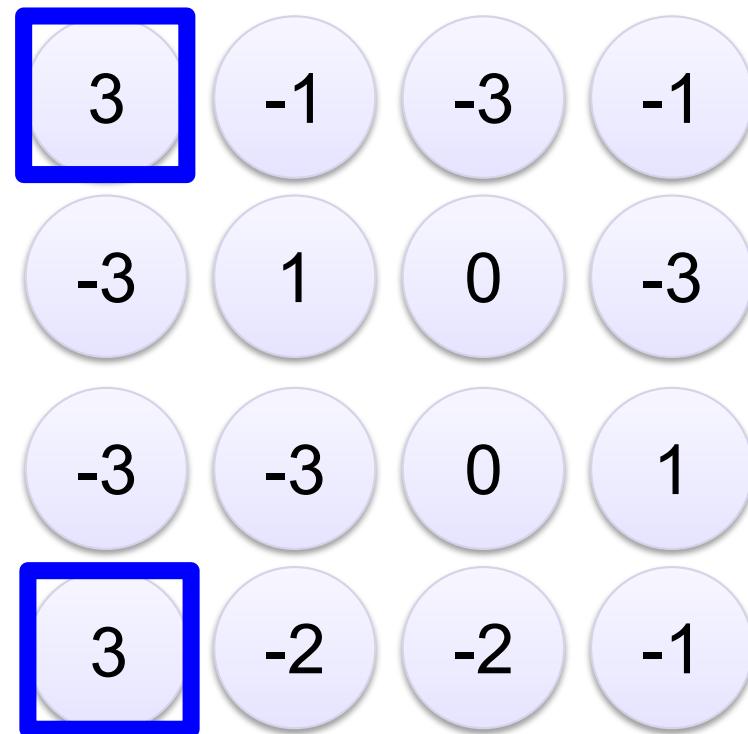
stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1



Convolution

stride=1

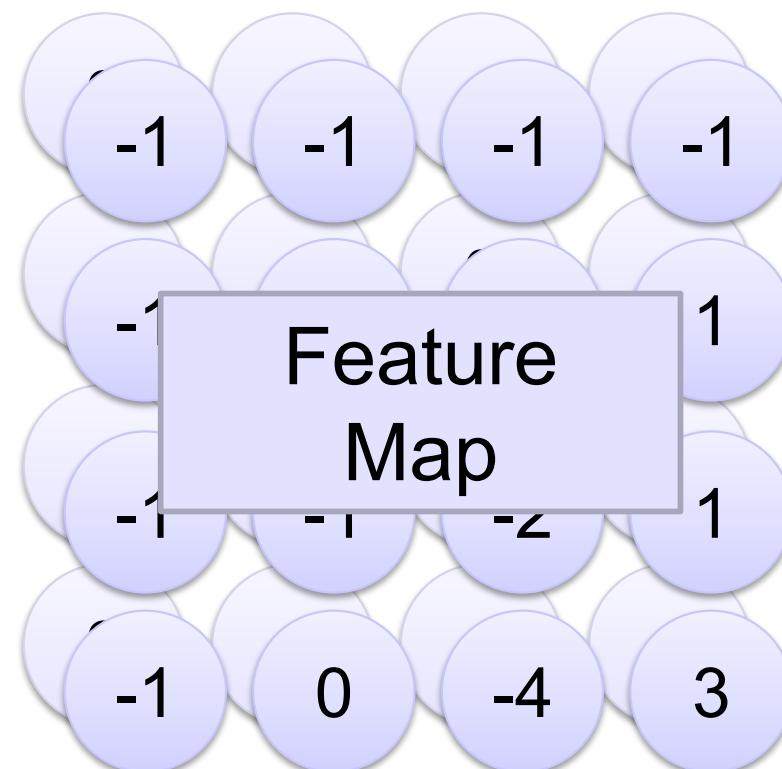
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

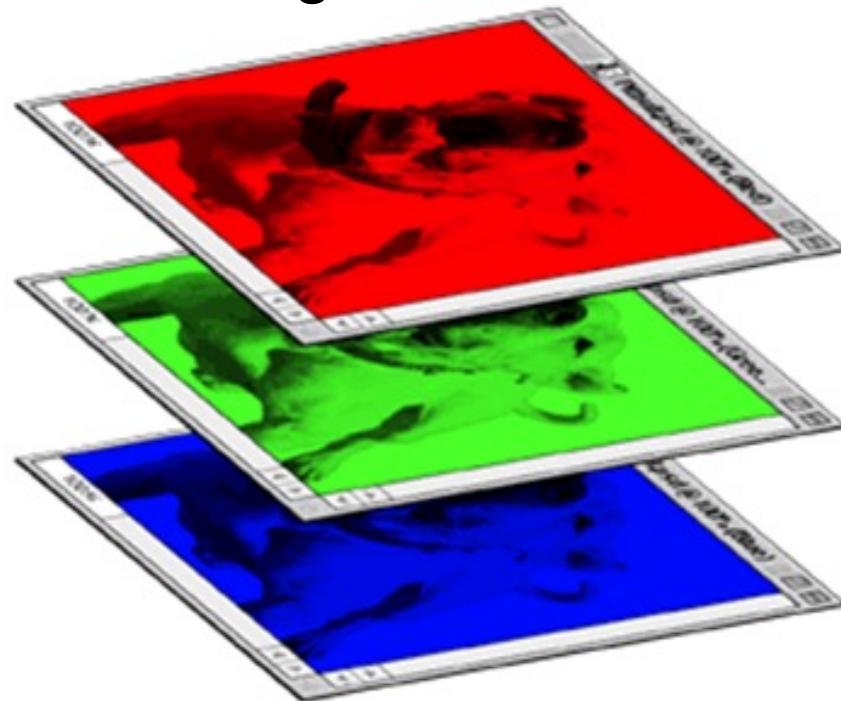
Repeat this for each filter



Two 4 x 4 images
Forming 2 x 4 x 4 matrix

Color image: RGB 3 channels

Color image



1	-1	-1
-1	1	-1
-1	-1	1

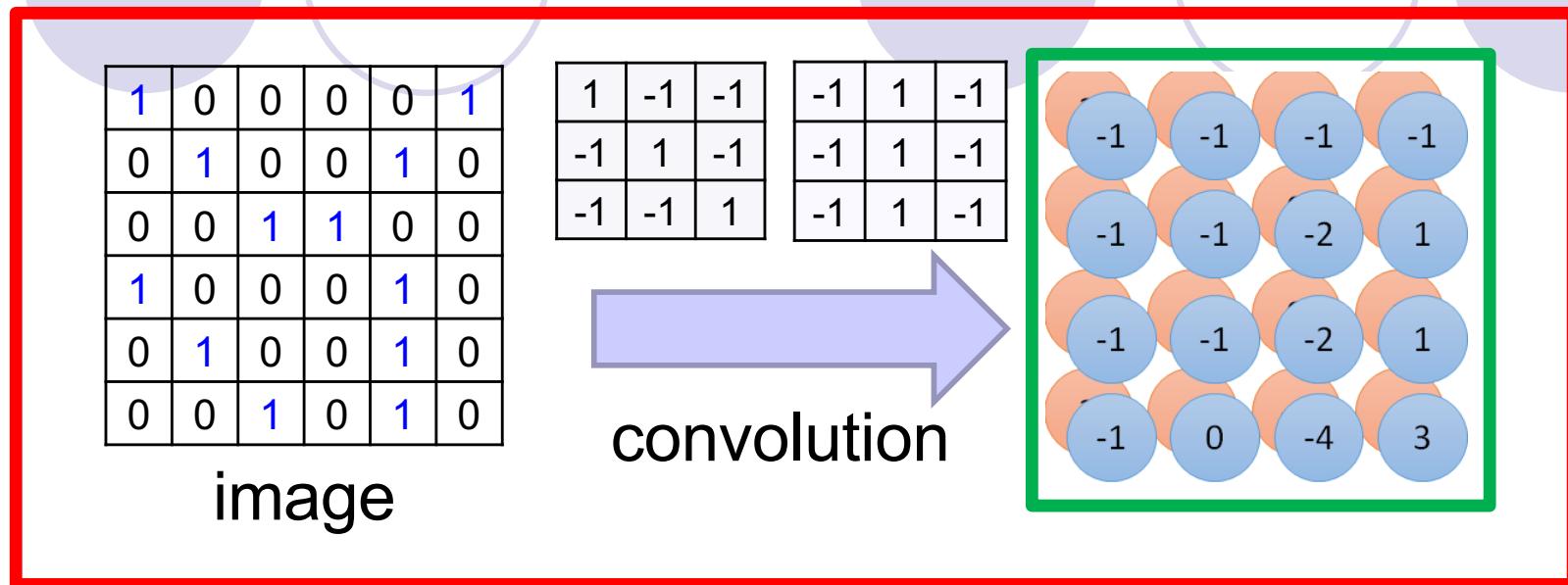
Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

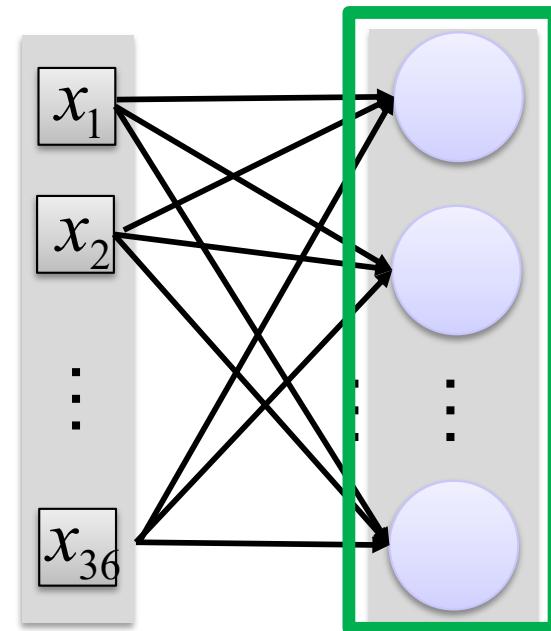
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

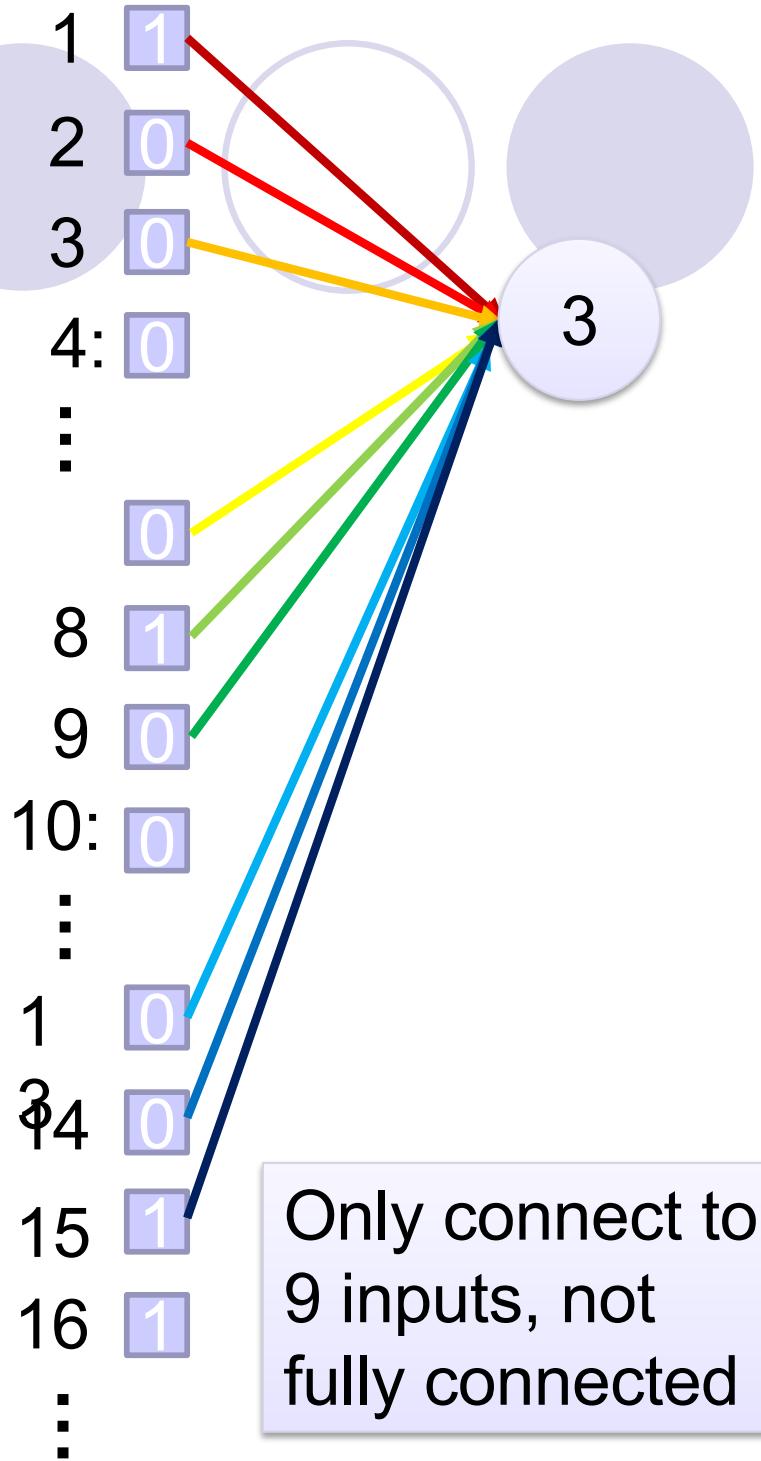
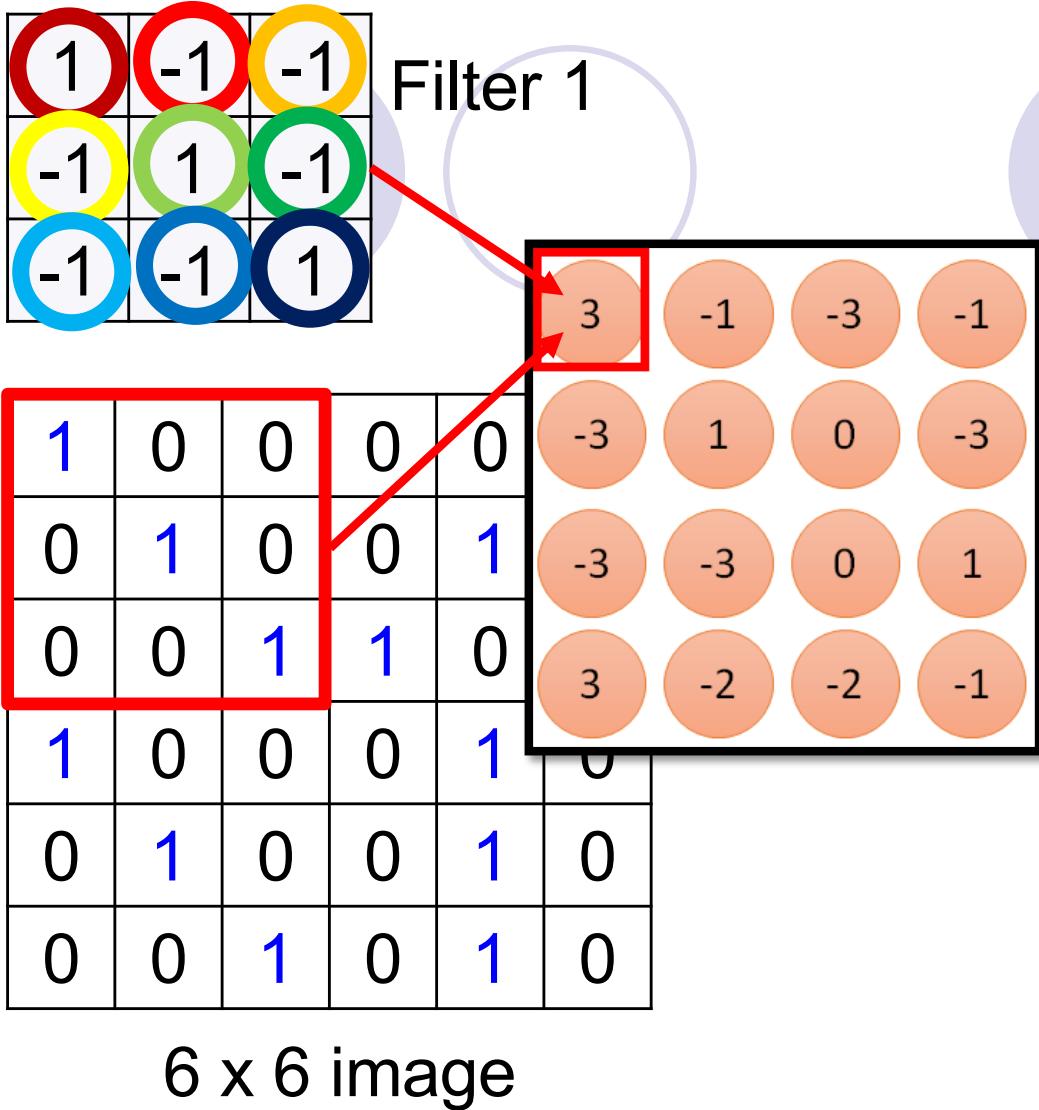
Convolution vs. Fully Connected



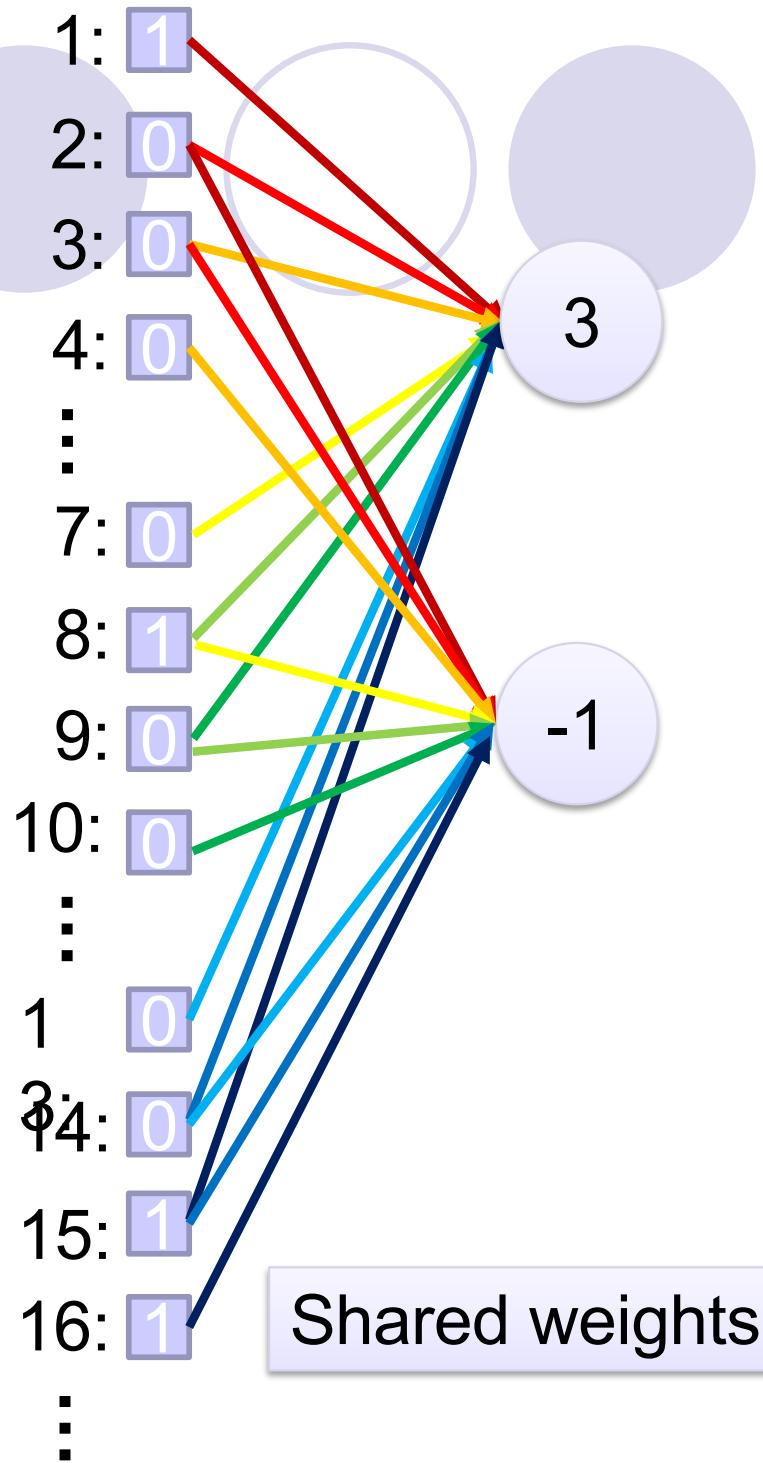
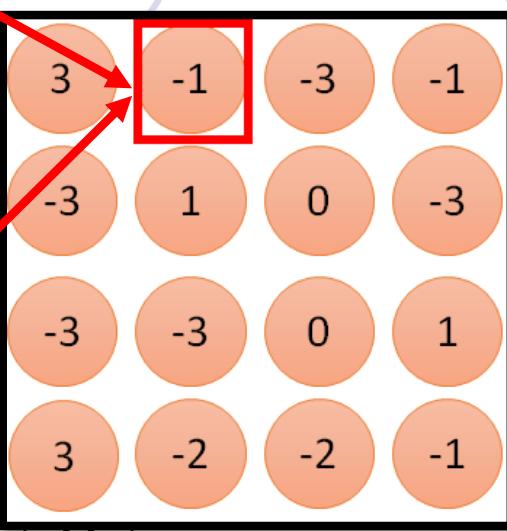
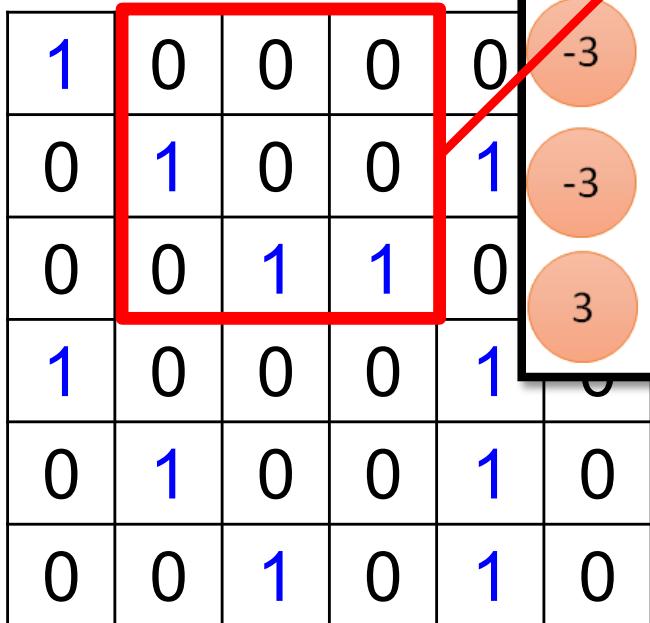
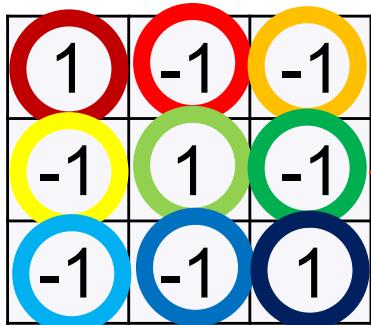
Fully-
connected

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0





fewer parameters!

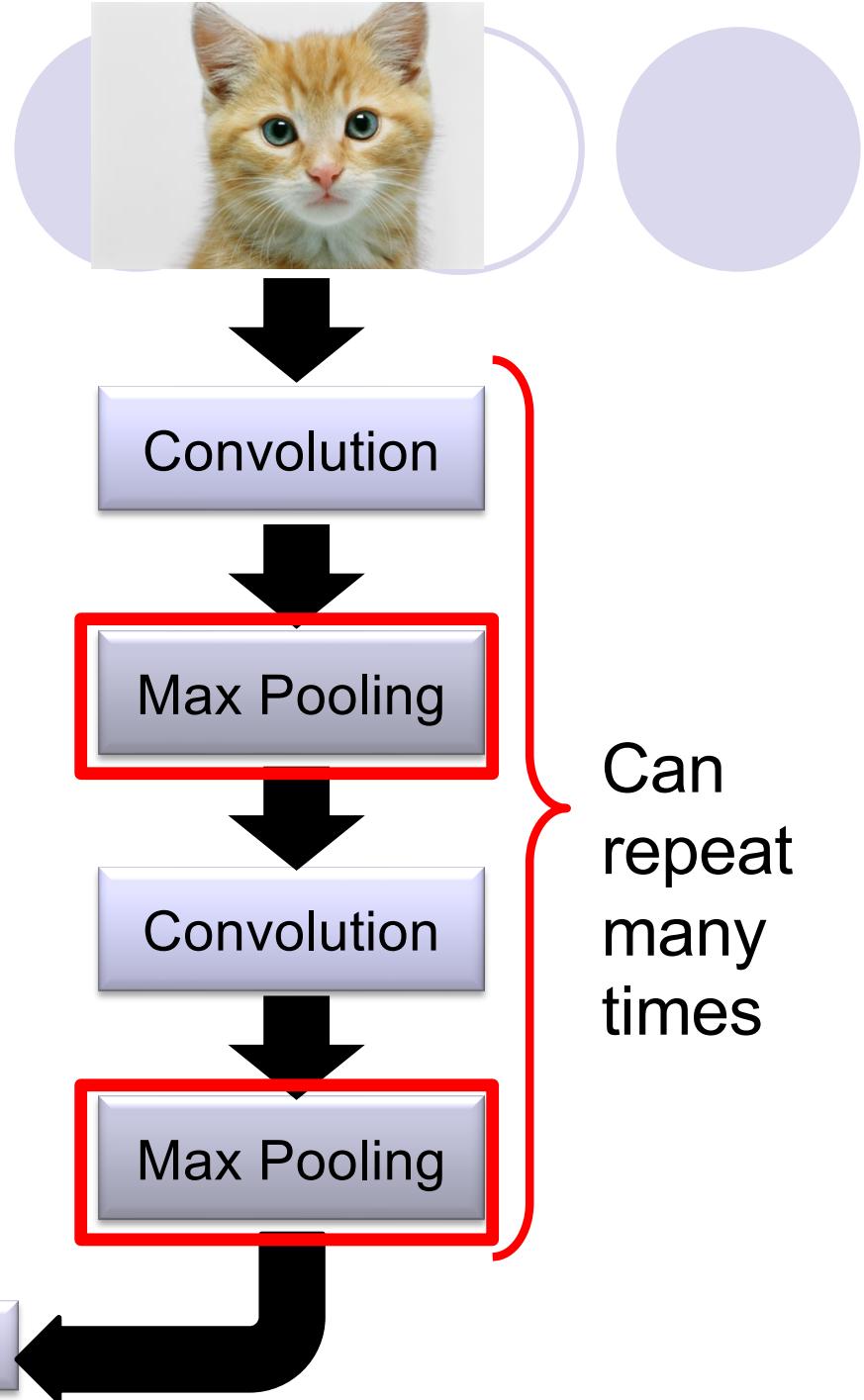
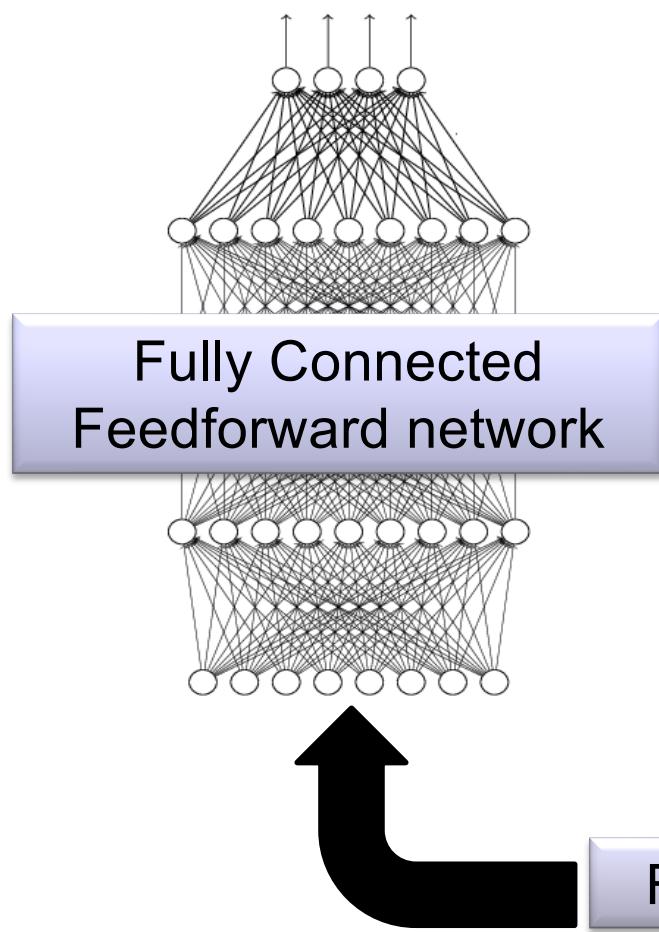


Fewer parameters

Even fewer parameters

The whole CNN

cat dog



Can
repeat
many
times

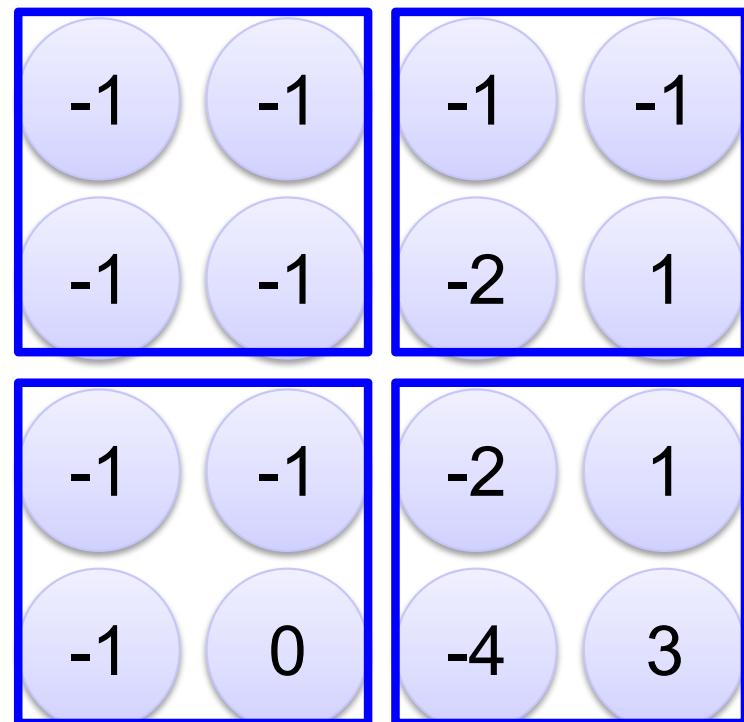
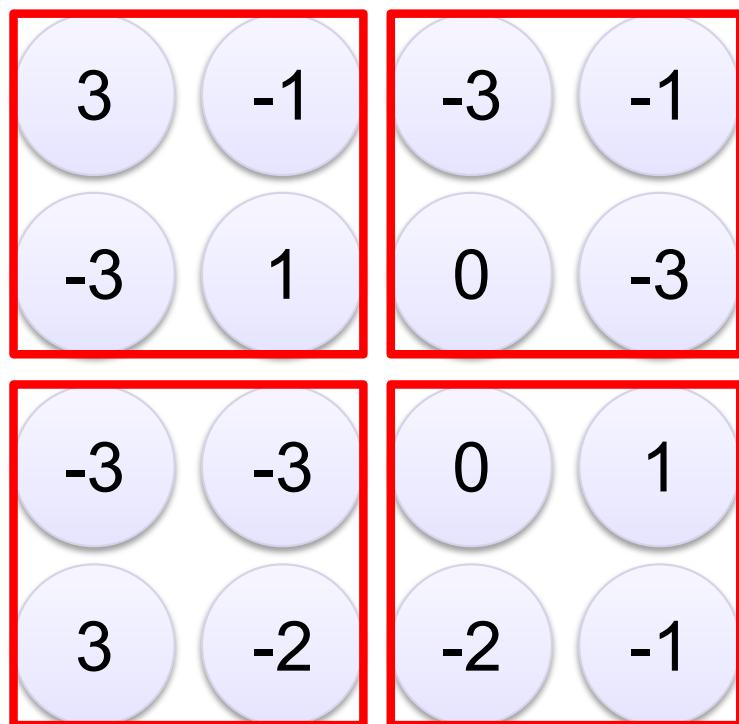
Max Pooling

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2



Why Pooling

- Subsampling pixels will not change the object bird



Subsampling

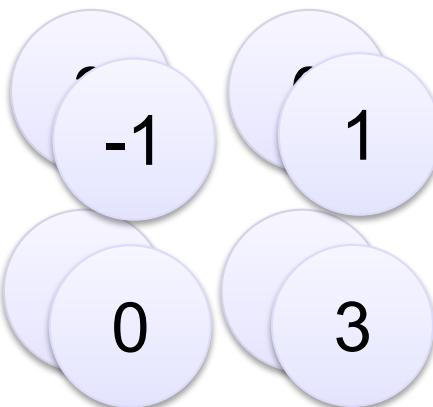


We can subsample the pixels to make image
smaller
→ fewer parameters to characterize the image

A CNN compresses a fully connected network in two ways:

- Reducing number of connections
- Shared weights on the edges
- Max pooling further reduces the complexity

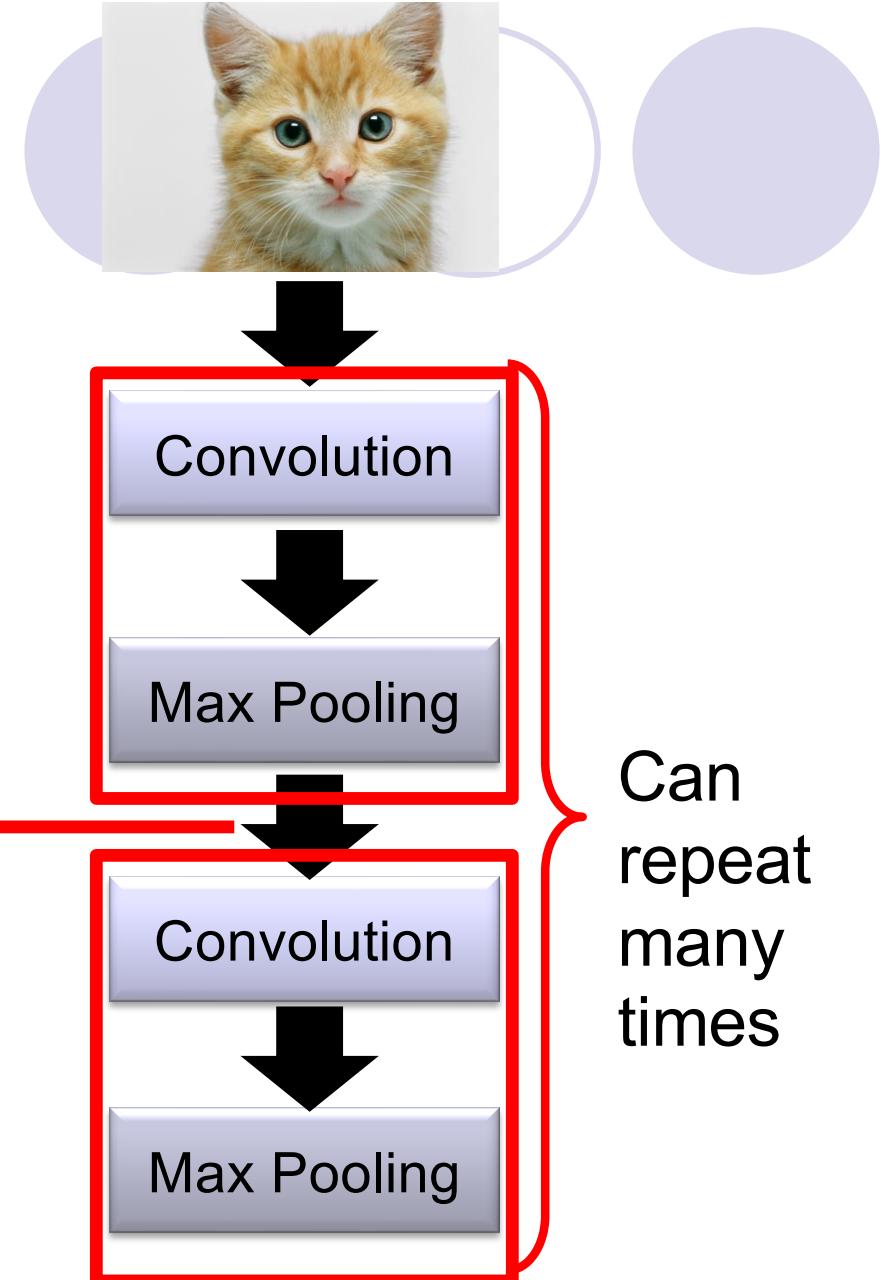
The whole CNN



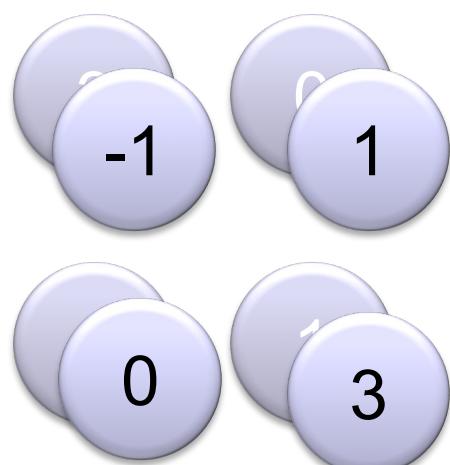
A new image

Smaller than the original image

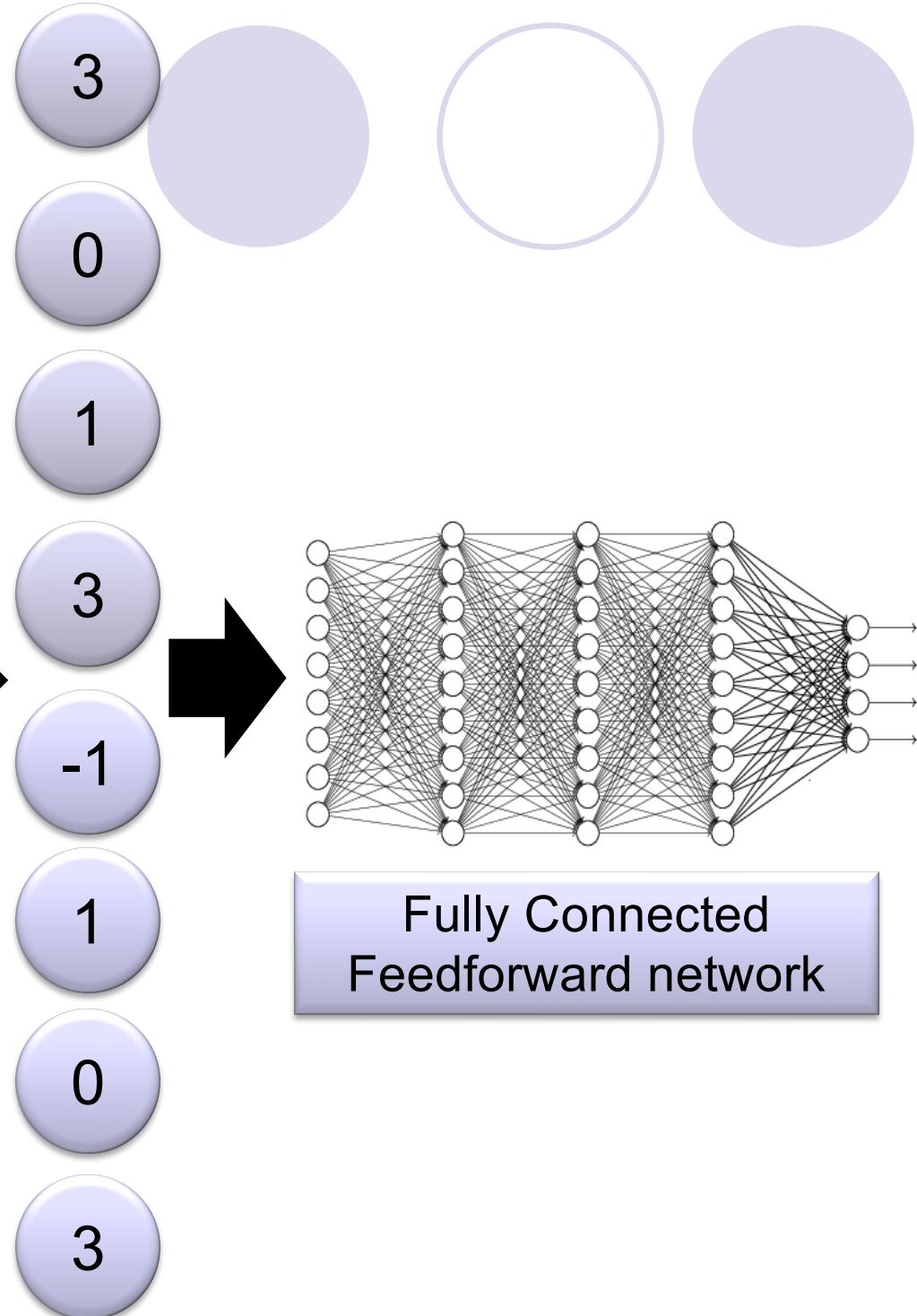
The number of channels
is the number of filters



Flattening

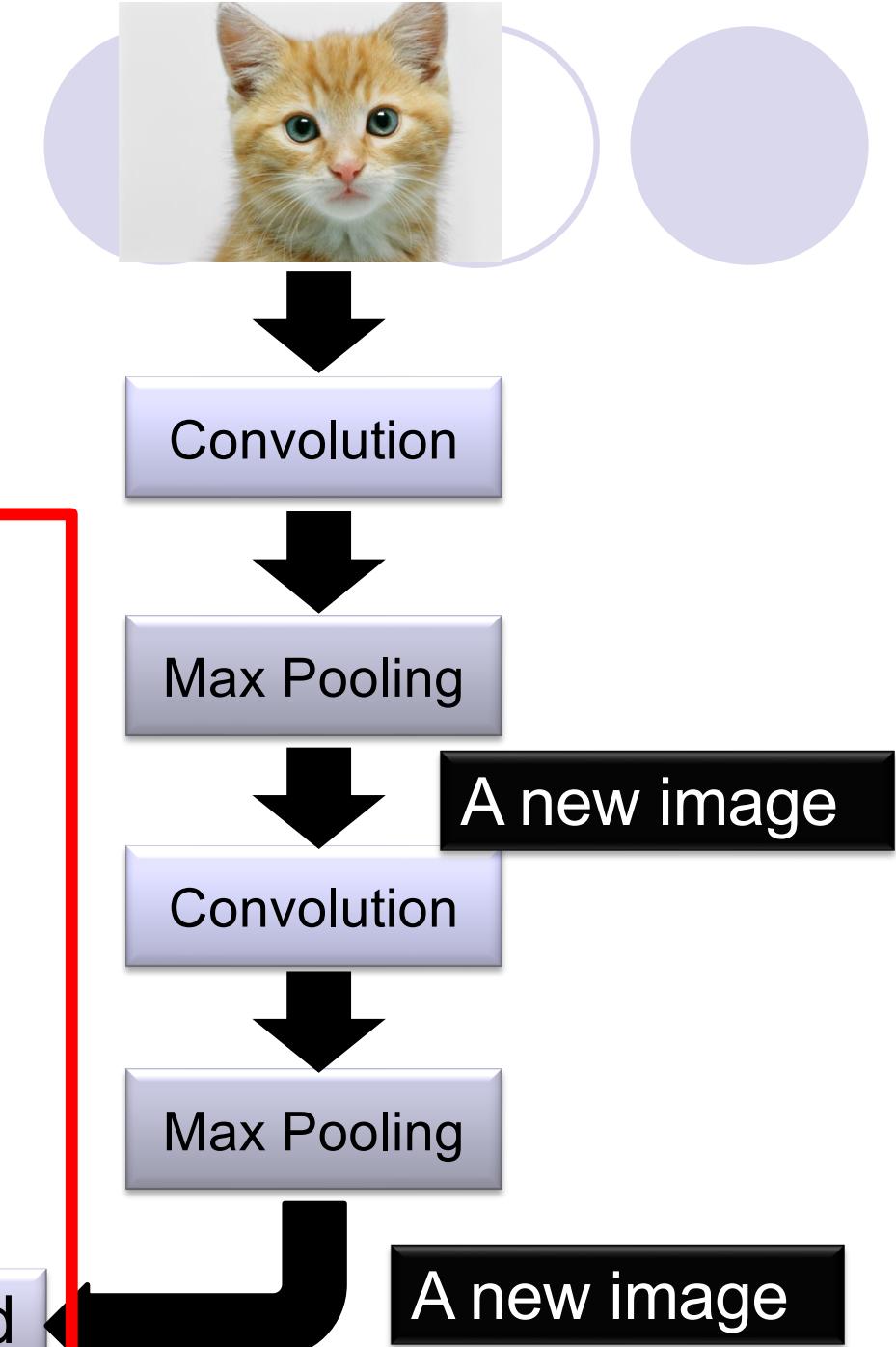
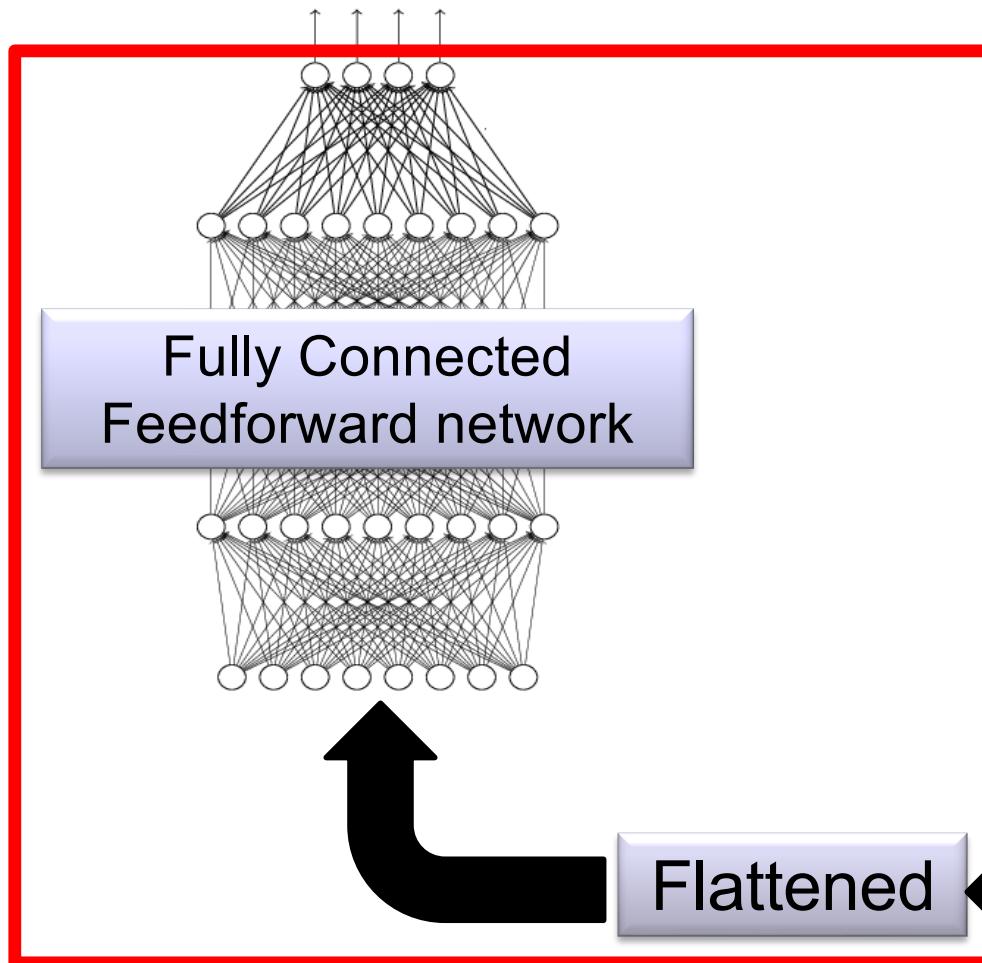


Flattened

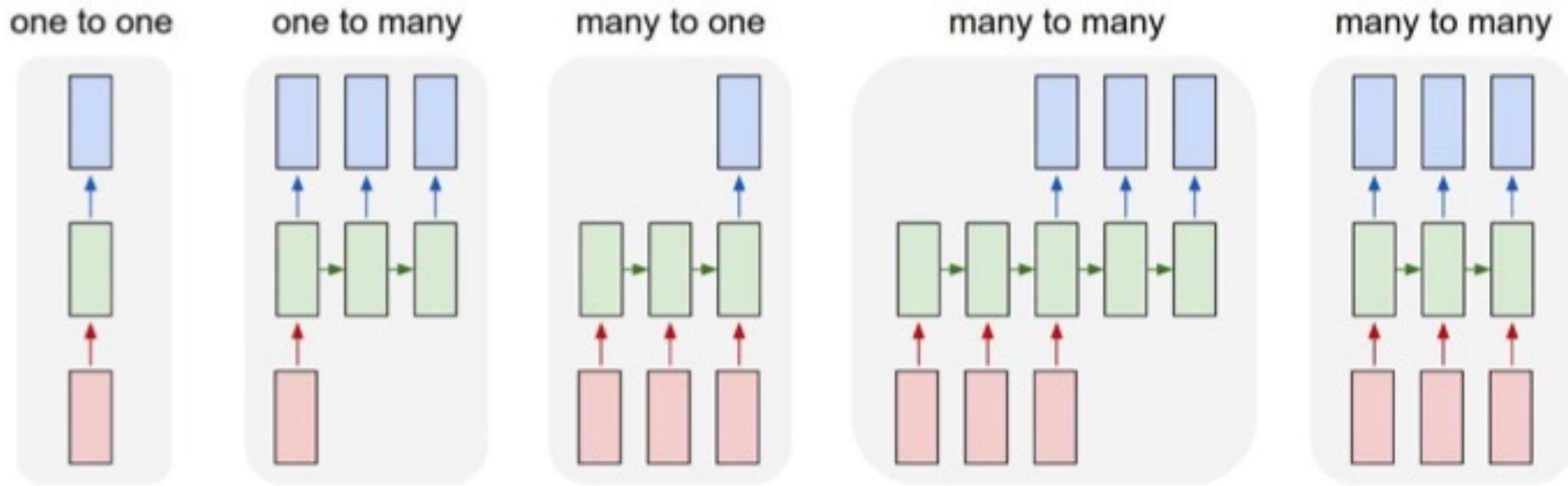


The whole CNN

cat dog



Recurrent Neural Networks



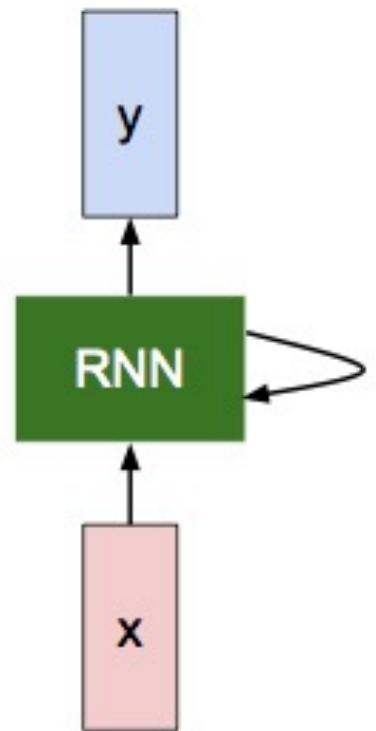
We've discussed the one to one classification case a lot but there are many cases where the input or output is sequential data.

Vanilla RNN

We can process a sequence of vectors \mathbf{x} by applying a **recurrence formula** at every time step:

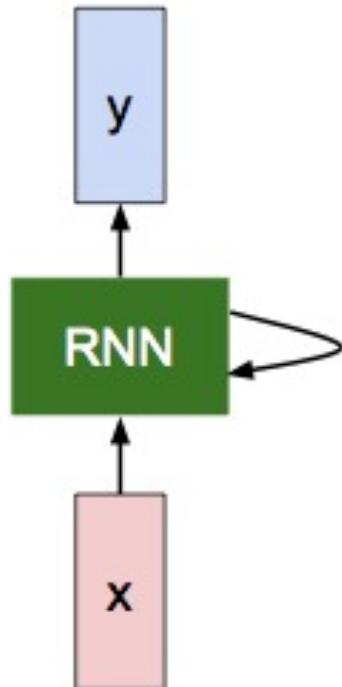
$$h_t = f_W(h_{t-1}, x_t)$$

new state / old state input vector at
some function | some time step
with parameters W



Vanilla RNN

The state consists of a single “*hidden*” vector \mathbf{h} :



$$\mathbf{h}_t = f_W(\mathbf{h}_{t-1}, \mathbf{x}_t)$$



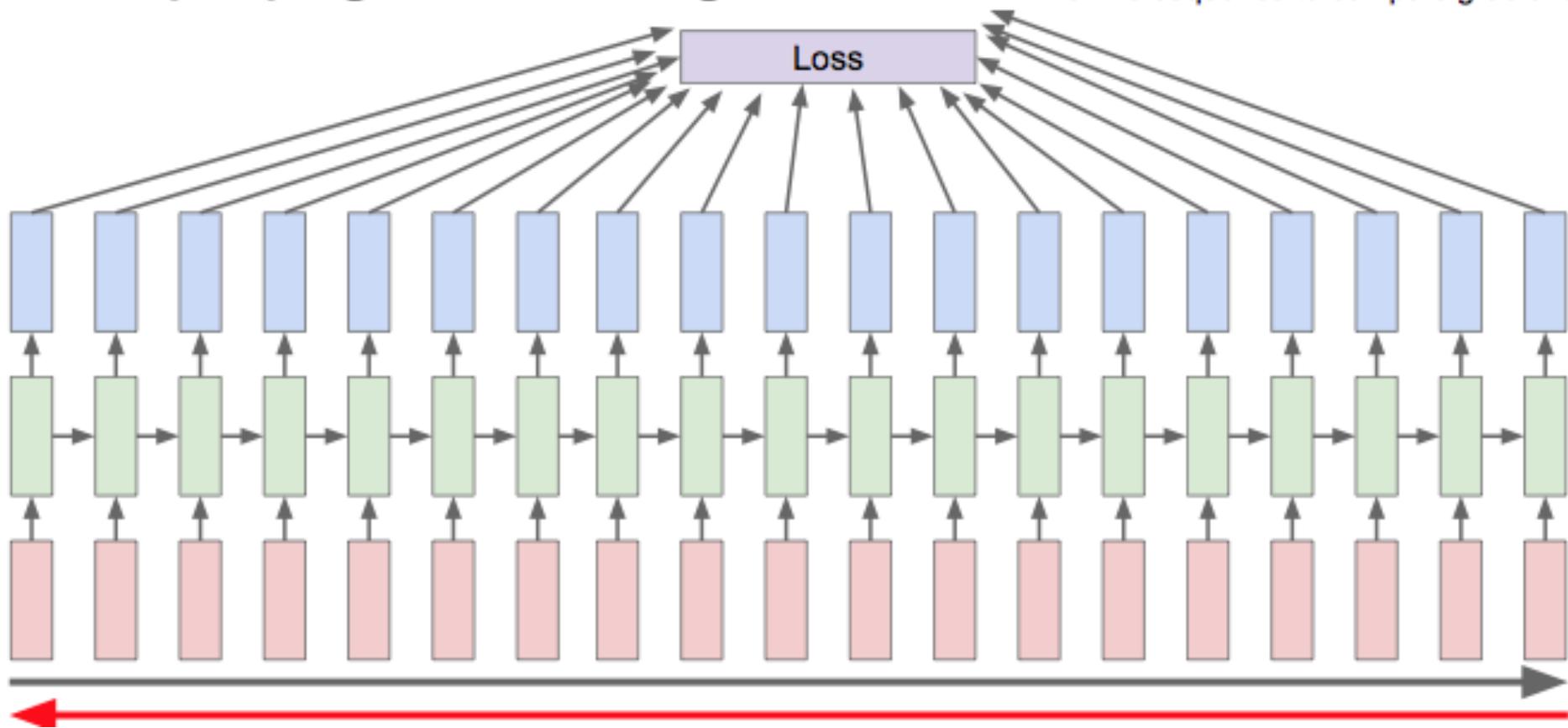
$$\mathbf{h}_t = \tanh(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t)$$

$$y_t = \mathbf{W}_{hy}\mathbf{h}_t$$

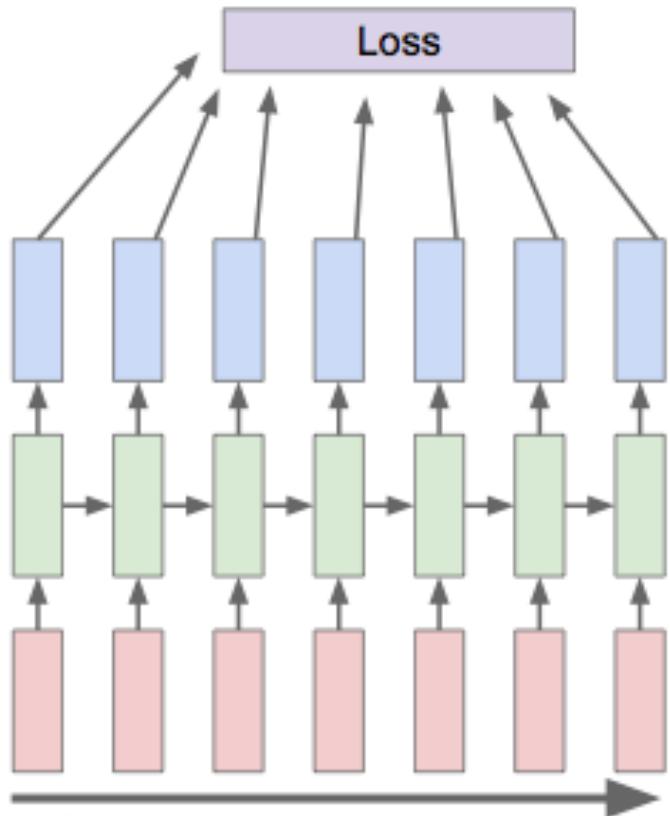
Backprop thru Time

Backpropagation through time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

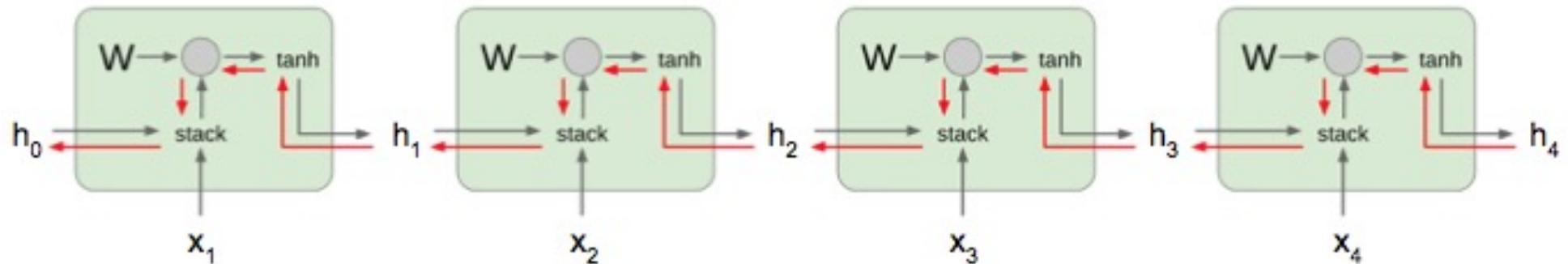


Truncated Backprop



Run forward and backward
through chunks of the
sequence instead of whole
sequence

Gradient Problems



Computing gradient
of h_0 involves many
factors of W
(and repeated tanh)

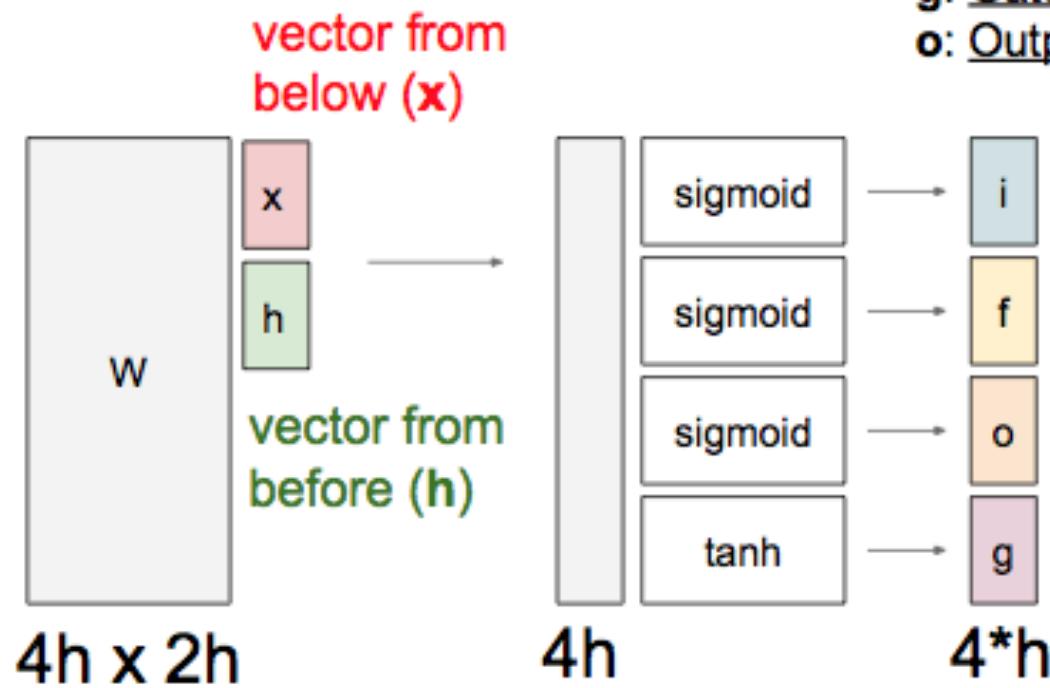
Largest singular value > 1 :
Exploding gradients

Largest singular value < 1 :
Vanishing gradients

LSTM

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

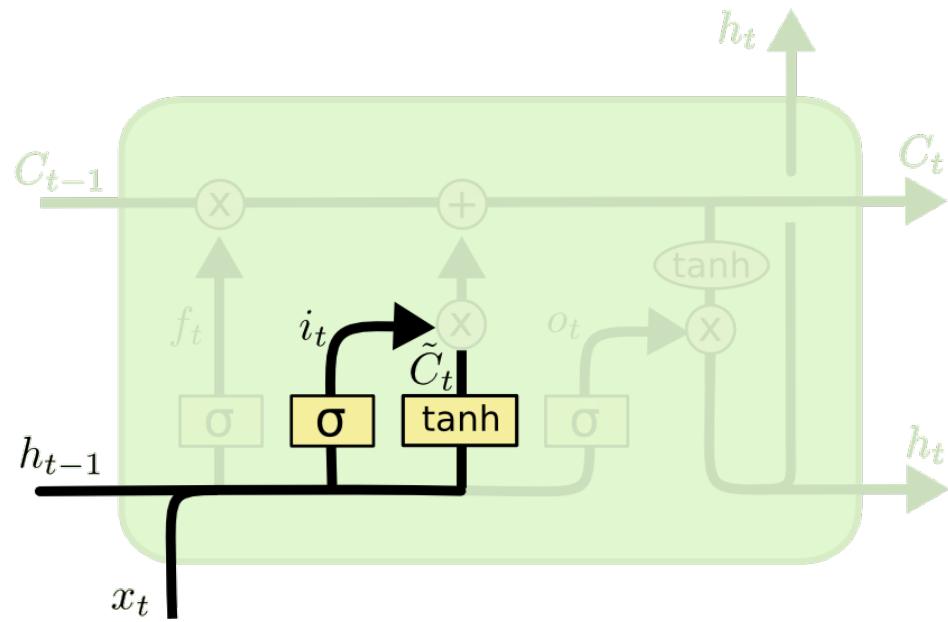


- f: Forget gate, Whether to erase cell
- i: Input gate, whether to write to cell
- g: Gate gate (?), How much to write to cell
- o: Output gate, How much to reveal cell

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

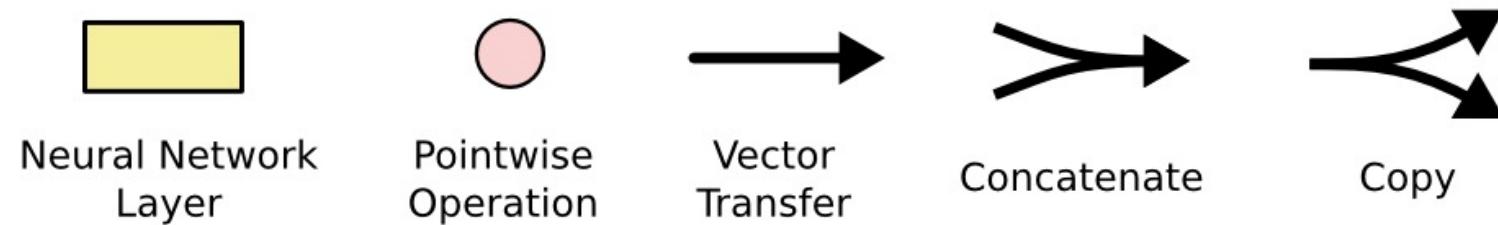
$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

LSTM

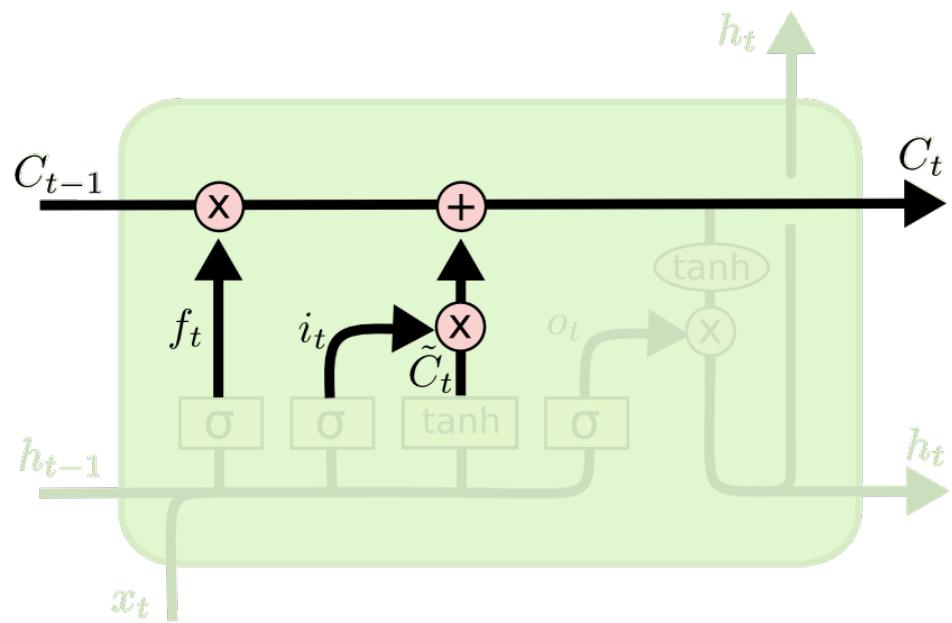


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

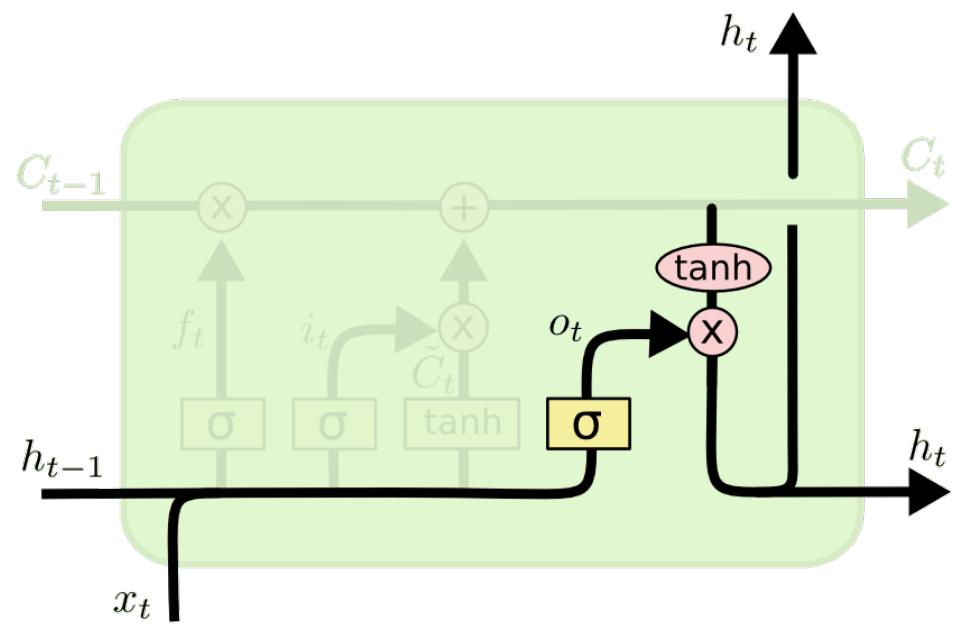


LSTM



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

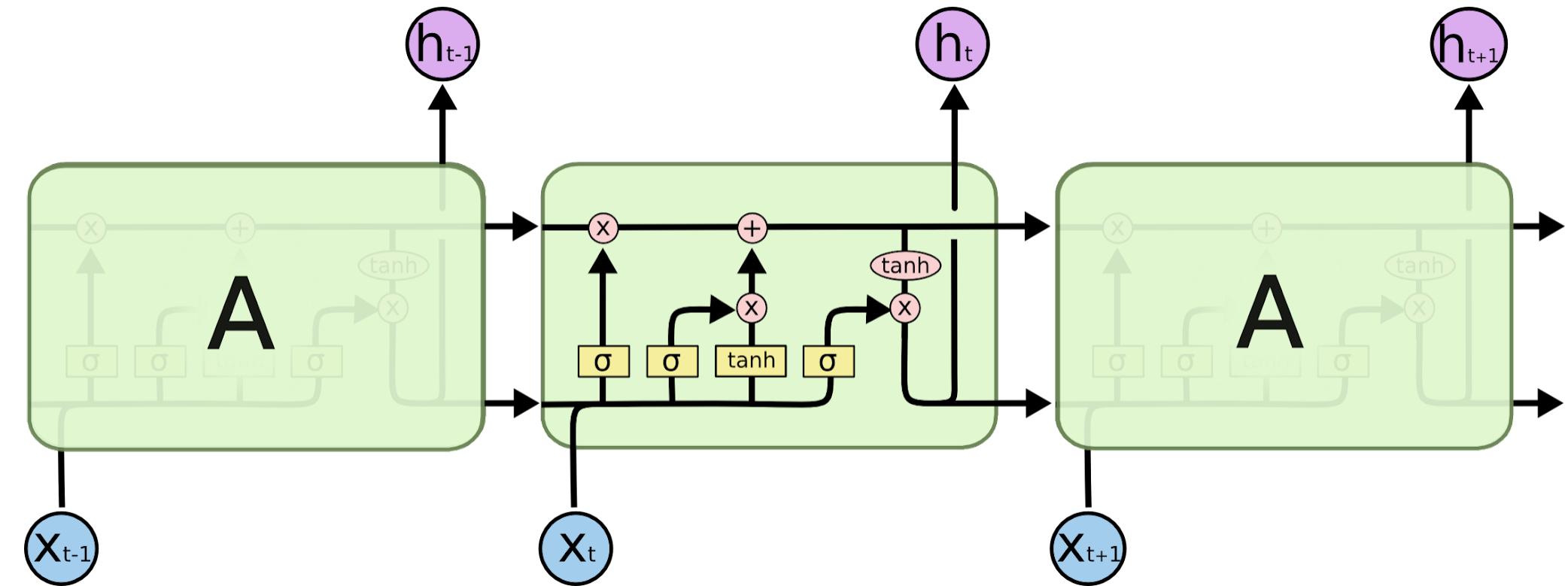
LSTM



$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

LSTM



Summary

- You've survived the whirlwind tour of neural networks, deep learning, convolutional neural networks, and RNNs.
- Next time: deep RL!

Slide Credits

- R. Mooney (neural networks)
- Y. LeCun (CNN)
- F. Li (RNN/LSTM)
- C. Olah (LSTM)

CAP6671 Intelligent Systems

Lecture 11: Deep Reinforcement Learning: DQN, PG, Actor-Critic

Instructor: Dr. Gita Sukthankar

Email: gitars@eeecs.ucf.edu

Slides: Sergey Levine, David Silver, Saif Alabachi

Announcement

- Reading: Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare and Joelle Pineau (2018), “An Introduction to Deep Reinforcement Learning”, Foundations and Trends in Machine Learning: Vol. 11, No. 3-4, pp 219–354. (Chap 4 and 5)

DeepMind RL

Atari

<https://youtu.be/Ih8EfvOzBOY>

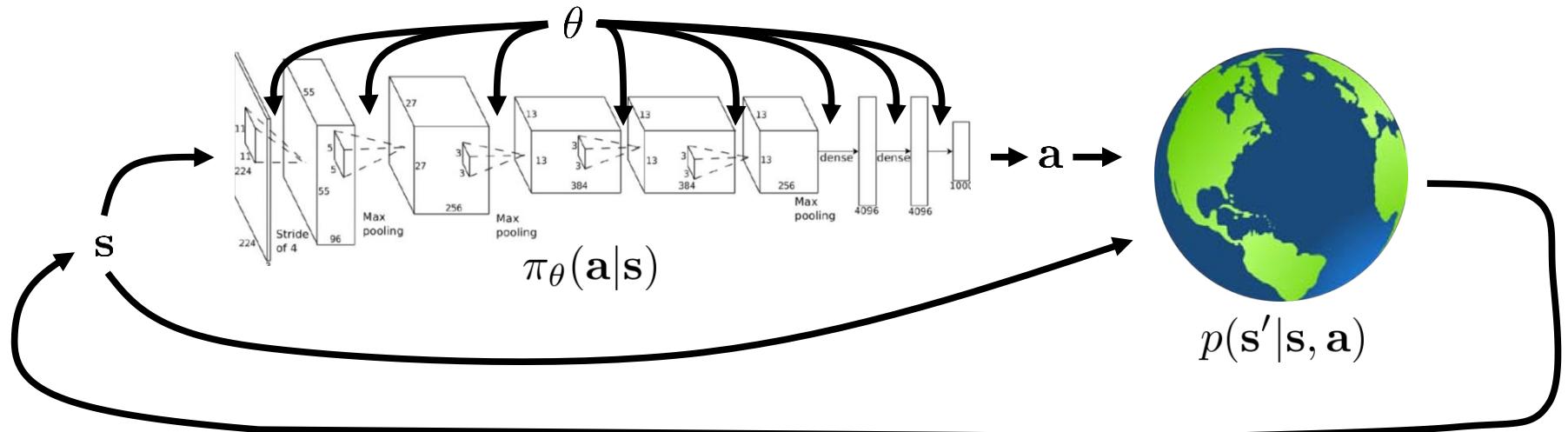
AlphaGo

<https://youtu.be/IFmj5M5Q5jg>

MuJoCo: Multi-Agent Joint Dynamics with Contact

<https://youtu.be/gn4nRCC9TwQ>

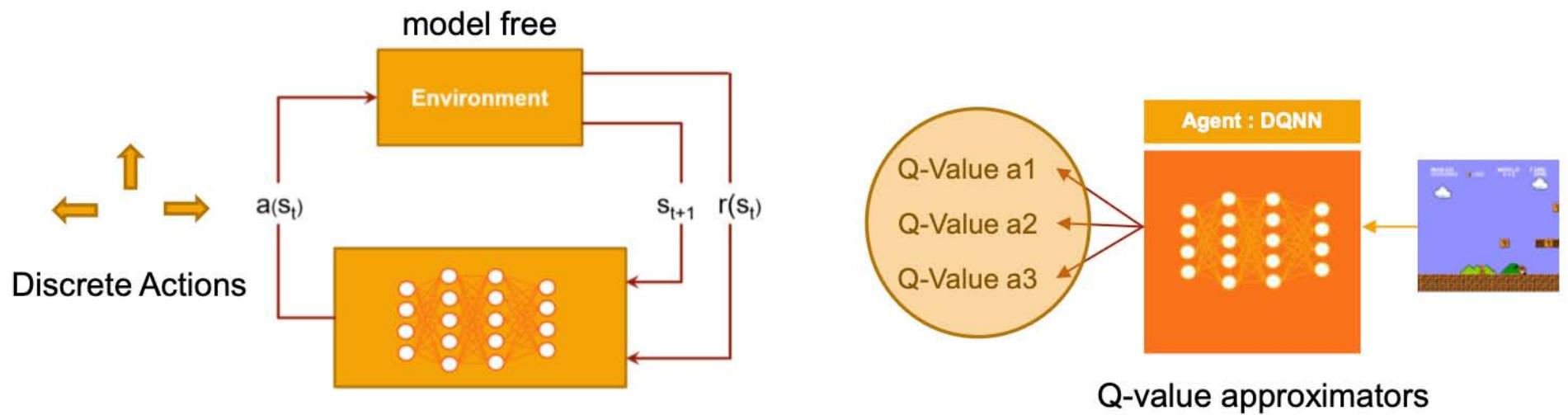
Deep RL Agent



$$p_\theta(s_1, a_1, \dots, s_T, a_T) = \underbrace{p(s_1)}_{p_\theta(\tau)} \prod_{t=1}^T \pi_\theta(a_t | s_t) p(s_{t+1} | s_t, a_t)$$

$$\theta^\star = \arg \max_{\theta} E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right]$$

Intuition



Moving from Q-learning to Deep Q-Networks

Deep RL Definitions

Deterministic Policy

$$a_t = \pi(s_t)$$

Stochastic Policy

$$a_t = \pi(a|s_t) = \mathbb{P}[A_t = a|S_t = s]$$

Return

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \text{ where } \gamma \in [0,1)$$

State value

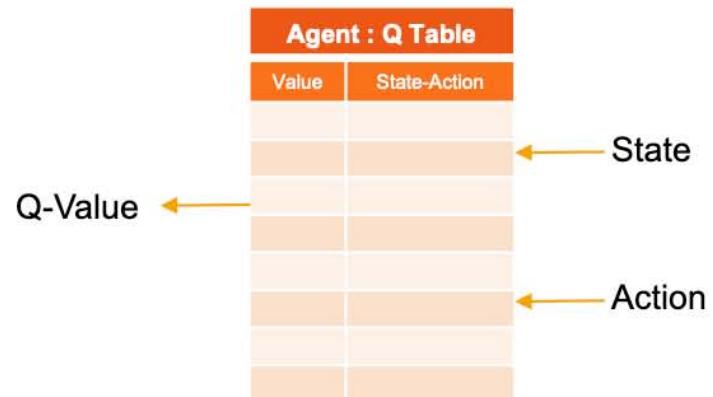
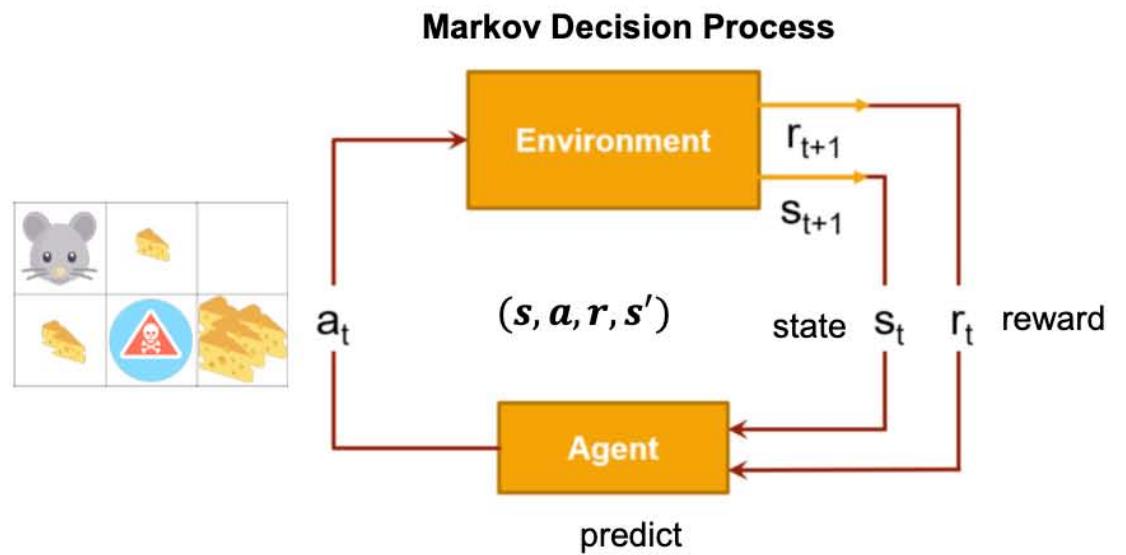
$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

Action State value

$$Q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a]$$

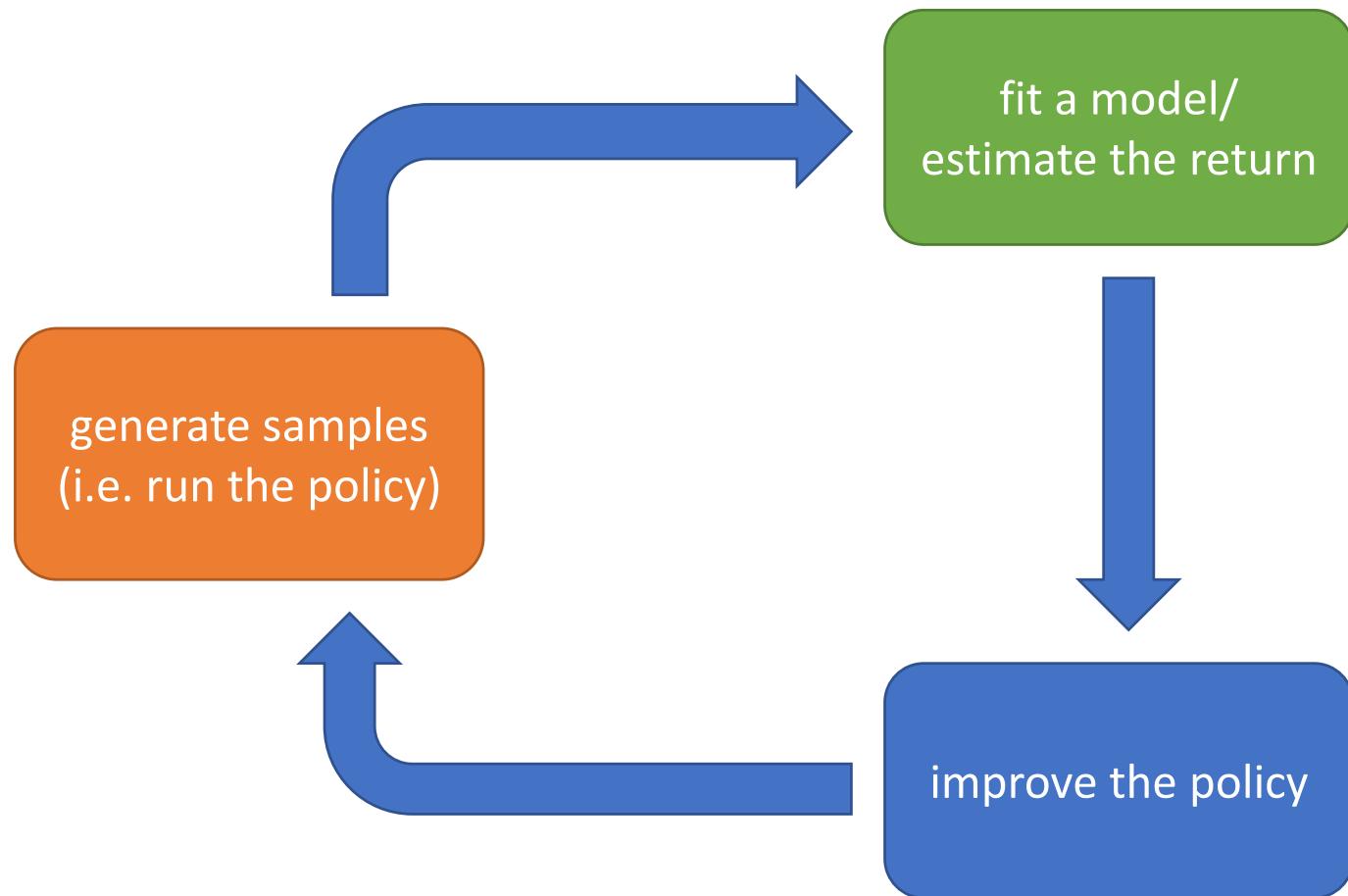
Advantage

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$$

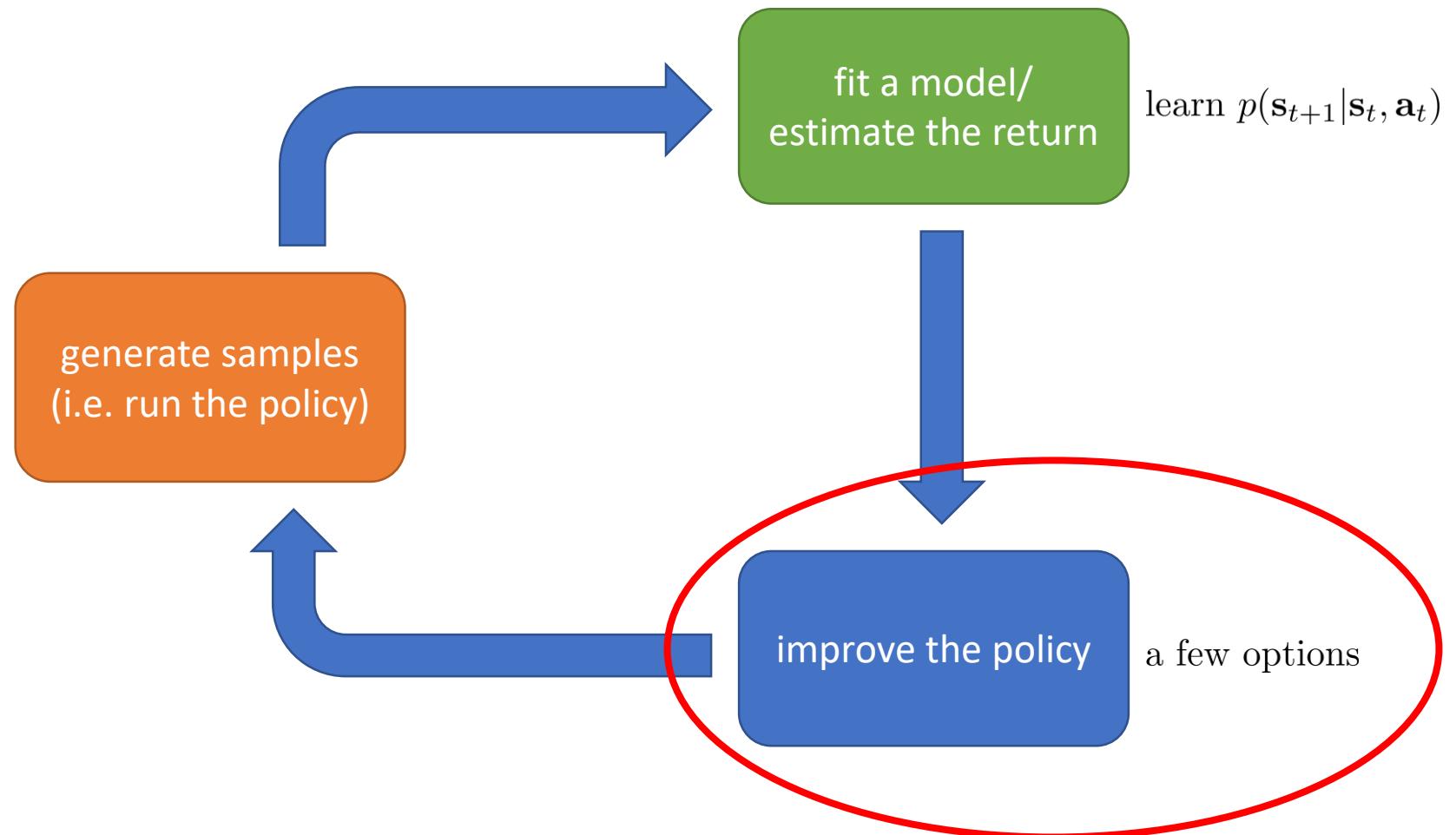


Note: often we work with stochastic policies since they are easier to optimize

General Procedure

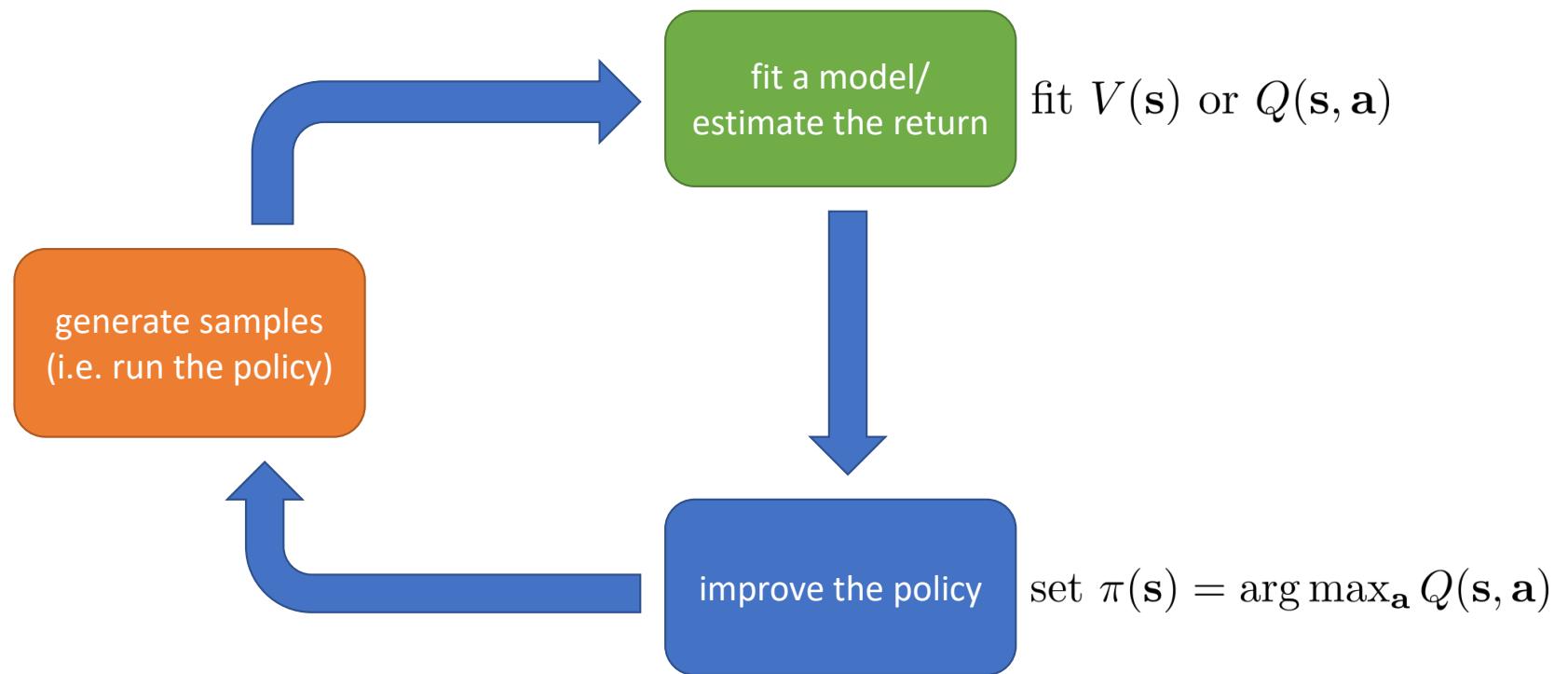


Model-based RL



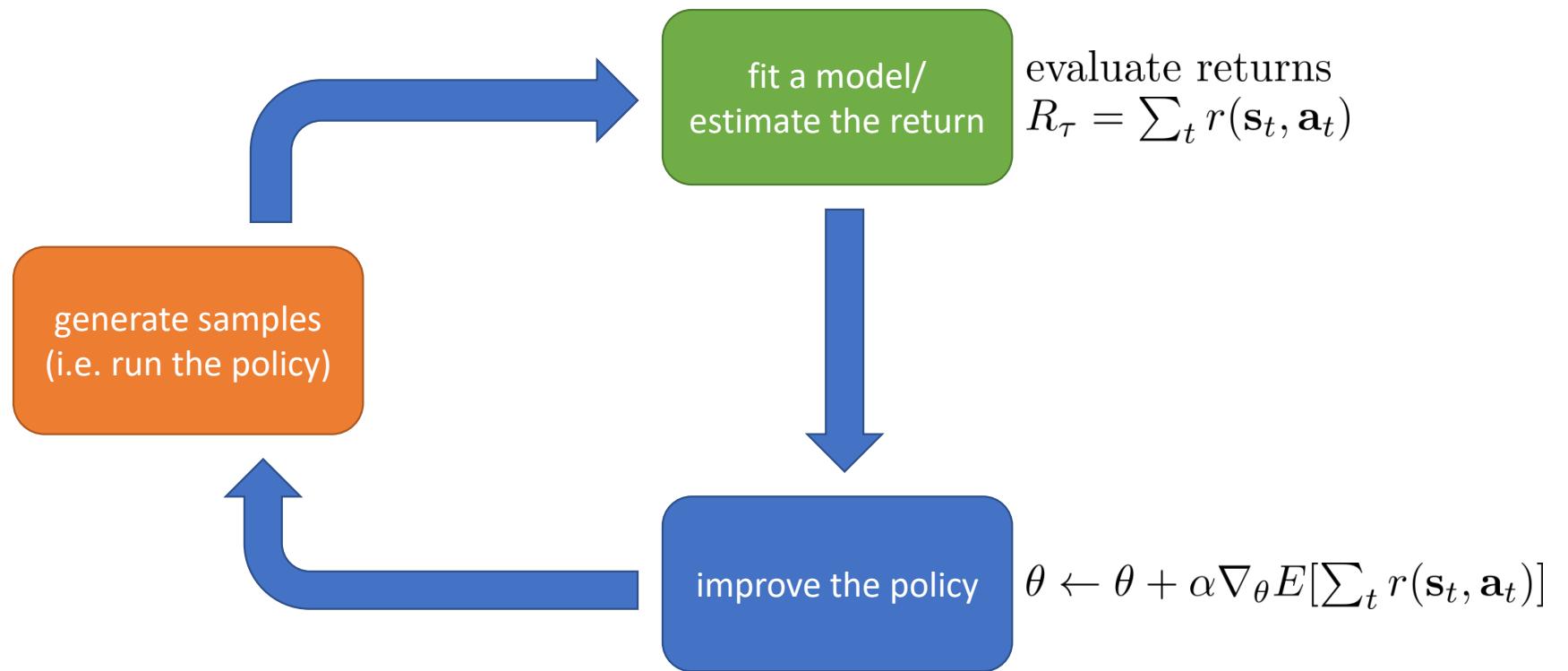
Learn the transition model and search for a good policy

Value-Function



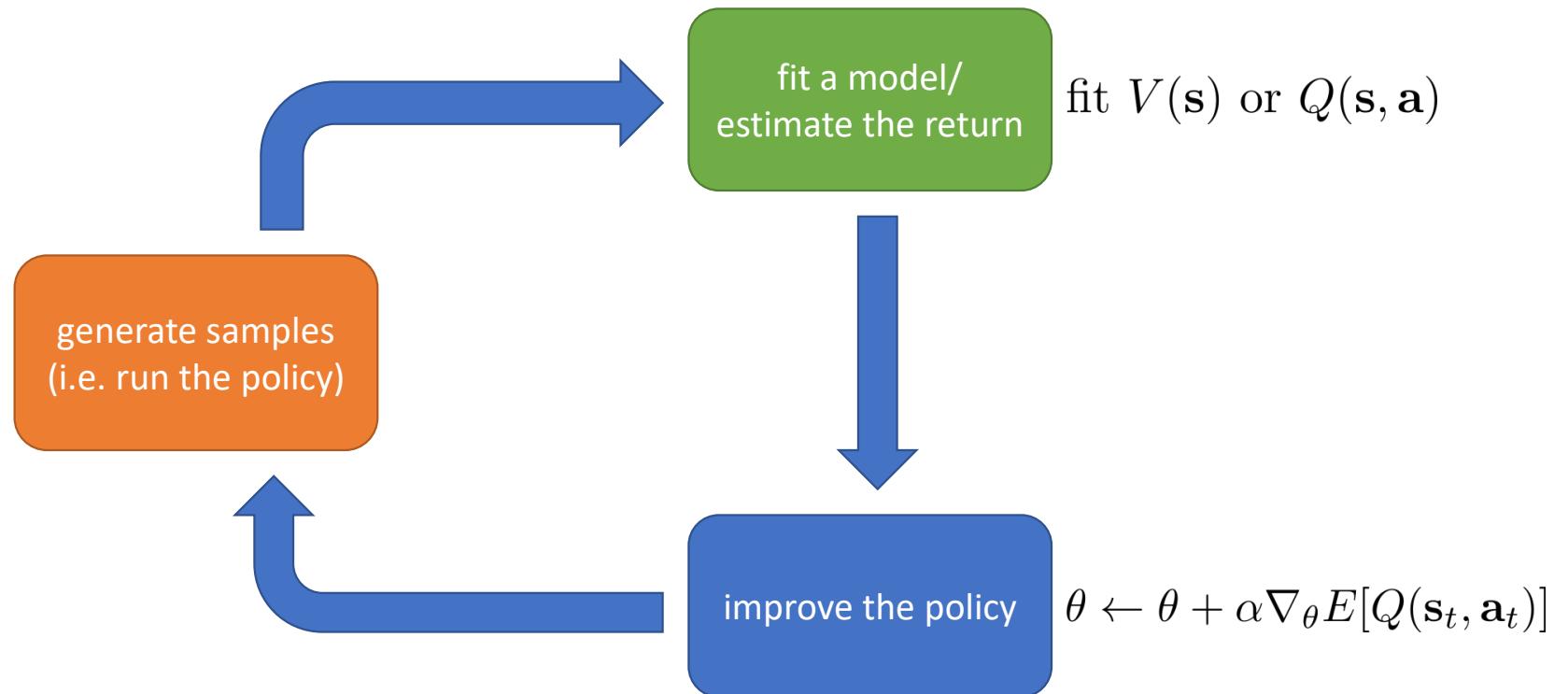
Learn the value function of state-action pairs and select the best action

Direct Policy Gradient



Learn the policy based on expected rewards

Actor-Critic



Alternating learning the value and improving the policy

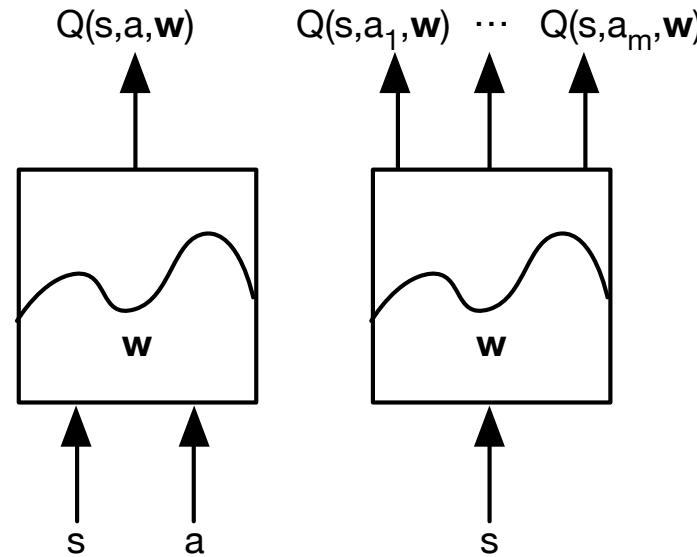
Summary

- Value function fitting
 - Q-learning, DQN
 - TD learning
 - Fitted value iteration
- Policy gradient
 - REINFORCE
 - DDPG (Direct Deterministic Policy Gradient)
- Actor-critic
 - Asynchronous advantage actor-critic
- Model-based
 - Dyna
 - Rmax

Q-Networks

Neural networks (parameterized by w) replace the Q-table

$$Q(s, a, w) \approx Q^*(s, a)$$



Unfortunately training these networks is not simple!

Q-Learning

- Optimal Q-values should obey the Bellman equation

$$Q^*(s, a) = \mathbb{E}_{s'} \left[r + \gamma \max_{a'} Q(s', a')^* \mid s, a \right]$$

- Right hand side should serve as the target
- Minimize MSE loss by stochastic gradient descent: $r + \gamma \max_{a'} Q(s', a', \mathbf{w})$

$$l = \left(r + \gamma \max_a Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w}) \right)^2$$

- This would converge in a Q-table but diverges in a neural network due to:
 - Correlations between samples
 - Non-stationary targets

online Q iteration algorithm:

- sequential states are strongly correlated

- target value is always changing

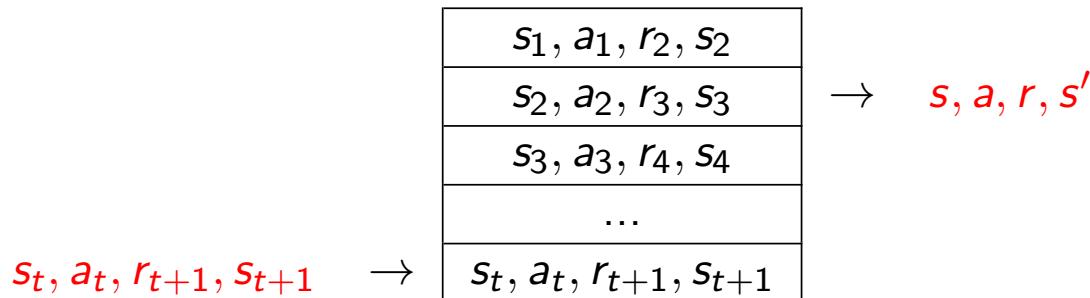
1. take some action \mathbf{a}_i and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$

2. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$



Experience Replay

- Remove correlations through sampling the agent's experience



- Apply update rule:

$$I = \left(r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

- Hold \mathbf{w}^- fixed to deal with the non-stationary process

Problem

Q-learning is *not* gradient descent!

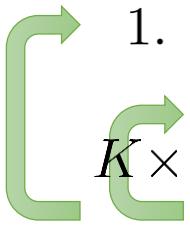
$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$$

no gradient through target value

Unlike in supervised learning, the target value changes which is problematic...

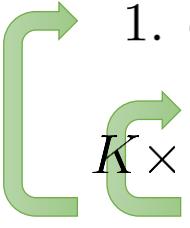
Q-Learning and Regression

full Q-learning with replay buffer:

- 
1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to \mathcal{B}
 2. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from \mathcal{B}
 3. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$
-

one gradient step, moving target

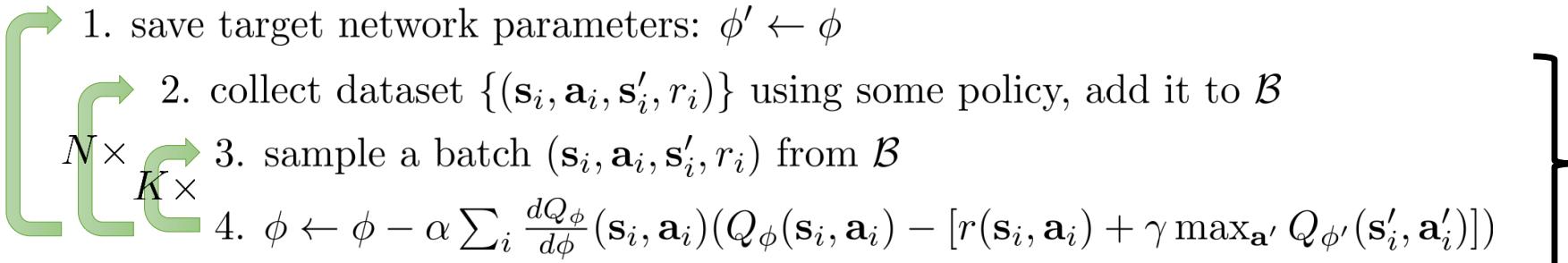
full fitted Q-iteration algorithm:

- 
1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
 2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
 3. set $\phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$
-

perfectly well-defined, stable regression

Target Network

Q-learning with replay buffer and target network:

- 
1. save target network parameters: $\phi' \leftarrow \phi$
 2. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to \mathcal{B}
 3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from \mathcal{B}
 4.
$$\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$$

targets don't change in inner loop!

supervised regression

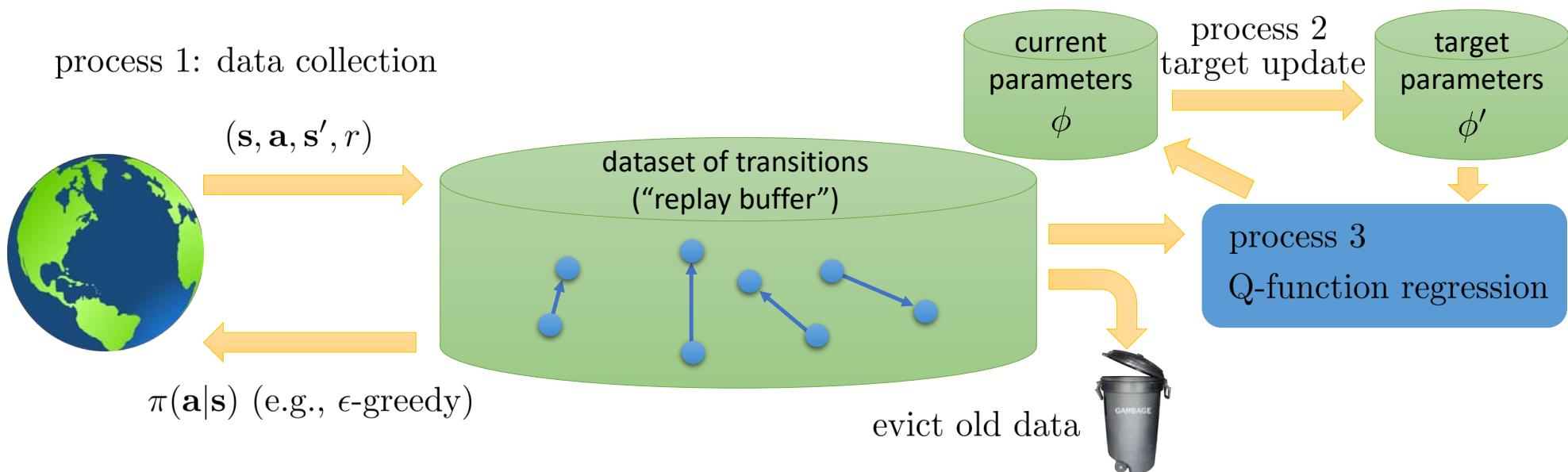
Add a second neural network which remains stable...

DQN

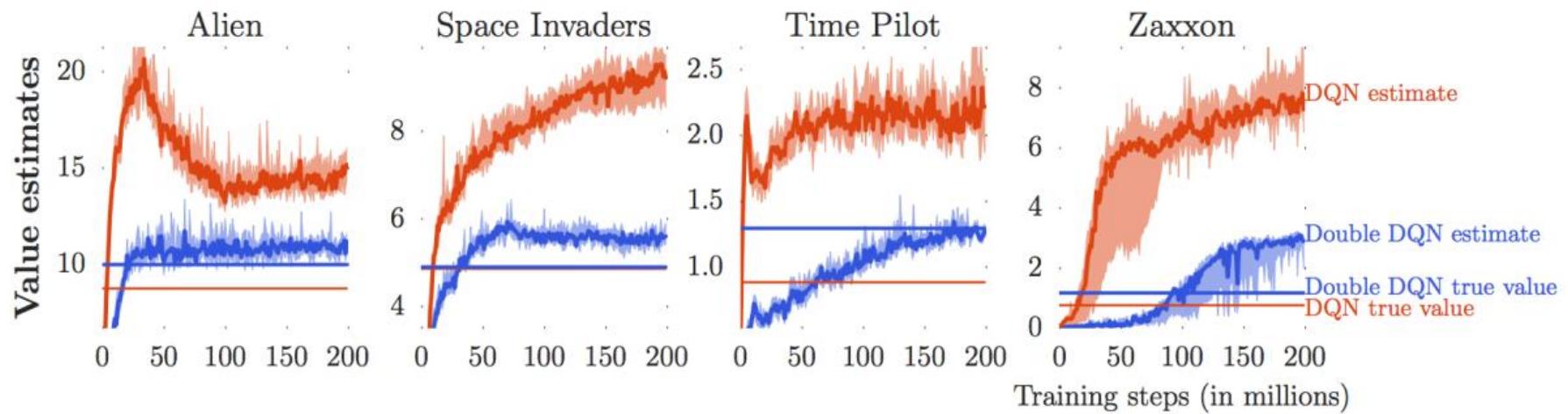
A more general view

Q-learning with replay buffer and target network:

1. save target network parameters: $\phi' \leftarrow \phi$
2. collect M datapoints $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add them to \mathcal{B}
- $N \times K \times$ 3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from \mathcal{B}
4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$



Accuracy of Q Values



DQN overestimates the Q-values--this can also be improved by adding a second target network

Improvements

- ▶ **Double DQN:** Remove upward bias caused by $\max_a Q(s, a, \mathbf{w})$
 - ▶ Current Q-network \mathbf{w} is used to **select** actions
 - ▶ Older Q-network \mathbf{w}^- is used to **evaluate** actions

$$l = \left(r + \gamma \underset{a'}{\operatorname{argmax}} Q(s', a', \mathbf{w}), \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

- ▶ **Prioritised replay:** Weight experience according to surprise
 - ▶ Store experience in priority queue according to DQN error

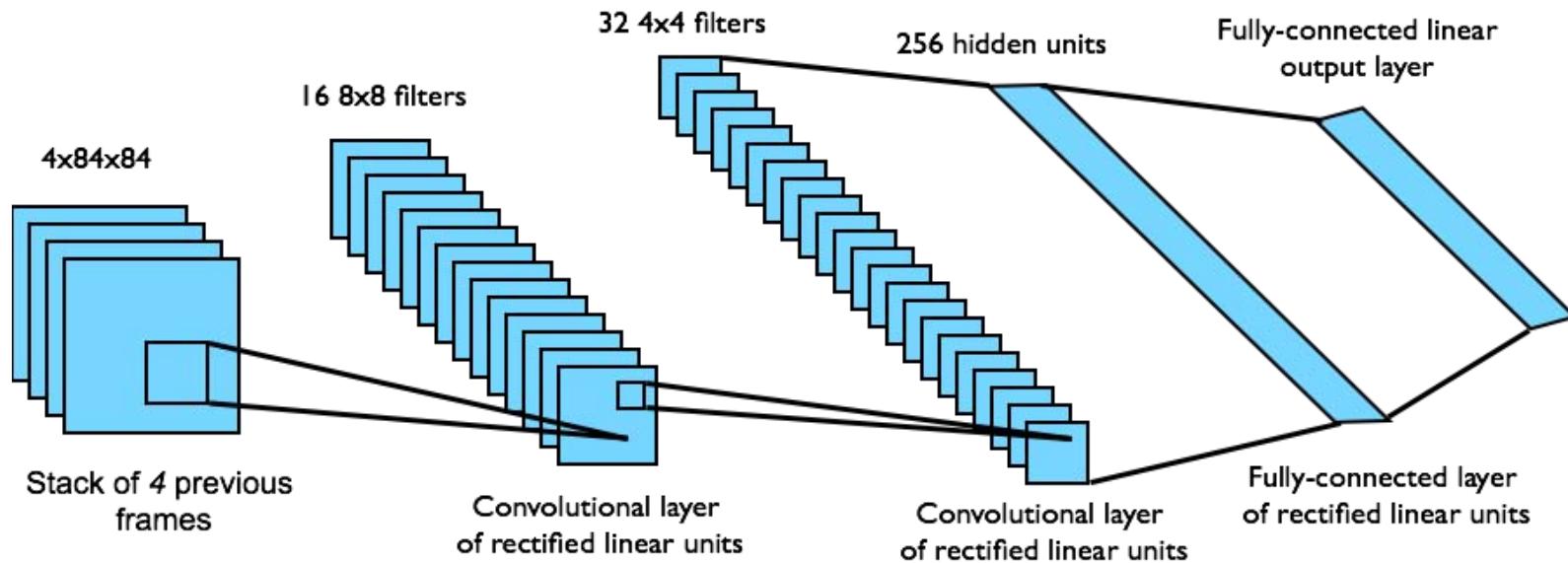
$$\left| r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right|$$

- ▶ **Duelling network:** Split Q-network into two channels
 - ▶ Action-independent **value function** $V(s, v)$
 - ▶ Action-dependent **advantage function** $A(s, a, \mathbf{w})$

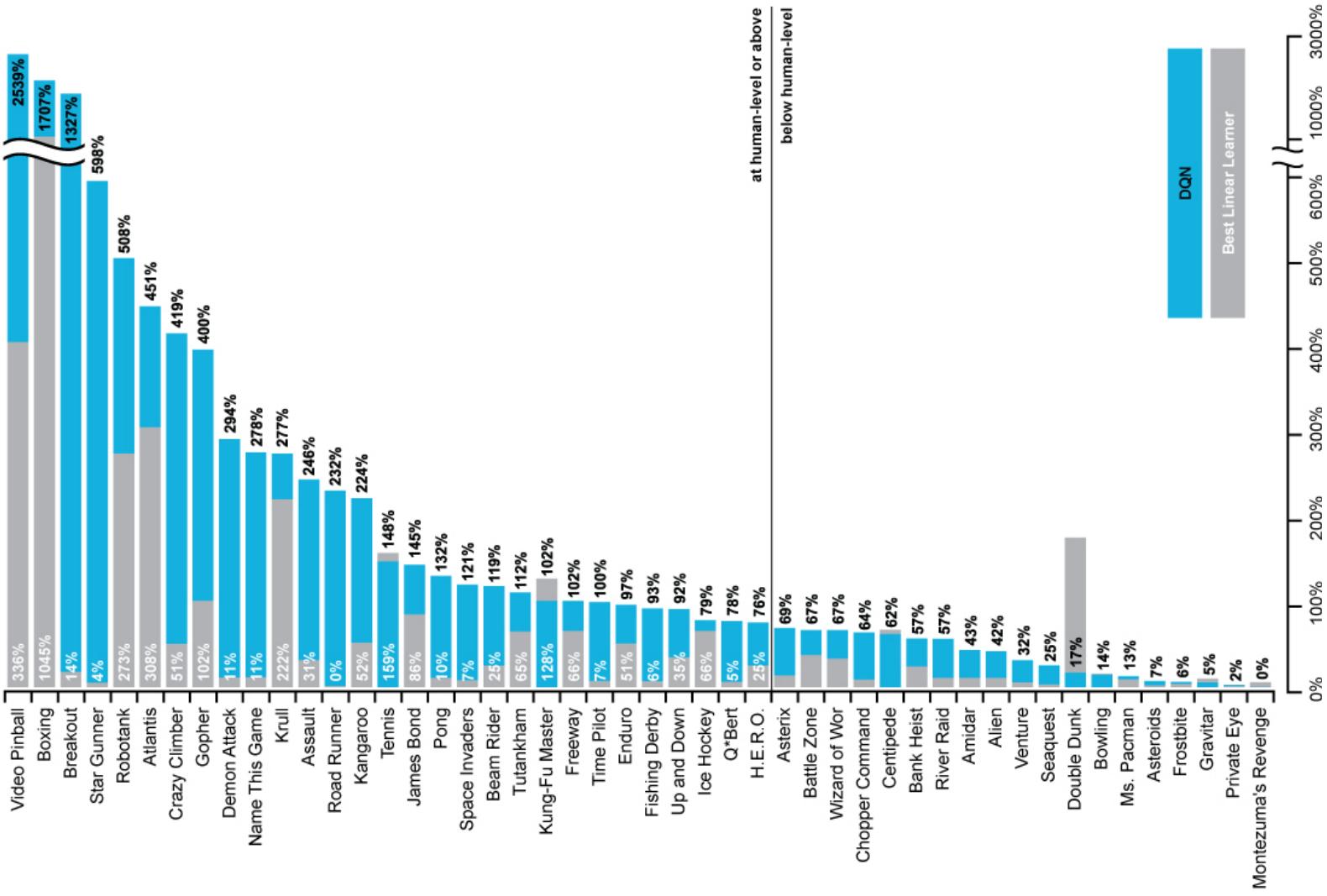
$$Q(s, a) = V(s, v) + A(s, a, \mathbf{w})$$

DQN in Atari

- End to end learning of $Q(s,a)$ from pixels
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s,a)$ for 18 joystick/button positions
- Reward is change in score



DQN Results



Deep Policy Networks

- Represent policy by deep network with weights \mathbf{u}

$$a = \pi(a|s, \mathbf{u}) \text{ or } a = \pi(s, \mathbf{u})$$

- Objective function: total discounted reward

$$L(\mathbf{u}) = \mathbb{E} [r_1 + \gamma r_2 + \gamma^2 r_3 + \dots | \pi(\cdot, \mathbf{u})]$$

- Optimize objective with stochastic gradient descent
- Adjust policy parameters

Policy Gradients

How to make high-value actions more likely:

- ▶ The gradient of a stochastic policy $\pi(a|s, \mathbf{u})$ is given by

$$\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \mathbb{E} \left[\frac{\partial \log \pi(a|s, \mathbf{u})}{\partial \mathbf{u}} Q^\pi(s, a) \right]$$

- ▶ The gradient of a deterministic policy $a = \pi(s)$ is given by

$$\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \mathbb{E} \left[\frac{\partial Q^\pi(s, a)}{\partial a} \frac{\partial a}{\partial \mathbf{u}} \right]$$

- ▶ if a is continuous and Q is differentiable

Direct Policy Gradient

$$\theta^* = \arg \max_{\theta} J(\theta)$$

$$J(\theta) = E_{\tau \sim \pi_\theta(\tau)}[r(\tau)]$$

$$\nabla_{\theta} J(\theta) = E_{\tau \sim \pi_\theta(\tau)}[\nabla_{\theta} \log \pi_{\theta}(\tau) r(\tau)]$$

$$\nabla_{\theta} \left[\cancel{\log p(\mathbf{s}_1)} + \sum_{t=1}^T \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) + \cancel{\log p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)} \right]$$

$$\nabla_{\theta} J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[\left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) \right) \left(\sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

log of both sides

$$\begin{aligned} \overbrace{\pi_{\theta}(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T, \mathbf{a}_T)}^{\pi_{\theta}(\tau)} &= p(\mathbf{s}_1) \prod_{t=1}^T \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) \\ \log \pi_{\theta}(\tau) &= \log p(\mathbf{s}_1) + \sum_{t=1}^T \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) + \log p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) \end{aligned}$$

REINFORCE

Evaluating the policy gradient

recall: $J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right] \approx \frac{1}{N} \sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$

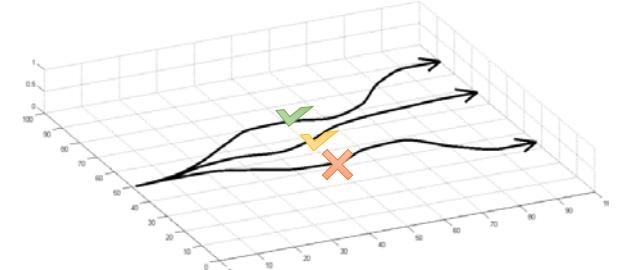
$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[\left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) \right) \left(\sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^T r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

REINFORCE algorithm:

- 1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$ (run the policy)
- 2. $\nabla_\theta J(\theta) \approx \sum_i (\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i | \mathbf{s}_t^i)) (\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i))$
- 3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$



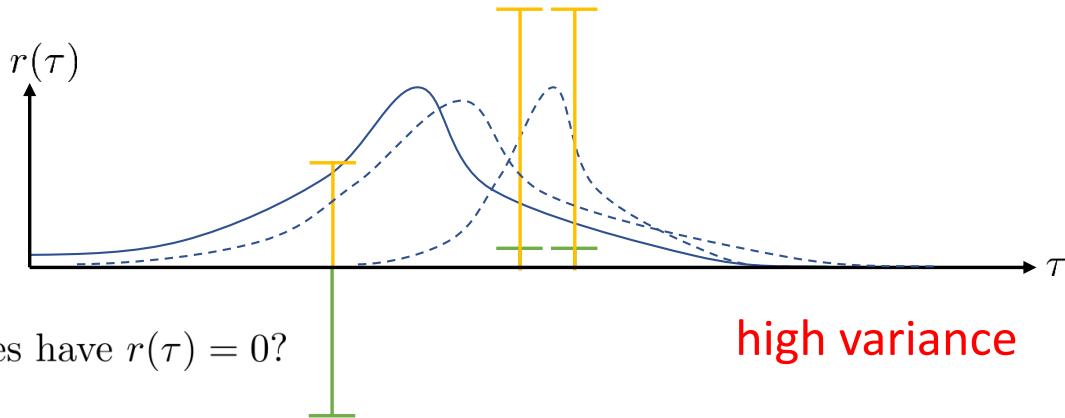
fit a model to estimate return

generate samples
(i.e. run the policy)

improve the policy

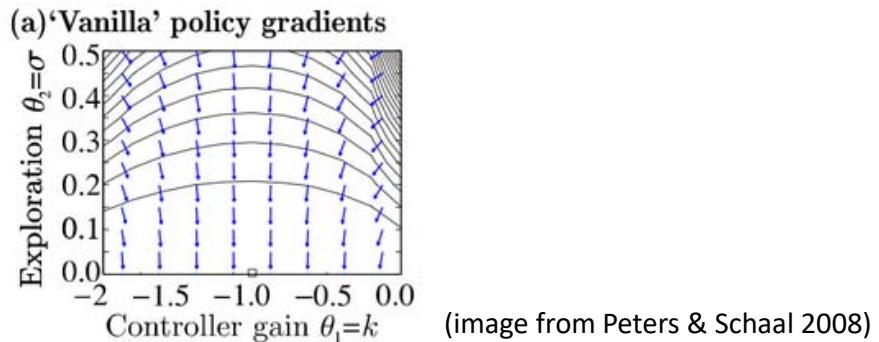
Problems with PG

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \nabla_{\theta} \log \pi_{\theta}(\tau) r(\tau)$$



$$\log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) = -\frac{1}{2\sigma^2} (\mathbf{k}\mathbf{s}_t - \mathbf{a}_t)^2 + \text{const} \quad \theta = (\mathbf{k}, \sigma)$$
$$r(\mathbf{s}_t, \mathbf{a}_t) = -\mathbf{s}_t^2 - \mathbf{a}_t^2$$

slow convergence
hard to choose learning rate



Actor-Critic

- ▶ Estimate value function $Q(s, a, \mathbf{w}) \approx Q^\pi(s, a)$
- ▶ Update policy parameters \mathbf{u} by stochastic gradient ascent

$$\frac{\partial l}{\partial \mathbf{u}} = \frac{\partial \log \pi(a|s, \mathbf{u})}{\partial \mathbf{u}} Q(s, a, \mathbf{w})$$

or

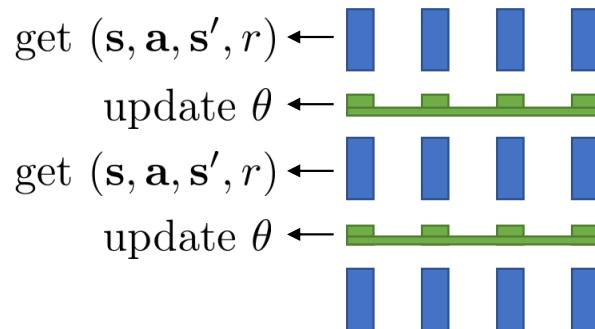
$$\frac{\partial l}{\partial \mathbf{u}} = \frac{\partial Q(s, a, \mathbf{w})}{\partial a} \frac{\partial a}{\partial \mathbf{u}}$$

Actor-Critic

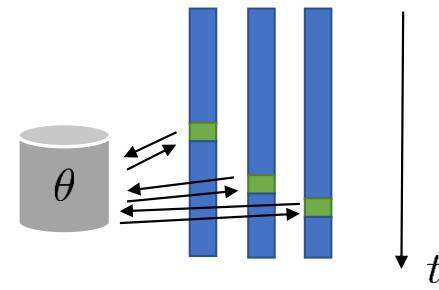
online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update \hat{V}_ϕ^π using target $r + \gamma \hat{V}_\phi^\pi(\mathbf{s}') \leftarrow$ works best with a batch (e.g., parallel workers)
3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}') - \hat{V}_\phi^\pi(\mathbf{s})$
4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a}) \leftarrow$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

synchronized parallel actor-critic



asynchronous parallel actor-critic



Asynchronous Advantage AC

- ▶ Estimate state-value function

$$V(s, \mathbf{v}) \approx \mathbb{E} [r_{t+1} + \gamma r_{t+2} + \dots | s]$$

- ▶ Q-value estimated by an n -step sample

$$q_t = r_{t+1} + \gamma r_{t+2} \dots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n}, \mathbf{v})$$

- ▶ Actor is updated towards target

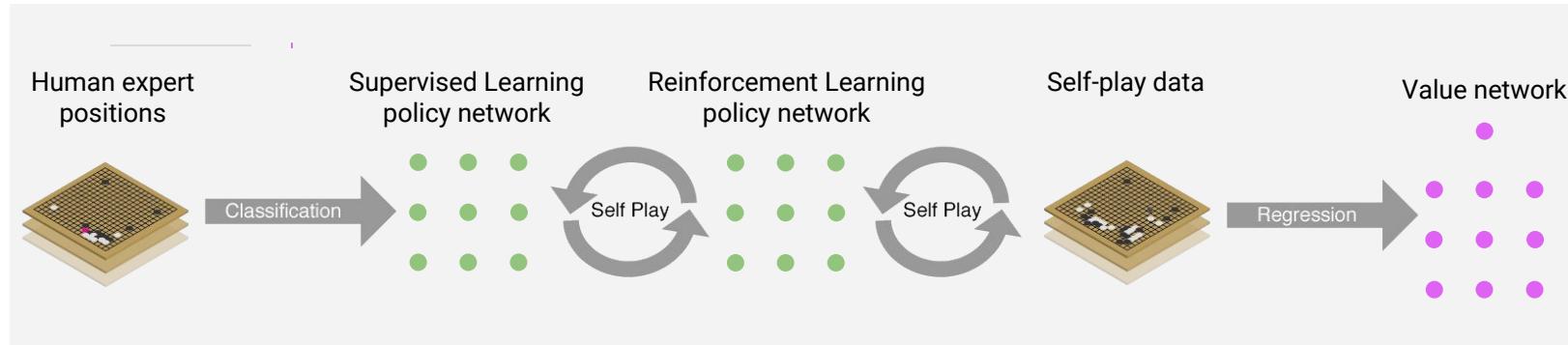
$$\frac{\partial l_u}{\partial \mathbf{u}} = \frac{\partial \log \pi(a_t | s_t, \mathbf{u})}{\partial \mathbf{u}} (q_t - V(s_t, \mathbf{v}))$$

- ▶ Critic is updated to minimise MSE w.r.t. target

$$l_v = (q_t - V(s_t, \mathbf{v}))^2$$

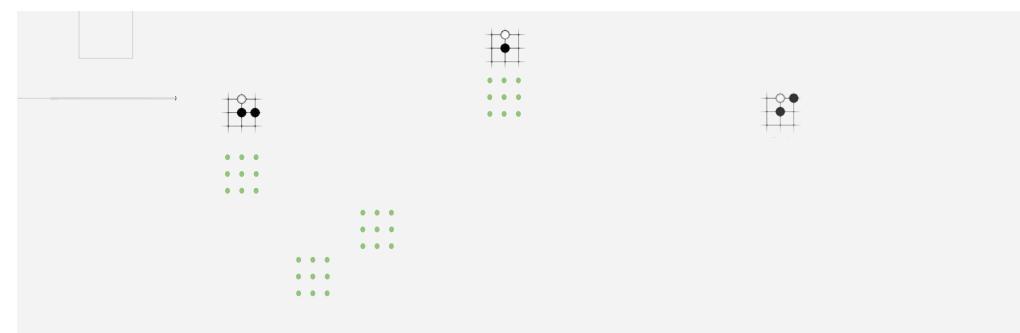
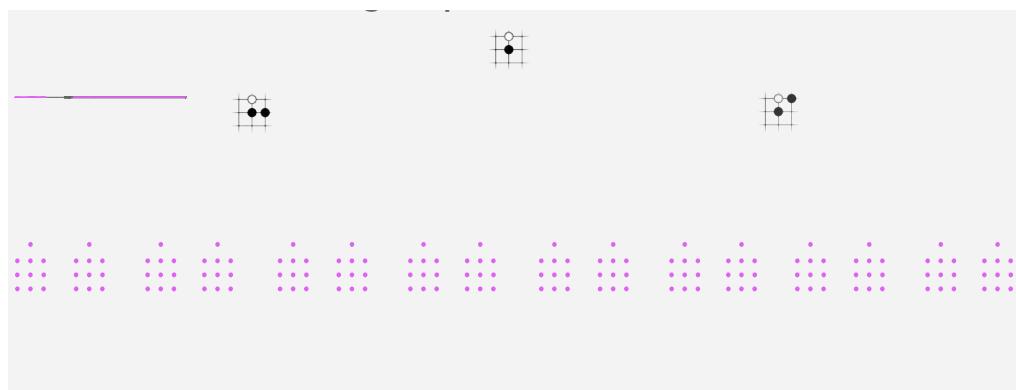
- ▶ 4x mean Atari score vs Nature DQN

AlphaGo



Reduce depth with value network

Reduce breadth with policy network



Deep DPG

DPG is the continuous analogue of DQN

- ▶ **Experience replay**: build data-set from agent's experience
- ▶ **Critic** estimates value of current policy by DQN

$$l_w = \left(r + \gamma Q(s', \pi(s', \mathbf{u}^-), \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

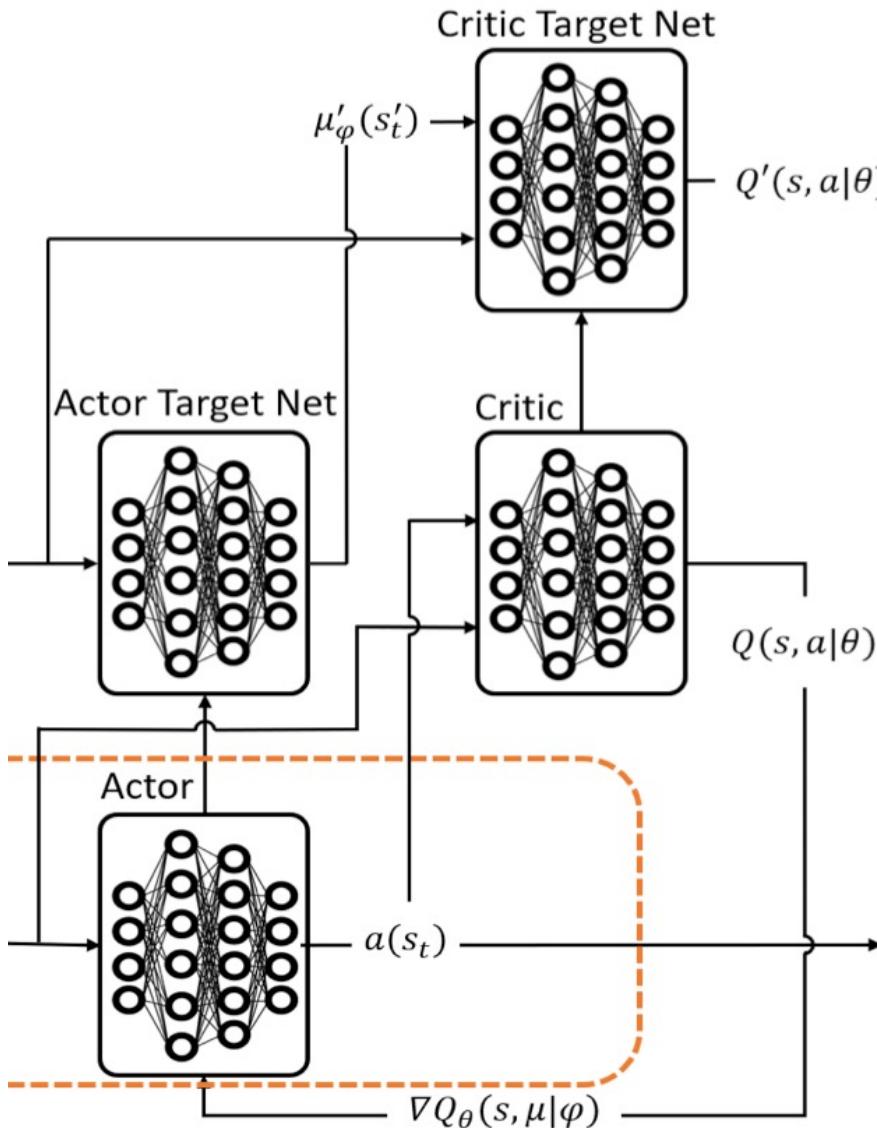
To deal with non-stationarity, targets \mathbf{u}^- , \mathbf{w}^- are held fixed

- ▶ **Actor** updates policy in direction that improves Q

$$\frac{\partial l_u}{\partial \mathbf{u}} = \frac{\partial Q(s, a, \mathbf{w})}{\partial a} \frac{\partial a}{\partial \mathbf{u}}$$

- ▶ In other words critic provides loss function for actor

DDPG



Transition State:
 (s, a, r, s', d)

Optimal Policy:

$$Q_{\theta}^*(s, a) = E_{s' \sim P} [r(s, a) + \gamma \max_{a'} Q_{\theta}^*(s', a')]$$

Compute targets:

$$y(r, s', d) = r + \gamma(1 - d)Q_{\theta}^{target}(s', \mu_{\varphi}^{target})$$

Q – value gradient descent update

$$L(\theta, N) = E_{(s, a, r, s', d) \sim N \sim D} [y(r, s', d) - Q_{\theta}(s, a)]$$

Policy gradient ascent update

$$\mu_{\varphi}(s) = \max_{\varphi} E_{s \sim N} [Q_{\theta}(s, \mu_{\varphi})]$$

Conclusion

- RL is tricky because what you do changes your evaluation and your evaluation changes what you do!
- Intuition: each term in the RL equations are replaced by a network that estimates that value
- Replay buffer allows us to turn the simulator into a dataset
- Source and target networks
- Lots of tricks required to make deep RL work!

CAP6671 Intelligent Systems

Lecture 13: Transfer Learning ROS (Robot Operating System)

Instructor: Dr. Gita Sukthankar

Email: gitars@eeecs.ucf.edu

Reminder

- Please finish the agent competition assignment by Mon!

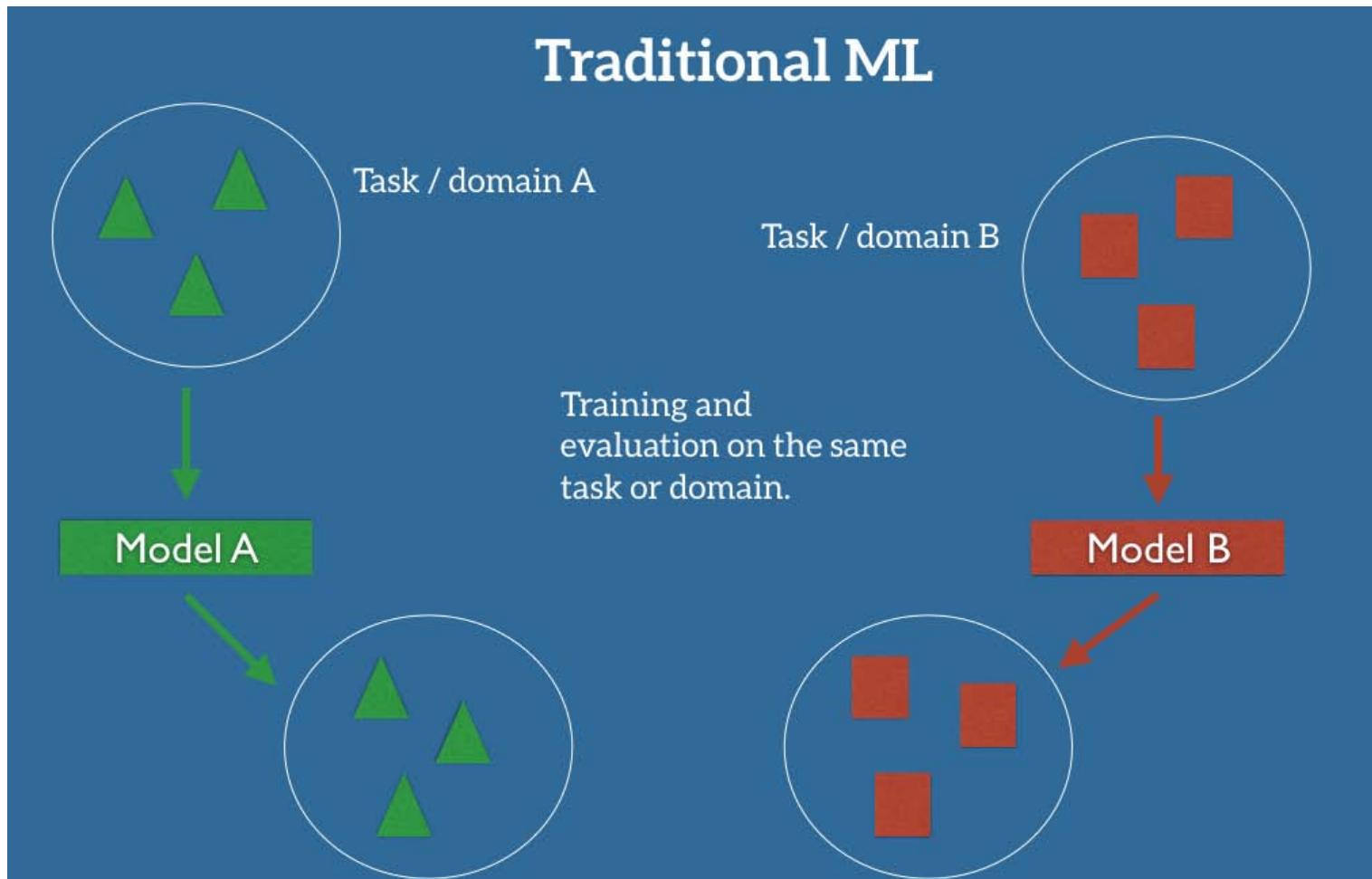
Reading to discuss:

- Matthew E. Taylor and Peter Stone. [Cross-Domain Transfer for Reinforcement Learning](#). In Proceedings of the Twenty-Fourth International Conference on Machine Learning, June 2007 (contains some of the same material)
- Pan and Yang, Survey on Transfer Learning, IEEE Transactions on KDE, 2009
- Transfer Learning <https://www.ruder.io/transfer-learning/>

Transfer Learning

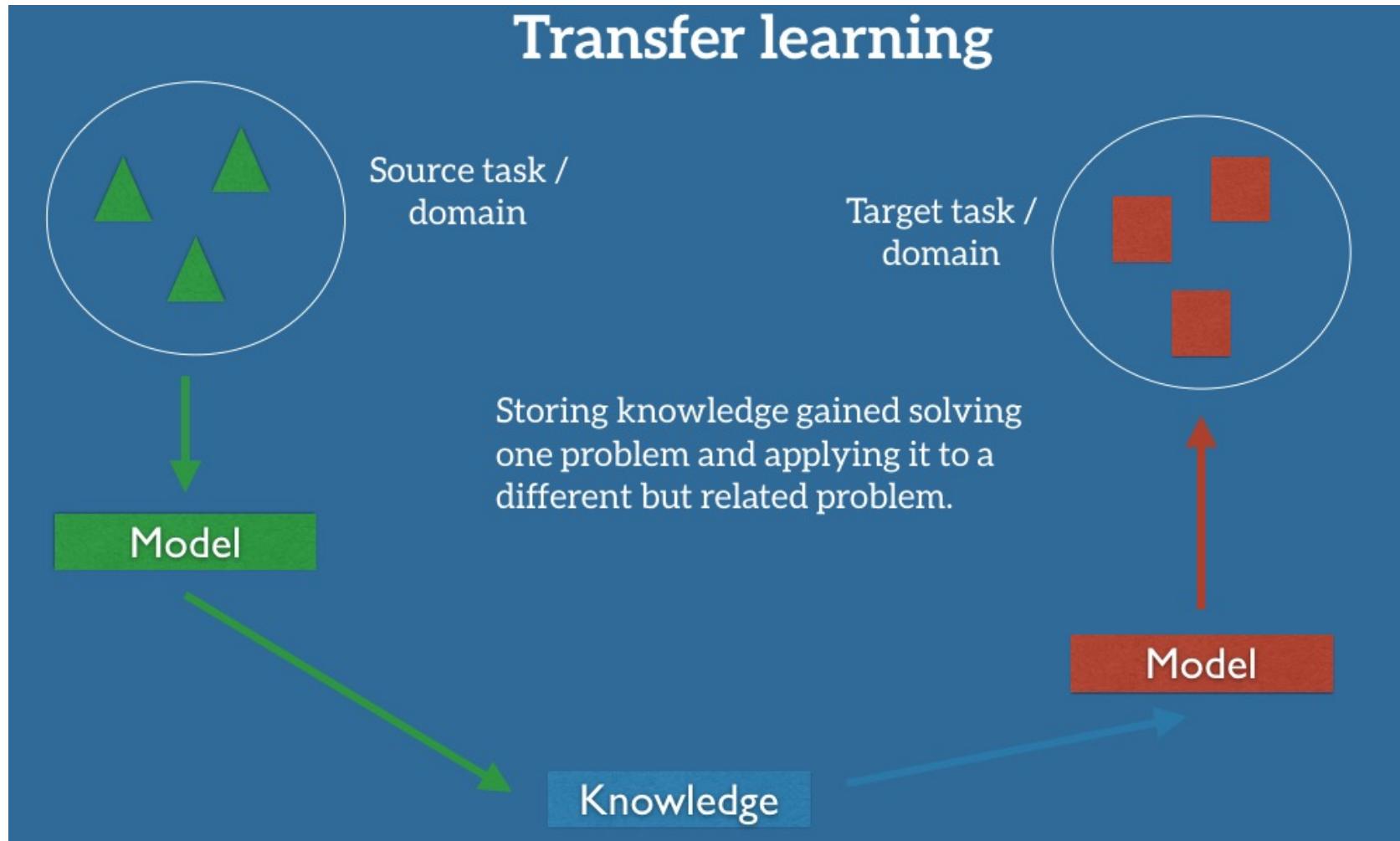
- Definition: Transfer learning (TL) is the practice of recognizing and applying knowledge and skills learned from one or more previous tasks to more efficiently and/or effectively learn to solve novel tasks (in new domains).
- Or: Situation where what has been learned in one setting is exploited to improve generalization in another setting.
- There are many techniques that can be used to do the transfer; no single dominant method has emerged yet.

Basic ML Paradigm



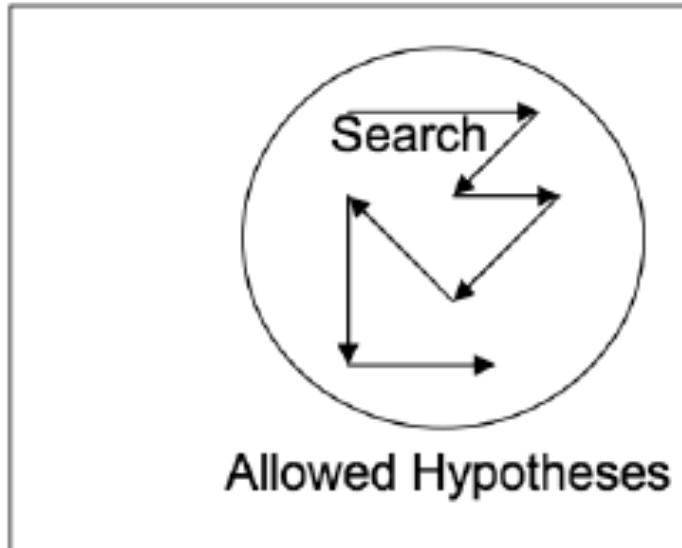
Transfer learning changes this paradigm.....

Transfer Learning



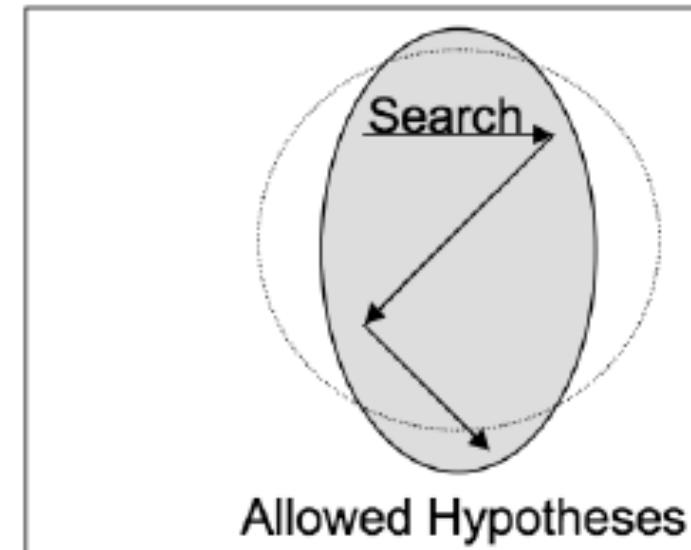
Bias the Search

Inductive Learning



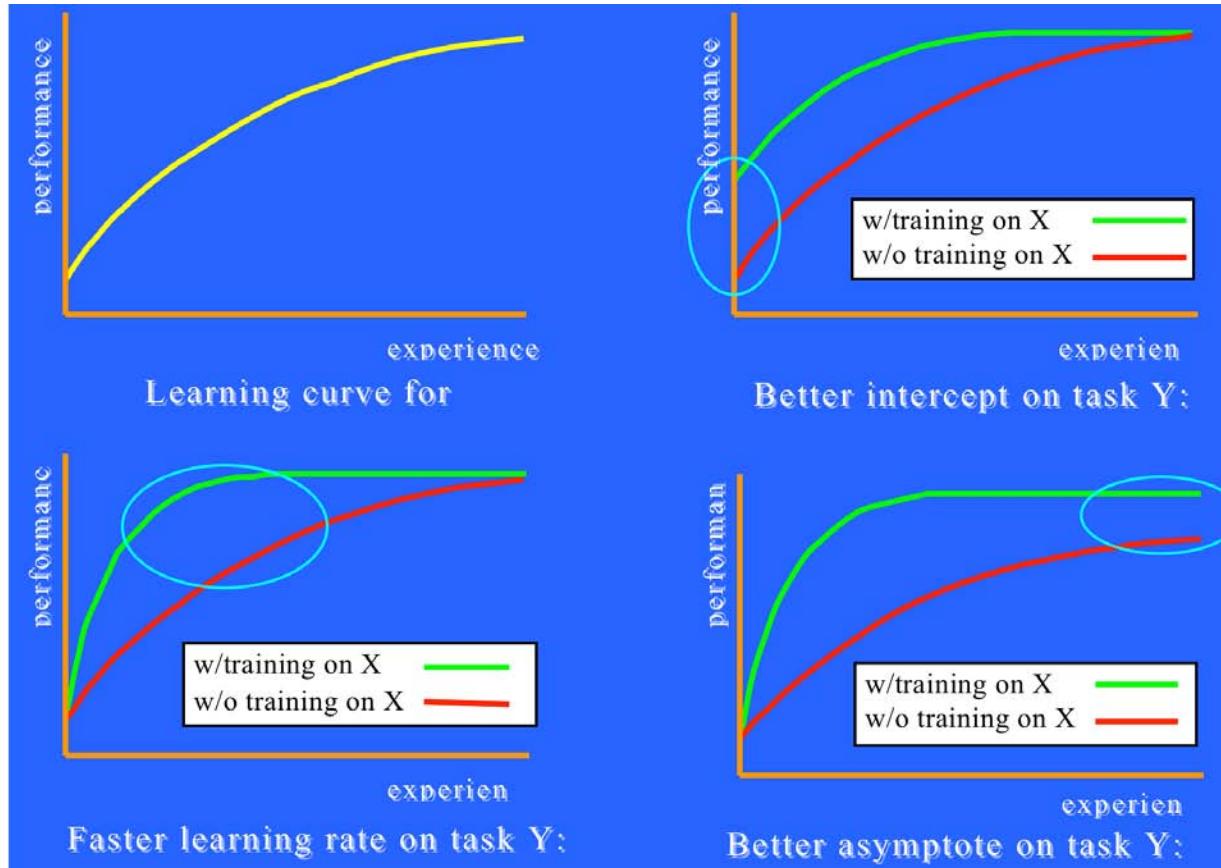
All Hypotheses

Inductive Transfer

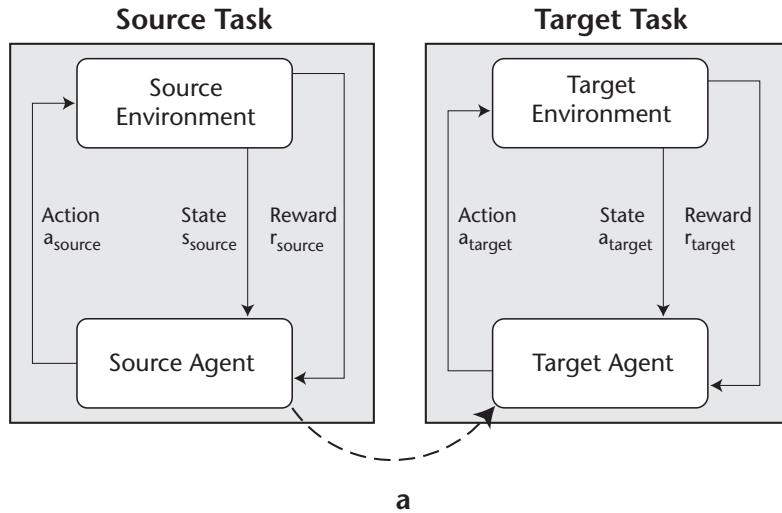


All Hypotheses

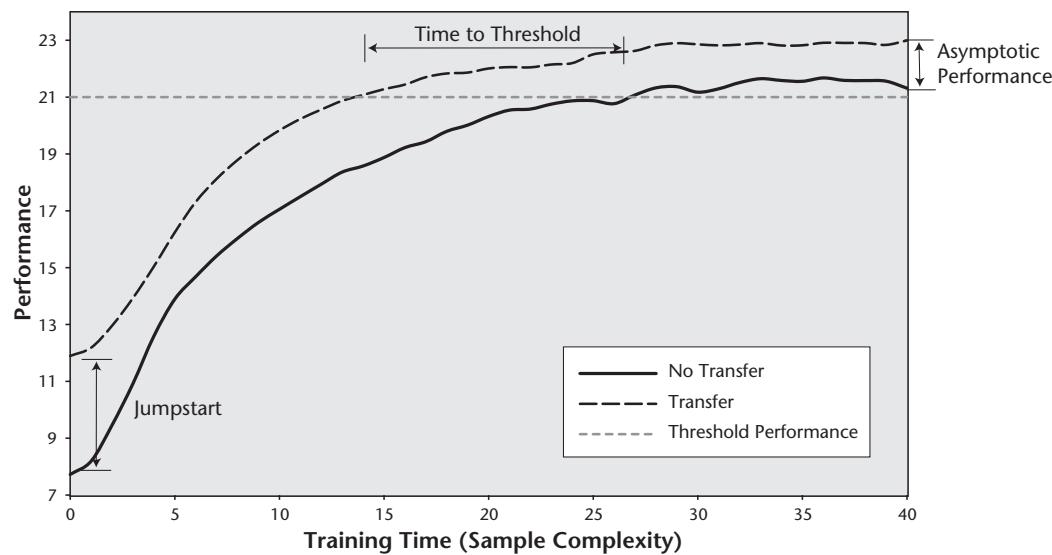
Evaluation Metrics



Transfer Learning in RL

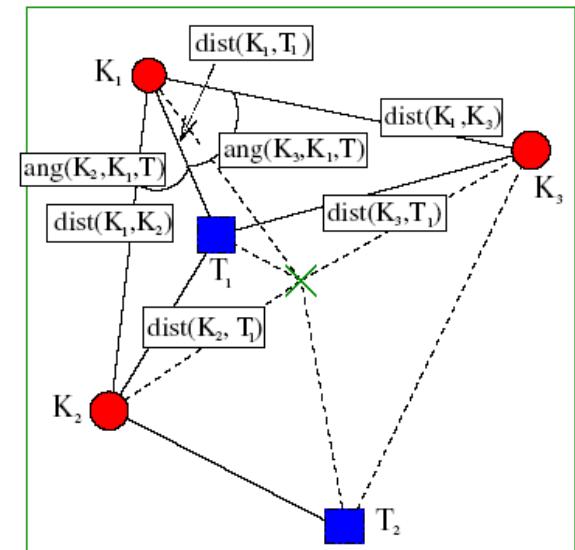
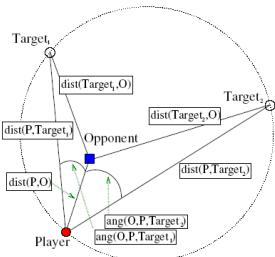
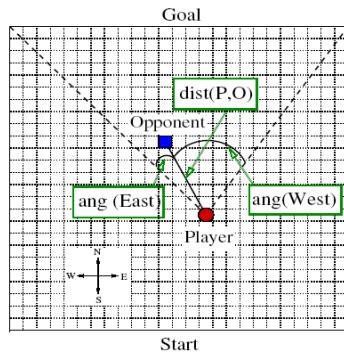


a

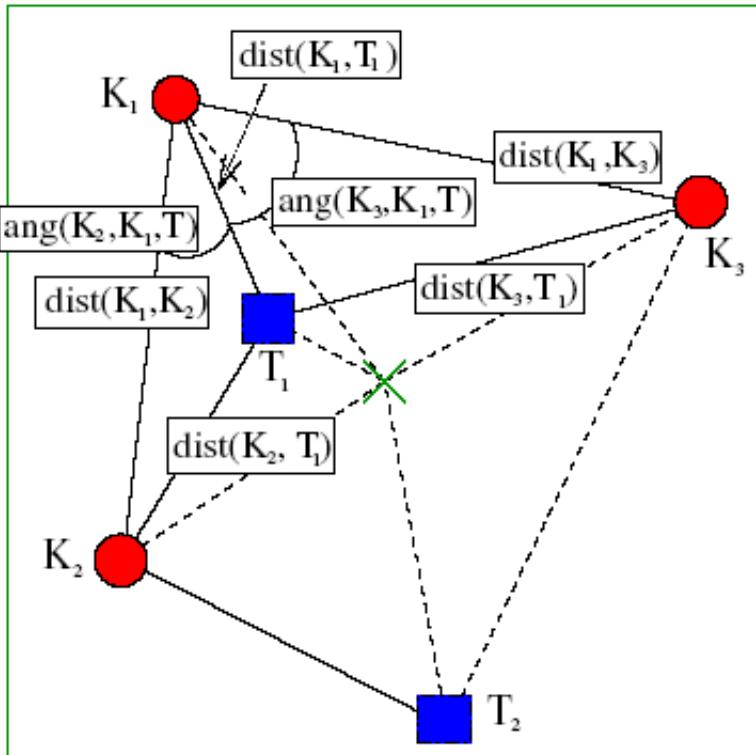


TL in Robocup

- How can we transfer learning from other types of tasks to Robocup or Robocup subtasks like Keepaway?

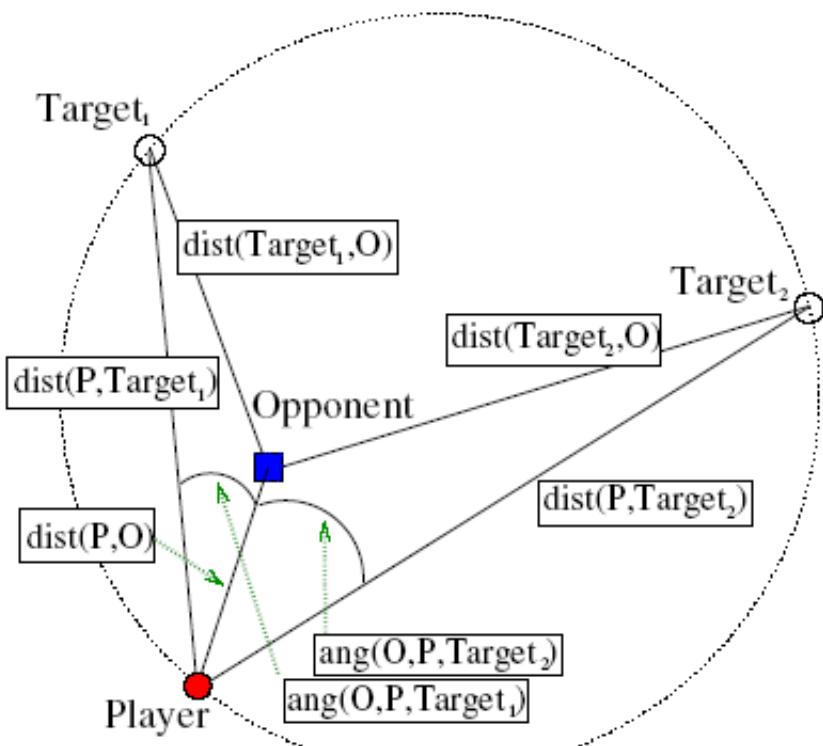


Keepaway



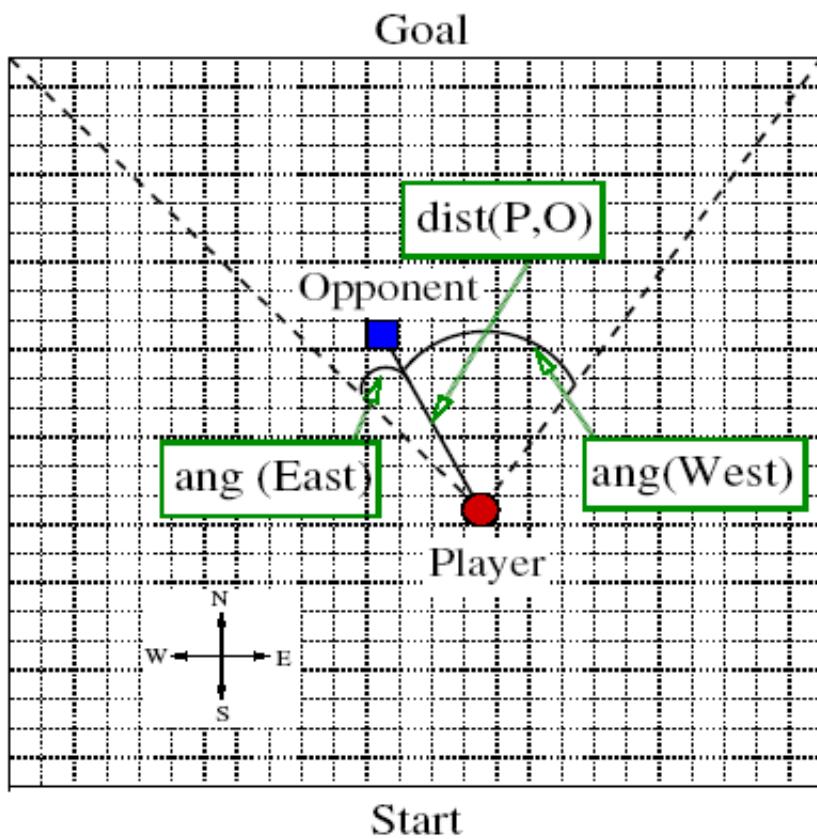
- 3 Keepers prevent 2 Takers from intercepting the ball
- Learn policy for Keeper holding the ball
- A={hold, Pass1, Pass2}
- Takers follow fixed strategy
- Keepers without ball either 1) attempt to capture an open ball or 2) get free for a pass
- Reward per timestep ball is in play
- Simulated noise in perception

Ringworld



- Opponent moves towards player on every timestep
- Player can either stay in current location or run towards a target
- As opponent approaches player the probability of the player being tagged increases
- $A = \{\text{Stay}, \text{RunNear}, \text{RunFar}\}$
- Size of ring/prob of tagging chosen to be similar to Keepaway
- Reward per timestep

Knight Joust



- Players alternate moves on a grid board
- Player can either move directly north or knight's move east or west
- Players' moves are deterministic and opponent has a fixed stochastic policy
- Similarity: favor distance between player and opponent
- Reward for advancing distance

What method would you use?

.

Translation between Tasks

- Define hand-coded translation functions between state variables and actions

Cross-Domain Mappings for Ringworld to Keepaway

Ringworld	Keepaway
	δ_A
Stay	Hold Ball
Run_{Near}	$Pass_1$: Pass to K_2
Run_{Far}	$Pass_2$: Pass to K_3
	δ_X
$dist(P, O)$	$dist(K_1, T_1)$
$dist(P, Target_1)$	$dist(K_1, K_2)$
$dist(Target_1, O)$	$\text{Min}(dist(K_2, T_1), dist(K_2, T_2))$
$ang(O, P, Target_1)$	$\text{Min}(ang(K_2, K_1, T_1)$ $ang(K_2, K_1, T_2))$
$dist(P, Target_2)$	$dist(K_1, K_3)$
$dist(Target_2, O)$	$\text{Min}(dist(K_3, T_1), dist(K_3, T_2))$
$ang(O, P, Target_2)$	$\text{Min}(ang(K_3, K_1, T_1),$ $ang(K_3, K_1, T_2))$

Rule Utilization

- **Value Bonus:** give constant bonus to Q-value as recommended by the translated decision list
- **Extra Action:** add action to target task such that when the agent selects this pseudo-action it follows the action recommended by D (have exploration policy favor this action)
- **Extra Variable:** add extra state variable to target state description that takes on the value of the index for the action recommended by D (have exploration policy favor this action)

RL Method

- SARSA: “State-Action State-Reward-State Action”
- Learning rule uses policy action instead of max value

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \phi Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

~ Remember: standard Q-learning rule

$$Q(s, a) := Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

- Radial basis function approximation to handle continuous state space

Procedure

- Use SARSA to learn Q-function for source domain
- Learn decision list summarizing source task policy (RIPPER, rule induction algorithm)
- Use decision list to train an agent in the target domain
- Measure
 - Initial performance
 - Asymptotic performance after learning plateaus (40 simulator hours)
 - Accumulated reward (sum of average reward per hour)

RIPPER

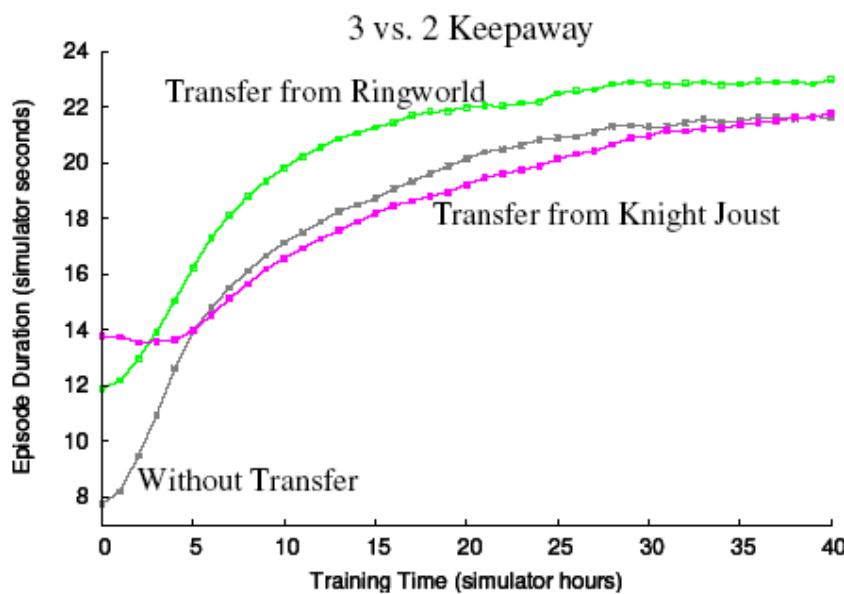
- Rule induction algorithm
- Split training data into growing set and pruning set
- GrowRule: add conditions to an empty conjunction
- PruneRule: deleting conditions from the rule to make it more general
- Example rule produced by algorithm

IF (($\text{dist}(K_1, T_1) \leq 4$) AND
 $(\text{Min}(\text{dist}(K_3, T_1), \text{dist}(K_3, T_2)) \geq 12.8)$ AND
 $(\text{ang}(K_3, K_1, T) \geq 36)$) THEN Pass to K_3

Evaluations

- Evaluate transfer from Keepaway to Keepaway to determine reasonable parameters
- Transfer of Ringworld to Keepaway produced benefits in all 3 metrics
- Transfer of KnightJoust to Keepaway only improves initial performance
- All 3 rule utilization schemes were effective with ExtraAction being slightly superior
- Also did a sensitivity analysis to show that the learning is not that dependent on parameters of RIPPER

Transfer Results



Ringworld to Keepaway			
Initial Performance	Asymptotic Performance	Accumulated Reward	
Without Transfer			
7.8 ± 0.1	21.6 ± 0.8	756.7 ± 21.8	
Added Constant Value Bonus			
5	11.1 ± 1.4	19.8 ± 0.6	722.3 ± 24.3
10	11.5 ± 1.7	22.2 ± 0.8	813.7 ± 23.6
Initial Episodes Extra Action			
100	11.9 ± 1.8	23.0 ± 0.5	842.0 ± 26.9
250	11.8 ± 1.9	23.0 ± 0.8	827.4 ± 33.0
Initial Episodes Extra Variable			
100	11.8 ± 1.9	21.9 ± 0.9	784.8 ± 27.0
250	11.7 ± 1.8	22.4 ± 0.8	793.5 ± 22.2

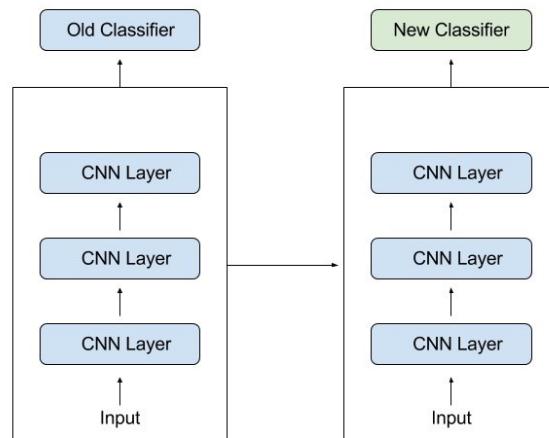
Knight Joust into Keepaway			
Param	Initial Performance	Asymptotic Performance	Accumulated Reward
Without Transfer			
	7.8 ± 0.1	21.6 ± 0.8	756.7 ± 21.8
Extra Action			
100	13.8 ± 1.1	21.8 ± 1.2	758.5 ± 29.3
250	13.5 ± 0.9	21.6 ± 0.9	747.9 ± 25.3

Future Work

- Automatically derive the rule translation function
- General approach for deriving translation:
 - Identify state variables that are near 0 when episode ends
 - Identify variable that causes those variables to decrease
 - Construct mapping between other distances and angles
- Drawback: still limited to fairly simple instances

Transfer in Deep Learning

- Use the representation learned by one neural network as a starting point as a second machine learning system
- Example: pretrained models for image processing to do feature extraction

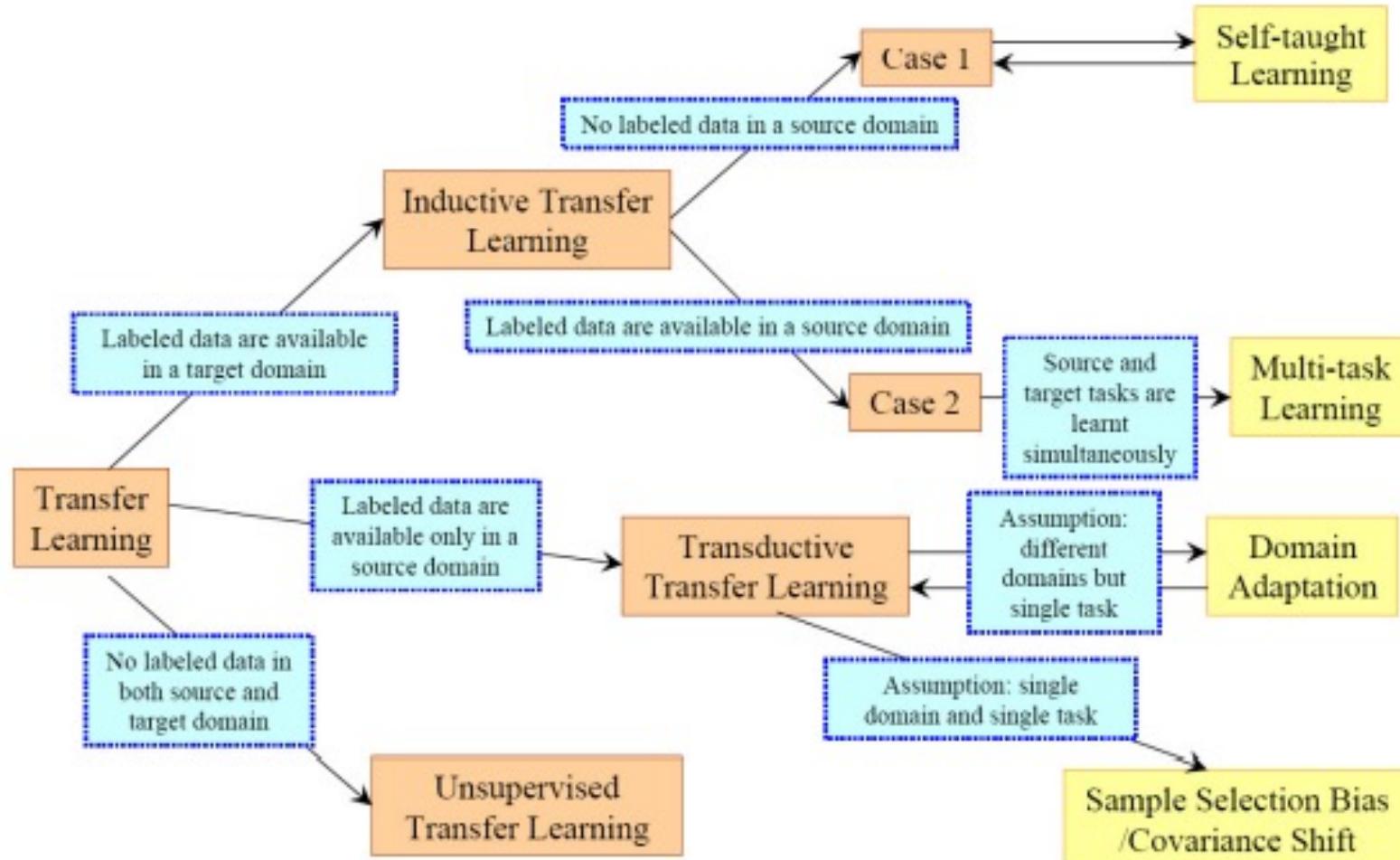


Deep Transfer

Given source and target domains \mathcal{D}_S and \mathcal{D}_T where $\mathcal{D} = \{\mathcal{X}, P(X)\}$ and source and target tasks \mathcal{T}_S and \mathcal{T}_T where $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$ source and target conditions can vary in four ways, which we will illustrate in the following again using our document classification example:

1. $\mathcal{X}_S \neq \mathcal{X}_T$. The feature spaces of the source and target domain are different, e.g. the documents are written in two different languages. In the context of natural language processing, this is generally referred to as cross-lingual adaptation.
2. $P(X_S) \neq P(X_T)$. The marginal probability distributions of source and target domain are different, e.g. the documents discuss different topics. This scenario is generally known as domain adaptation.
3. $\mathcal{Y}_S \neq \mathcal{Y}_T$. The label spaces between the two tasks are different, e.g. documents need to be assigned different labels in the target task. In practice, this scenario usually occurs with scenario 4, as it is extremely rare for two different tasks to have different label spaces, but exactly the same conditional probability distributions.
4. $P(Y_S|X_S) \neq P(Y_T|X_T)$. The conditional probability distributions of the source and target tasks are different, e.g. source and target documents are unbalanced with regard to their classes. This scenario is quite common in practice and approaches such as over-sampling, under-sampling, or SMOTE are widely used

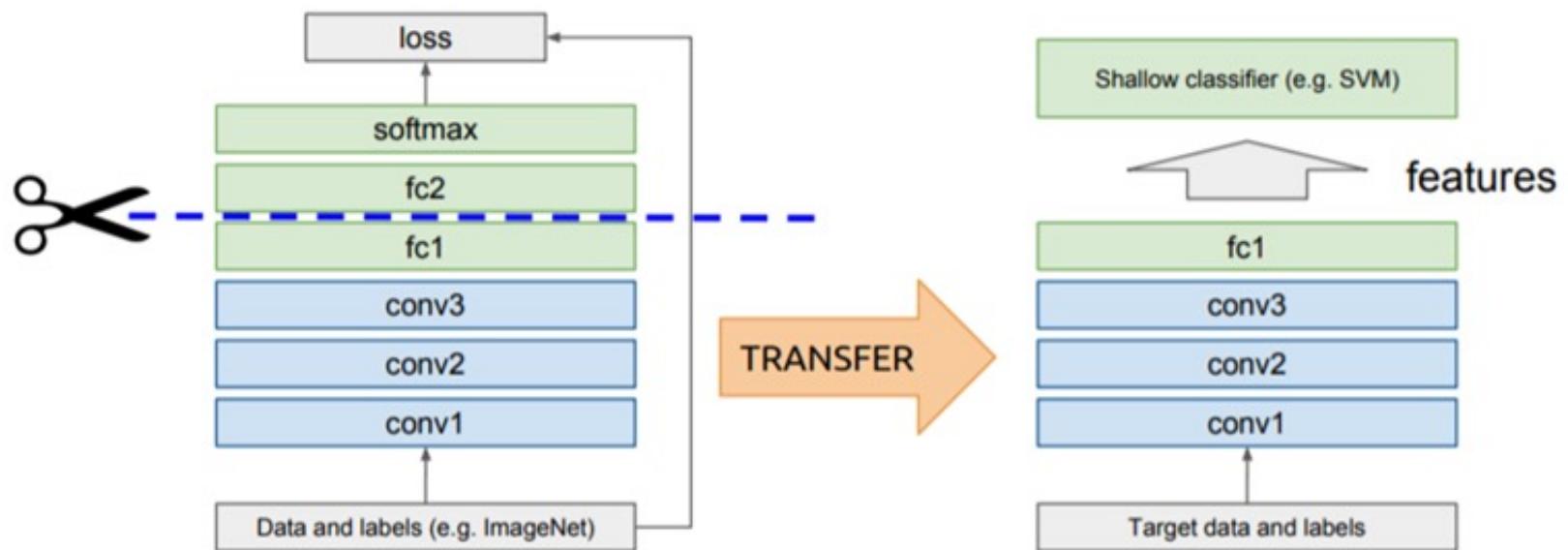
Possible Transfer Learning



Feature Extraction

Idea: use outputs of one or more layers of a network trained on a different task as generic feature detectors. Train a new shallow model on these features.

Assumes that $D_S = D_T$

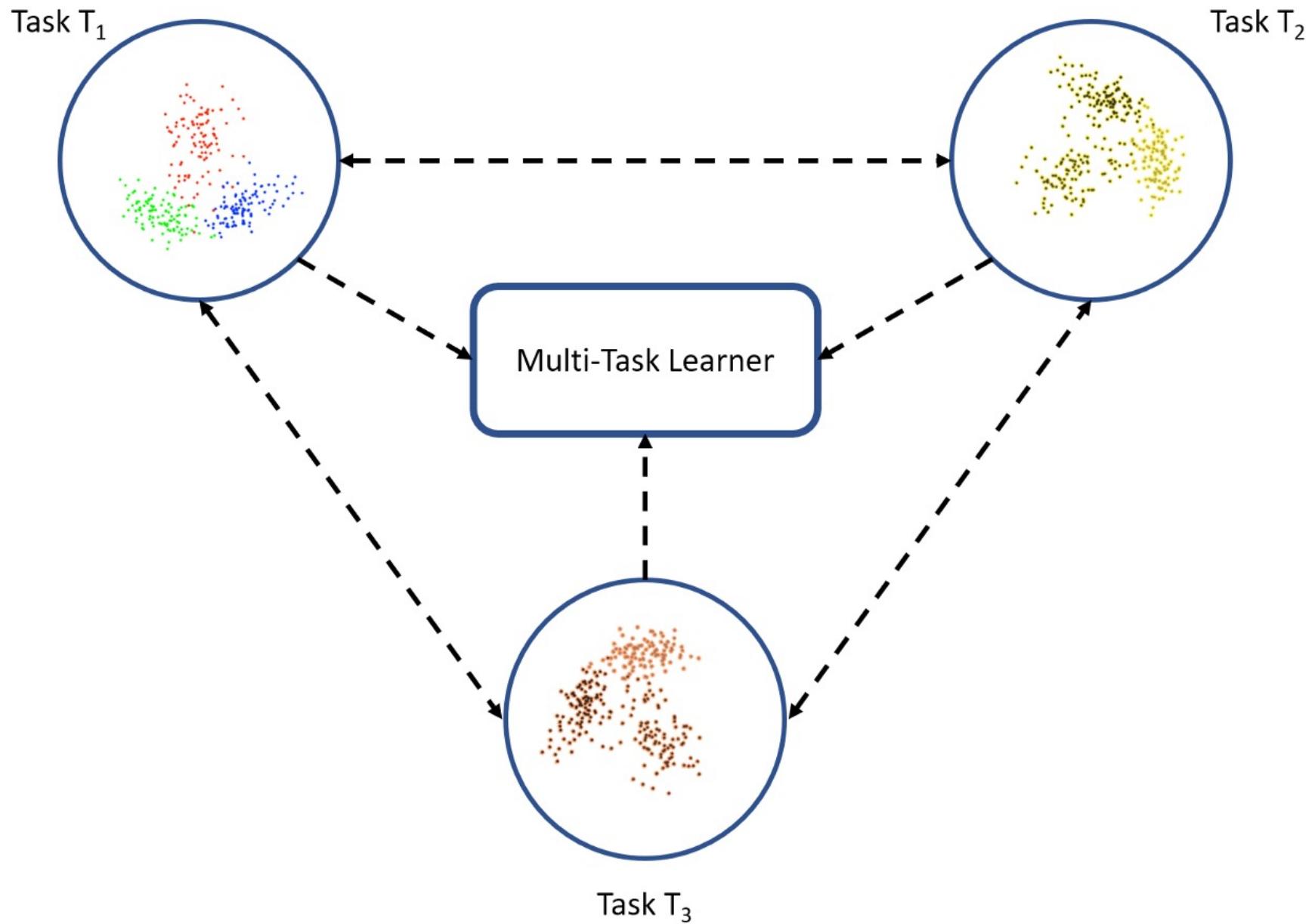


Need to decide about freezing or fine tuning

Domain Adaptation



Multi-Task Learning



Related Research

- Simulation to real learning in robotics
- Generalizability
- One shot/zero shot learning
- Continual learning

Summary

- Many of the tasks that we consider just “machine learning” can only be achieved through transfer learning.
- Still an ongoing area of research!

CAP6671: Intelligent Systems

Lecture 15: Mapping and Coverage

**Instructor: Dr. Gita Sukthankar
Email: gitars@eeecs.ucf.edu**

Paper Presentations

- Next month will feature student presentations.
 - Will be conducted over Zoom
 - **Email me your choice of date by Mar 20.**
 - Presentation dates: Mar 27, Mar 29, Apr 3, Apr 5, Apr 10, Apr 12, Apr 17
 - Select a paper written in the past 5 years that has appeared in an AI conference (not just on arxiv)
 - Present as if you were a presenter
 - 15 mins strictly timed
 - You are graded on your slides, oral presentation, and the technical depth.
 - **Email me your paper selection at least one week before you present.**

Final Project

- Due Apr 19
- Students who are working as a group should join a single final project group and submit ONE copy of their project.
- Submit key sections of CS conference paper:
 - Introduction, Method, Results
- Code repository is optional:
 - Recommend inclusion of a github link in your paper
- Paper should be formatted according to ACM/IEEE guidelines.
- A key aspect of the grading is in the evaluation section: how do you evaluate your AI system?

Optional: RL Agent

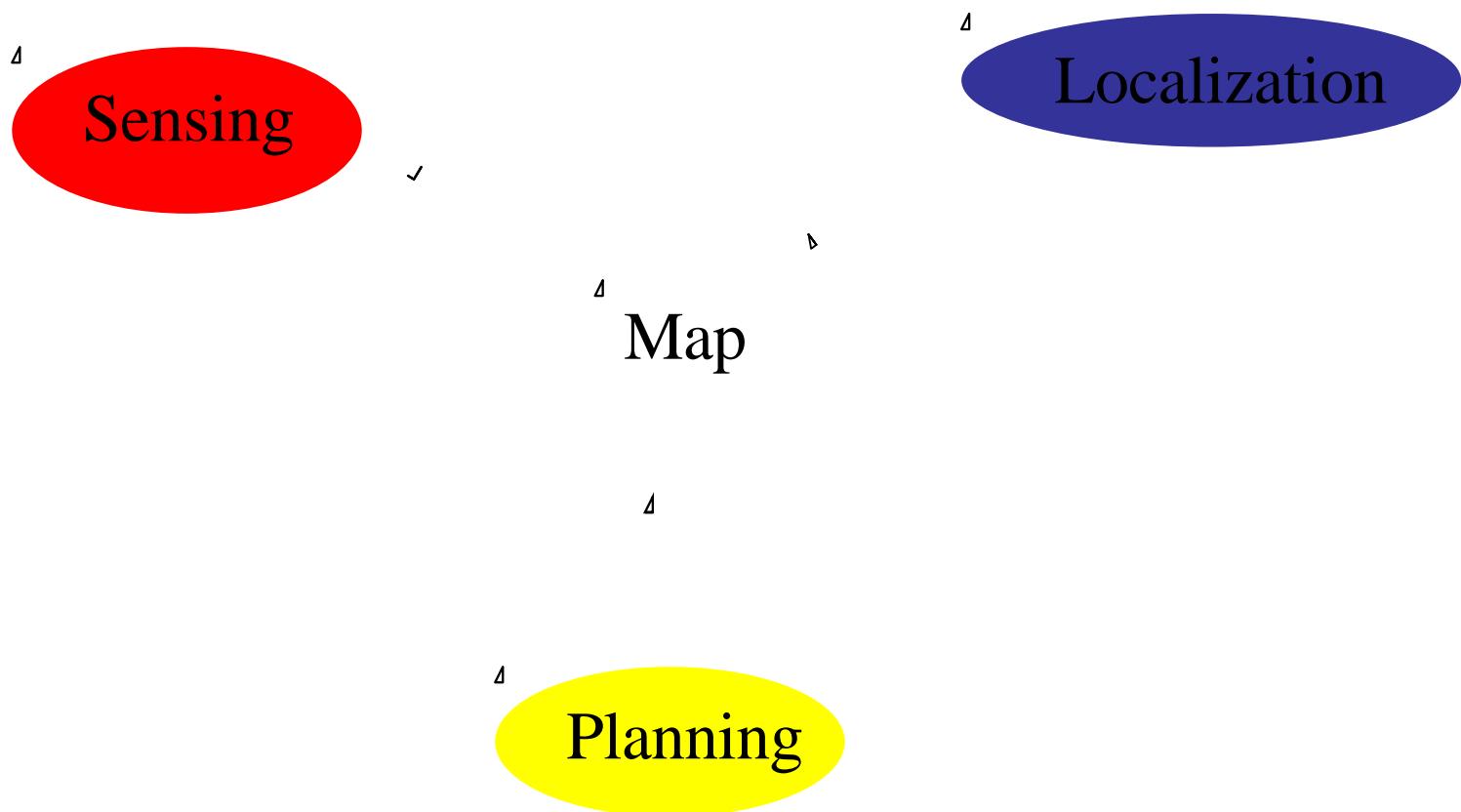
- Due Mar 29
- Over ½ the students have selected a final assignment with RL!
 - Thus, I've made the RL assignment optional.
 - Assignment grade for the course will be $\max(\text{ass1}, \text{ass2})$
 - No extra credit for completing the course
- Implement RL Agent using stable-baselines3
 - <https://github.com/DLR-RM/stable-baselines3>
 - Choose between DQN + Atari games OR robotics and DDPG

Reading

- Reading: D. Fox, W. Burgard, F. Dellaert, S. Thrun,
Monte Carlo Localization: Efficient Position Estimation for
Mobile Robots, AAAI 1999
- Reading: D. Haehnel, W. Burgard, D. Fox, and S. Thrun,
An Efficient FastSLAM Algorithm for Generating Maps of
Large-scale Cyclic Environments From Raw Laser Range
Measurements, IROS-03.

Robot Internals

Architecture



Definitions

- Mapping
- Coverage
- Localization
- Simultaneous Localization and Mapping

Definitions

- Mapping: updating the world model with information gained from the sensors
- Coverage: creating/executing a plan or policy that will guarantee (to some reasonable degree of certainty) that the robot has visited all reachable areas
- Localization: identifying the robot's location in an absolute coordinate frame on a map known in advance
- Simultaneous Localization and Mapping: localizing in an unknown region based on combining discovered landmarks with sensor data

Questions

- How would you go about mapping an area?
- What makes this task hard?

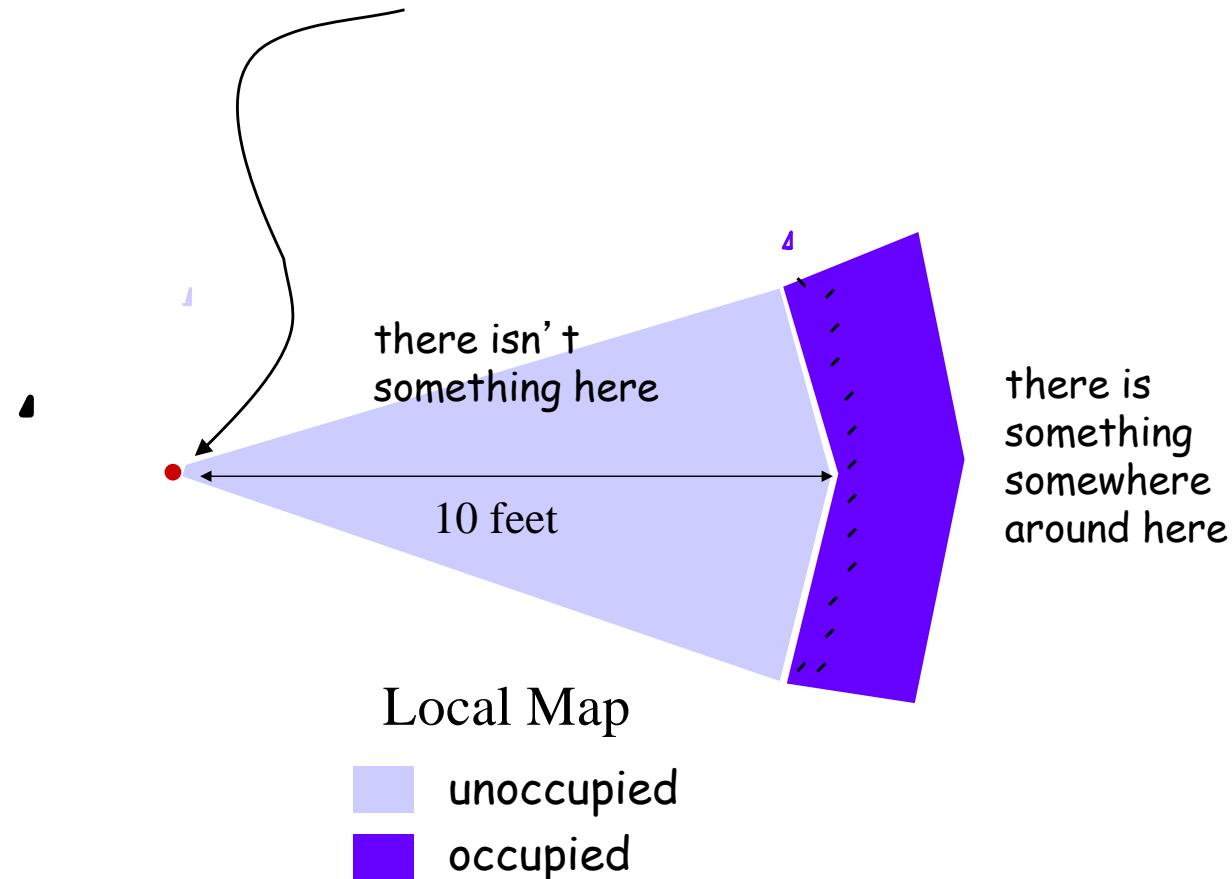
Challenges of Mapping

- Maps are highly labor-intensive to create:
 - Exploration (global coverage)
 - Measurement (local coverage)
 - Validity (correctness, error bounds)
 - Currency (freshness)
- Map creation is an ideal robotics task!
 - Achieving robust, sustained, large-area autonomous mapping capability remains an open problem

Sensory-based maps

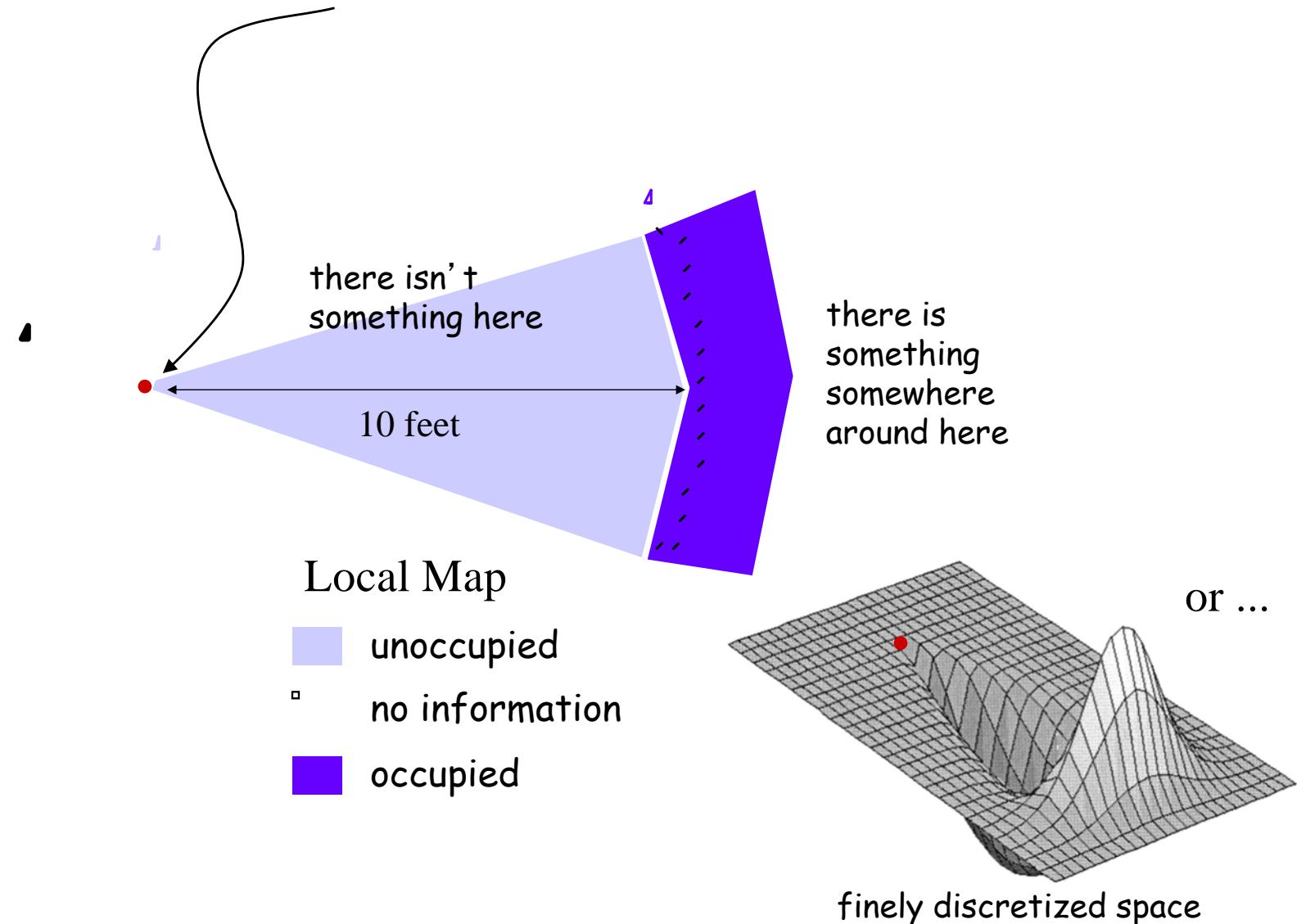


What should we conclude if this sonar reads 10 feet?



Local map (sonar-based)

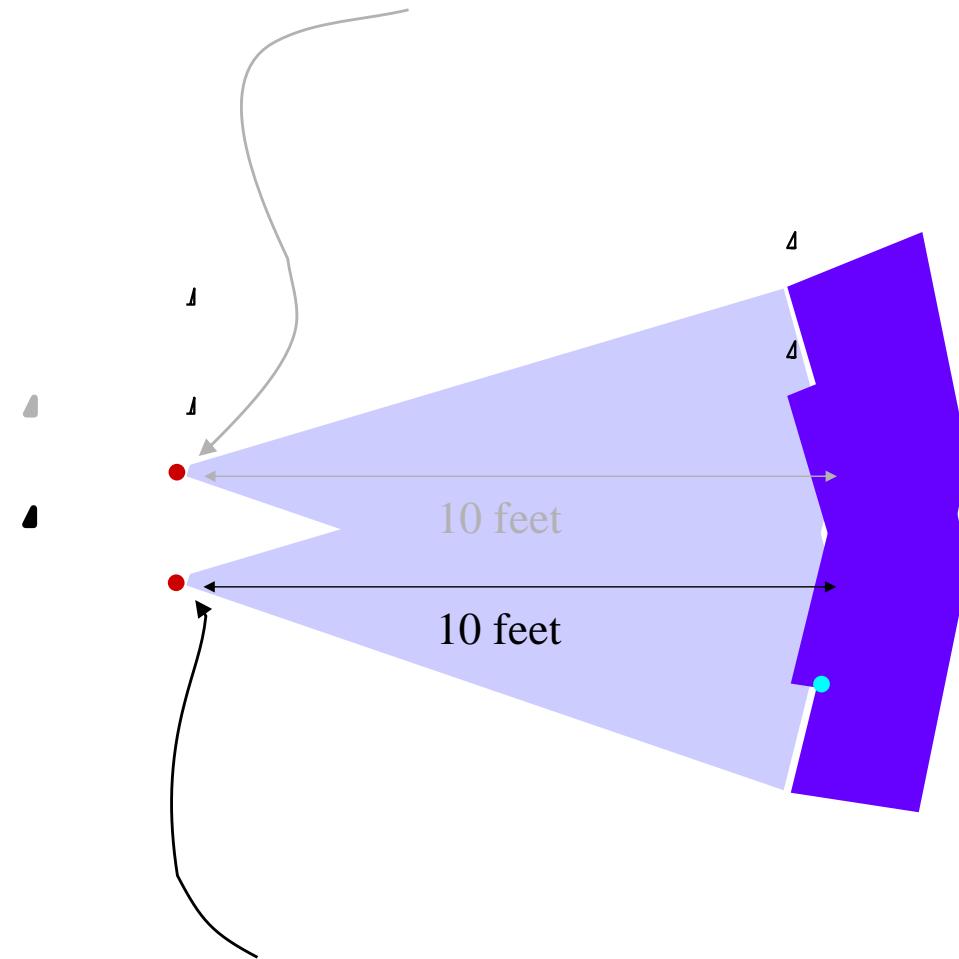
What should we conclude if this sonar reads 10 feet?



Combining maps



What should we conclude if these two offset sonars each read 10 feet... ?



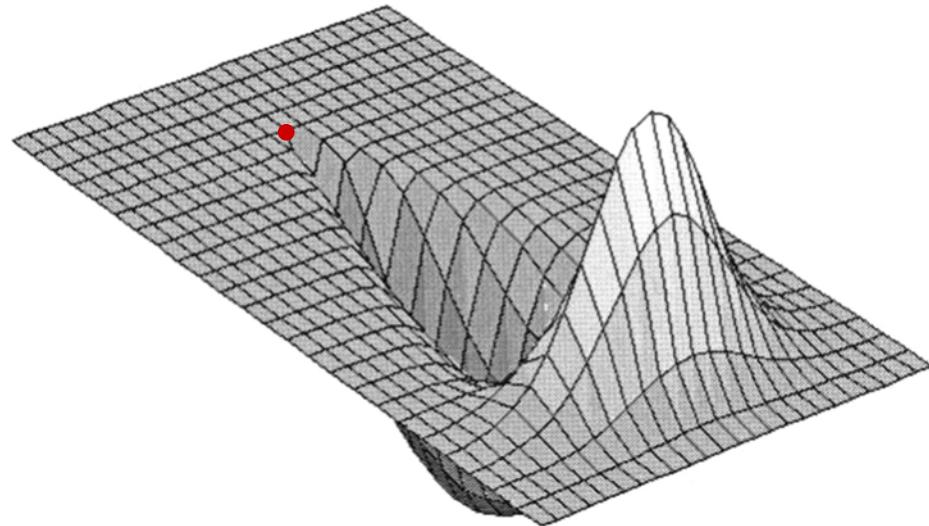
and how do we fuse these results together naturally?

How to represent space?



- The key to making accurate maps is *combining* lots of data.
- But combining these numbers means we have to decide what they are !

What should our map contain ?



what is in each cell of this local map ?

occupancy information

continuous vs. discrete ?

features vs. cells ?

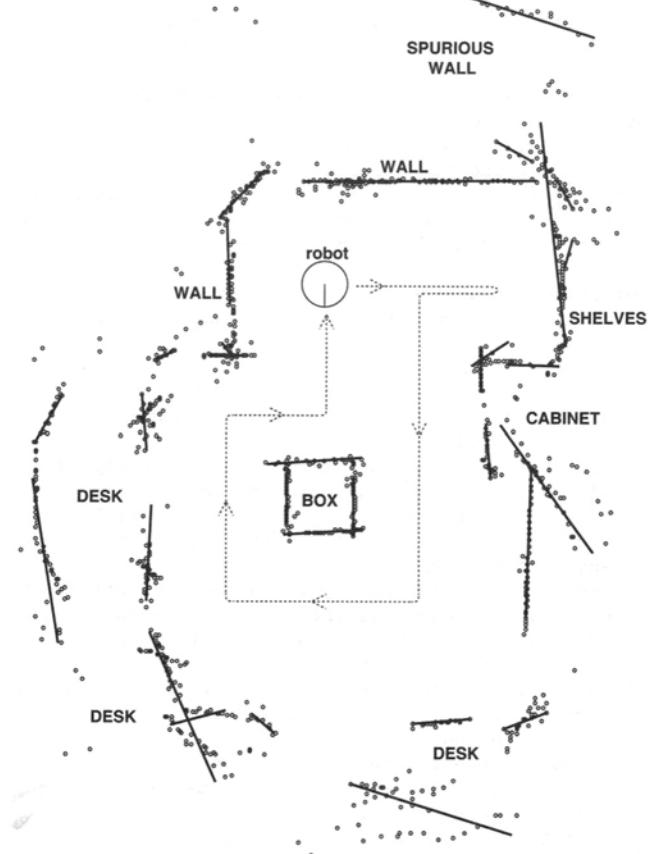
representation is the key...

What is it a map of?



Several answers to this question have been tried:

pre '83 It's a map of obstacle locations. (Points in a continuous space)



Sonar (or IR or even just tactile information!) provides indications of where obstacles are located...

How might we build features (e.g., walls) out of such a large set of discrete points?

What is it a map of?

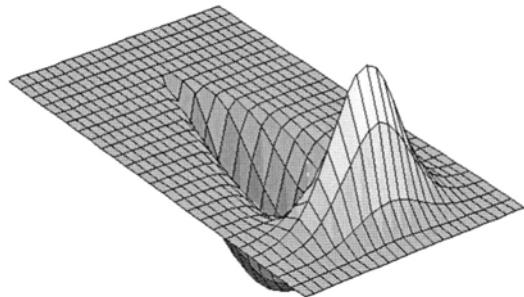


Several answers to this question have been tried:

pre '83 It's a map of obstacle locations. (Points in a continuous space)

Stanford
Cart

'83 - It's a map of probabilities: $p(o|S)$



$p(o|S)$ The certainty that a cell is **occupied**, given the sensor readings so far

\circ cell (x,y) is unoccupied

\circ cell (x,y) is occupied

What is it a map of?



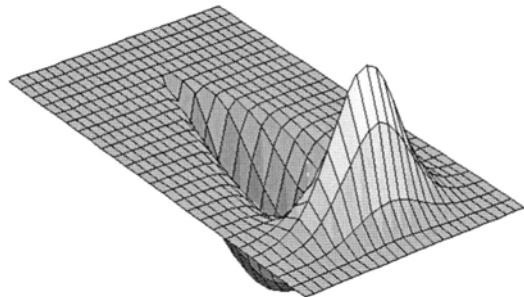
Several answers to this question have been tried:

pre '83 It's a map of obstacle locations. (Points in a continuous space)

Stanford
Cart

'83 - It's a map of probabilities: $p(o|S)$

The certainty that a cell is **occupied**, given the sensor readings so far



$p(\bar{o}|S)$

The certainty that a cell is **unoccupied**, given the sensor readings so far

\bar{o}

cell (x,y) is unoccupied

o

cell (x,y) is occupied

Note that the map is not exactly the sensor model:

$p(S|o)$ sensor model

What is it a map of ?

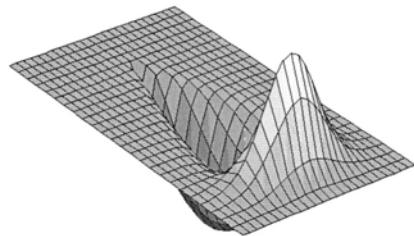


Several answers to this question have been tried:

pre '83 It's a map of obstacle locations. (Points in a continuous space)

Stanford
Cart

'83 - It's a map of probabilities: $p(o|S)$ ↗ The certainty that a cell is **occupied**, given the sensor readings so far



$p(\bar{o}|S)$ ↗ The certainty that a cell is **unoccupied**, given the sensor readings so far

'89 - It's a map of *odds*. The odds of an event are expressed *relative to the complement* of that event.

probabilities ↗

The odds that a cell is **occupied**, given the sensor readings S (taken so far) ↗ $\text{odds}(o|S) = \frac{p(o|S)}{p(\bar{o}|S)}$

0 ← → ∞
an event's odds range from 0 to ∞

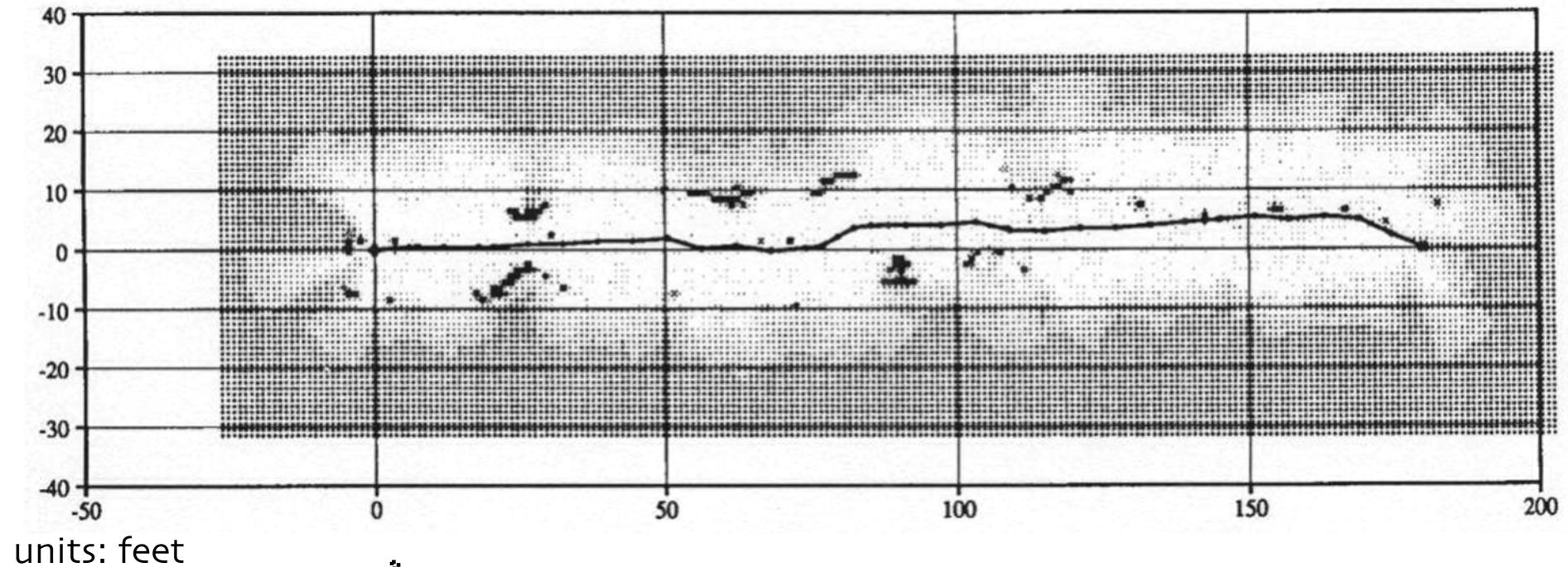
Odds

- The odds representation magnifies differences since as the numerator increases the denominator also decreases.
- Often a log(odds) representation is used:

probability	odds	log(odds)
0.3/0.7	.42	-.36
0.5/0.5	1	0
0.9/0.1	9	0.95
0.99/0.01	99	1.99

.probability,.certainty, occupancy, evidence grids...

H. Moravec and Alberto Elfes, CMU



units: feet

Evidence grid of a tree-lined outdoor path



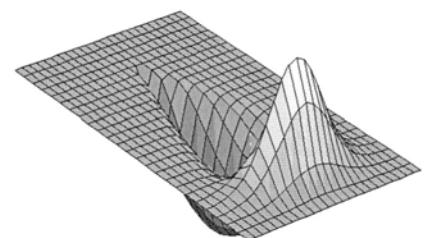
lighter areas: *lower* odds of obstacles being present



intermediate areas: *who knows?*

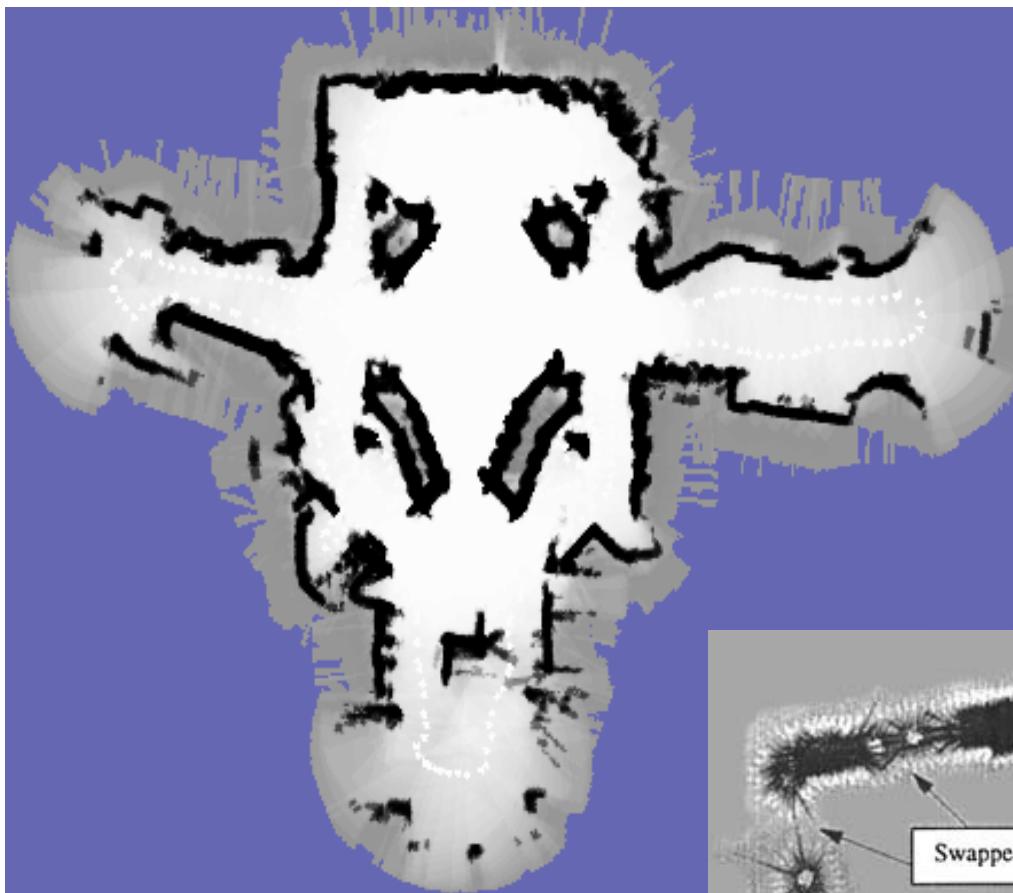


darker areas: *higher* odds of obstacles being present



how to combine lots of
local, single-sonar maps?

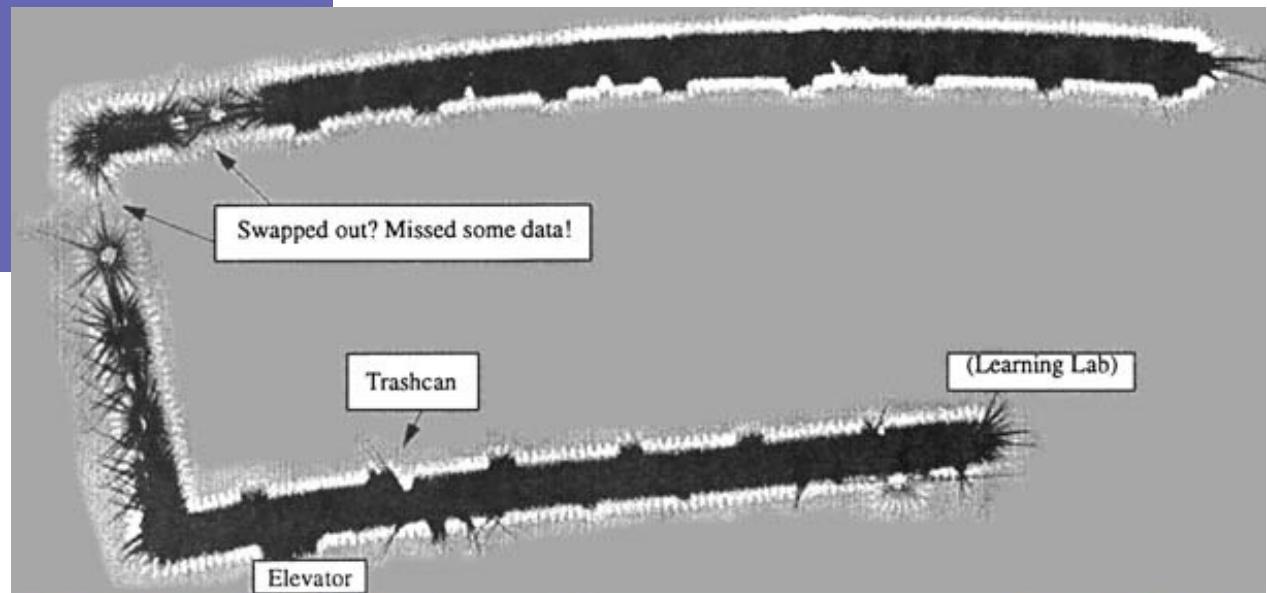
Evidence grid examples



Any range sensor
will work...

Smithsonian museum
of Natural History

CMU's Wean Hall

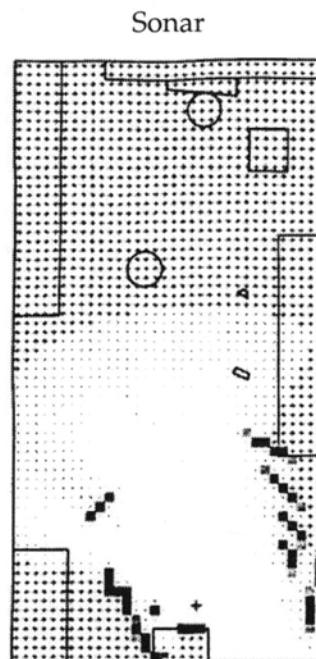


Sensor fusion



Incorporating data
from other sensors --
e.g., IR rangefinders
and stereo vision...

After 1 reading



(1) create another sensor model

After 10 readings

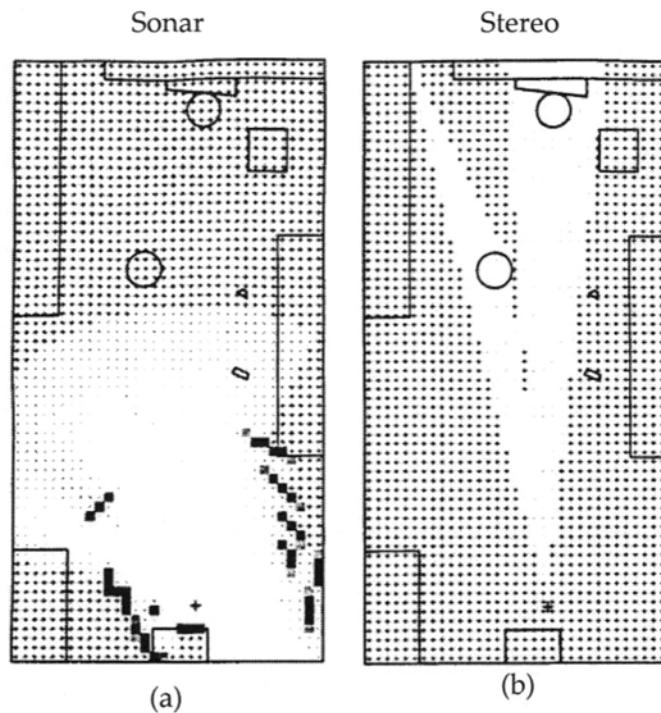


Sensor fusion



Incorporating data
from other sensors --
e.g., IR rangefinders
and stereo vision...

After 1 reading



(a)

(b)

(1) create another sensor model

(2) update along with the sonar

After 10 readings



(d)

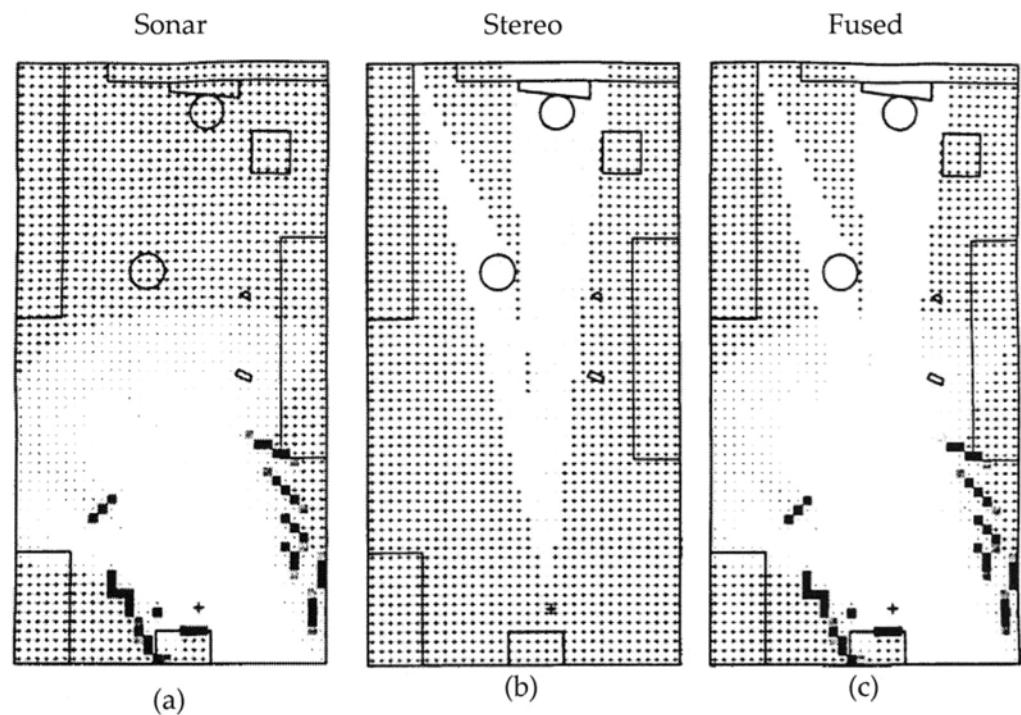
(e)

Sensor fusion



Incorporating data
from other sensors --
e.g., IR rangefinders
and stereo vision...

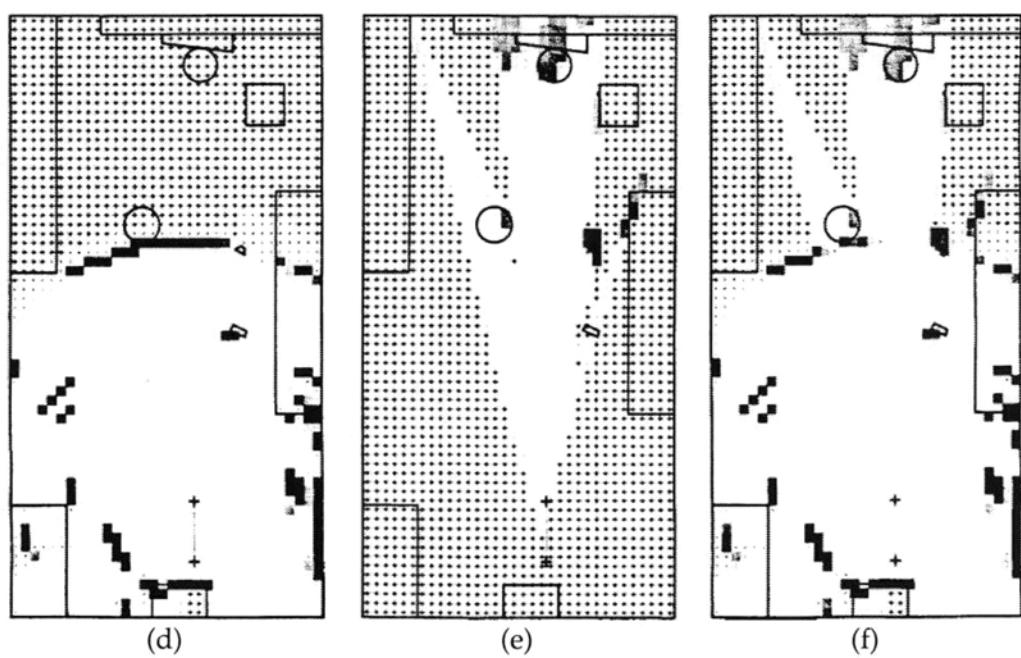
After 1 reading



(1) create another sensor model

(2) update along with the sonar

After 10 readings



Probabilistic Robotics

- Sebastian Thrun



- Probability of two events:

$$\begin{aligned} &= \begin{cases} \text{p(o } \wedge S) = p(o | S) p(S) \\ \text{p(S } \wedge o) = p(S | o) p(o) \end{cases} \end{aligned}$$

Probabilistic Robotics

- Sebastian Thrun



- Probability of two events:

$$\begin{aligned} p(o \wedge S) &= p(o|S)p(S) \\ p(S \wedge o) &= p(S|o)p(o) \end{aligned}$$

▪

- Bayes rule switches
the conditioning event

$$p(o|S) = \frac{p(S|o)p(o)}{p(S)}$$

Bayes rule

Conditional Prob. & Bayes' Rule

- Probability of two events:

$$= \begin{cases} p(o \wedge S) = p(o|S)p(S) \\ p(S \wedge o) = p(S|o)p(o) \end{cases}$$

Bayes rule

$$p(o|S) = \frac{p(S|o)p(o)}{p(S)}$$

- Bayes rule switches
the conditioning event

- Independence of events

$$p(S_2 \wedge S_1) = p(S_2)p(S_1)$$

Why might this be an unreasonable assumption?

Conditional Prob. & Bayes' Rule

- Probability of two events:

$$\begin{aligned} p(o \wedge S) &= p(o|S)p(S) \\ p(S \wedge o) &= p(S|o)p(o) \end{aligned}$$

▪

Bayes rule

- Bayes rule switches
the conditioning event

$$p(o|S) = \frac{p(S|o)p(o)}{p(S)}$$

- Independence of events

$$p(S_2 \wedge S_1) = p(S_2)p(S_1)$$

- Conditional independence

$$p(S_2 \wedge S_1 | o) = p(S_2|o)p(S_1|o)$$

More reasonable...

Probability vs. Odds

The probability of tossing *heads* is $\frac{1}{2}$.

The odds of tossing *heads* is 1-to-1.

The odds spans the range from 0 to ∞ instead of 0 to 1.

The map holds

$$\text{odds}(o \mid S_1) = \frac{p(o \mid S_1)}{p(\bar{o} \mid S_1)}$$

prob. that the cell is occupied,
given (sonar) data so far

prob. that the cell is unoccupied,
given (sonar) data so far

Representation



$$\text{odds}(o | S_1) = \frac{p(o | S_1)}{p(\bar{o} | S_1)}$$

We know the old
occupancy odds

The map holds

$$\text{odds}(o | S_1) = \frac{p(o | S_1)}{p(\bar{o} | S_1)}$$

prob. that the cell is occupied,
given (sonar) data so far

prob. that the cell is unoccupied,
given (sonar) data so far

Combining evidence



$$\text{odds}(o | S_2 \wedge S_1) = \frac{p(o | S_2 \wedge S_1)}{p(\bar{o} | S_2 \wedge S_1)}$$

We want the *new* occupancy odds



What do we need to integrate the reading S_2 ?



$$\text{odds}(o | S_1) = \frac{p(o | S_1)}{p(\bar{o} | S_1)}$$

We *know* the old occupancy odds

The map holds

$$\text{odds}(o | S_1) = \frac{p(o | S_1)}{p(\bar{o} | S_1)}$$

prob. that the cell is occupied,
given (sonar) data so far

prob. that the cell is unoccupied,
given (sonar) data so far

Combining evidence

$$\text{odds}(o | S_2 \wedge S_1) = \frac{p(o | S_2 \wedge S_1)}{p(\bar{o} | S_2 \wedge S_1)}$$

We want the *new* occupancy odds

what we'd like ↑
what we have ↓

$$\text{odds}(o | S_1) = \frac{p(o | S_1)}{p(\bar{o} | S_1)}$$

We *know* the old occupancy odds

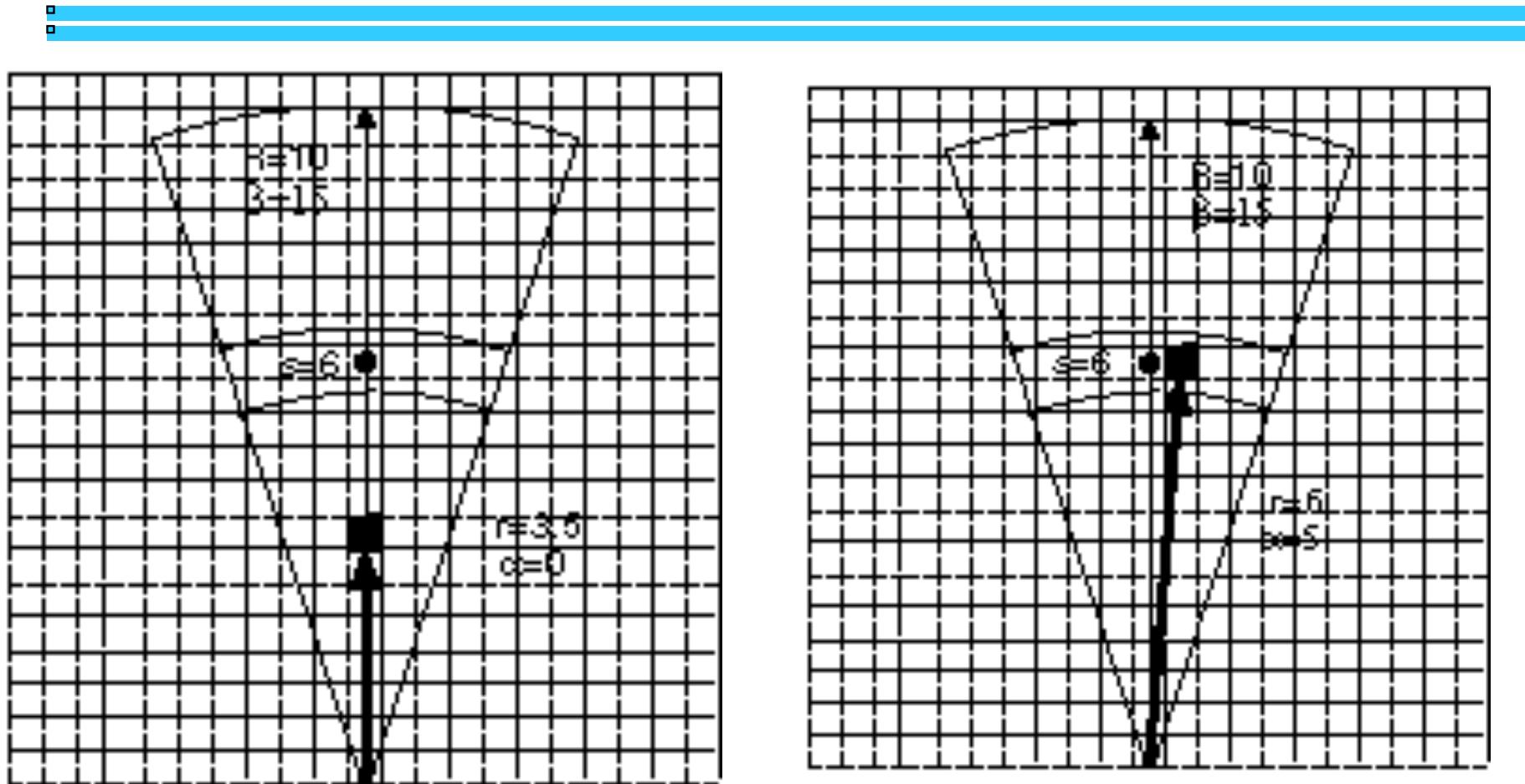
We *know* the sensor model



$$\begin{aligned} & p(S_2 | o) \\ & p(S_2 | \bar{o}) \end{aligned}$$

How S_2 arises == the "sensor model"

Sonar Model



Region-based

Region I = probably unoccupied

Region II = probably occupied

Region III = unknown

Combining evidence

$$\text{odds}(o \mid S_2 \wedge S_1) = \frac{p(o \mid S_2 \wedge S_1)}{p(\bar{o} \mid S_2 \wedge S_1)}$$

We want the new occupancy odds

Challenge: Represent ... in terms of

$$\text{odds}(o \mid S_1) = \frac{p(o \mid S_1)}{p(\bar{o} \mid S_1)}$$

We know the old occupancy odds

We know the sensor model

$$\frac{p(S_2 \mid o)}{p(S_2 \mid \bar{o})}$$

Bag o' Tricks

- Probability of two events:

$$= \begin{cases} p(o \wedge S) = p(o|S)p(S) \\ p(S \wedge o) = p(S|o)p(o) \end{cases}$$

Bayes rule

$$p(o|S) = \frac{p(S|o)p(o)}{p(S)}$$

- Bayes rule switches
the conditioning event

- Independence of events

$$p(S_2 \wedge S_1) = p(S_2)p(S_1)$$

- Conditional independence

$$p(S_2 \wedge S_1 | o) = p(S_2|o)p(S_1|o)$$

Combining evidence

:
:

$$\text{odds}(o | S_2 \wedge S_1) = \frac{p(o | S_2 \wedge S_1)}{p(\bar{o} | S_2 \wedge S_1)}$$

def' n of odds

Combining evidence

:
:

$$\begin{aligned}\text{odds}(o | S_2 \wedge S_1) &= \frac{p(o | S_2 \wedge S_1)}{p(\bar{o} | S_2 \wedge S_1)} && \text{def' n of odds} \\ &= \frac{p(S_2 \wedge S_1 | o) p(o)}{p(S_2 \wedge S_1 | \bar{o}) p(\bar{o})}\end{aligned}$$

Combining evidence

•

$$\text{odds}(o | S_2 \wedge S_1) = \frac{p(o | S_2 \wedge S_1)}{p(\bar{o} | S_2 \wedge S_1)} \quad \text{def' n of odds}$$

$$= \frac{p(S_2 \wedge S_1 | o) p(o)}{p(S_2 \wedge S_1 | \bar{o}) p(\bar{o})} \quad \text{Bayes' rule (+)}$$

$$= \frac{p(S_2 | o) p(S_1 | o) p(o)}{p(S_2 | \bar{o}) p(S_1 | \bar{o}) p(\bar{o})}$$

Combining evidence

•

$$\text{odds}(o | S_2 \wedge S_1) = \frac{p(o | S_2 \wedge S_1)}{p(\bar{o} | S_2 \wedge S_1)} \quad \text{def' n of odds}$$

$$= \frac{p(S_2 \wedge S_1 | o) p(o)}{p(S_2 \wedge S_1 | \bar{o}) p(\bar{o})} \quad \text{Bayes' rule (+)}$$

$$= \frac{p(S_2 | o) p(S_1 | o) p(o)}{p(S_2 | \bar{o}) p(S_1 | \bar{o}) p(\bar{o})} \quad \text{conditional independence of } S_1 \text{ and } S_2$$

$$= \frac{p(S_2 | o) p(o | S_1)}{p(S_2 | \bar{o}) p(\bar{o} | S_1)} \quad \text{Bayes' rule (+)}$$

Combining evidence

;

$$\begin{aligned}\text{odds}(o \mid S_2 \wedge S_1) &= \frac{p(o \mid S_2 \wedge S_1)}{p(\bar{o} \mid S_2 \wedge S_1)} && \text{def' n of odds} \\ &= \frac{p(S_2 \wedge S_1 \mid o) p(o)}{p(S_2 \wedge S_1 \mid \bar{o}) p(\bar{o})} && \text{Bayes' rule (+)} \\ &= \frac{p(S_2 \mid o) p(S_1 \mid o) p(o)}{p(S_2 \mid \bar{o}) p(S_1 \mid \bar{o}) p(\bar{o})} && \text{conditional independence of } S_1 \text{ and } S_2 \\ &= \frac{p(S_2 \mid o) p(o \mid S_1)}{p(S_2 \mid \bar{o}) p(\bar{o} \mid S_1)} && \text{Bayes' rule (+)}\end{aligned}$$

the sensor model

previous odds

Combining evidence

;

$$\begin{aligned}\text{odds}(o \mid S_2 \wedge S_1) &= \frac{p(o \mid S_2 \wedge S_1)}{p(\bar{o} \mid S_2 \wedge S_1)} && \text{def' n of odds} \\ &= \frac{p(S_2 \wedge S_1 \mid o) p(o)}{p(S_2 \wedge S_1 \mid \bar{o}) p(\bar{o})} && \text{Bayes' rule (+)} \\ &= \frac{p(S_2 \mid o) p(S_1 \mid o) p(o)}{p(S_2 \mid \bar{o}) p(S_1 \mid \bar{o}) p(\bar{o})} && \text{conditional independence of } S_1 \text{ and } S_2 \\ &= \frac{p(S_2 \mid o) p(o \mid S_1)}{p(S_2 \mid \bar{o}) p(\bar{o} \mid S_1)} && \text{Bayes' rule (+)}\end{aligned}$$

the sensor model

previous odds

Update step = ?

Combining evidence

;

$$\begin{aligned}\text{odds}(o \mid S_2 \wedge S_1) &= \frac{p(o \mid S_2 \wedge S_1)}{p(\bar{o} \mid S_2 \wedge S_1)} && \text{def' n of odds} \\ &= \frac{p(S_2 \wedge S_1 \mid o) p(o)}{p(S_2 \wedge S_1 \mid \bar{o}) p(\bar{o})} && \text{Bayes' rule (+)} \\ &= \frac{p(S_2 \mid o) p(S_1 \mid o) p(o)}{p(S_2 \mid \bar{o}) p(S_1 \mid \bar{o}) p(\bar{o})} && \text{conditional independence of } S_1 \text{ and } S_2 \\ &= \frac{p(S_2 \mid o) p(o \mid S_1)}{p(S_2 \mid \bar{o}) p(\bar{o} \mid S_1)} && \text{Bayes' rule (+)}\end{aligned}$$

the sensor model

previous odds

Update step = multiplying the previous odds by a precomputed weight.

“evidence” = $\log(\text{odds})$

why log?

Combining evidence

$$\text{odds}(o | S_2 \wedge S_1) = \frac{p(o | S_2 \wedge S_1)}{p(\bar{o} | S_2 \wedge S_1)} \quad \text{def' n of odds}$$

$$= \frac{p(S_2 \wedge S_1 | o) p(o)}{p(S_2 \wedge S_1 | \bar{o}) p(\bar{o})} \quad \text{Bayes' rule (+)}$$

$$= \frac{p(S_2 | o) p(S_1 | o) p(o)}{p(S_2 | \bar{o}) p(S_1 | \bar{o}) p(\bar{o})} \quad \text{conditional independence of } S_1 \text{ and } S_2$$

$$= \frac{p(S_2 | o) p(o | S_1)}{p(S_2 | \bar{o}) p(\bar{o} | S_1)} \quad \text{Bayes' rule (+)}$$

the sensor model

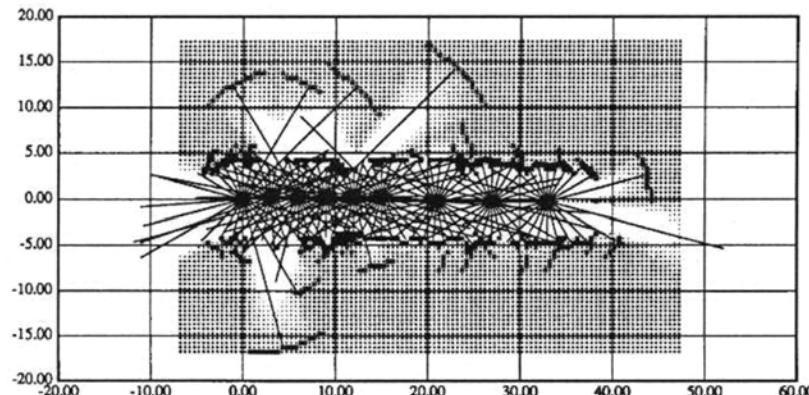
previous odds

Update step = multiplying the previous odds by a precomputed weight.

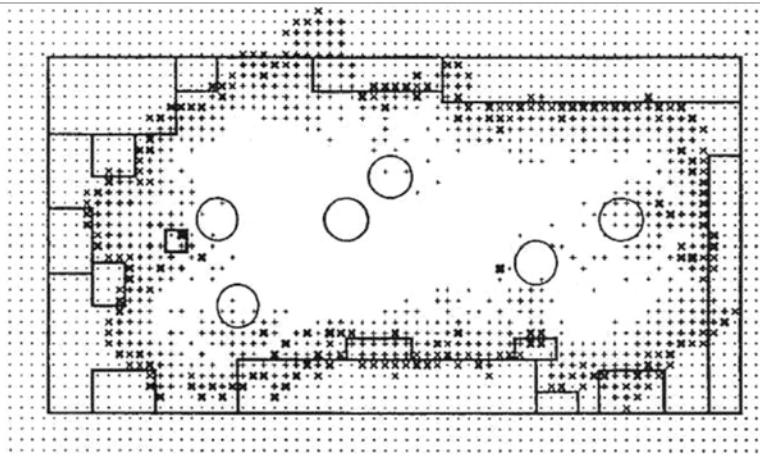
“evidence” = $\log(\text{odds})$

add vs. multiply

Evidence grid examples

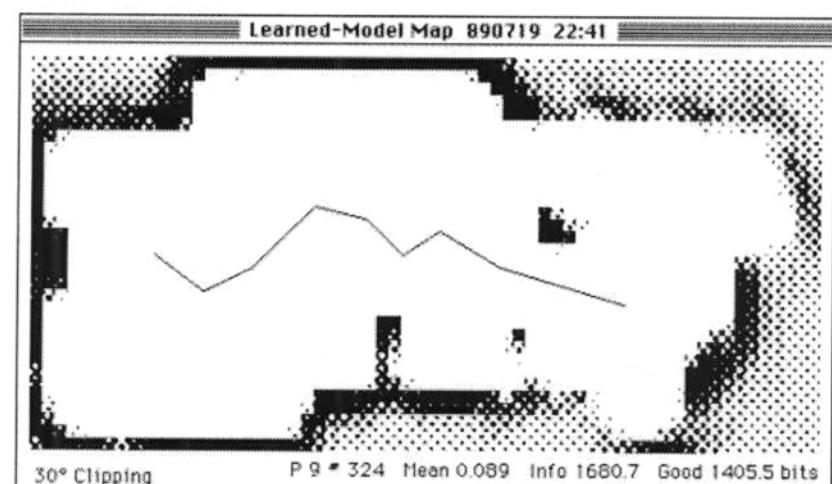
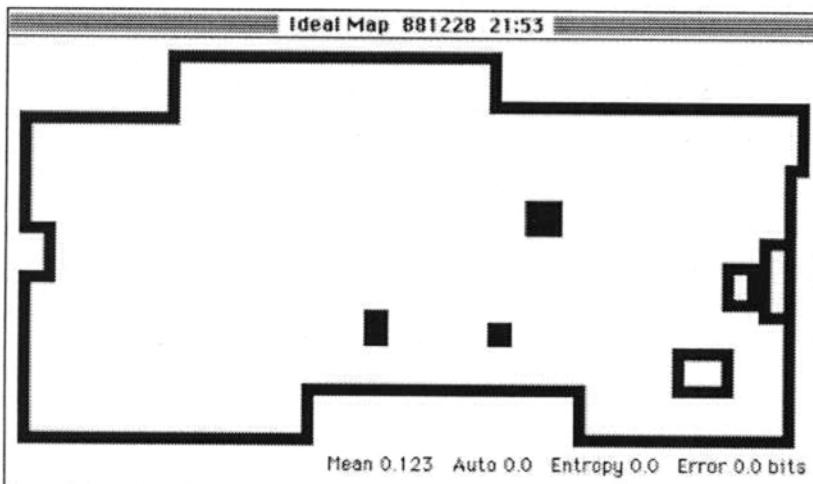


hallway with some open doors



lab space

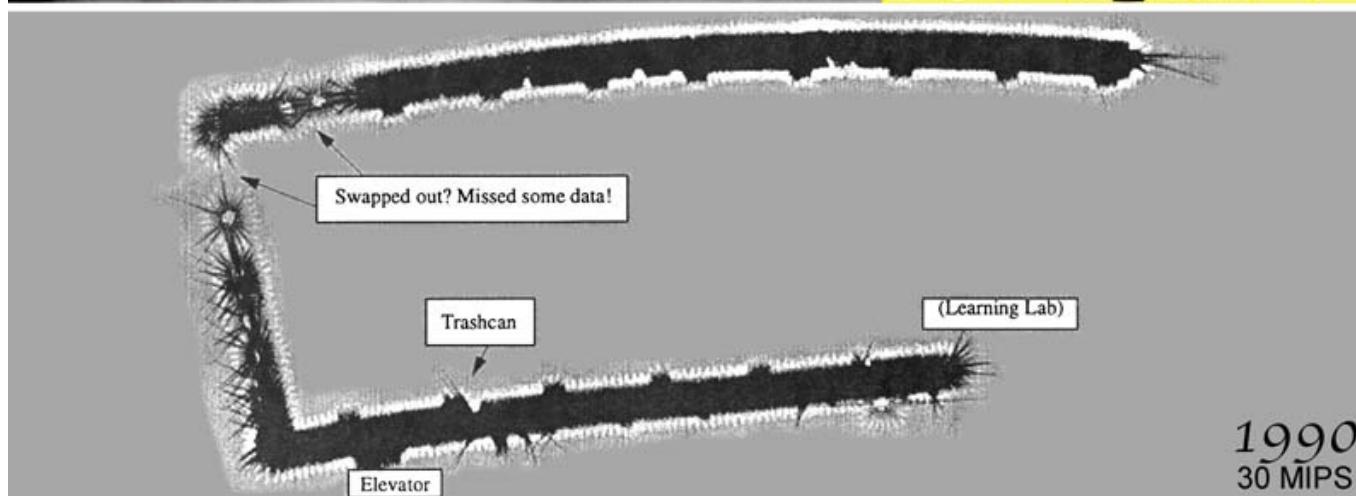
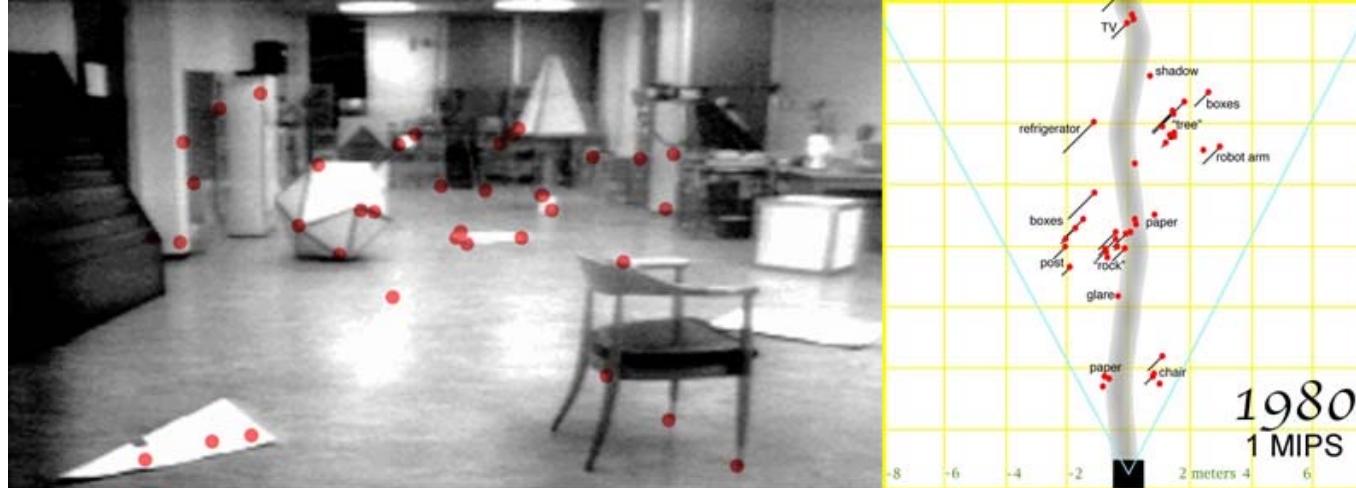
known map and estimated evidence grid



Spatial Representations

the evolution of
evidence grids

Individual Points



2d maps



3d maps

Next problem: coverage

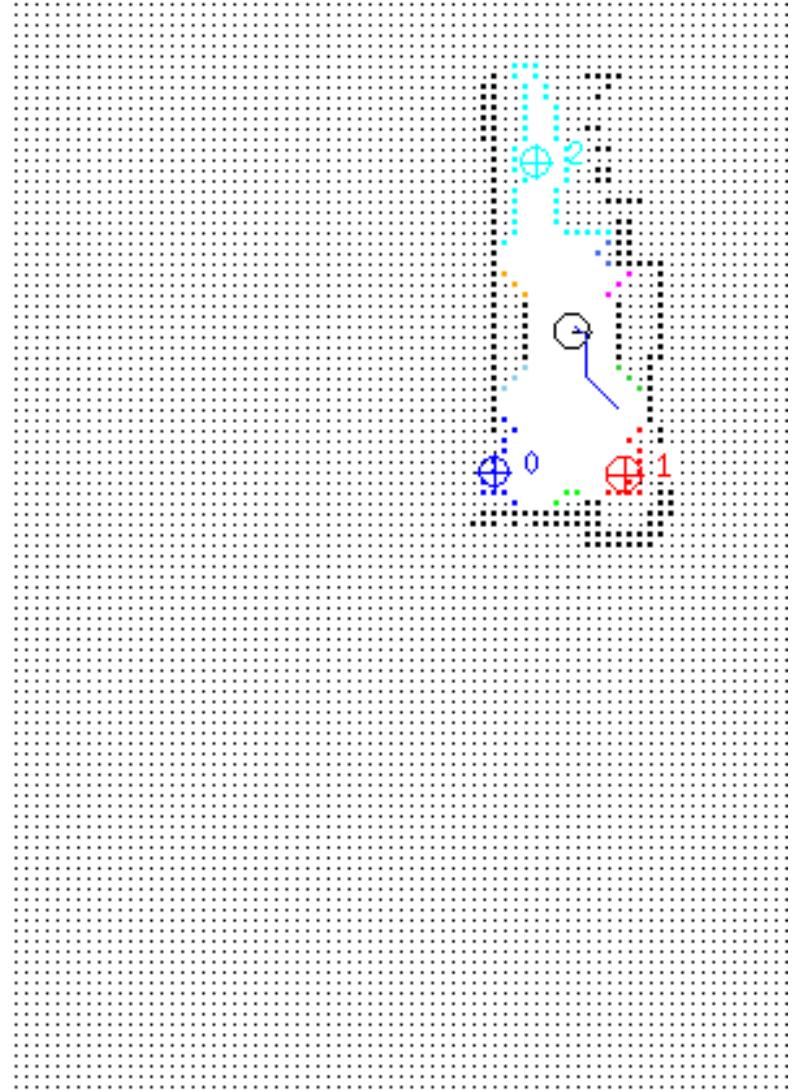
- Now we know how to create a map---how do we explore the environment in a principled way?

Coverage

- Random-walk: inefficient since the robot covers the same area multiple times
- Pattern based: spirals, sweeps
- Frontier-based: prioritize and investigate unexplored areas
- Generalized Voronoi graph: maximize distance between robot and obstacles while creating a Voronoi graph

Frontier Based Exploration

- Robot senses environment
- Borders of low certainty form *frontiers*
- Rate the frontiers
 - Centroid
 - Utility of exploring (big? Close?)
- Move robot to the centroid and repeat
- (continuously localize and map as you go)



Next time:

- Monte Carlo Localization and SLAM!

CAP6671: Intelligent Systems

Lecture 16: Monte Carlo Localization; Simultaneous Localization and Mapping

Instructor: Dr. Gita Sukthankar
Email: gitars@eeecs.ucf.edu

Announcements

- **Please sign up for your presentation time by tonight!**
 - **Most of the remaining presentation slots are next week.**
 - Class will be conducted entirely over Zoom from Mar 27-Apr 17.
 - All presentations will be done over Zoom since most of the students are virtual this semester.
 - Wed will be the last in person class for the next month.
- Final project: due Apr 19
 - Please make sure to get started on this!
- Final exam: scheduled for Apr 26
 - Webcourses quiz centering on lecture content
 - Short answer, simple math, no coding
 - Open book, open notes, no collaboration

Agent Competition Assignment

- Almost everyone did very well (avg=8.5)!
- Most students selected StarCraft or RoboCup.
- Popular theme: applications of deep learning models to competitions
- Problems:
 - **Little connection between papers.**
 - Overly long quotes
 - Proofreading errors (capitalization, grammar, spaces)

RLLab (Apr 5)

- Optional assignment
 - Assignment grade for the course will be $\max(\text{AgentCompetition}, \text{RLLab})$
 - Would not recommend completing this assignment if you received an 8 on the first assignment.
- Choose either Atari/Breakout (DQN) or BipedalWalker (DDPG) for your basic RL agent
 - StableBaselines3 has implemented versions of these algorithms and pre-trained versions of the models.
 - Evaluate performance of algorithm, plot graph of reward vs. epochs, calculate mean over multiple runs, tune hyperparameters, compare to a 2nd algorithm

Localization

- Assuming the robot has a map, how does it figure out where it is on the map?

Localization

- Assuming the robot has a map, how does it figure out where it is on the map?
 - Landmarks/features
 - Matching features in environment to features on map
 - Tracking distance moved
 - Combining information from multiple sequential readings
 - Method of updating beliefs based on observations (Bayes rule)

Localization Techniques

- Open loop pose estimation:
 - Maintain pose estimate based on expected results of motion commands (no sensing)
- Dead reckoning:
 - Use proprioception (odometry, inertial) to estimate pose w.r.t. *initial* coordinate frame
 - Multiple error sources:
 - Wheel slip, gear backlash
 - Noise (e.g. from encoders)
 - Sensor, processor quantization errors
 - Pose error accumulates with time and motion
 - Accurate, long-term pose estimation probably impossible without an external reference

Bayesian robotics (and vision)



Sebastian Thrun
and Stanley (in background)



University of Bonn
↓
Carnegie Mellon
↓
Stanford AI Lab
↓
Stops and starts...

Reading

Fox, Burgard, Dellaert, and Thrun, Monte Carlo Localization: Efficient Position Estimation for Mobile Robots, AAAI 1999

Haehnel, Burgard, Fox, and Thrun, An Efficient FastSLAM Algorithm for Generating Maps of Large-scale Cyclic Environments from Raw Laser Range Measurements, IROS 2003

Also recommend looking at the Probabilistic Robotics book for further background

Let's review the notation used in the reading....

Bayes Formula

$$P(x, y) = P(x \mid y)P(y) = P(y \mid x)P(x)$$

\Rightarrow

$$P(x \mid y) = \frac{P(y \mid x) P(x)}{P(y)} = \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}}$$

Normalization

$$P(x \mid y) = \frac{P(y \mid x) P(x)}{P(y)} = \eta P(y \mid x) P(x)$$

$$\eta = P(y)^{-1} = \frac{1}{\sum_x P(y \mid x) P(x)}$$

Algorithm:

$$\forall x : \text{aux}_{x|y} = P(y \mid x) P(x)$$

$$\eta = \frac{1}{\sum_x \text{aux}_{x|y}}$$

$$\forall x : P(x \mid y) = \eta \text{ aux}_{x|y}$$

Bayes Rule with Background Knowledge

$$P(x \mid y, z) = \frac{P(y \mid x, z) P(x \mid z)}{P(y \mid z)}$$

Conditioning

- Total probability:

$$P(x) = \int P(x, z) dz$$

$$P(x) = \int P(x \mid z) P(z) dz$$

$$P(x \mid y) = \int P(x \mid y, z) P(z) dz$$

Conditional Independence

$$P(x, y \mid z) = P(x \mid z)P(y \mid z)$$

equivalent to

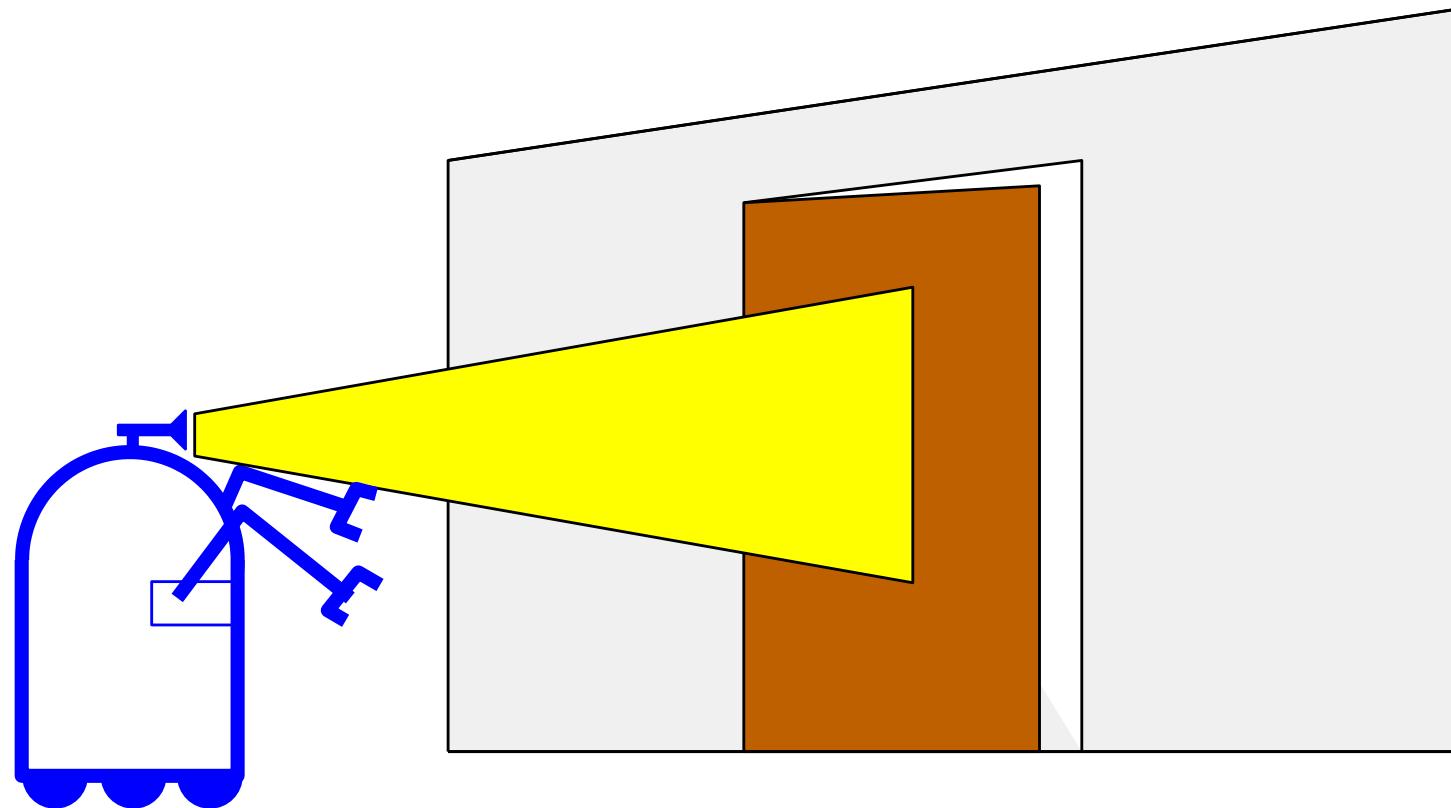
$$P(x \mid z) = P(x \mid z, y)$$

and

$$P(y \mid z) = P(y \mid z, x)$$

Simple Example of State Estimation

- Suppose a robot obtains measurement z
- What is $P(\text{open}|z)$?



Example

- $P(z|open) = 0.6 \quad P(z|\neg open) = 0.3$
- $P(open) = P(\neg open) = 0.5$

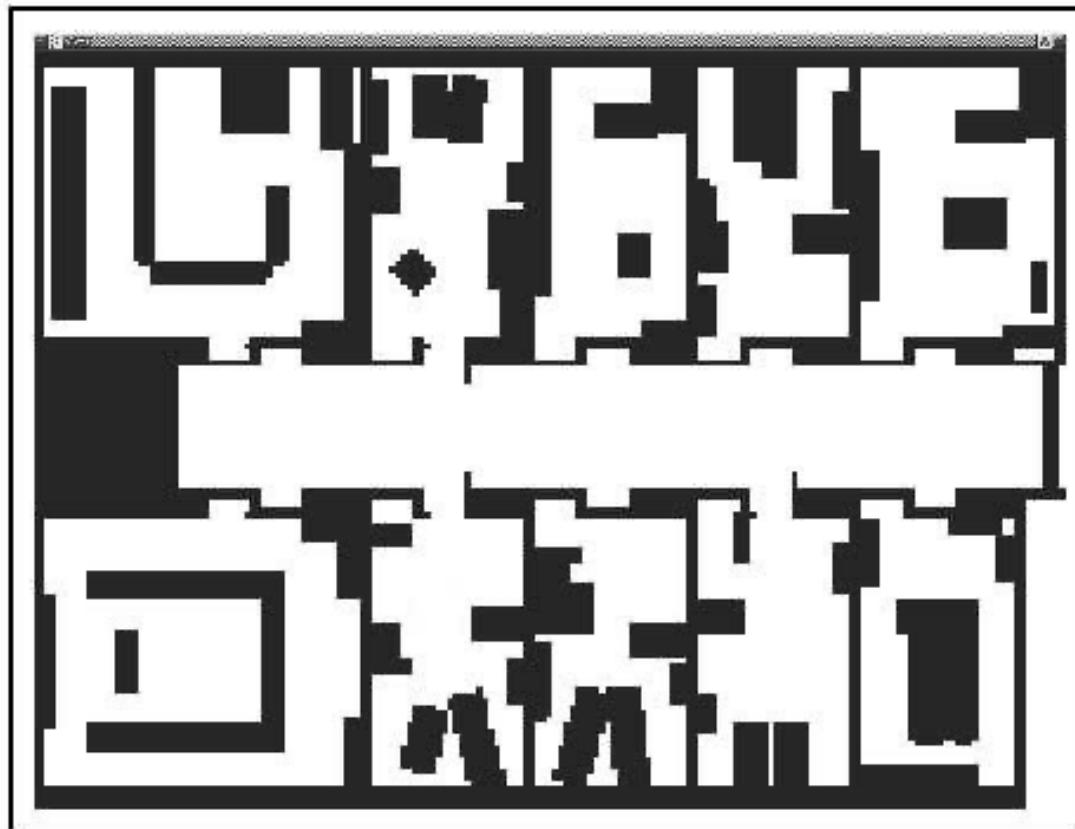
$$P(open | z) = \frac{P(z | open)P(open)}{P(z | open)p(open)+P(z | \neg open)p(\neg open)}$$

$$P(open | z) = \frac{0.6 \cdot 0.5}{0.6 \cdot 0.5 + 0.3 \cdot 0.5} = \frac{2}{3} = 0.67$$

- z raises the probability that the door is open.

Robot Localization

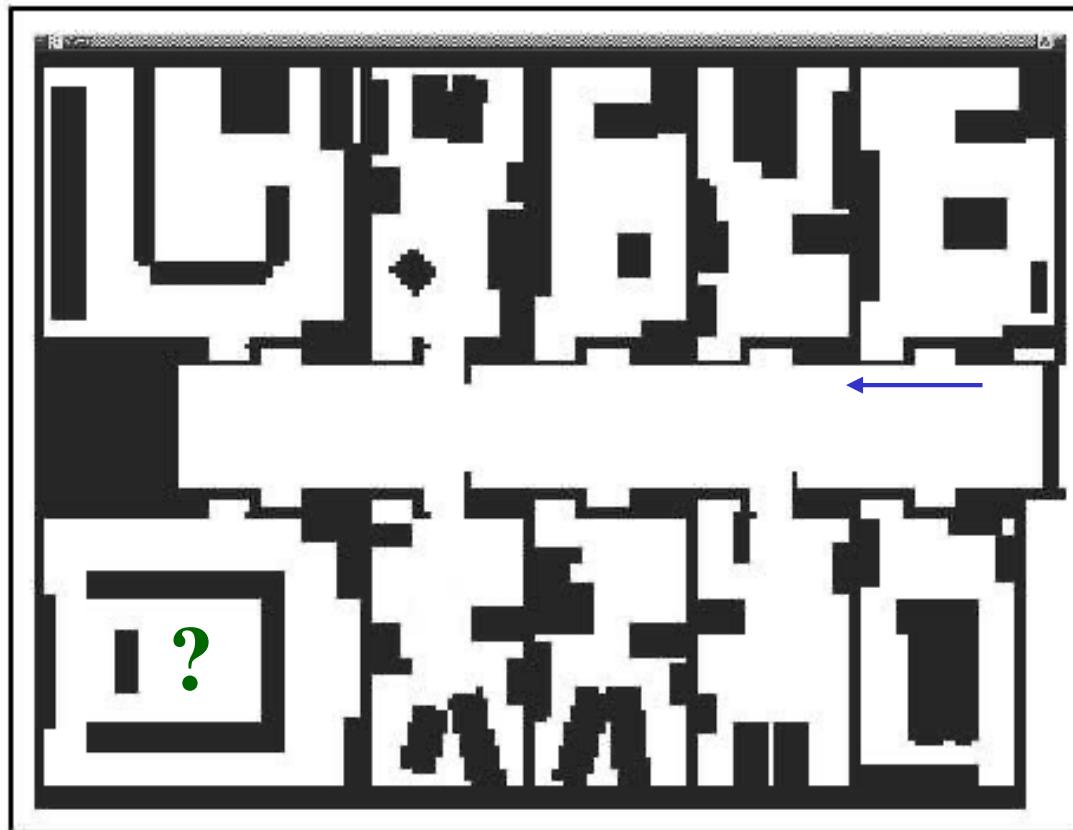
Where am I?



Rhino's home, Bonn

Robot Localization

Where am I?



robot tracking

local problem

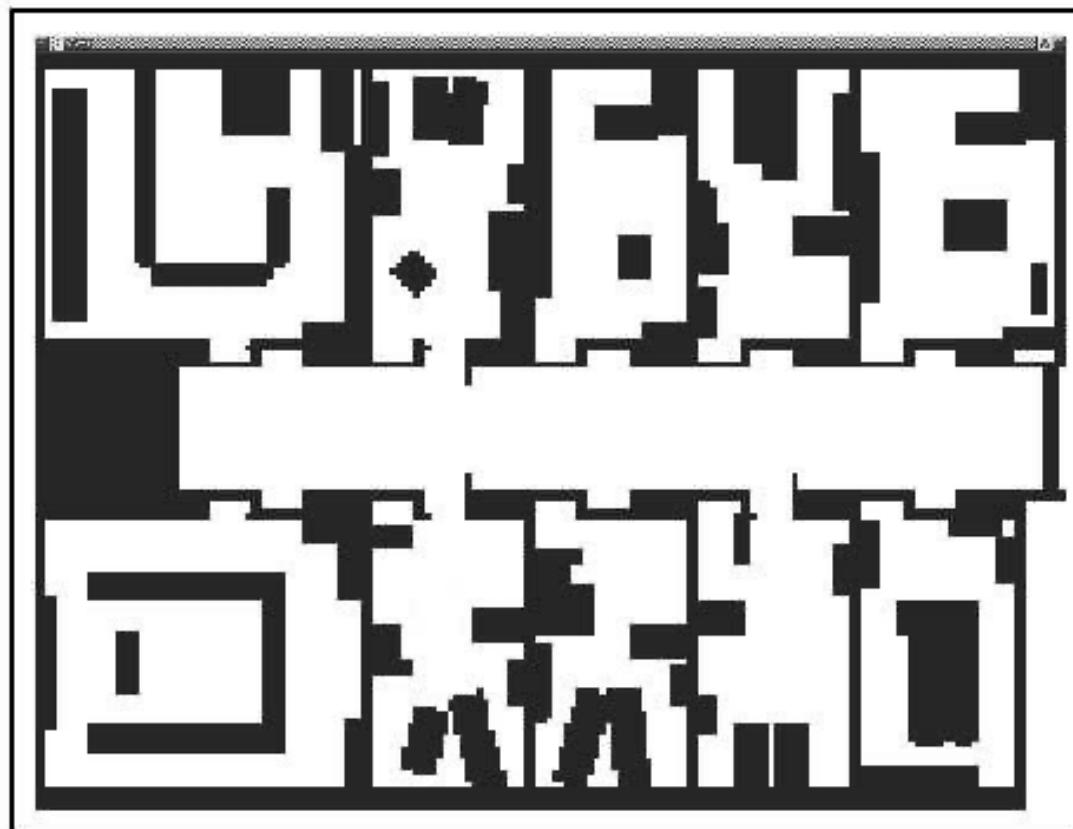
robot kidnapping

global problem

What's the problem?

only local data!

(even perfect data)



robot tracking

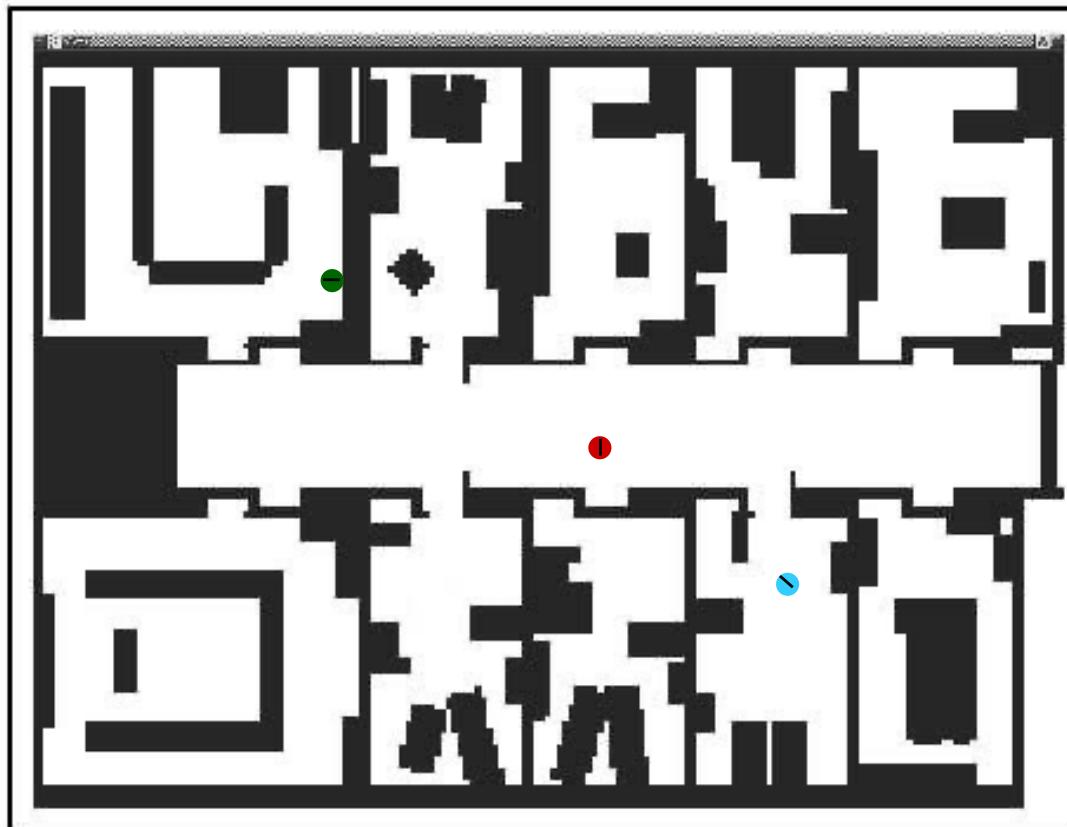
local problem

robot kidnapping

global problem

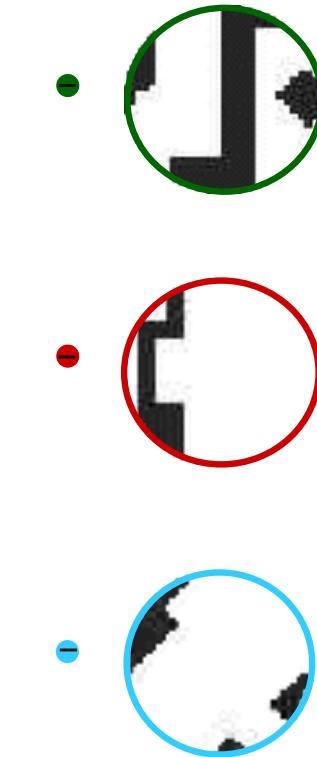
What's the problem?

direct map-matching can be overwhelming



robot tracking

local problem

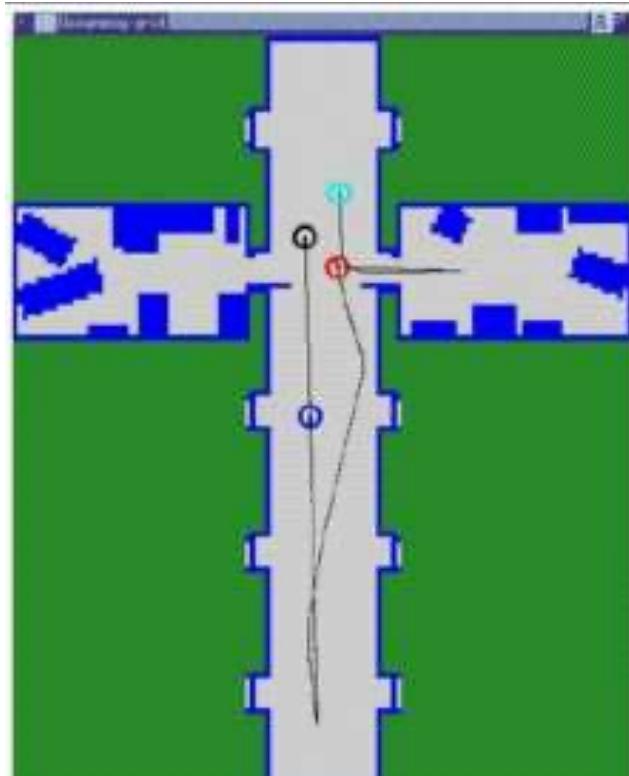


robot kidnapping

global problem

Monte Carlo Localization

Key idea: keep track of a *probability distribution* for where the robot might be in the known map

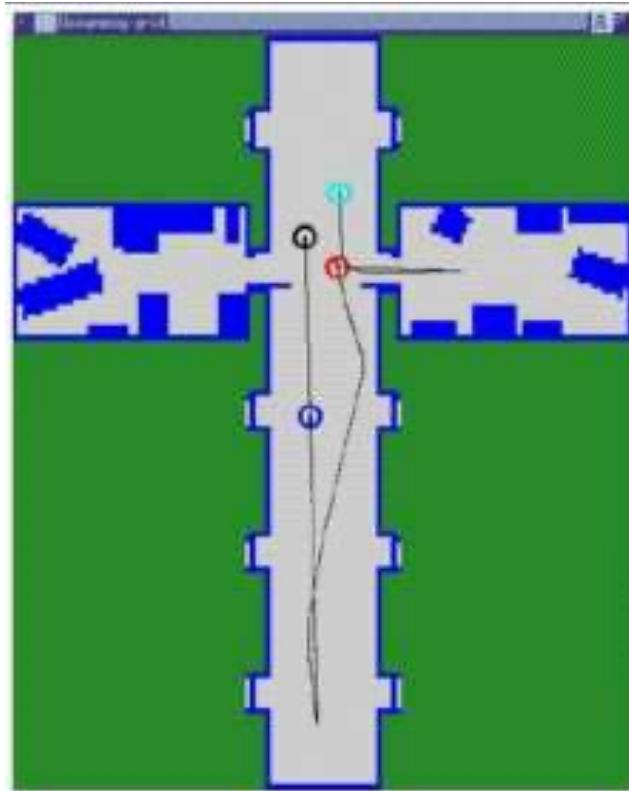


Initial (uniform) distribution

black - blue - red - cyan

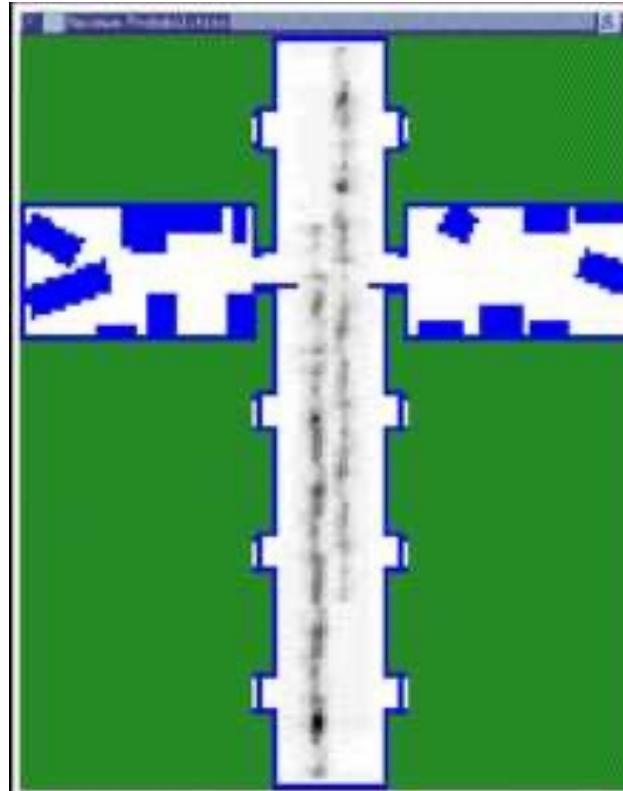
Monte Carlo Localization

Key idea: keep track of a *probability distribution* for where the robot might be in the known map



Initial (uniform) distribution

black - blue - red - cyan

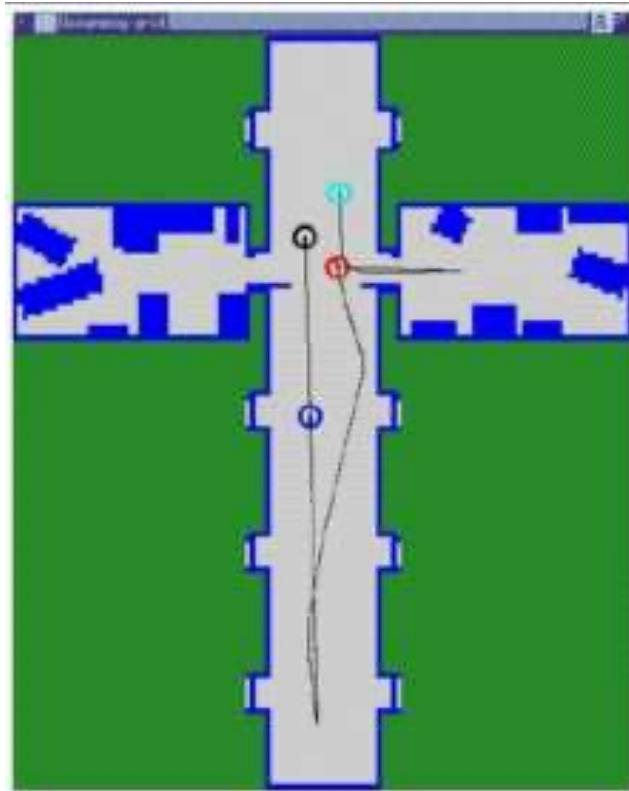


Intermediate stage 1

blue

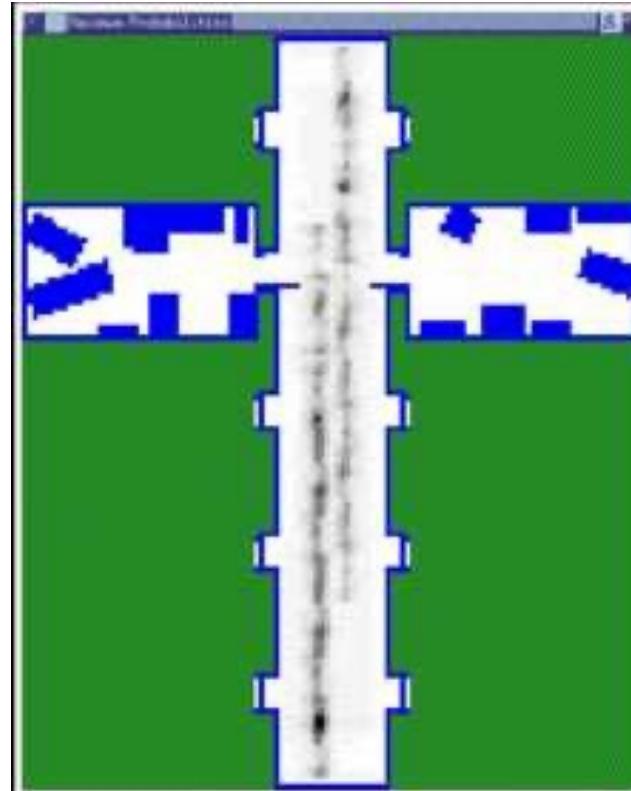
Monte Carlo Localization

Key idea: keep track of a *probability distribution* for where the robot might be in the known map



Initial (uniform) distribution

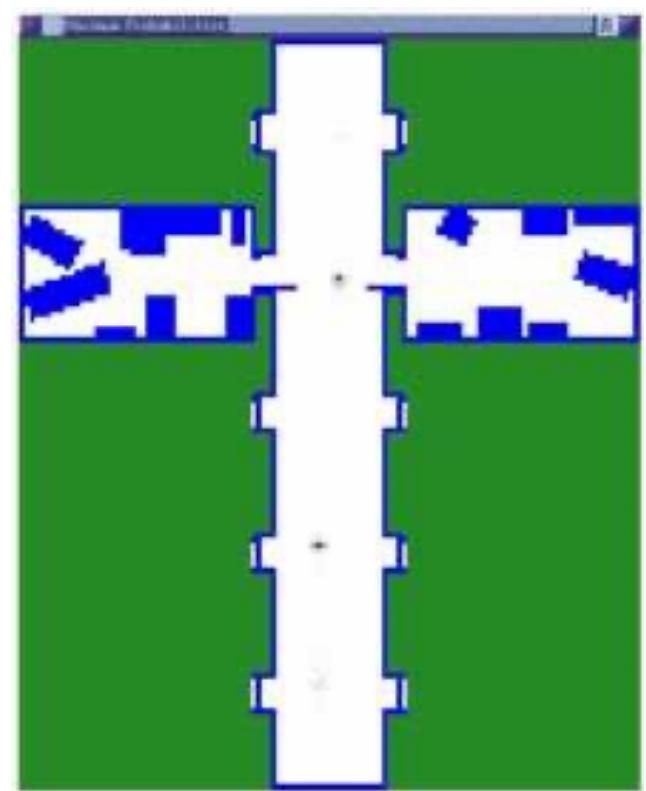
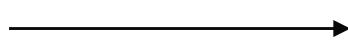
black - blue - red - cyan



Intermediate stage 2

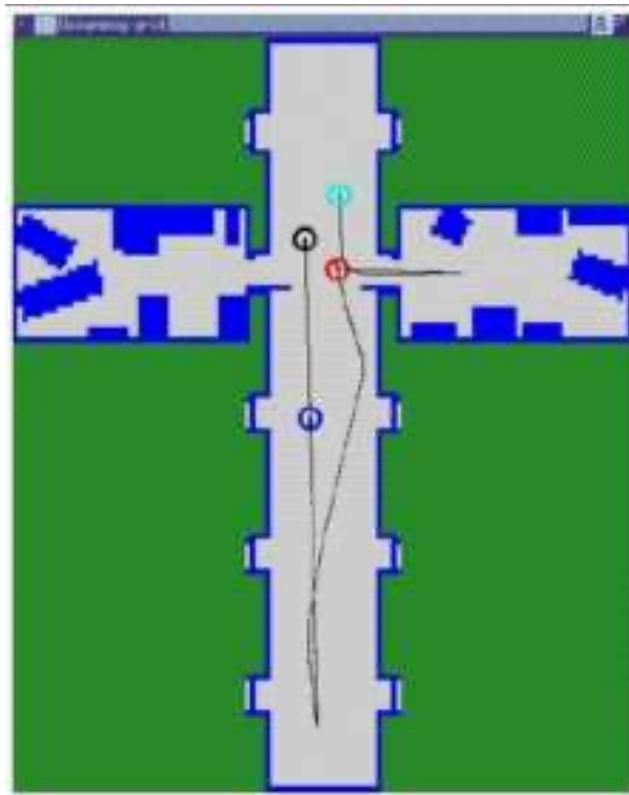
blue

red



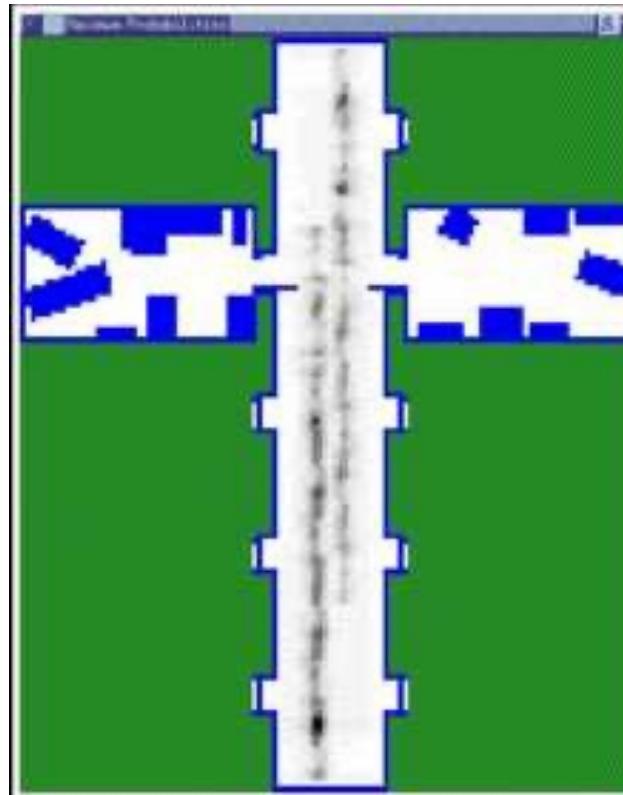
Monte Carlo Localization

Key idea: keep track of a *probability distribution* for where the robot might be in the known map

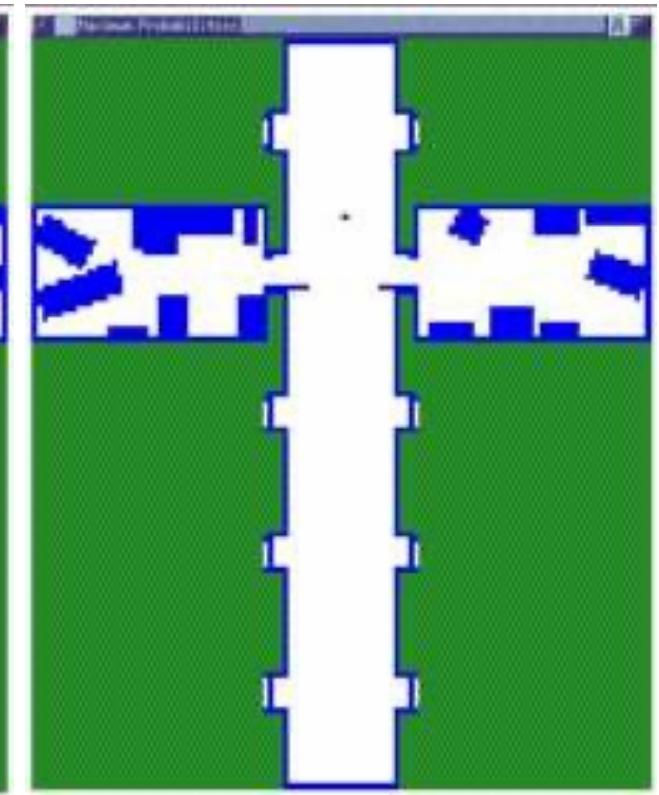


Initial (uniform) distribution

black - blue - red - cyan



Intermediate stages



cyan

Final distribution

But how?

Deriving MCL

- Sebastian Thrun
- Wolfram Burgard
- Dieter Fox

Bag o' tricks

- Bayes' rule

$$p(A | B) = \frac{p(B | A) \cdot p(A)}{p(B)}$$

- Definition of *conditional* probability

$$p(A \wedge B) = p(A | B) \cdot p(B)$$

-
- Definition of *marginal* probability

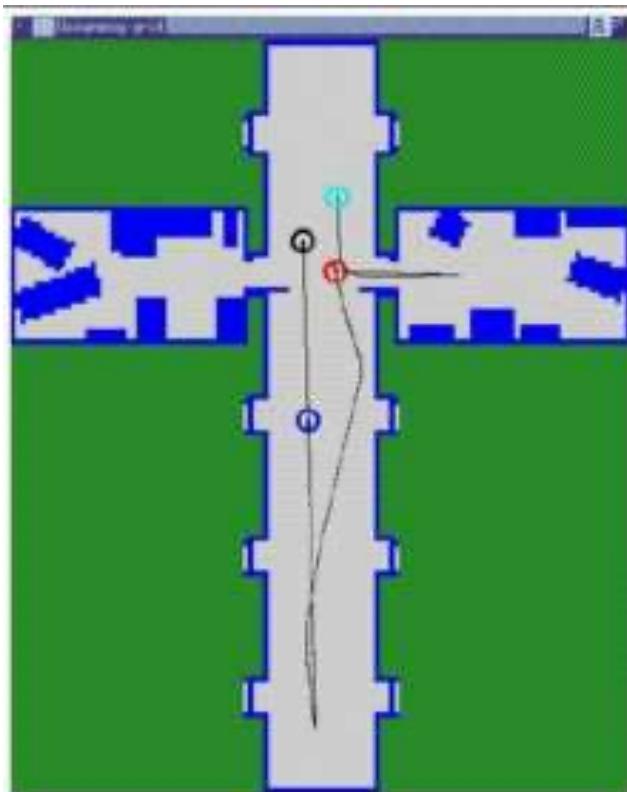
$$p(A) = \sum_{\text{all } B} p(A \wedge B)$$

$$p(A) = \sum_{\text{all } B} p(A | B) \cdot p(B)$$

Setting up the problem

The robot alternates between

- sensing -- getting range observations $o_1, o_2, o_3, \dots, o_{t-1}, o_t$ “local maps”
- acting -- driving around (or ferrying?) $a_1, a_2, a_3, \dots, a_{t-1}$ whence?



Setting up the problem

The robot alternates between

- sensing -- getting range observations $o_1, o_2, o_3, \dots, o_{t-1}, o_t$ “local maps”
 - acting -- driving around (or ferrying?) $a_1, a_2, a_3, \dots, a_{t-1}$ whence?
-

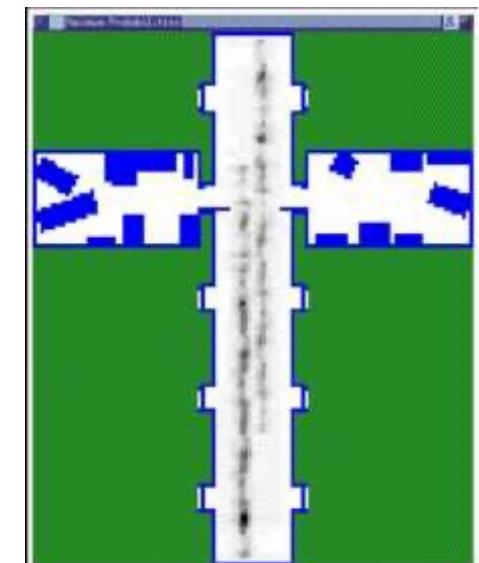
We want to know r_t -- the position of the robot at time t

- but we'll settle for $p(r_t)$ -- a probability distribution for r_t



What *kind* of thing is $p(r_t)$?

What *do* we know?



Setting up the problem

The robot does (or can be modeled to) alternate between

- sensing -- getting range observations $o_1, o_2, o_3, \dots, o_{t-1}, o_t$ “local maps”
 - acting -- driving around (or ferrying?) $a_1, a_2, a_3, \dots, a_{t-1}$ whence?
-

We want to know r_t -- the position of the robot at time t

- but we'll settle for $p(r_t)$ -- a probability distribution for r_t



What *kind* of thing is $p(r_t)$?

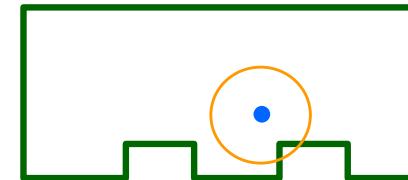
We do know m (or will know)	-- the map of the environment
$p(o r, m)$	-- the sensor model
$p(r_{\text{new}} r_{\text{old}}, a, m)$	-- the motion model = the accuracy of performing action a

Sensor Model

$p(o | r, m)$ sensor model



map m and location r



$$p(\text{ } \cdot \text{ } | r, m) = .95$$

$$p(\text{ } \cdot \text{ } | r, m) = .05$$

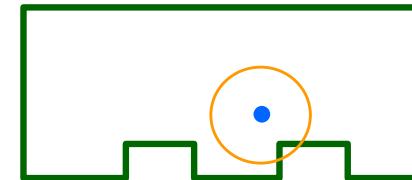
potential observations O

Sensor Model

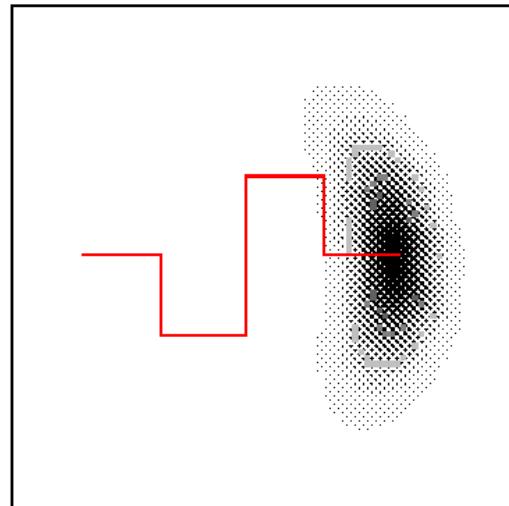
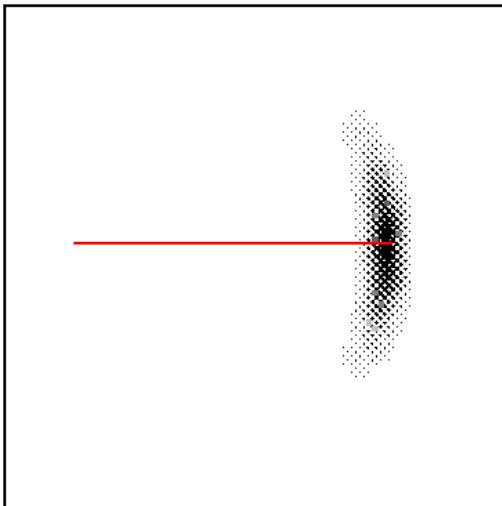
$p(o | r, m)$ sensor model



map m and location r



$p(r_{\text{new}} | r_{\text{old}}, a, m)$ action model



$$p(\text{circle} | r, m) = .95$$

$$p(\text{circle} | r, m) = .05$$

potential observations O

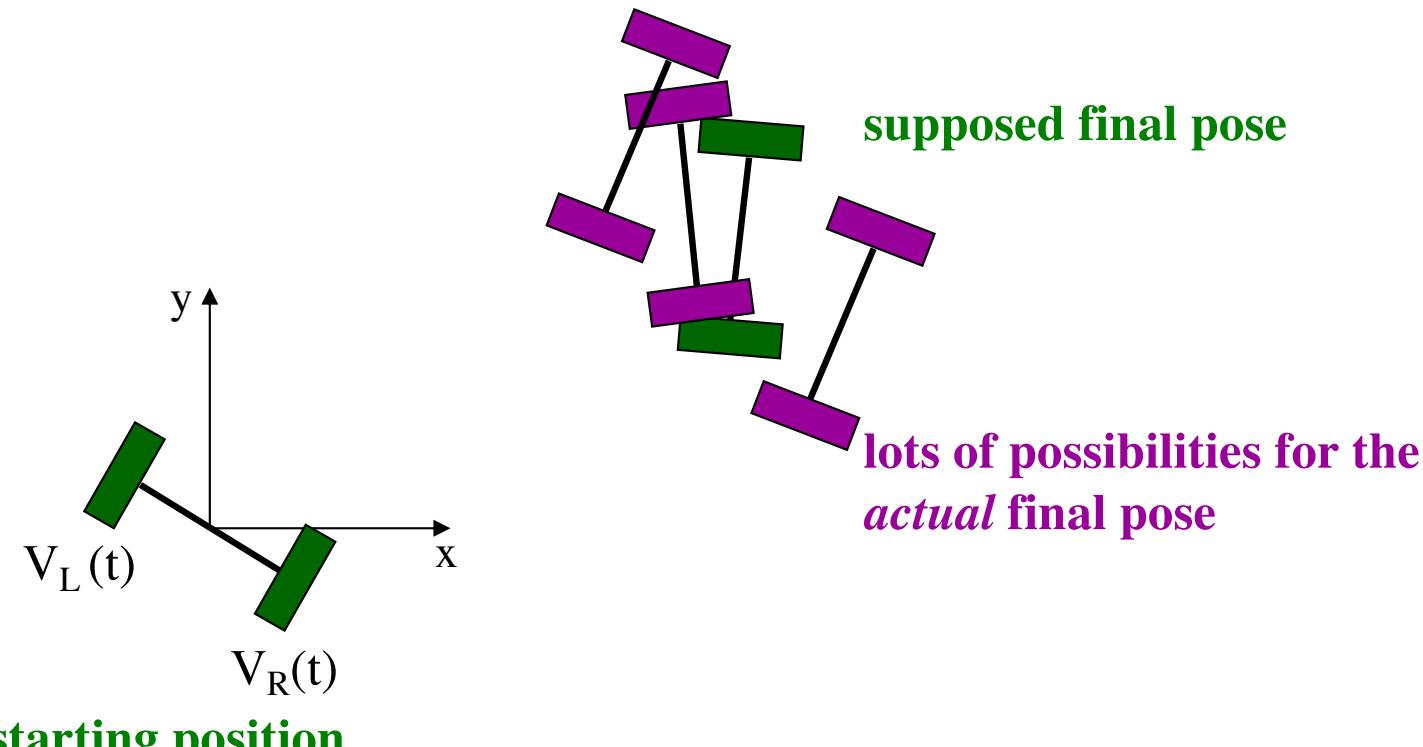
“probabilistic kinematics” -- encoder uncertainty

- red lines indicate commanded action
- the cloud indicates the likelihood of various final states

Probabilistic Kinematics

Key question:

We may know where our robot is *supposed to be*,
but in reality it might be somewhere else...



What should we do?

Robot models: how-to

$p(o | r, m)$ sensor model

$p(r_{\text{new}} | r_{\text{old}}, a, m)$ action model

- (0) Model the physics of the sensor/actuators
(with error estimates)

theoretical
modeling

Robot models: how-to

$p(o | r, m)$ sensor model

$p(r_{\text{new}} | r_{\text{old}}, a, m)$ action model

(0) Model the physics of the sensor/actuators
(with error estimates)

theoretical
modeling

(1) Measure lots of sensing/action results
and create a model from them

empirical
modeling

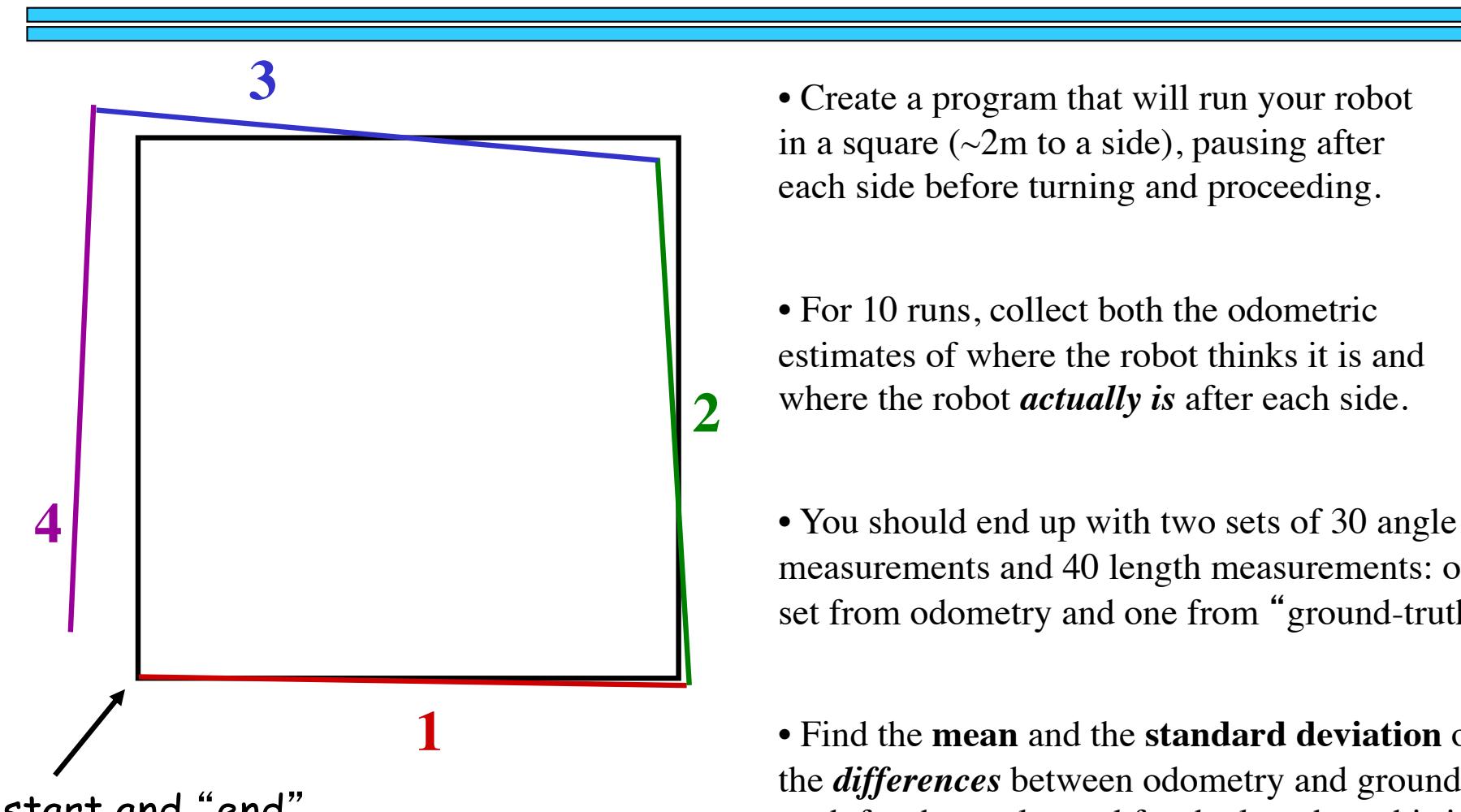
- take N measurements, find mean (m) and st. dev. (σ) and then use a Gaussian model
- or, some other easily-manipulated (*probability?*) model...

$$p(x) = \begin{cases} 0 & \text{if } |x-m| > s \\ 1 & \text{otherwise} \end{cases}$$

$$p(x) = \begin{cases} 0 & \text{if } |x-m| > s \\ 1 - |x-m|/s & \text{otherwise} \end{cases}$$

MODEL the error in order to reason about it!

Running around in squares

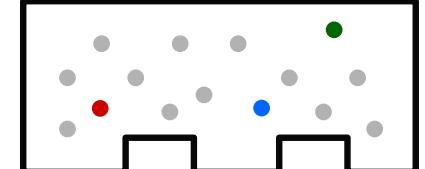


This provides a *probabilistic kinematic* model.

Monte Carlo Localization

Start by assuming $p(r_0)$ is the uniform distribution.

take K samples of r_0 and weight each with a “probability” of $1/K$



dimensionality??

“Particle Filter” representation
of a probability distribution

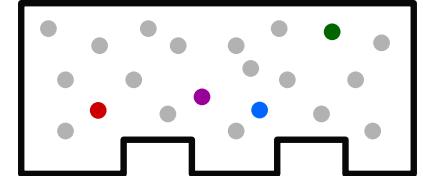
Monte Carlo Localization



Start by assuming $p(r_0)$ is the uniform distribution.

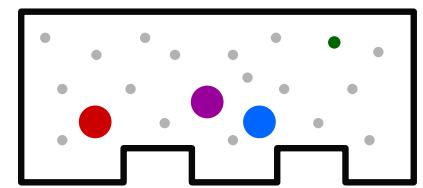
take K samples of r_0 and weight each with a “probability” of $1/K$

Get the current sensor observation, o_1



For each sample point r_0 multiply the importance factor by $p(o_1 | r_0, m)$

← “probability” →



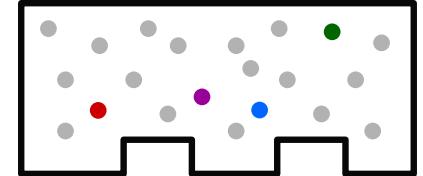
Monte Carlo Localization



Start by assuming $p(r_0)$ is the uniform distribution.

take K samples of r_0 and weight each with a “probability” of $1/K$

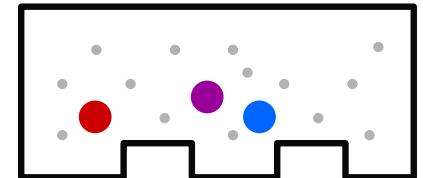
Get the current sensor observation, o_1



For each sample point r_0 multiply the importance factor by $p(o_1 | r_0, m)$

Normalize (make sure the importance factors add to 1)

You now have an approximation of $p(r_1 | o_1, \dots, m)$
and the distribution is no longer uniform

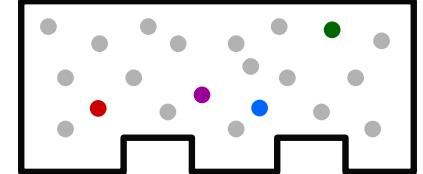


Monte Carlo Localization

Start by assuming $p(r_0)$ is the uniform distribution.

take K samples of r_0 and weight each with a “probability” of $1/K$

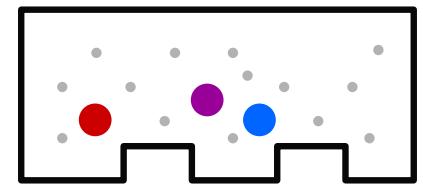
Get the current sensor observation, o_1



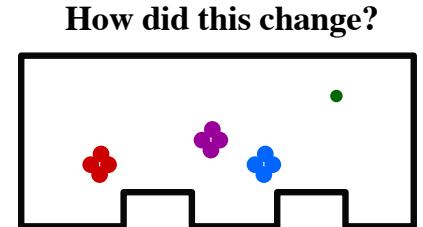
For each sample point r_0 multiply the importance factor by $p(o_1 | r_0, m)$

Normalize (make sure the importance factors add to 1)

You now have an approximation of $p(r_1 | o_1, \dots, m)$
and the distribution is no longer uniform



Create new samples by dividing up large clumps
each point spawns new ones in proportion to its importance factor

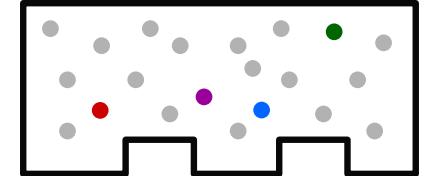


Monte Carlo Localization

Start by assuming $p(r_0)$ is the uniform distribution.

take K samples of r_0 and weight each with a “probability” of $1/K$

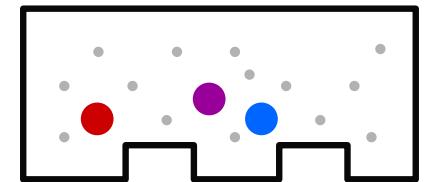
Get the current sensor observation, o_1



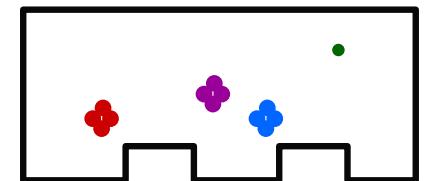
For each sample point r_0 multiply the importance factor by $p(o_1 | r_0, m)$

Normalize (make sure the importance factors add to 1)

You now have an approximation of $p(r_1 | o_1, \dots, m)$
and the distribution is no longer uniform



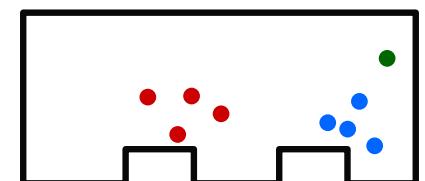
How did this change?



Create new samples by dividing up large clumps
each point spawns new ones in proportion to its importance factor

The robot moves, a_1

For each sample r_1 , move it according to the model $p(r_2 | a_1, r_1, m)$

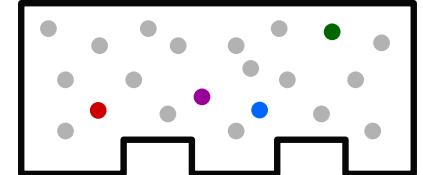


Where do the purple ones go?

Monte Carlo Localization

Start by assuming $p(r_0)$ is the uniform distribution.

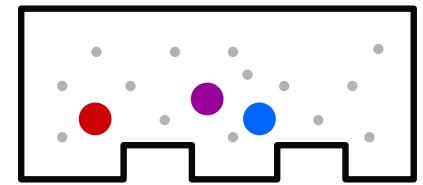
take K samples of r_0 and weight each with a “probability” of $1/K$



- Get the current sensor observation, o_1



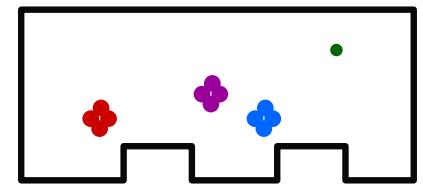
For each sample point r_0 multiply the importance factor by $p(o_1 | r_0, m)$



Normalize (make sure the importance factors add to 1)

You now have an approximation of $p(r_1 | o_1, \dots, m)$
and the distribution is no longer uniform

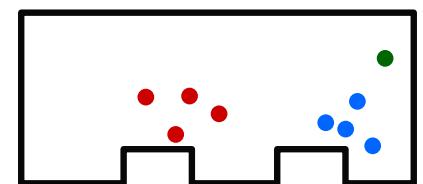
How did this change?



Create new samples by dividing up large clumps
each point spawns new ones in proportion to its importance factor

The robot moves, a_1

For each sample r_1 , move it according to the model $p(r_2 | a_1, r_1, m)$

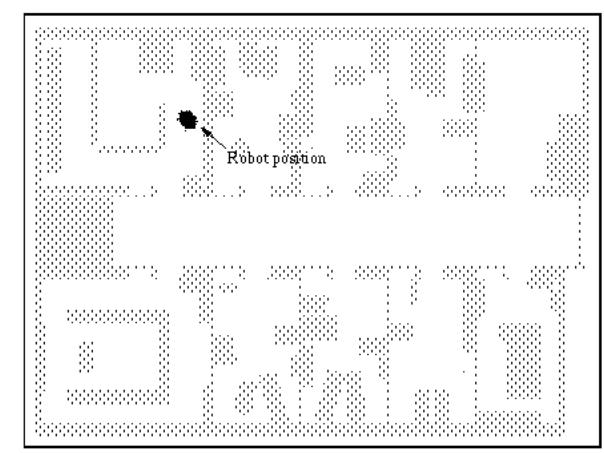
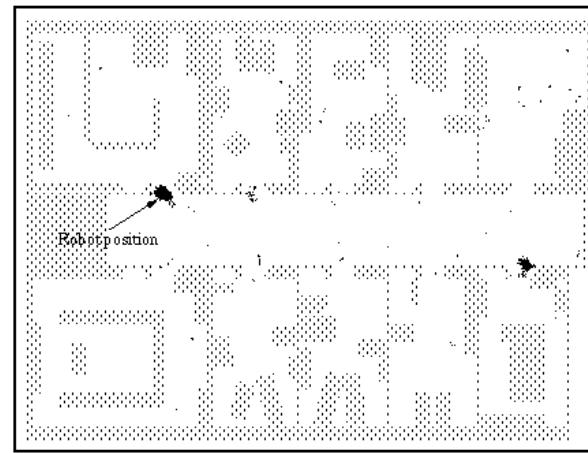
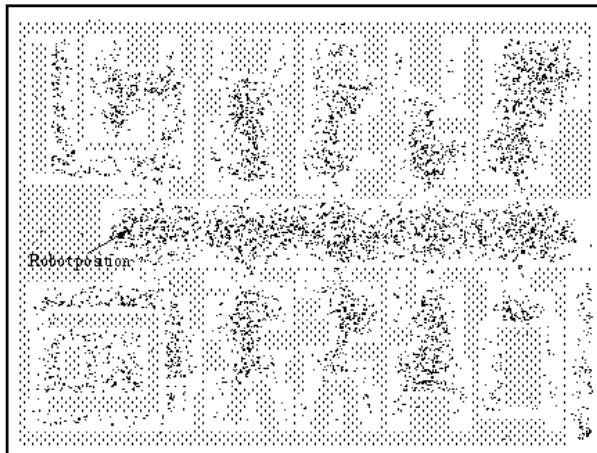


- Increase all the indices by 1 and keep going!

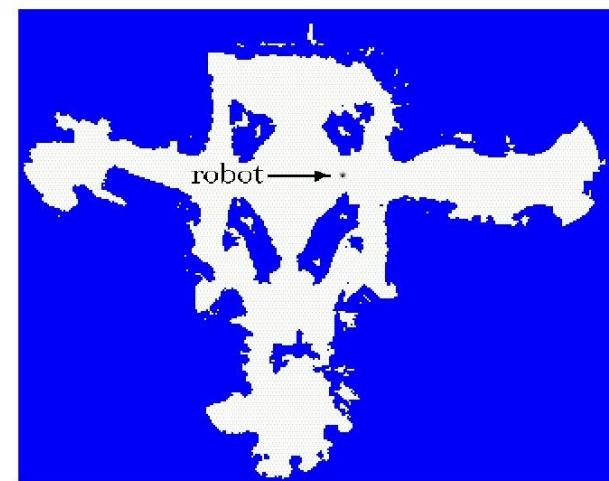
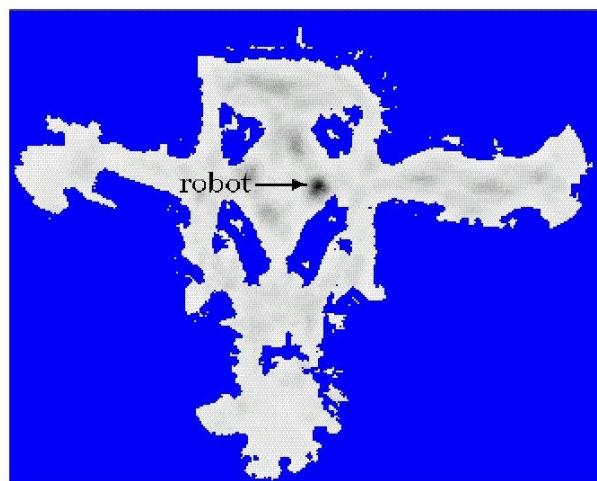
Where do the purple ones go?

MCL in action

“Monte Carlo” Localization -- refers to the resampling of the distribution each time a new observation is integrated



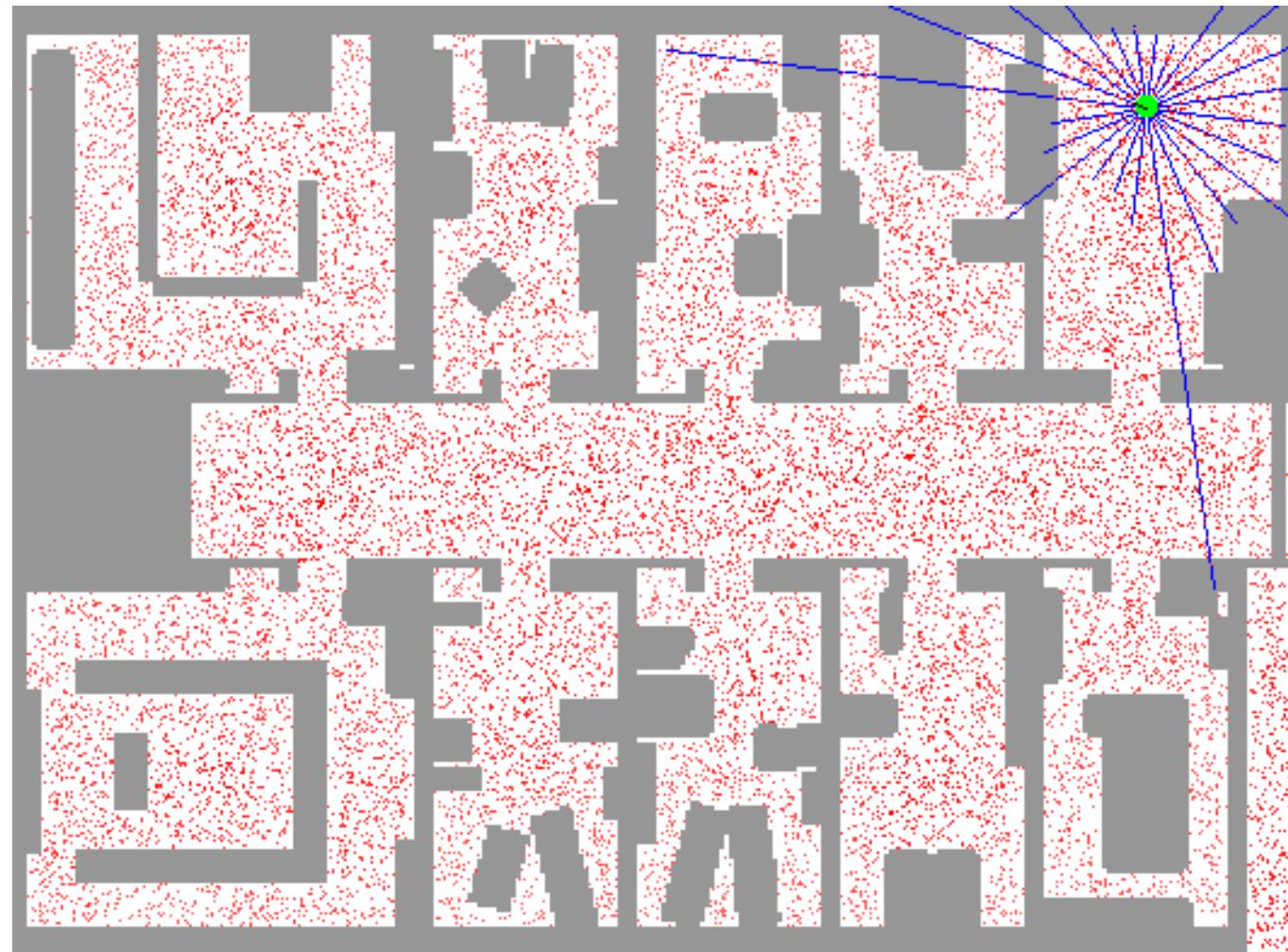
Rhino



Minerva

MCL in action

“Monte Carlo” Localization -- refers to the resampling of the distribution each time a new observation is integrated



in action...

Deriving the MCL algorithm...

How do we find

$p(r_t)$ given all the information available... ?

$$= p(r_t | o_1, a_1, \dots, o_{t-1}, a_{t-1}, o_t, m)$$

$$= \frac{p(o_t | o_1, a_1, \dots, o_{t-1}, a_{t-1}, r_t, m) \cdot p(r_t | o_1, \dots, a_{t-1}, m)}{p(o_t | o_1, \dots, a_{t-1}, m)}$$

$$= \lambda p(o_t | o_1, a_1, \dots, o_{t-1}, a_{t-1}, r_t, m) \cdot p(r_t | o_1, \dots, a_{t-1}, m)$$

$$= \lambda p(o_t | r_t, m) \cdot p(r_t | o_1, \dots, a_{t-1}, m)$$

$$= \lambda p(o_t | r_t, m) \cdot \sum_{\text{all } r_{t-1}} p(r_t | o_1, \dots, a_{t-1}, r_{t-1}, m) \cdot p(r_{t-1} | o_1, \dots, a_{t-1}, m)$$

$$= \underbrace{\lambda p(o_t | r_t, m)}_{\text{all } r_{t-1}} \cdot \underbrace{\sum_{\text{all } r_{t-1}} p(r_t | a_{t-1}, r_{t-1}, m)}_{\text{all } r_{t-1}} \cdot \underbrace{p(r_{t-1} | o_1, \dots, a_{t-1}, m)}_{\text{all } r_{t-1}}$$

Proof

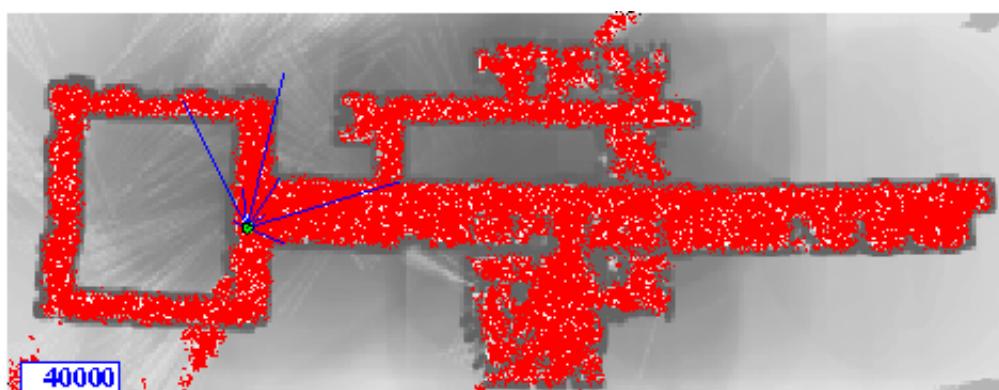
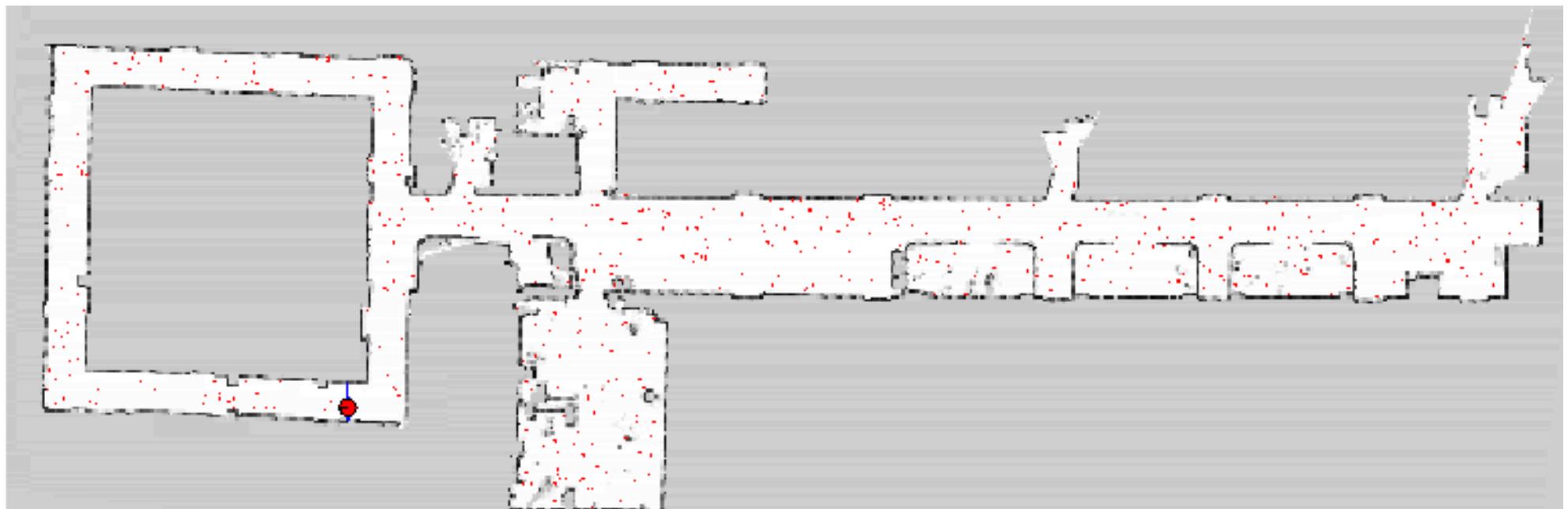
How do we find

$$\begin{aligned} p(r_t) & \text{ given all the information available... ?} \\ = p(r_t | o_1, a_1, \dots, o_{t-1}, a_{t-1}, o_t, m) & \\ = \frac{p(o_t | o_1, a_1, \dots, o_{t-1}, a_{t-1}, r_t, m) \cdot p(r_t | o_1, \dots, a_{t-1}, m)}{p(o_t | o_1, \dots, a_{t-1}, m)} & \text{ Bayes' Rule} \\ = \lambda p(o_t | o_1, a_1, \dots, o_{t-1}, a_{t-1}, r_t, m) \cdot p(r_t | o_1, \dots, a_{t-1}, m) & \text{ No } r \text{ in denom.} \\ = \lambda p(o_t | r_t, m) \cdot p(r_t | o_1, \dots, a_{t-1}, m) & \leftarrow \text{ "Markov" assumption} \\ = \lambda p(o_t | r_t, m) \cdot \sum_{\text{all } r_{t-1}} p(r_t | o_1, \dots, a_{t-1}, r_{t-1}, m) \cdot p(r_{t-1} | o_1, \dots, a_{t-1}, m) & \text{ marginal probabilities} \\ = \underbrace{\lambda p(o_t | r_t, m)}_{\text{this sensor reading}} \cdot \underbrace{\sum_{\text{all } r_{t-1}} p(r_t | a_{t-1}, r_{t-1}, m)}_{\text{possible results of the last motion}} \cdot \underbrace{p(r_{t-1} | o_1, \dots, a_{t-1}, m)}_{\text{previous distribution}} & \text{ conditional independence} \end{aligned}$$

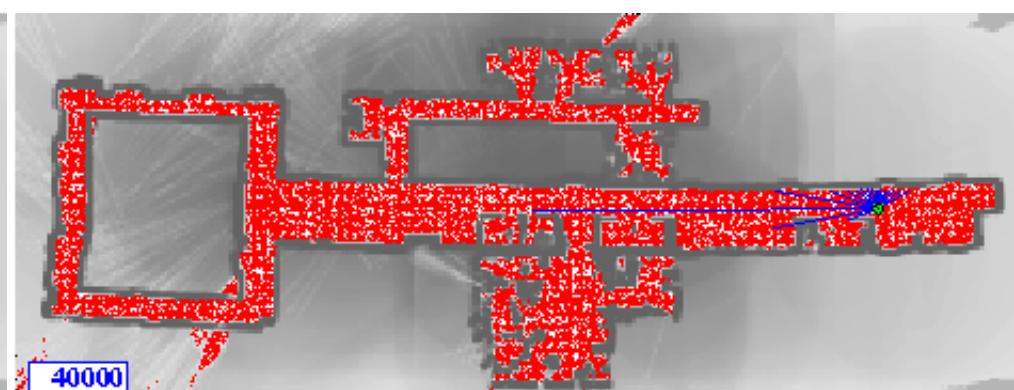
What are these terms?

MCL in action

Color-coded propagation of uncertainty...

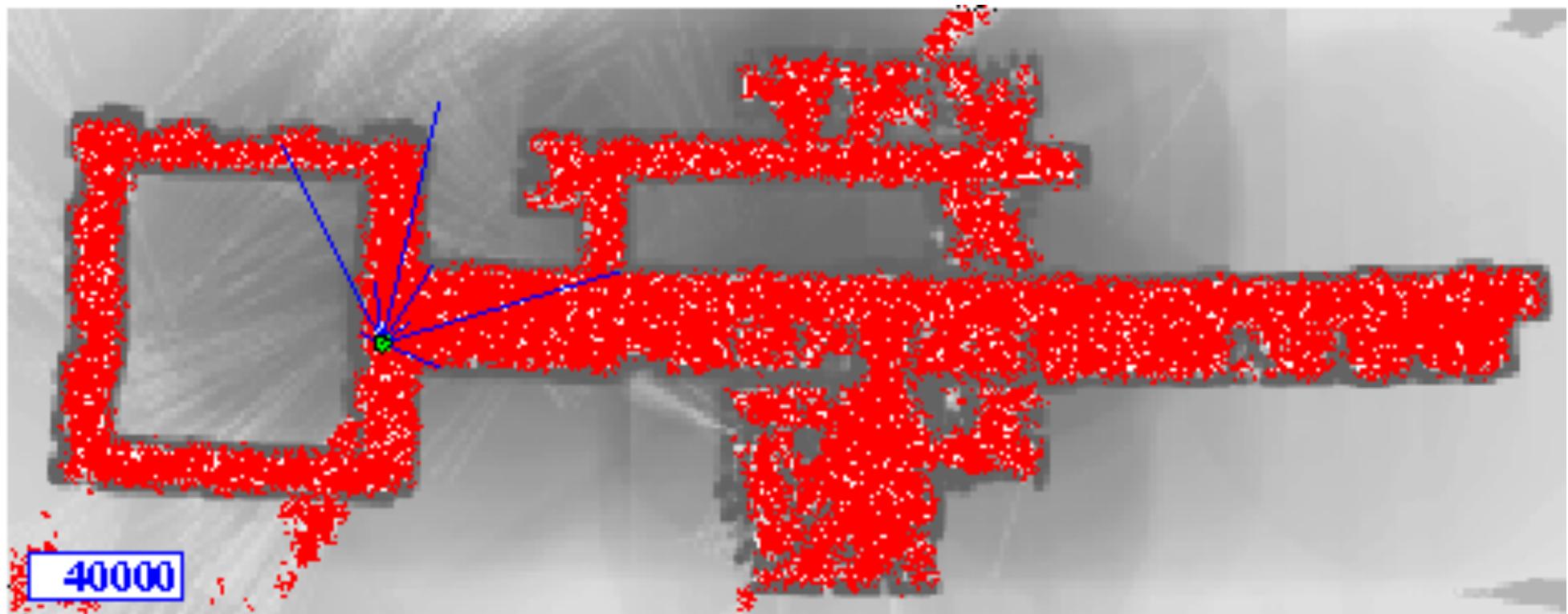


sonar



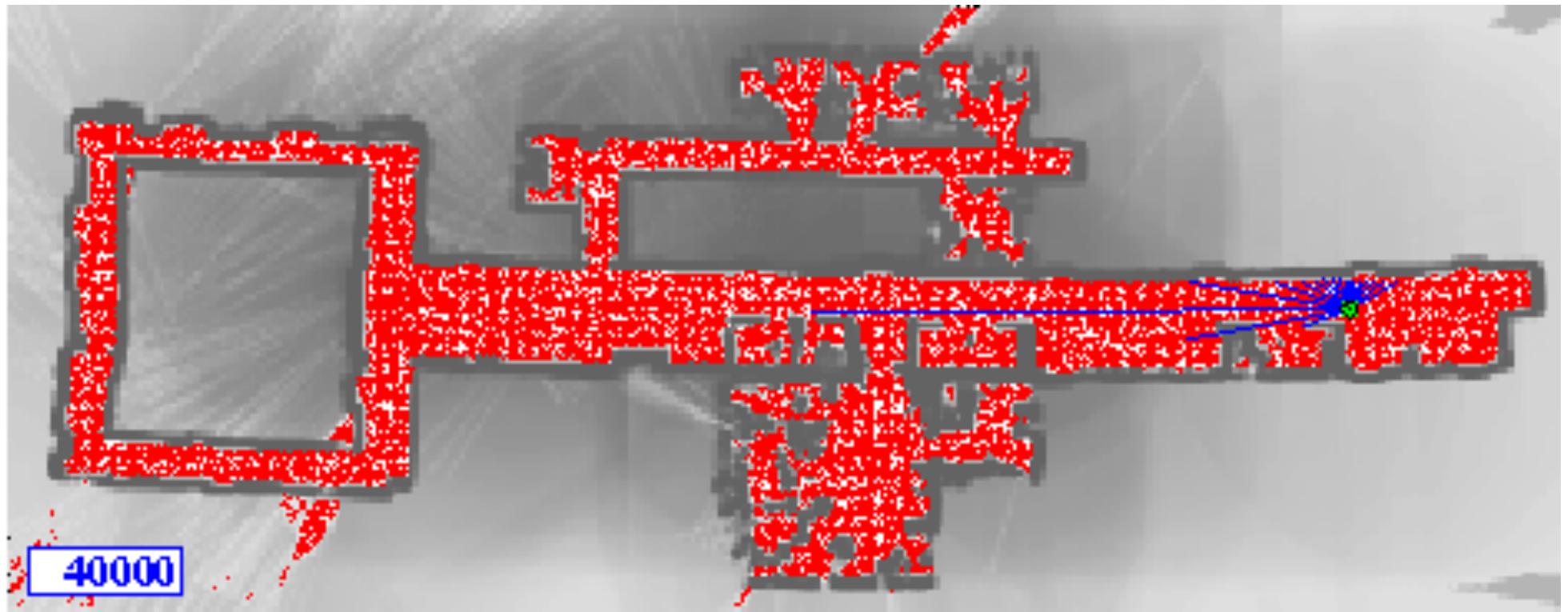
laser rangefinder

MCL in action



using sonar

MCL in action



using a laser rangefinder

Taking a step back...

Plusses

It has worked well in practice!

Simple algorithm

Well-motivated via probabilistic reasoning

Naturally fuses data from very disparate sensors!

Doesn't require control of the robot: *passive* localization

It's an *any-time* algorithm

Handles multi-modal pdfs

Drawbacks

Any-time may not be enough !

Need large number of particles to represent high state spaces

Doesn't *use* the robot control available: *active* localization



SLAM Problem Statement

Simultaneous Localization and Mapping

- Inputs:
 - No external coordinate reference
 - Time sequence of proprioceptive and exteroceptive measurement made as robot moves through an initially unknown environment
- Outputs:
 - A *map* of the environment
 - A robot *pose estimate* associated with each measurement, in the coordinate system in which the map is defined

The SLAM Problem

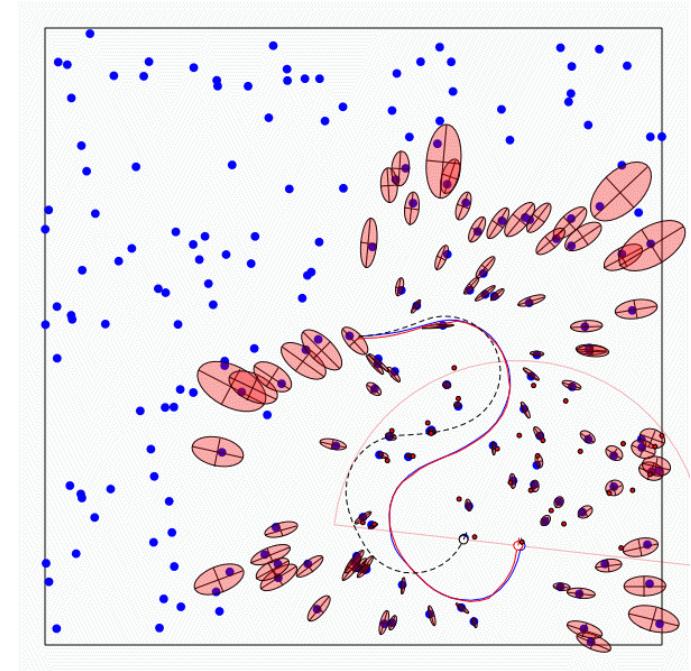
A robot is exploring an unknown, static environment.

Given:

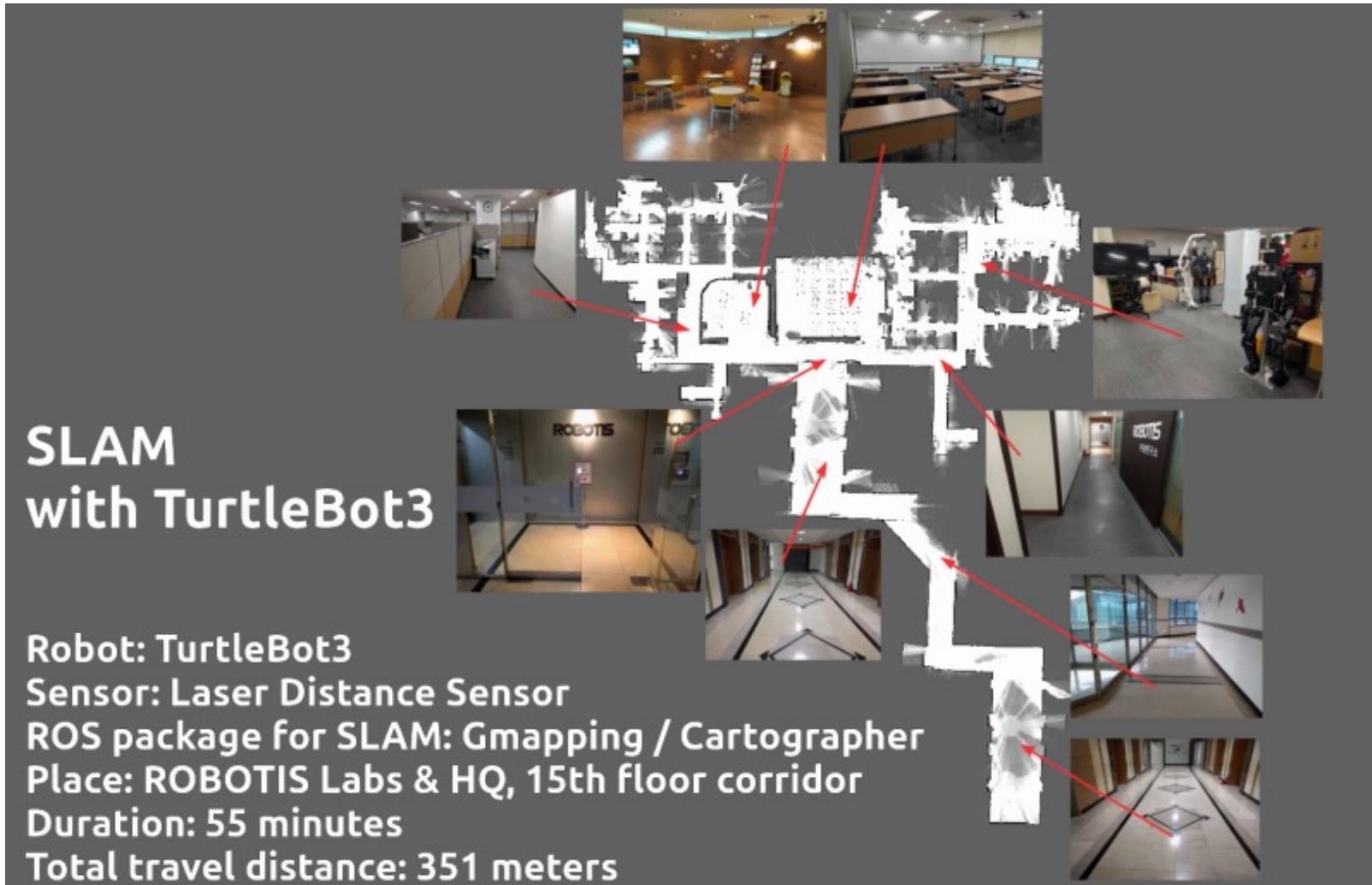
- The robot's controls
- Observations of nearby features

Estimate:

- Map of features
- Path of the robot



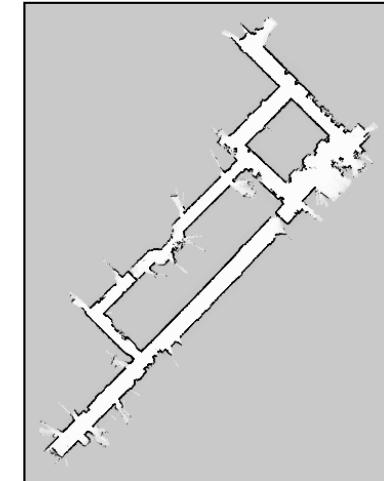
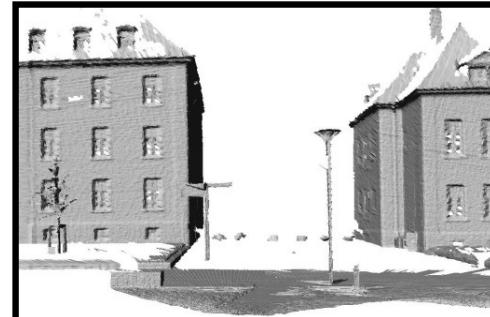
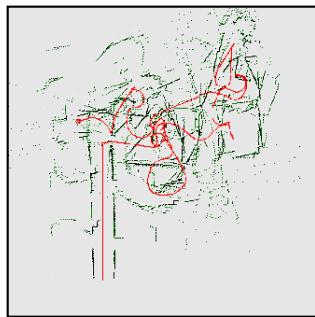
SLAM with Turtlebot3



<https://youtu.be/7iM2ynZEuf0>

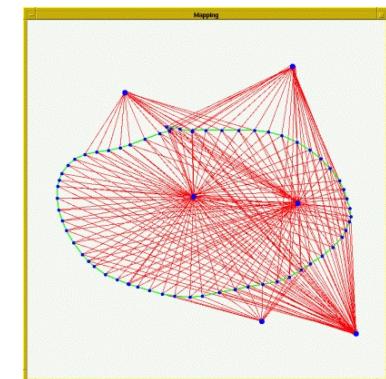
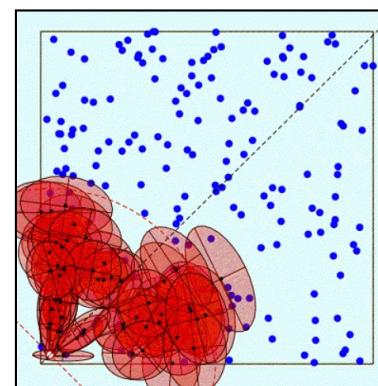
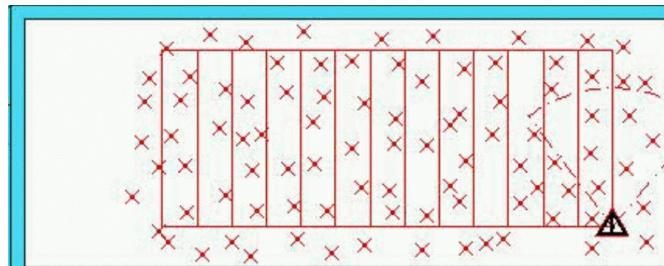
Representations

- Grid maps or scans



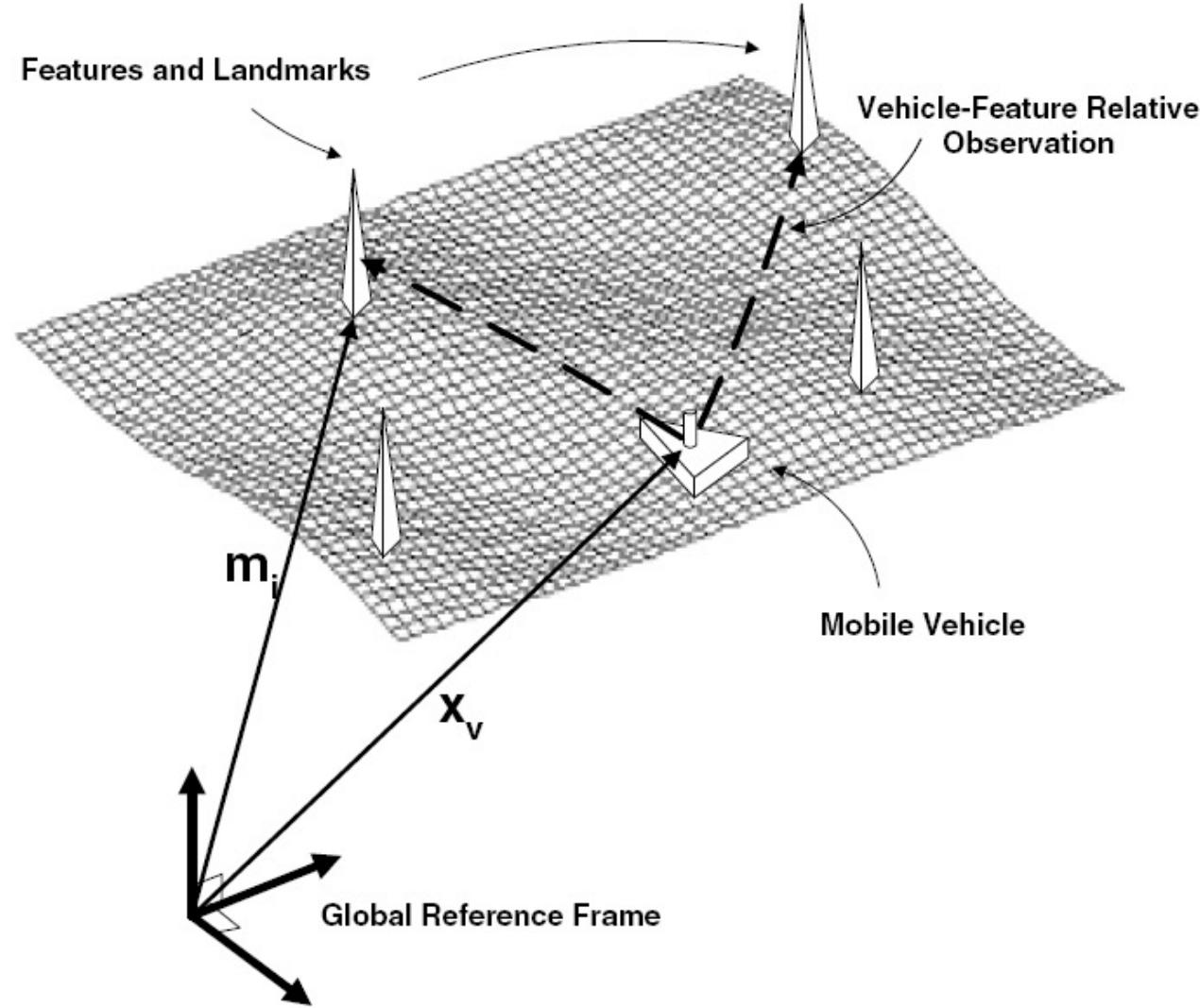
[Lu & Milios, 97; Gutmann, 98; Thrun 98; Burgard, 99; Konolige & Gutmann, 00; Thrun, 00; Arras, 99; Haehnel, 01;...]

- Landmark-based



[Leonard et al., 98; Castelanos et al., 99; Dissanayake et al., 2001; Montemerlo et al., 2002;...]

Landmark-based SLAM-Problem



How to tackle SLAM?

- Global: reason about all available information from previous observations simultaneously
- Local: match information from successive observations
- Intuition: solving a SLAM problem requires tweaking many possible options until you find the best fit solution that explains the data

Illustration of SLAM without Landmarks

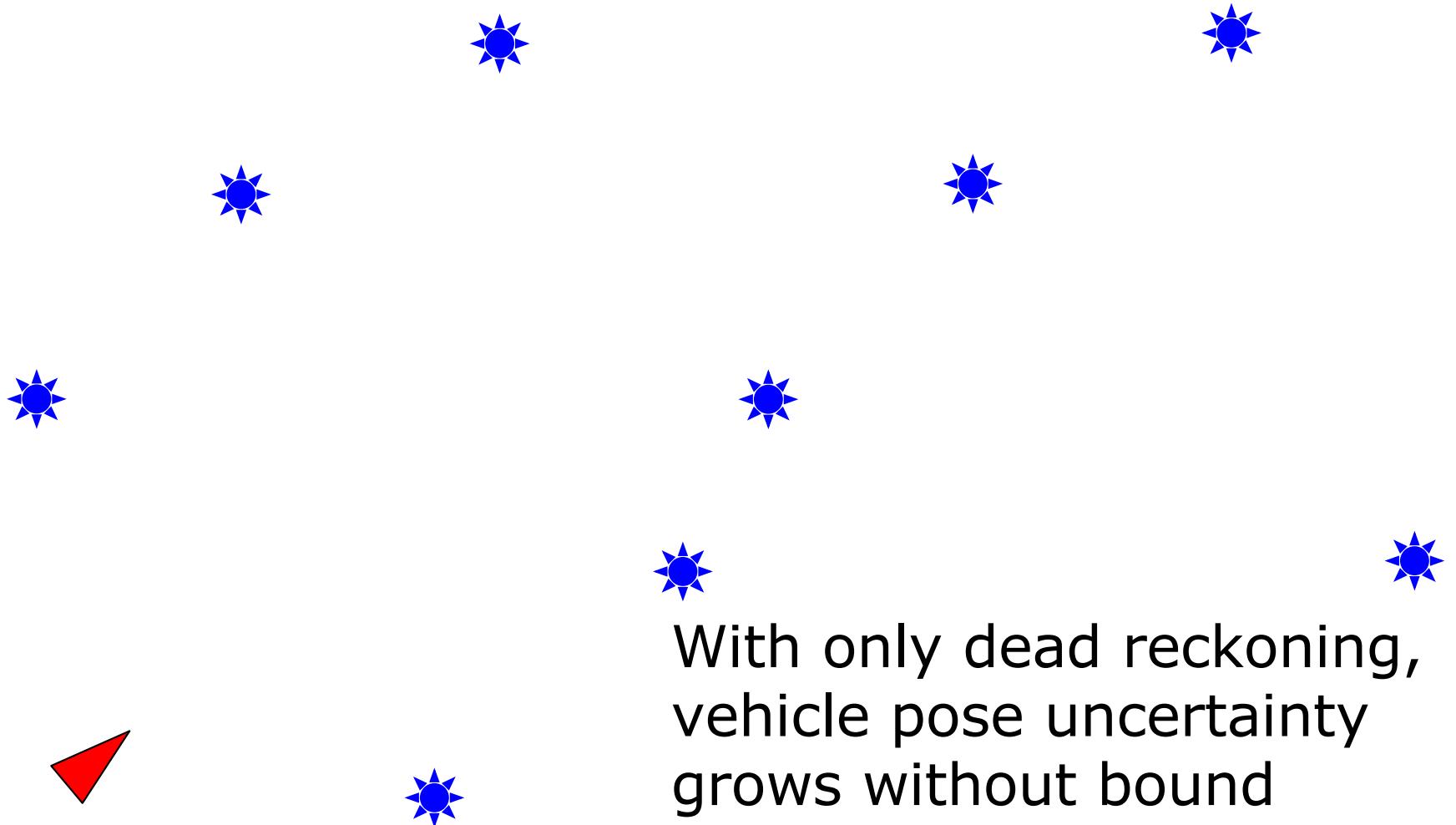


Illustration of SLAM without Landmarks

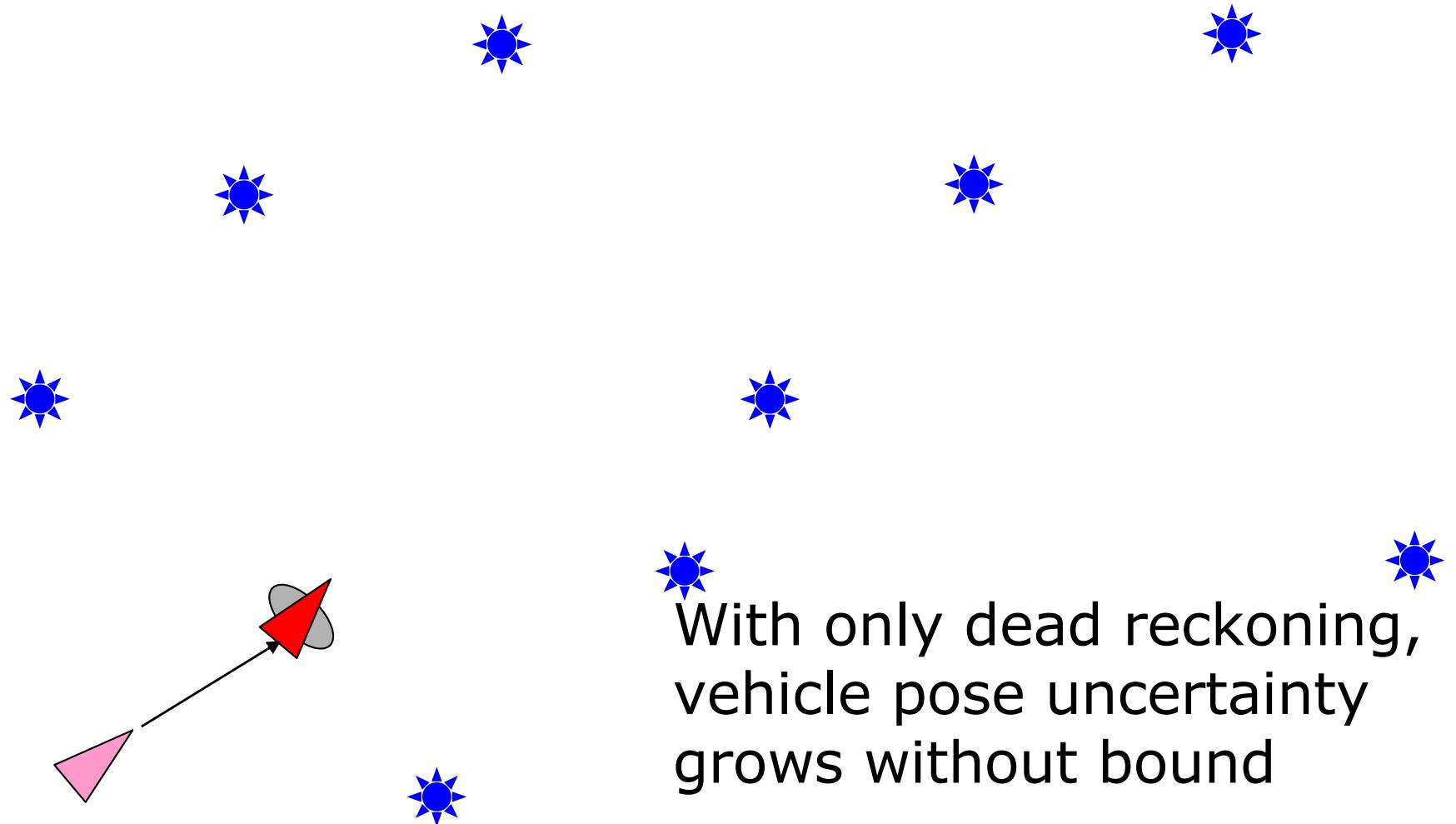


Illustration of SLAM without Landmarks

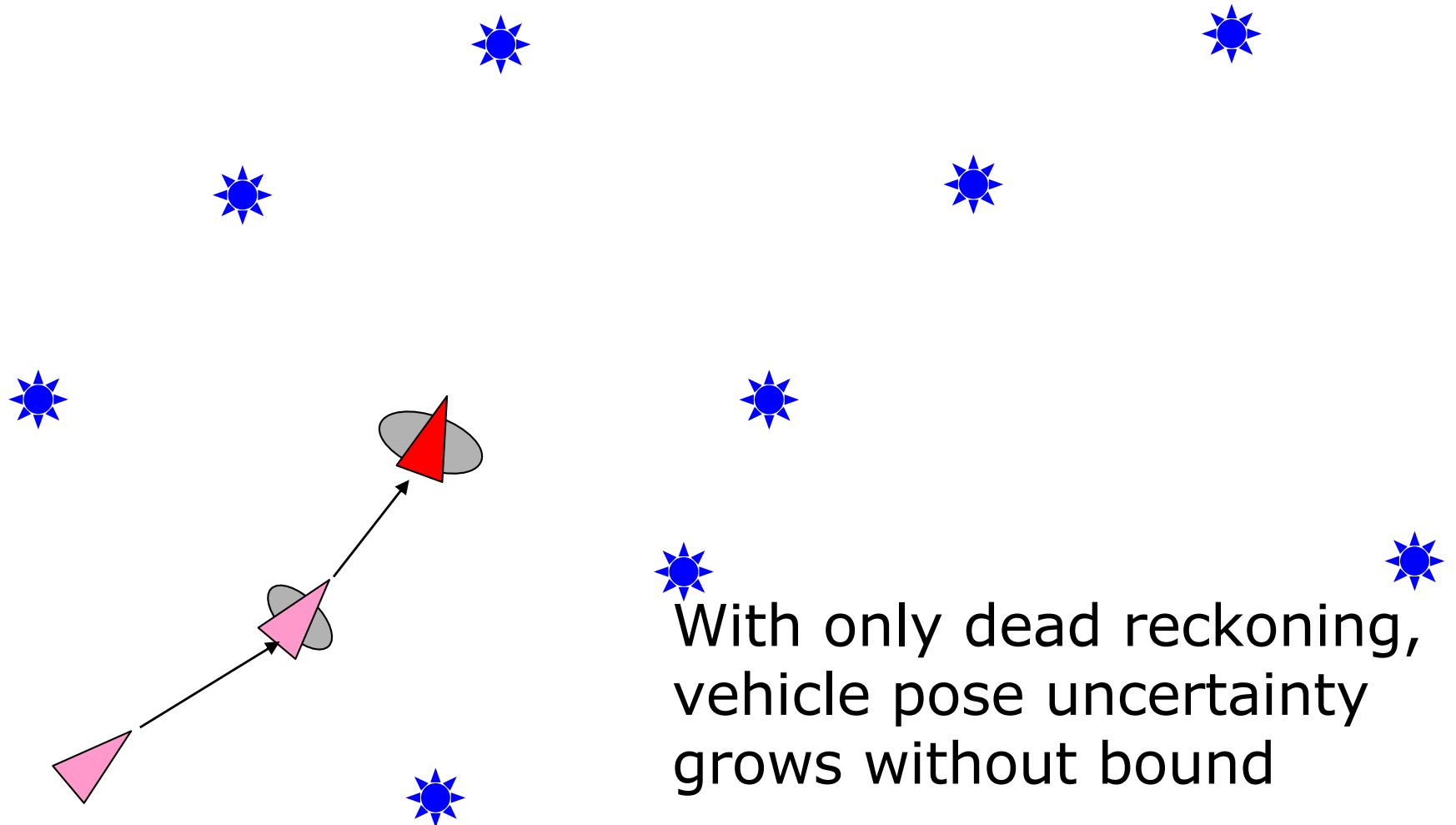


Illustration of SLAM without Landmarks

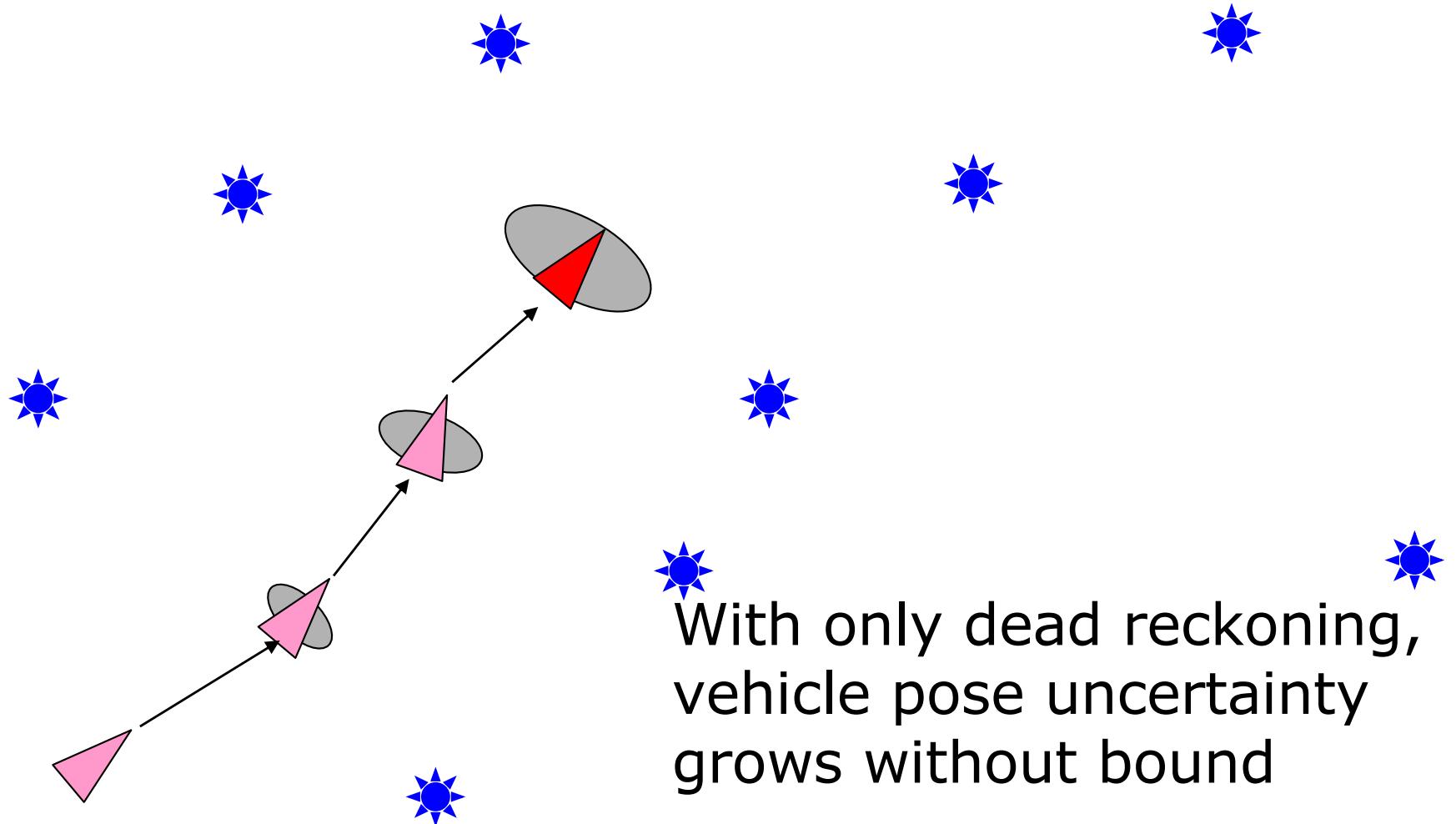


Illustration of SLAM without Landmarks

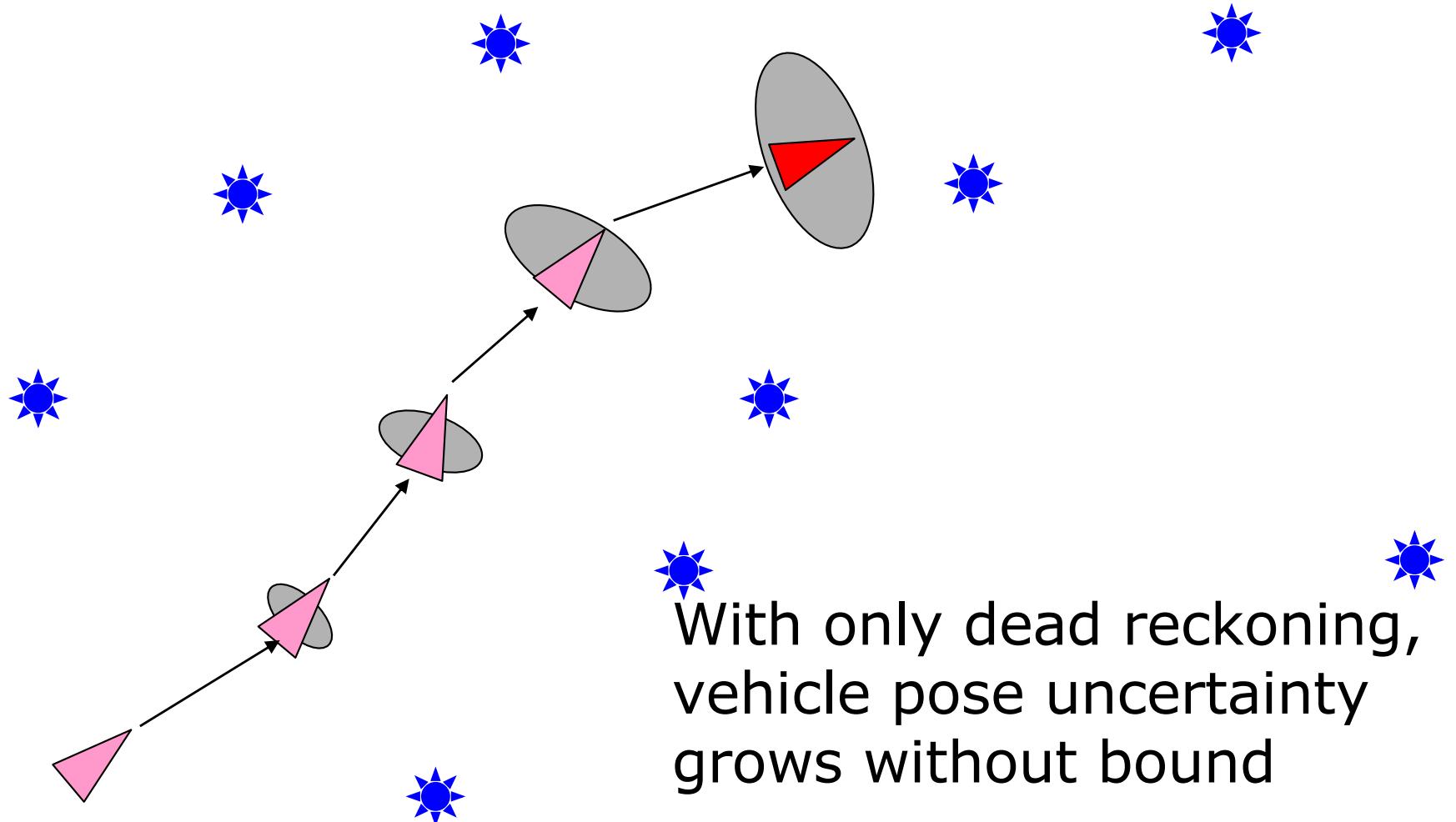
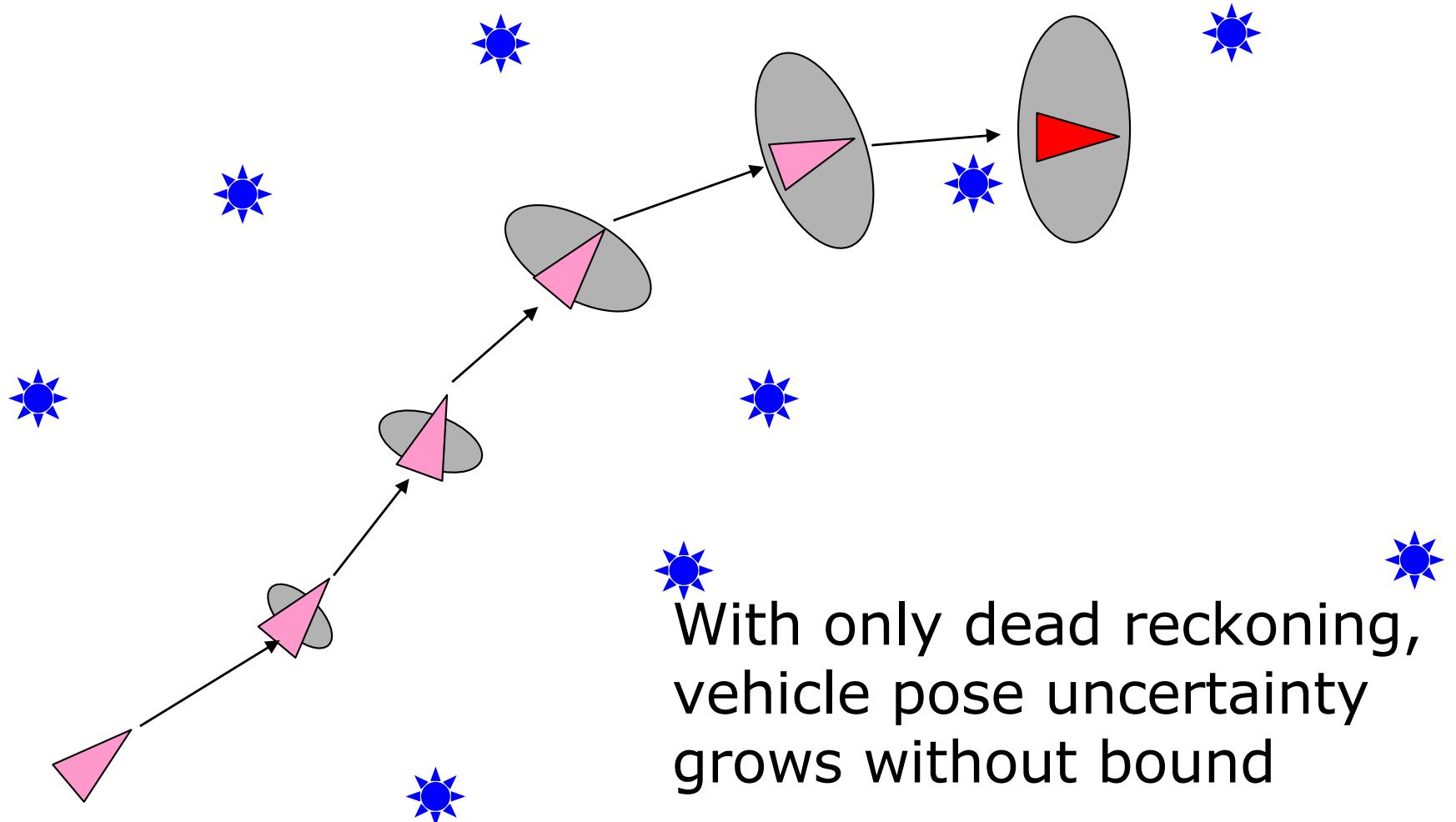


Illustration of SLAM without Landmarks



Repeat, with Measurements of Landmarks

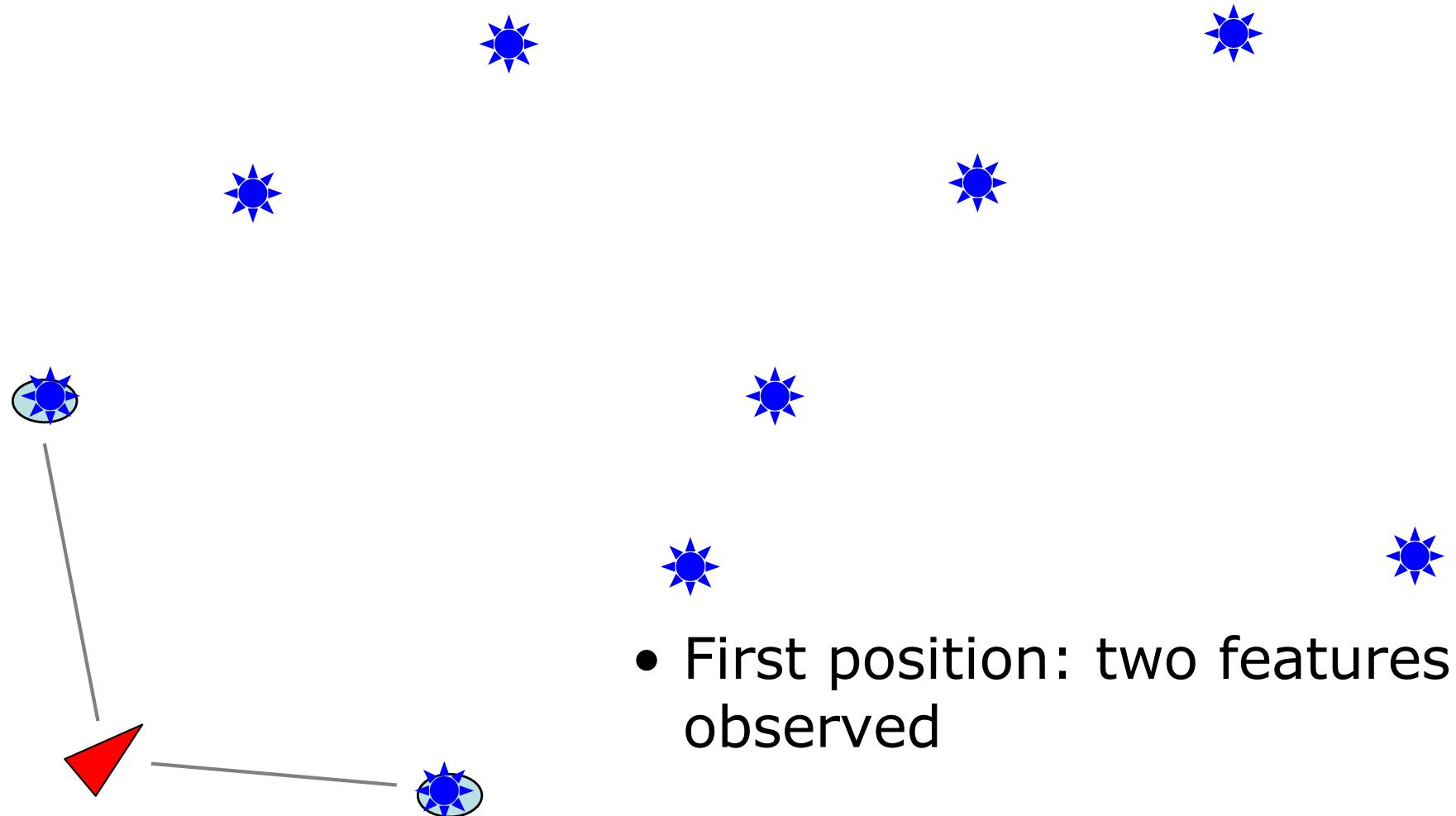


Illustration of SLAM with Landmarks

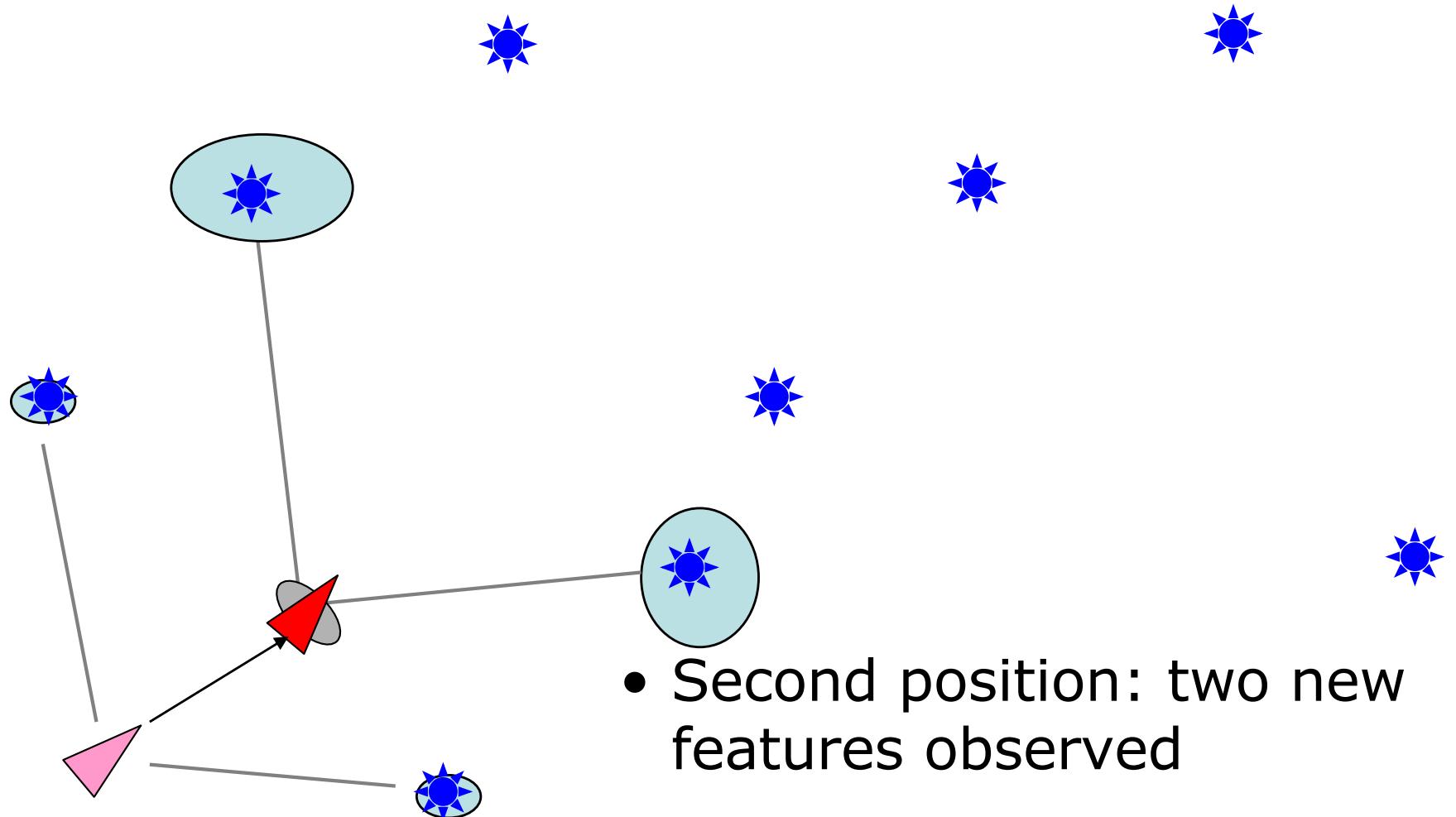


Illustration of SLAM with Landmarks

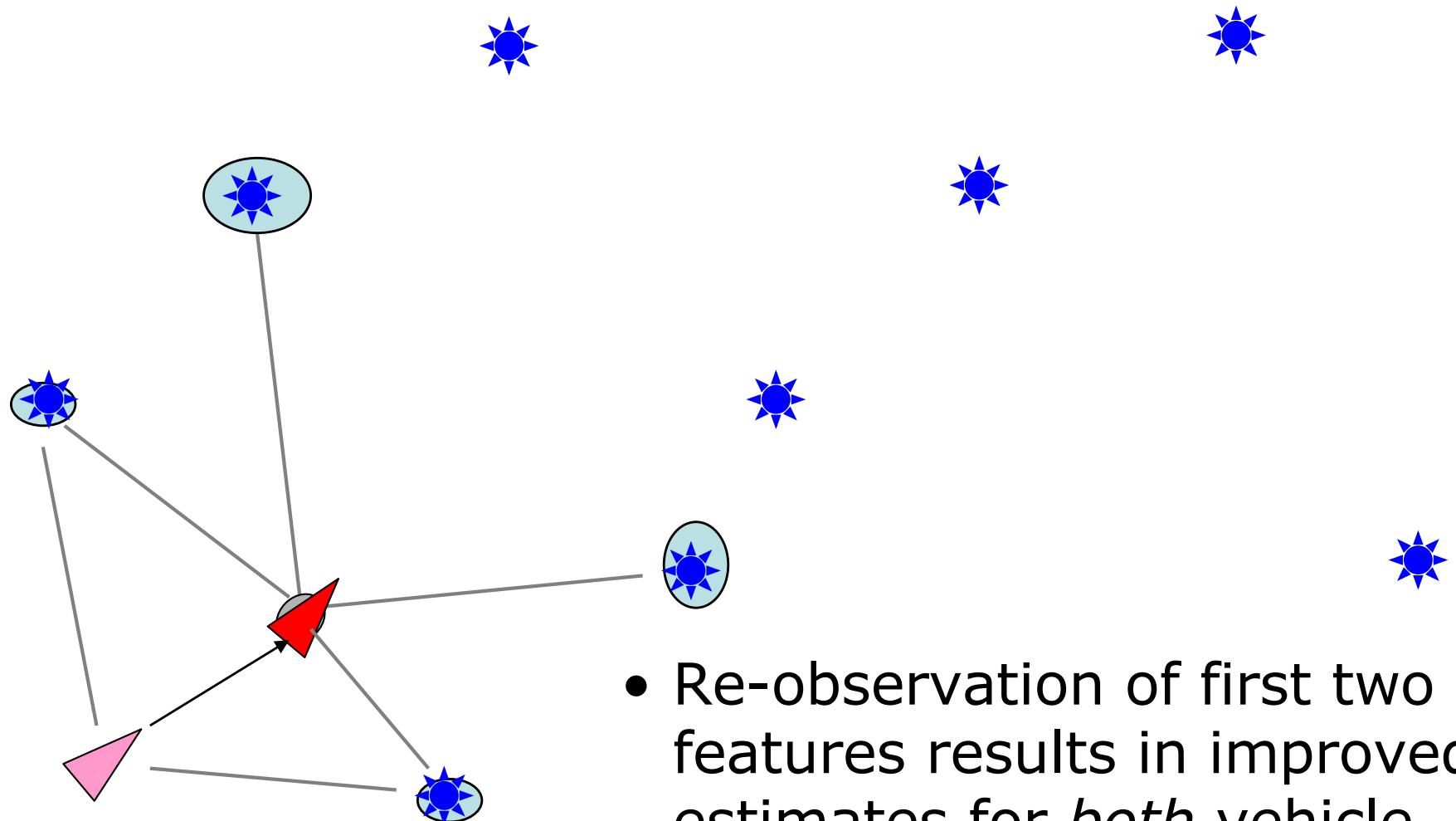


Illustration of SLAM with Landmarks

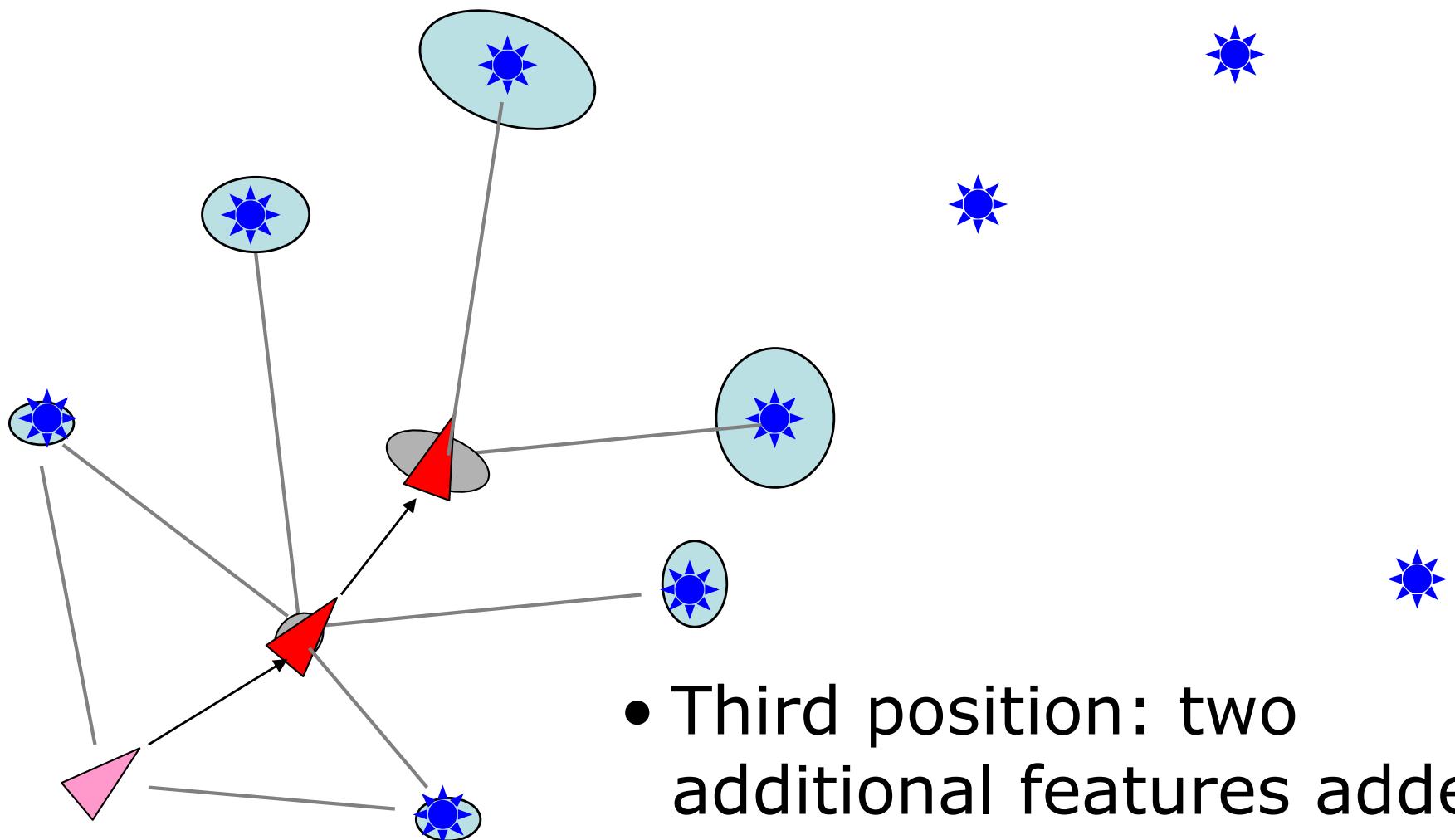


Illustration of SLAM with Landmarks

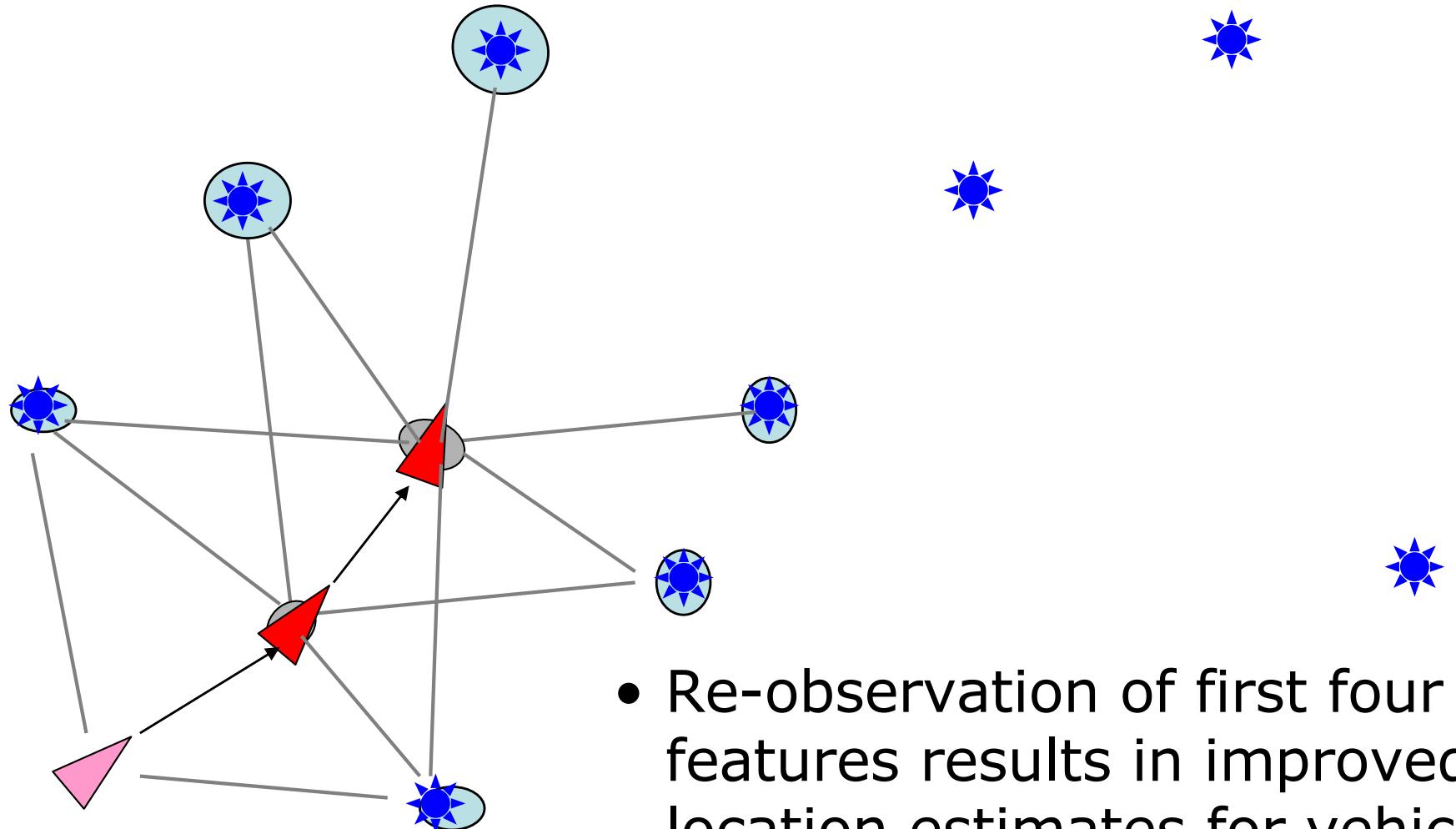
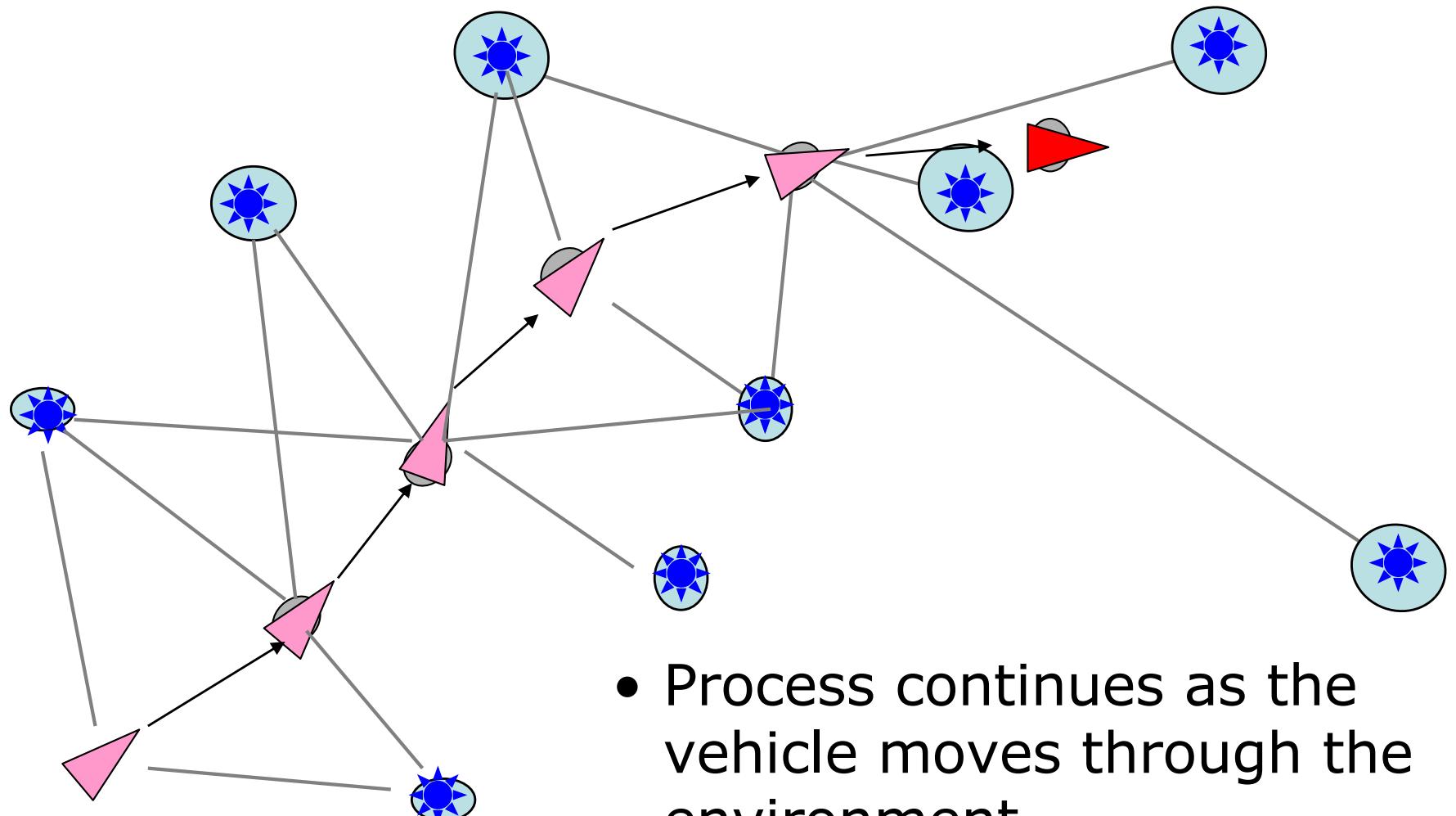
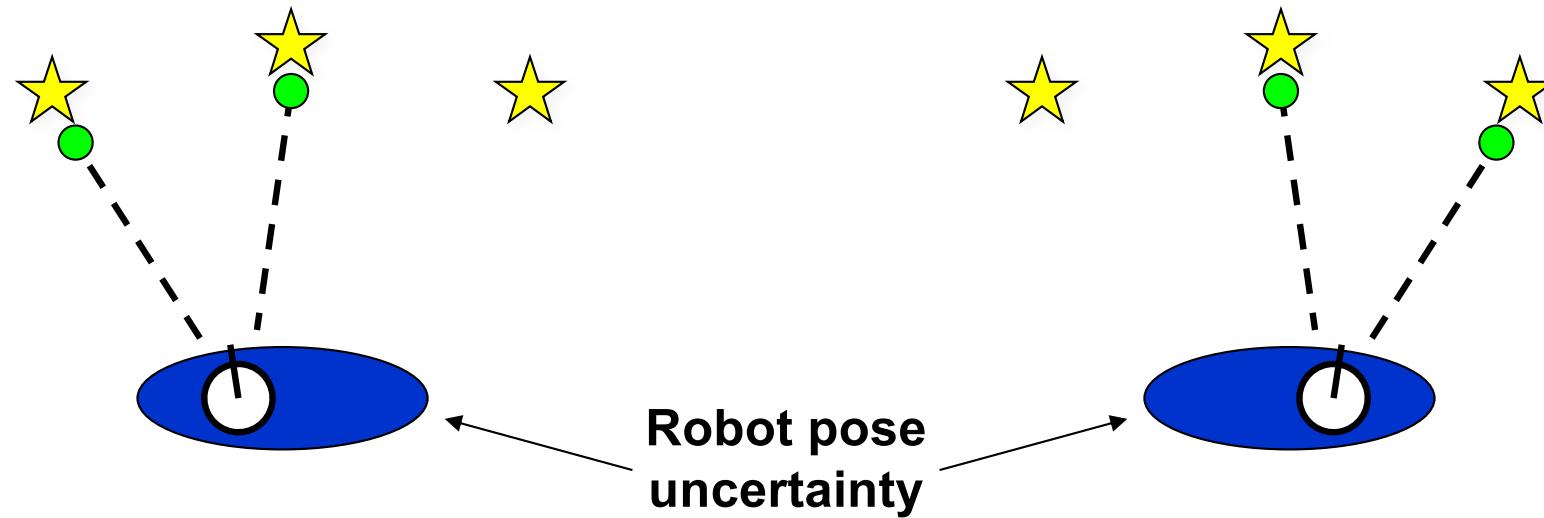


Illustration of SLAM with Landmarks

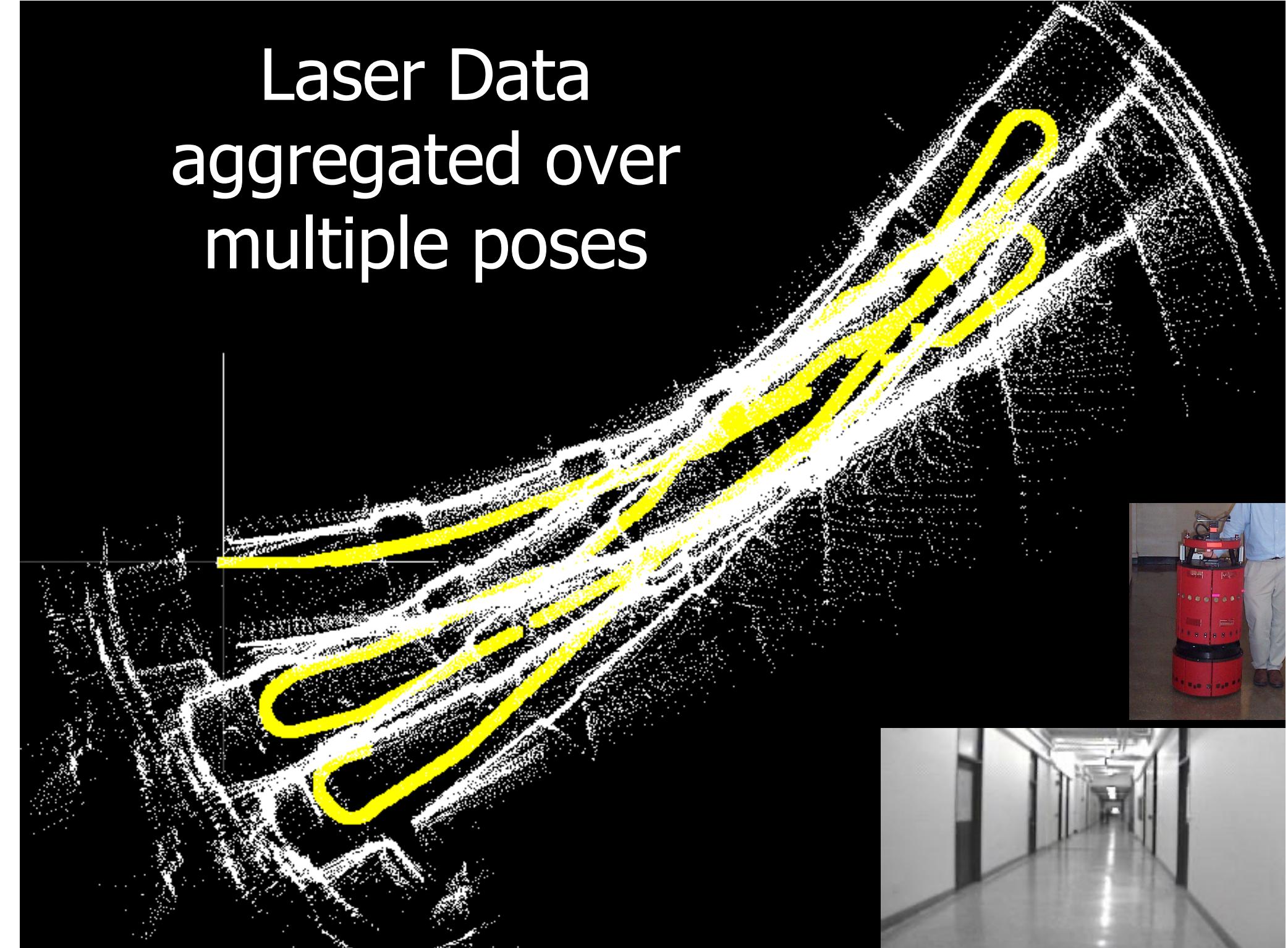


Why is SLAM a hard problem?



- In the real world, the mapping between observations and landmarks is unknown
- Picking wrong data associations can have catastrophic consequences
- Pose error correlates data associations

Laser Data aggregated over multiple poses



SLAM: **Simultaneous Localization and Mapping**

- Full SLAM: Estimates entire path and map!

$$p(x_{1:t}, m | z_{1:t}, u_{1:t})$$

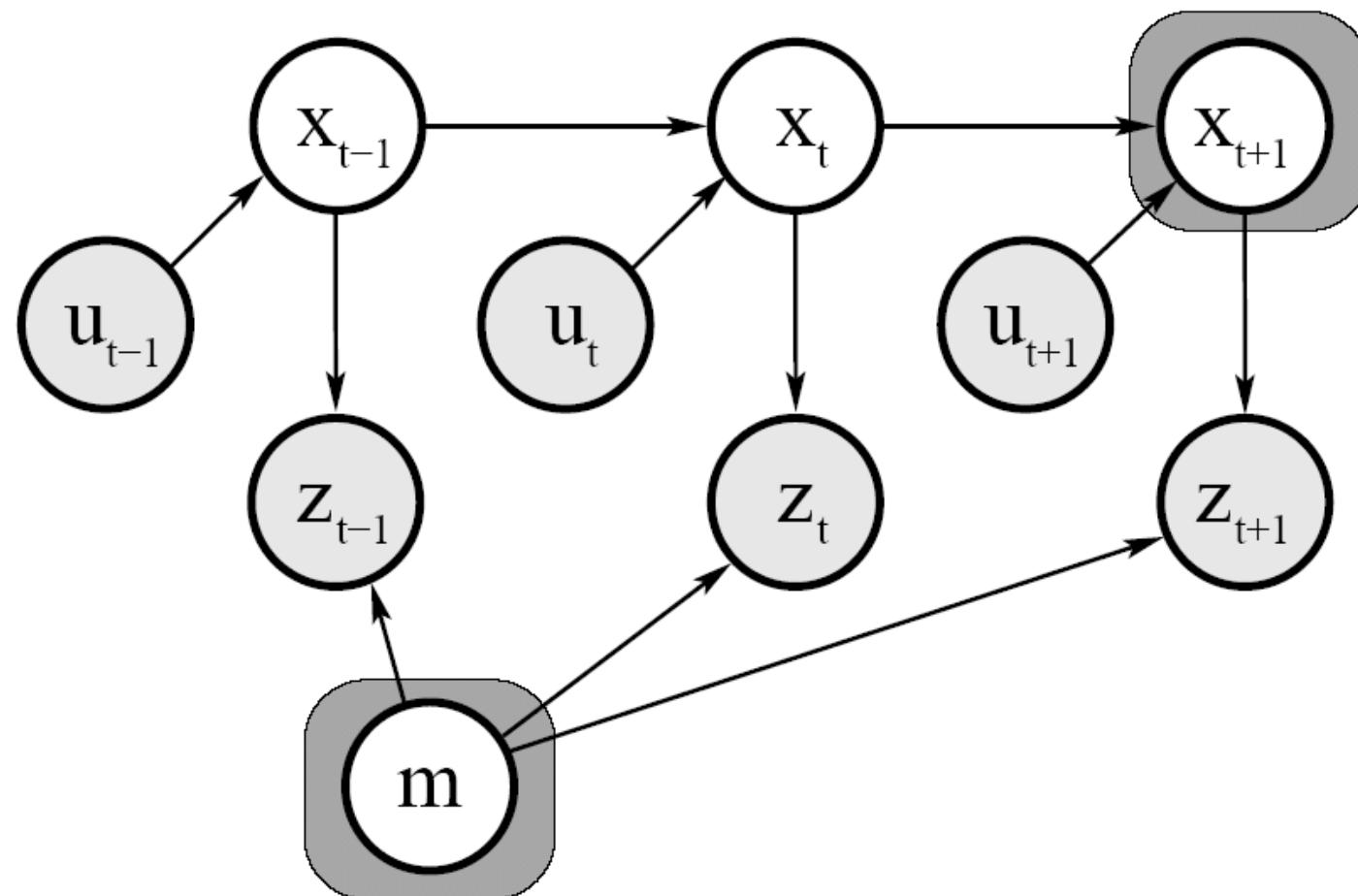
- Online SLAM:

$$p(x_t, m | z_{1:t}, u_{1:t}) = \int \int \dots \int p(x_{1:t}, m | z_{1:t}, u_{1:t}) dx_1 dx_2 \dots dx_{t-1}$$

Integrations typically done one at a time

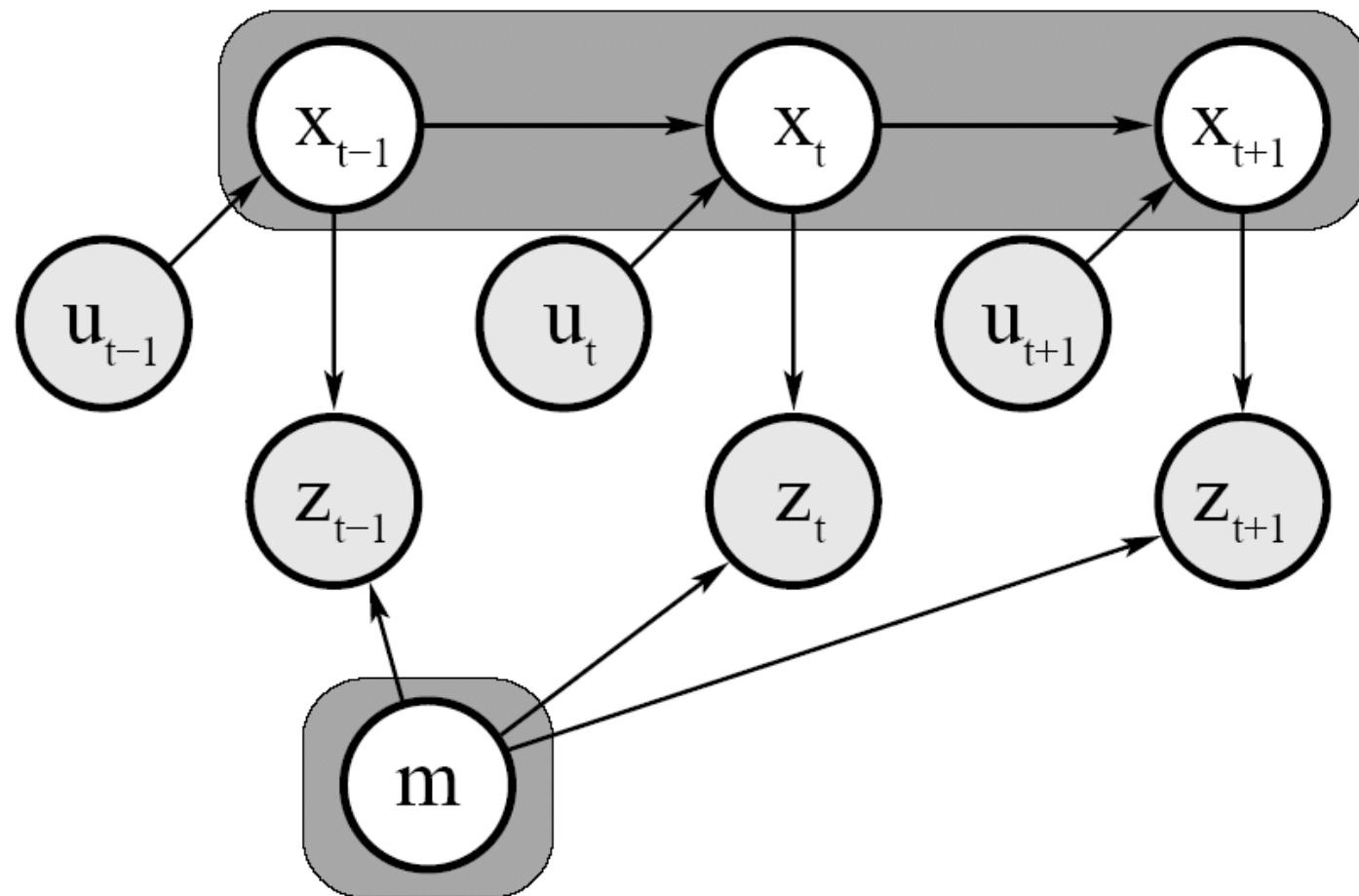
Estimates most recent pose and map!

Graphical Model of Online SLAM:



$$p(x_t, m \mid z_{1:t}, u_{1:t}) = \int \int \dots \int p(x_{1:t}, m \mid z_{1:t}, u_{1:t}) dx_1 dx_2 \dots dx_{t-1}$$

Graphical Model of Full SLAM:



$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t})$$

FastSLAM

Key insight: the landmarks' poses are conditionally independent, given the robot's path...

“Rao-Blackwellization” of a statistical problem...

FastSLAM

Key insight: the landmarks' poses are conditionally independent, given the robot's path...

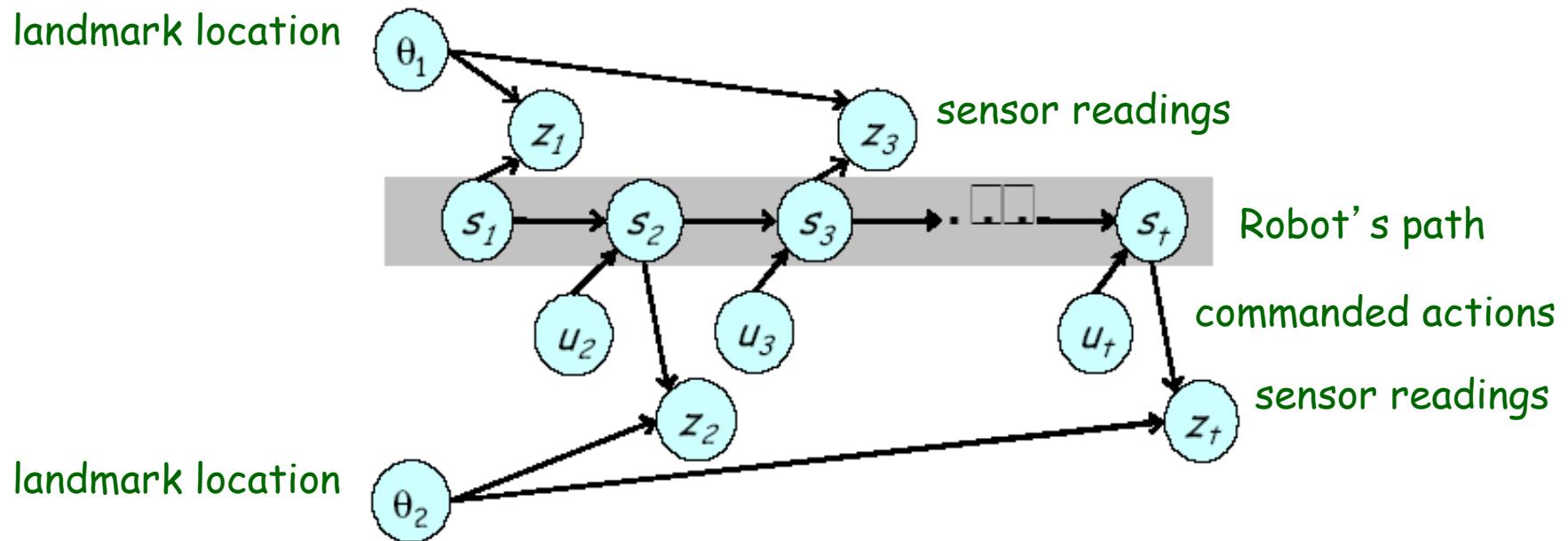


Figure 3: The SLAM problem: The robot moves from pose s_1 through a sequence of controls, u_1, u_2, \dots . As it moves, it measures nearby landmarks. At time $t = 1$, it observes landmark θ_1 out of two landmarks, $\{\theta_1, \theta_2\}$. The measurement is denoted z_1 (range and bearing). At time $t = 1$, it observes the other landmark, θ_2 , and at time $t = 3$, it observes θ_1 again. The SLAM problem is concerned with estimating the locations of the landmarks and the robot's path from the controls u^t and the measurements z^t . The gray shading illustrates a conditional independence relation.

FastSLAM

Key insight:

the landmarks' poses are conditionally independent, given the robot's path...

In other words:

given a guess as to the robot's path so far,
sensing a LM affects *only that LM's pose!*



How to take
advantage of this?

FastSLAM

Key insight:

the landmarks' poses are conditionally independent, given the robot's path...

In other words:

given a guess as to the robot's path so far,
sensing a LM affects *only that LM's pose!*



Create a **particle filter** to keep track of hypotheses of the **robot's path** so far... .

How to take advantage of this?

Each particle holds a **full map of landmarks**, with each LM represented by a Kalman Filter.

Each particle guesses **which LM is which for itself**, based on the path its taken so far...

Factored Posterior

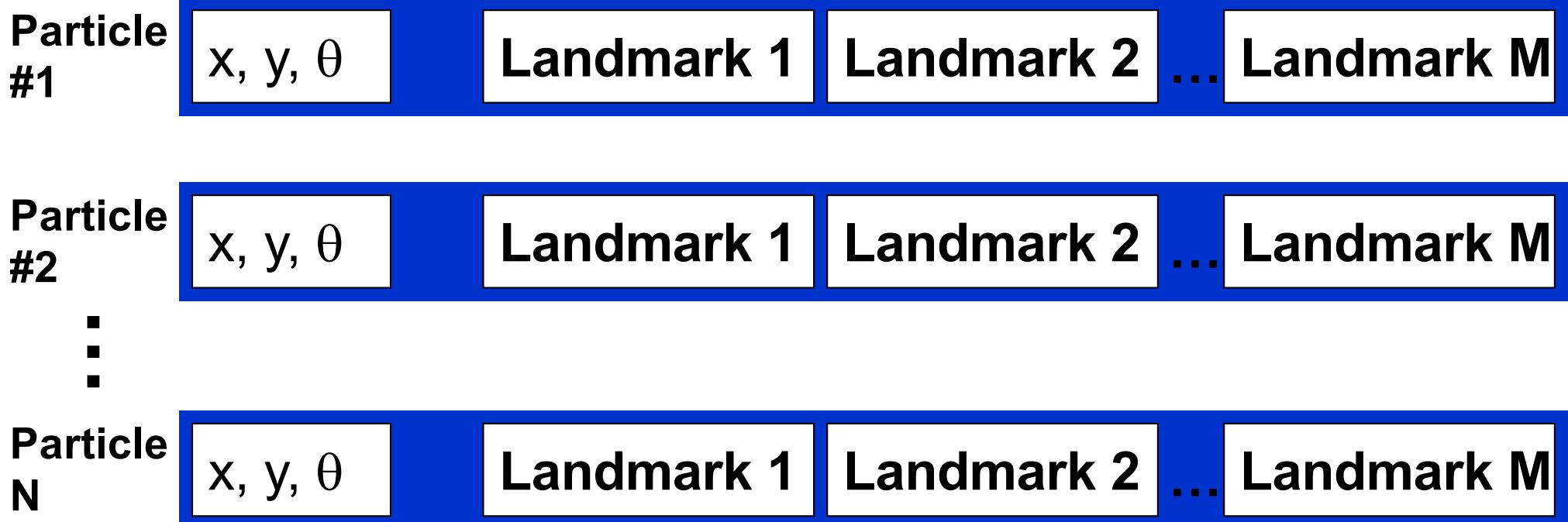
$$\begin{aligned} p(x_{1:t}, l_{1:m} \mid z_{1:t}, u_{0:t-1}) \\ = p(x_{1:t} \mid z_{1:t}, u_{0:t-1}) \cdot p(l_{1:m} \mid x_{1:t}, z_{1:t}) \\ = p(x_{1:t} \mid z_{1:t}, u_{0:t-1}) \cdot \prod_{i=1}^M p(l_i \mid x_{1:t}, z_{1:t}) \end{aligned}$$

Robot path posterior
(localization problem)

Conditionally
independent
landmark positions

FastSLAM

- Rao-Blackwellized particle filtering based on landmarks [Montemerlo et al., 2002]
- Each landmark is represented by a 2x2 Extended Kalman Filter (EKF)
- Each particle therefore has to maintain M EKFs



Components of a Kalman Filter

A_t

Matrix ($n \times n$) that describes how the state evolves from t to $t-1$ without controls or noise.

B_t

Matrix ($n \times l$) that describes how the control u_t changes the state from t to $t-1$.

C_t

Matrix ($k \times n$) that describes how to map the state x_t to an observation z_t .

ε_t

Random variables representing the process and measurement noise that are assumed to be independent and normally distributed with covariance R_t and Q_t respectively.

δ_t

Linear Gaussian Systems: Initialization

- Initial belief is normally distributed:

$$bel(x_0) = N(x_0; \mu_0, \Sigma_0)$$

Linear Gaussian Systems: Dynamics

- Dynamics are linear function of state and control plus additive noise:

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t$$

$$p(x_t | u_t, x_{t-1}) = N(x_t; A_t x_{t-1} + B_t u_t, R_t)$$

$$\overline{bel}(x_t) = \int p(x_t | u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}$$



$$\sim N(x_t; A_t x_{t-1} + B_t u_t, R_t) \quad \sim N(x_{t-1}; \mu_{t-1}, \Sigma_{t-1})$$

Linear Gaussian Systems: Observations

- Observations are linear function of state plus additive noise:

$$z_t = C_t x_t + \delta_t$$

$$p(z_t | x_t) = N(z_t; C_t x_t, Q_t)$$

$$bel(x_t) = \eta p(z_t | x_t)$$



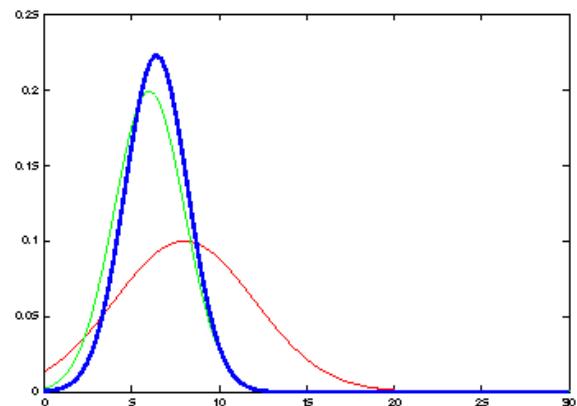
$$\sim N(z_t; C_t x_t, Q_t)$$

$$\overline{bel}(x_t)$$



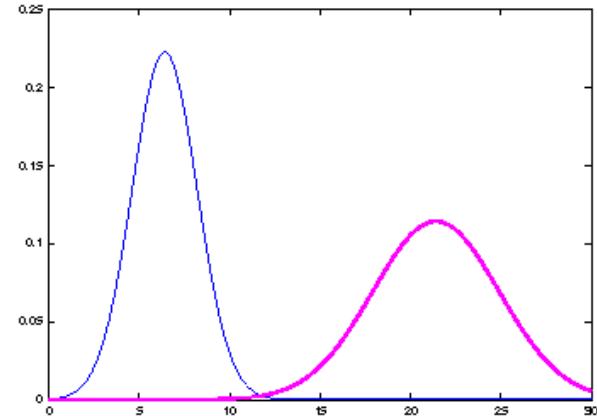
$$\sim N(x_t; \bar{\mu}_t, \bar{\Sigma}_t)$$

The Prediction-Correction-Cycle

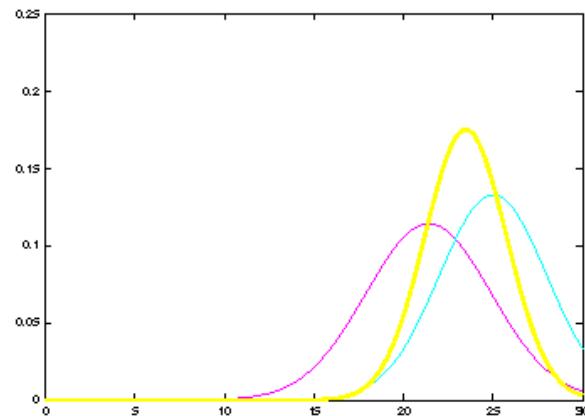


$$\overline{bel}(x_t) = \begin{cases} \bar{\mu}_t = a_t \mu_{t-1} + b_t u_t \\ \bar{\sigma}_t^2 = a_t^2 \sigma_t^2 + \sigma_{act,t}^2 \end{cases}$$

$$\overline{bel}(x_t) = \begin{cases} \bar{\mu}_t = A_t \mu_{t-1} + B_t u_t \\ \Sigma_t = A_t \Sigma_{t-1} A_t^T + R_t \end{cases}$$

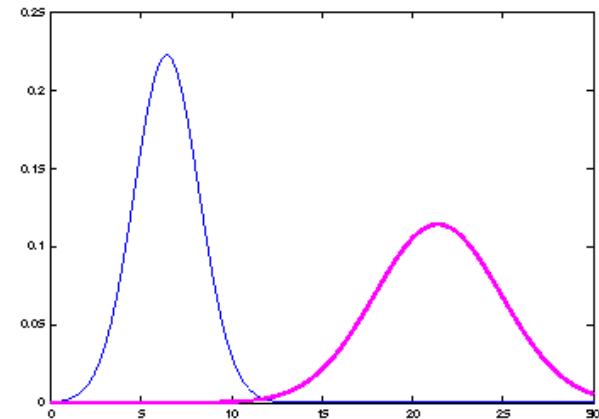


The Prediction-Correction-Cycle



$$bel(x_t) = \begin{cases} \mu_t = \bar{\mu}_t + K_t(z_t - \bar{\mu}_t), \\ \sigma_t^2 = (1 - K_t)\bar{\sigma}_t^2, \end{cases}, K_t = \frac{\bar{\sigma}_t^2}{\bar{\sigma}_t^2 + \bar{\sigma}_{obs,t}^2}$$

$$bel(x_t) = \begin{cases} \mu_t = \bar{\mu}_t + K_t(z_t - C_t \bar{\mu}_t), \\ \Sigma_t = (I - K_t C_t) \bar{\Sigma}_t \end{cases}, K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$$



The Prediction-Correction-Cycle



$$bel(x_t) = \begin{cases} \mu_t = \bar{\mu}_t + K_t(z_t - \bar{\mu}_t) \\ \sigma_t^2 = (1 - K_t)\bar{\sigma}_t^2 \end{cases}, K_t = \frac{\bar{\sigma}_t^2}{\bar{\sigma}_t^2 + \bar{\sigma}_{obs,t}^2}$$

$$bel(x_t) = \begin{cases} \mu_t = \bar{\mu}_t + K_t(z_t - C_t \bar{\mu}_t) \\ \Sigma_t = (I - K_t C_t) \bar{\Sigma}_t \end{cases}, K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$$

$$\overline{bel}(x_t) = \begin{cases} \bar{\mu}_t = a_t \mu_{t-1} + b_t u_t \\ \bar{\sigma}_t^2 = a_t^2 \sigma_t^2 + \sigma_{act,t}^2 \end{cases}$$

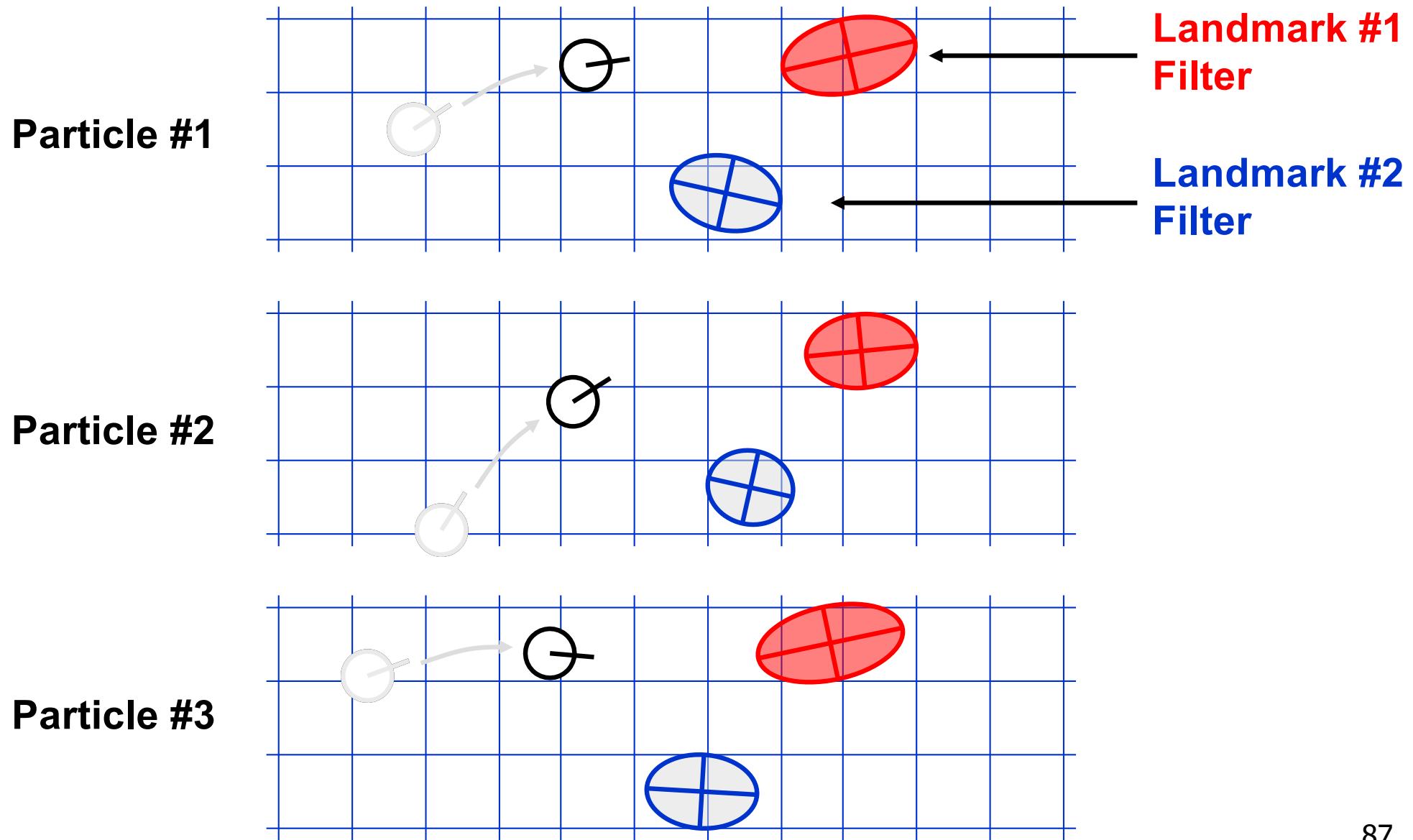
$$\overline{bel}(x_t) = \begin{cases} \bar{\mu}_t = A_t \mu_{t-1} + B_t u_t \\ \bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t \end{cases}$$



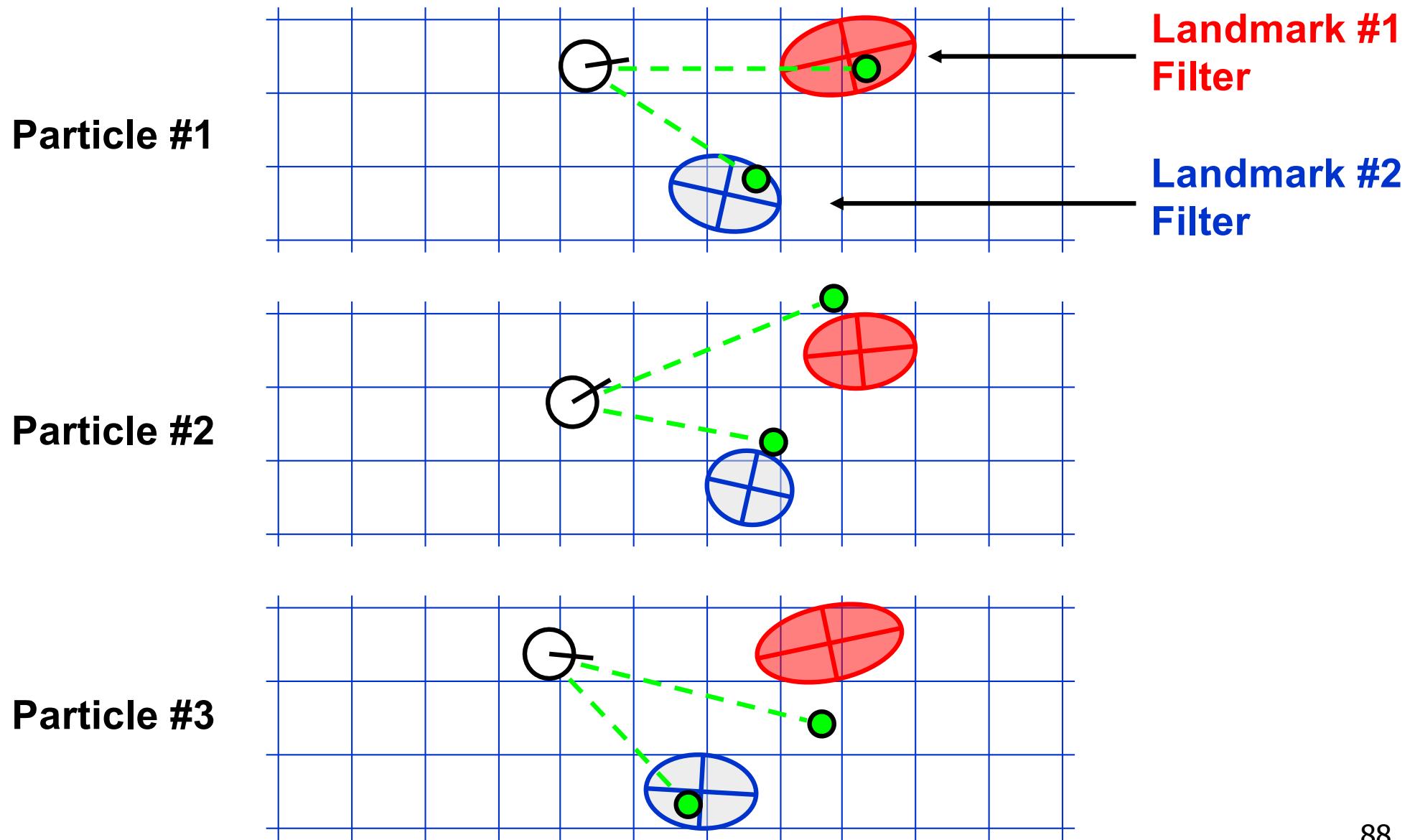
Kalman Filter Algorithm

1. Algorithm **Kalman_filter**(μ_{t-1} , Σ_{t-1} , u_t , z_t):
2. Prediction:
3. $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
4. $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
5. Correction:
6. $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$
7. $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$
8. $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$
9. Return μ_t , Σ_t

FastSLAM – Action Update

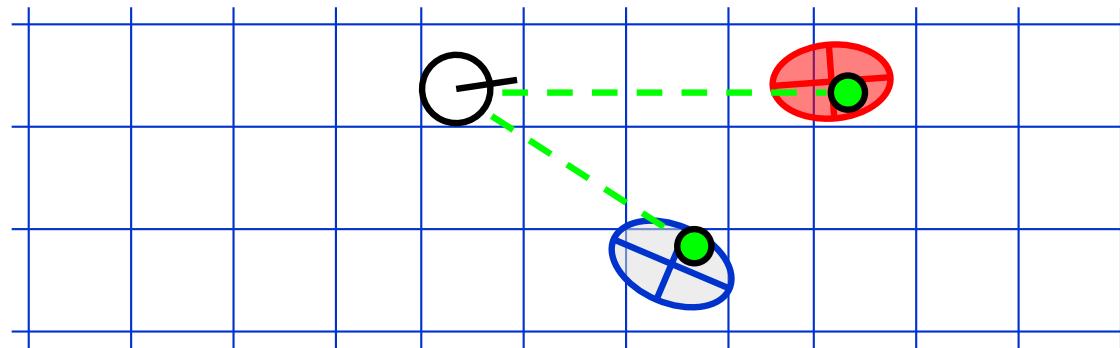


FastSLAM – Sensor Update



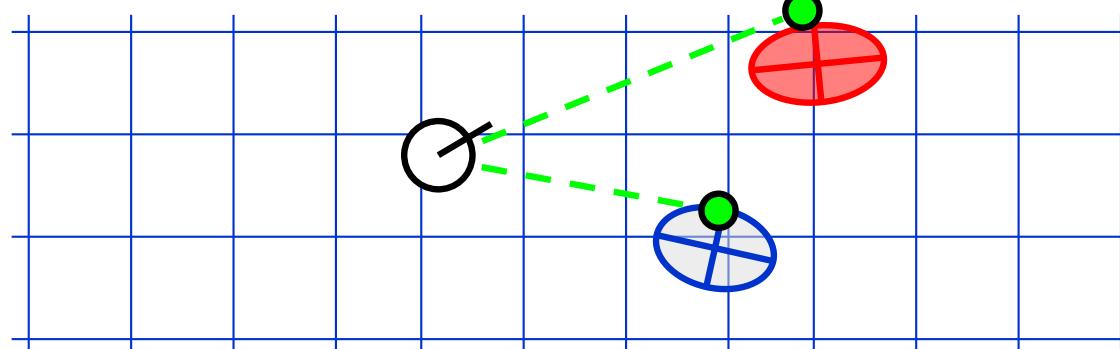
FastSLAM – Sensor Update

Particle #1



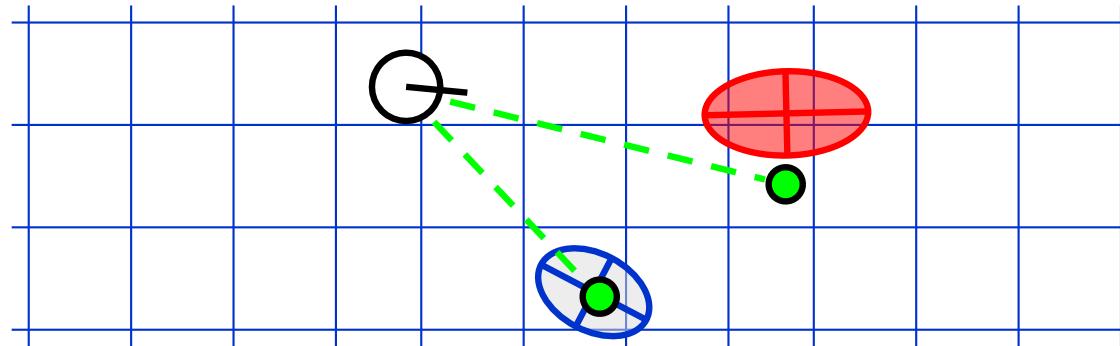
Weight = 0.8

Particle #2



Weight = 0.4

Particle #3



Weight = 0.1

Video Demos

- <http://www.youtube.com/watch?v=U6VFweDGn1o>
- <http://www.youtube.com/watch?v=m3L8OfbTXH0>

CAP6671 Intelligent Systems

Lecture 17:

Multi-robot Coordination

Instructor: Dr. Gita Sukthankar

Email: gitars@eeecs.ucf.edu

Reading

- Reading: N. Kalra et al., Market-based Multirobot Coordination: A survey, CMU-RI Tech Report 2005
- Reading: M. Koes et al., Constraint Optimization Coordination Architecture for Search and Rescue Robotics, in Proceedings of ICRA 2006, Orlando, FL, May 2006, pp. 3977-3982.
- S. Koenig: <http://mapf.info>

Coordination Problems

- Task allocation
 - Which robot does what?
- Plan deconfliction
 - Do separately constructed plans interfere with each other?
- Information fusion
 - Sharing sensor information across robots
- Other issues:
 - Robot heterogeneity
 - Robot death/failure
 - Minimizing communication

NP hard problem (comparable to multi-traveler TSP problem)

Foraging Robots

- Bucket-Brigade Forager

Warehouse Robots

- Kiva Robots
- Commercial warehousing robots
- One of the few successfully commercialized robotics systems

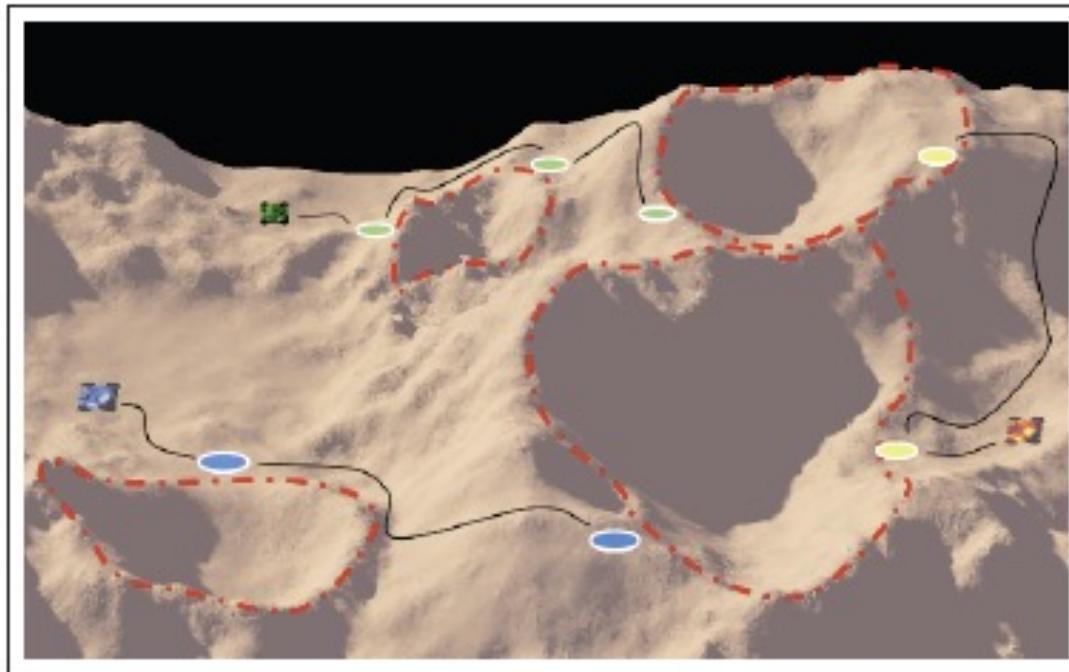
Swarm (Small) Robots

- Swarmanoids
 - <http://www.youtube.com/watch?v=M2nn1X9Xlps>

Future: Smart Matter

- Claytronics catoms
 - <https://www.youtube.com/watch?v=Dc6y4JIGwZc>
 - <https://www.youtube.com/watch?v=HKIAOMz6N-E&feature=youtu.be>
- Kilobots
 - <https://www.youtube.com/watch?v=xK54Bu9HFRw&feature=youtu.be>
- Termes
 - <https://youtu.be/nFjtRONfae4>

Example Task



Robots exploring Mars need to gather data around the highlighted craters.
Which robot goes where?

Problem Characteristics

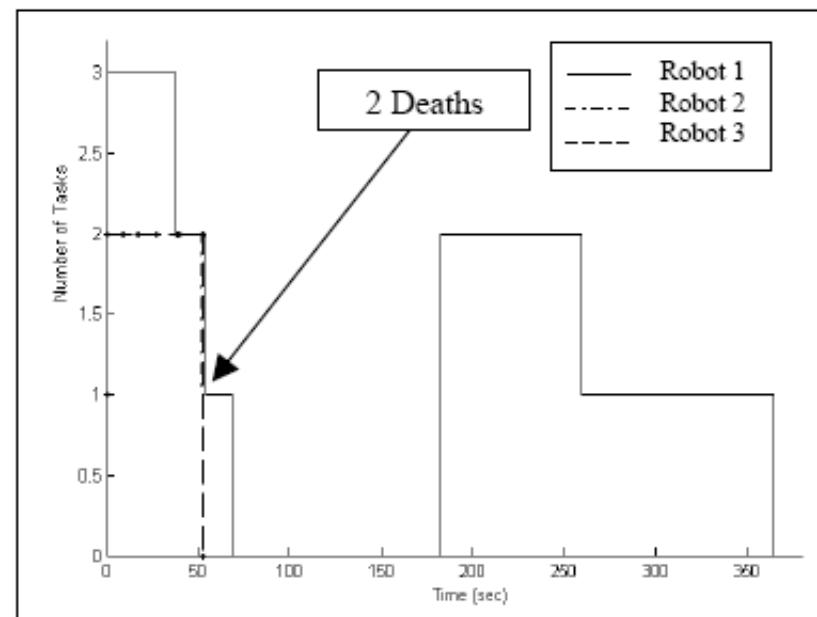
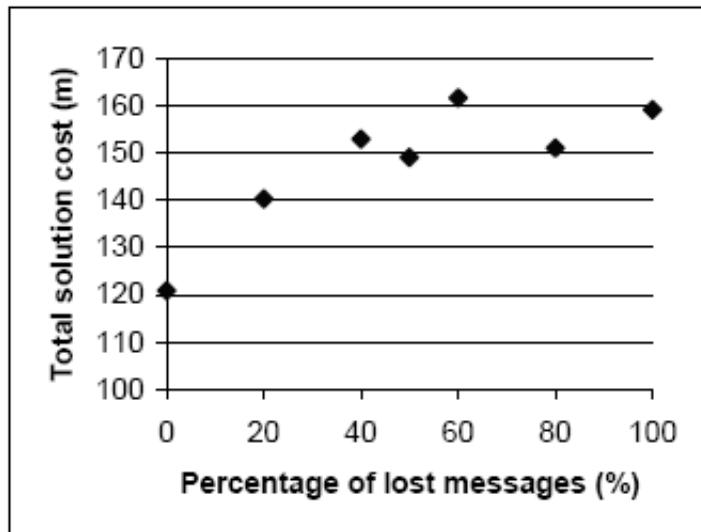
- Team has an objective which can be decomposed into separate tasks.
- Global objective function quantifies designer's preference over possible solutions
- Individual utility/cost functions quantify robot's preferences toward resource usage.
- Resources and tasks are redistributed using some type of task allocation mechanism.
- Auctions are a popular example of a task allocation mechanism.

Market-based Approaches

- Formulate the robot task allocation problem as a group auction
- Robots bid their cost for completing a task
- Central auctioneer injects tasks into the market
- Robots can re-auction tasks to other robots
- Market can be made increasingly robust through:
 - Distributed auctions
 - Monitoring awarded tasks and reauctioning them if not completed within a certain period of time

TraderBots (Dias, Stentz, et al.)

- Demonstrated system robust to
 - simulated message loss
 - partial robot failures
 - robot death
 - intermittent robot function



Parameters

- More robots
- Resources
 - Time
 - Network bandwidth
 - Computer power
 - Sensors
- Tasks
 - Constraints between tasks
 - Dynamically changing tasks

Auction Properties

- Are tasks allocated individually (single-item) or in combinations?
 - Combinatorial auctions: consider task interactions but can be computationally expensive (exponential number of bundles to consider)
 - Multi-item auctions: robots bid on multiple items but are only awarded one
- Price
 - First price: sale price equal to winning bid
 - Vickrey: sale price equal to 2nd highest bid
 - Equivalent in case of honest robots

Taxonomy

- Gerkey and Mataric developed a famous MRTA taxonomy that is often used to describe the properties of the tasks.
- ST: single-task robots
- MT: multi-task robots
- SR: single-robot tasks
- MR: multiple-robot task
- IA: instantaneous, robots don't plan for future
- TA: time-extended assignment
- Most architectures fall into the SR-ST category.

Optimality Results

Approach	Theoretical guarantees	Experimental results
Combinatorial auctions [4, 47]	Optimal (if all bundles are considered) [55]	Good solutions with limited number of task bundles [4, 47, 31, 16]
Central single task iterated auctions [40, 67]	Approximation bounds for 18 cases (3 objective functions, 6 bidding rules) [40]	Close to optimal results when using the appropriate bidding rules [67]
Central instantaneous assignment (IA) [24, 39]	Optimal possible; commonly used greedy algorithm is a 2-approximation; greedy algorithm for online version is 3-competitive [25]	
Peer-to-peer trading [11, 28, 50, 75, 53]	Optimal solution possible in a finite number of trades with a sufficiently expressive set of contract types [54]	In a limited number of rounds, a combination of single- and multi-task trades outperforms all other combinations of single-task, multi-task, swap, and multi-party contracts [1]; allowing non-individual rational trades can lead to better solutions [69]
Central multi-task auctions followed by peer-to-peer trading [14]		Increasing the maximum number of tasks awarded per multi-task auction results in poorer solution quality [14]

Time/Communication

Time complexity

Auction type	Bid valuation	Winner determination	Number of auctions
Single-item	v	$O(r)$	n
Multi-item (greedy)	$O(n \cdot v)$	$O(n \cdot r \cdot m)$	$\lceil n/m \rceil$
Multi-item (optimal)	$O(n \cdot v)$	$O(r \cdot n^2)$ [25]	$\lceil n/m \rceil$
Combinatorial	$O(2^n \cdot V)$	$O((b+n)^n)$ [55]	1

Communication complexity

Auction type	Auction call	Bid submission	Award	Award (+ losers)
Single-item	$O(r)$	$O(r)$	$O(1)$	$O(r)$
Multi-item	$O(r \cdot n)$	$O(r \cdot n)$	$O(m)$	$O(r)$
Combinatorial	$O(r \cdot n)$	$O(r \cdot 2^n)$	$O(n)$	$O(r + n)$

COCOA

- Constraint Optimization Coordination Architecture
- Capable of reasoning about joint goals
- Outperforms token-based coordination and market-based architecture under certain conditions
- Problem formulation is similar to deliberate team planning framework but solution method uses constraint optimization

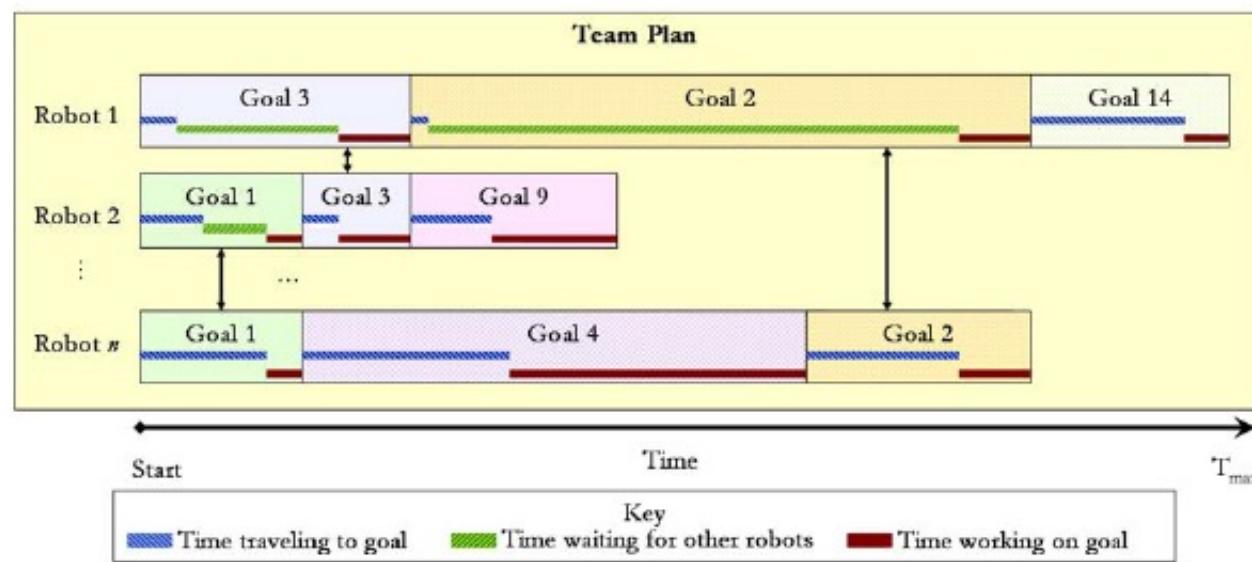
MILP

- Stands for **Mixed-integer Linear Programming**
- Linear programming
 - Optimizing a linear objective function subject to linear equality and inequality constraints
 - Used to pose questions like “how can I maximize my utility subject to certain constraints”
 - Simplex method
- Mixed-integer linear programming
 - Certain variables are required to have integer values (e.g. number of robots)
 - Solved by relaxing constraint that variables be integer

Constraints

- Goal constraints:
 - Allen's 13 temporal relationships (before, equal, meets)
- Robot constraints
 - ParticipateIn (robot must be part of mission)
 - EndAt (robot must end at a certain location)
- Resource constraints
 - LimitFuelTo
 - LimitResourceTo
- Constraints can be combined using 1st order logic and converted to numeric constraints using binary constraint variables

Example Plan



Constraint Formulation

GOAL CONSTRAINTS	
$G^x \text{before } G^y$	$G_x.\text{start} + d^x G_x < G_y.\text{start}^*$ $G_x.\text{start}^* \leq G_y.\text{start}^*$
$G^x \text{ equal } G^y$	$G_x.\text{start} = G_y.\text{start}$ $G_x.d^x = G_y.d^y$
$G^x \text{meets } G^y$	$G_x.\text{start} + d^x G_x = G_y.\text{start}$
$G^x \text{overlaps } G^y$	$G_x.\text{start} < G_y.\text{start}^*$ $G_x.\text{start} + d^x G_x < G_y.\text{start}^* + d^y G_y$ $G_x.\text{start}^* + d^x > G_y.\text{start}$ $G_x = G_y$
$G^x \text{during } G^y$	$G_x.\text{start}^* > G_y.\text{start}$ $G_x.\text{start} + d^x G_x < G_y.\text{start}^* + d^y G_y$ $G_x = G_y$
$G^x \text{starts } G^y$	$G_x.\text{start} = G_y.\text{start}$ $G_x.\text{start} + d^x G_x < G_y.\text{start}^* + d^y G_y$
$G^x \text{finishes } G^y$	$G_x.\text{start} + d^x G_x = G_y.\text{start} + d^y G_y$ $G_x.\text{start}^* > G_y.\text{start}$ $G_x = G_y$
$\text{do } G^x$	$G_x = 1$

ROBOT CONSTRAINTS	
$R^n \text{ participantIn } G^m$	$R_n G_m = G_m$
$R^n \text{ endAt } L$	We create a new goal G_{end_L} $R_n G_{end_i} = 1$

RESOURCE CONSTRAINTS	
$\mathcal{R}_s \text{ limitFuelTo } F$	$\sum_{n \text{ s.t. } R^n \in \mathcal{R}_s} \sum_{i < \omega} R_n O_i.\text{travel} \leq F$
$\mathcal{R}_s \text{ limitResourcesOf } (f(R^n, G^m), \rho)$	$\sum_{R^n \in \mathcal{R}_s} \sum_{G^m \in \mathcal{G}} f(R^n, G^m) R_n G_m \leq \rho$

Making MILP Real-time

- Generate solution using heuristic and use it as a starting point for MILP solver
- Use solutions for subsets of goals to build up to full solution
- Use solutions for shorter time-horizons to build up to complete time horizon

Heuristics

- Myopic
 - Robots recursively schedule goals over an increasing planning horizon
 - Problem size grows exponentially with planning horizon so subproblem is much smaller than original problem
- Greedy goal
 - Similar to market-based task allocation
 - Sort goals in decreasing order of reward
 - Robots bid cost
 - Lowest bidder assigned to goal capability
 - No goal reauctioning

Evaluation

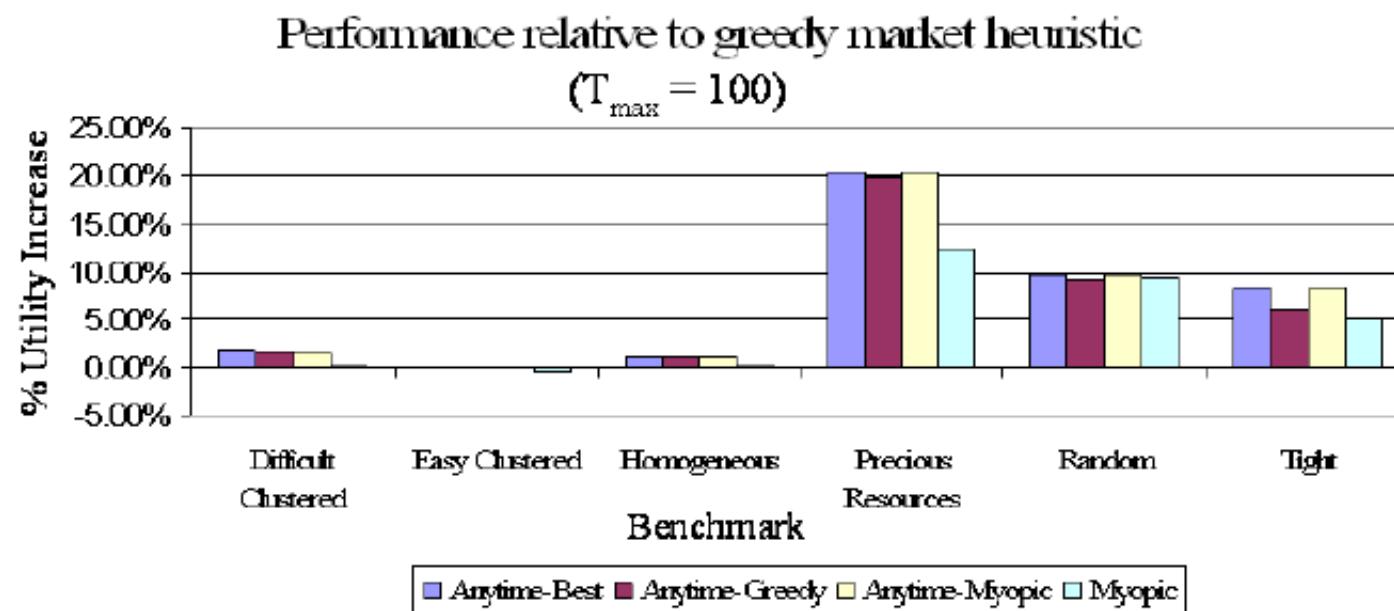
Benchmark Problems

Benchmark	Robot Capabilities	Goal Requirements	Goal Location
Homogeneous	All	All	Random
Tight	Single	All	Random
Easy Clustered	All	All	Clustered
Difficult Clustered	Single	All	Clustered
Precious Resources	Super and weak robots	Easy and hard goals	Random
Random	Random	Random	Random

- Compared myopic heuristic, greedy heuristic, anytime algorithm with myopic, anytime with greedy, anytime with best

Results

- Showed significant performance improvements over the greedy algorithm (which is comparable to a market-based strategy)

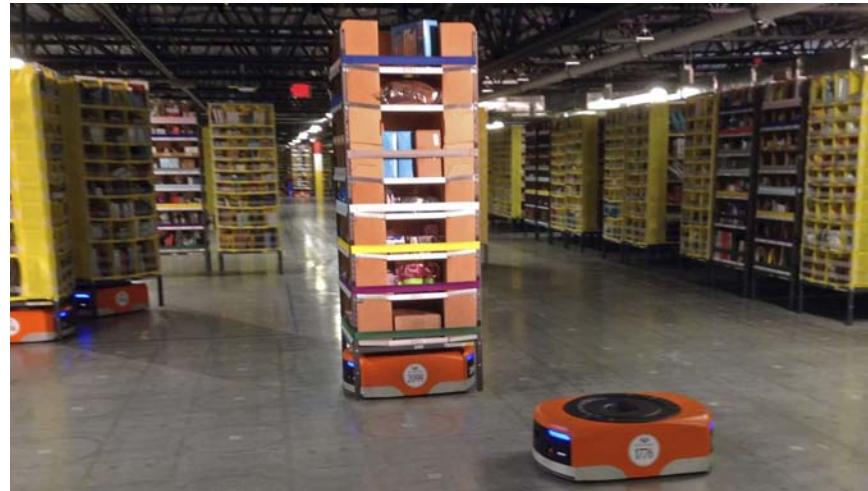


Multi-Agent Path Finding

- Application: Amazon fulfillment centers
- 2003 Kiva Systems founded
- 2012 Amazon acquires Kiva Systems for \$775 million
- 2015 Kiva Systems becomes Amazon Robotics



[www.npr.org – Getty Images]

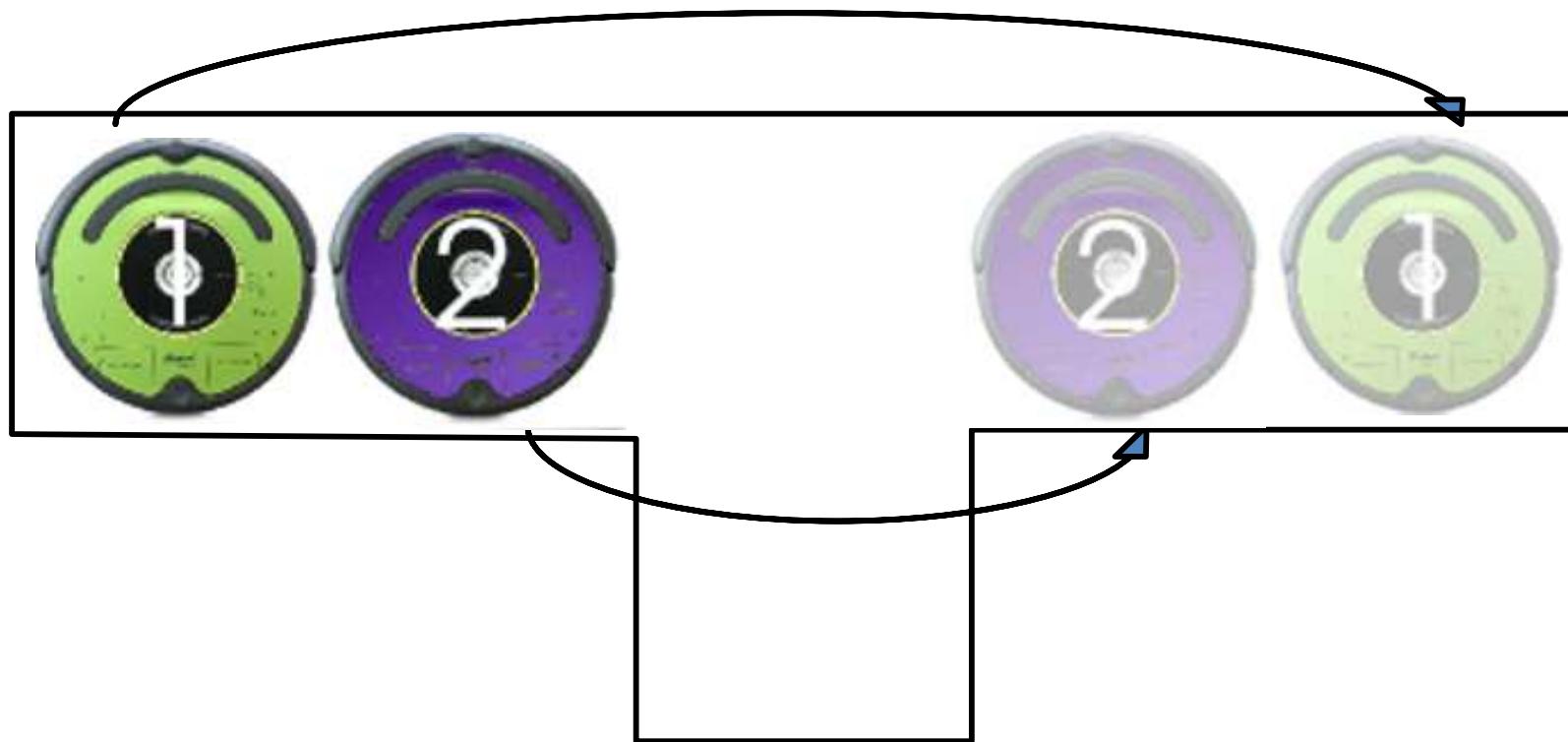


[www.theguardian.com - AP]

- > 3,000 robots on > 110,000 square meters in Tracy, California

(slides courtesy of S. Koenig <http://mapf.info>)

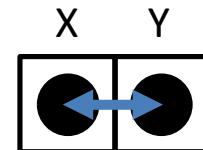
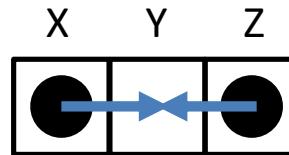
Multi-Agent Path Finding



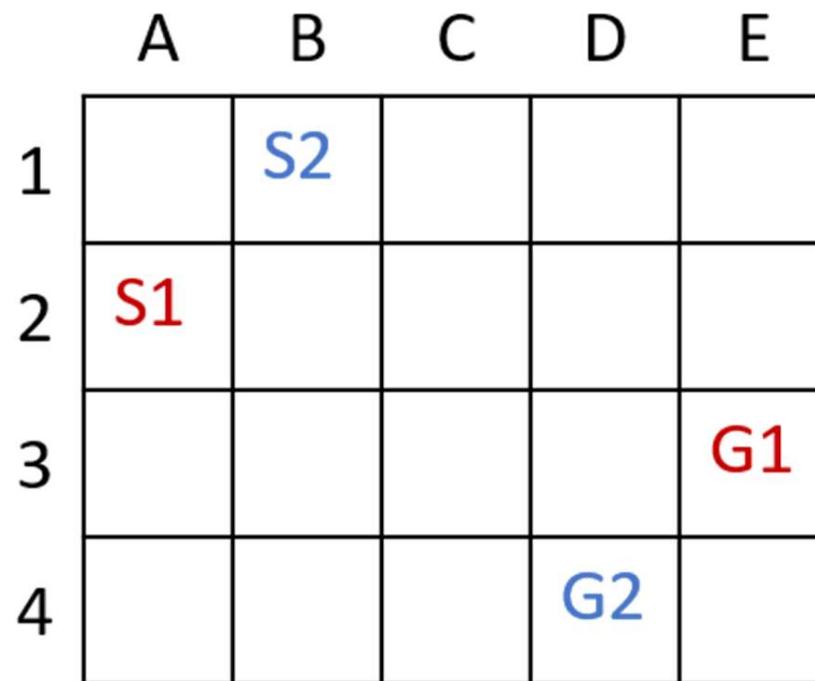
Individually planned paths will collide with one another!

MAPF

- Each agent can move N, E, S or W into any adjacent unblocked cell (provided an agent already in that cell leaves it while the agent moves into it or earlier) or wait in its current cell
- Not allowed (“vertex collision”)
 - Agent 1 moves from X to Y
 - Agent 2 moves from Z to Y
- Not allowed (“edge collision”)
 - Agent 1 moves from X to Y
 - Agent 2 moves from Y to X



MAPF Example

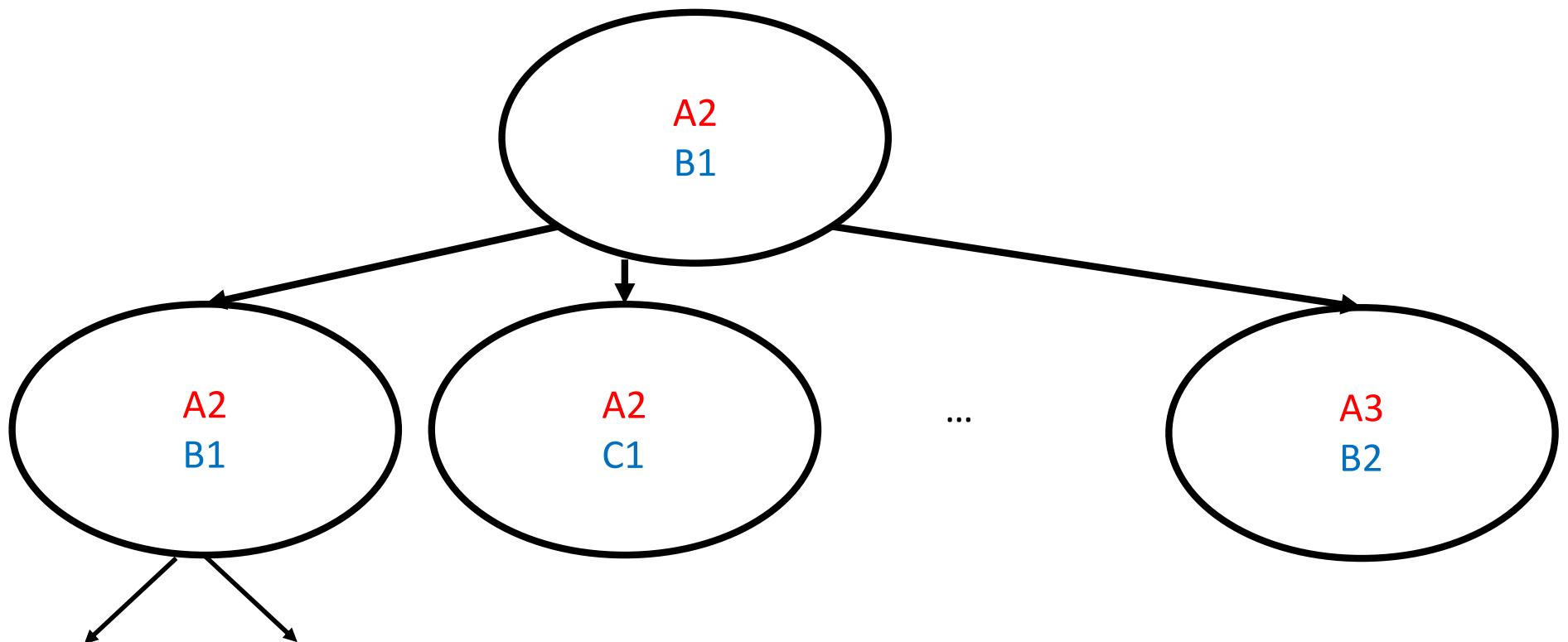


S1 (S2) = start cell of the **red (blue)** agent
G1 (G2) = goal cell of the **red (blue)** agent

A*-Based Search

	A	B	C	D	E
1		S2			
2	S1				
3				G1	
4			G2		

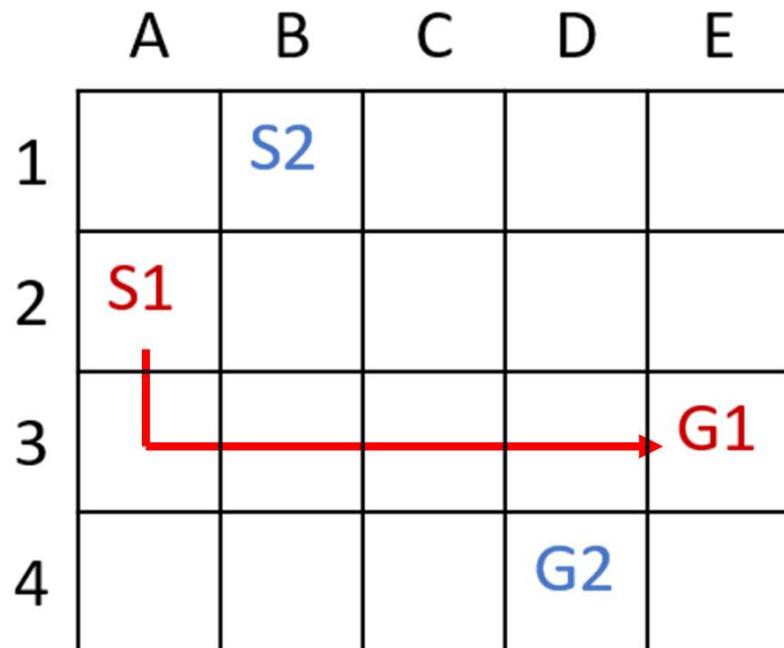
- A*-based search in the joint cell space: Optimal (or bounded-suboptimal) but extremely inefficient MAPF solver



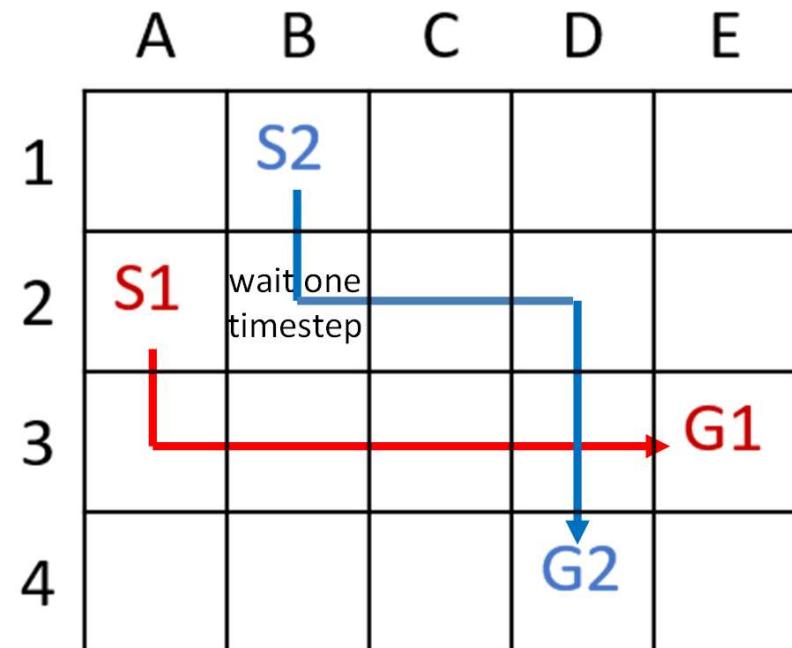
A	B	C	D	E
1	S2			
2	S1			
3				G1
4			G2	

Priority-Based Search

- Priority-based (= sequential) search (plan for one agent after another in space (= cell)-time space in a given order): efficient but suboptimal (and even incomplete) MAPF solver



First, find a time-minimal path for the agent with priority 1.



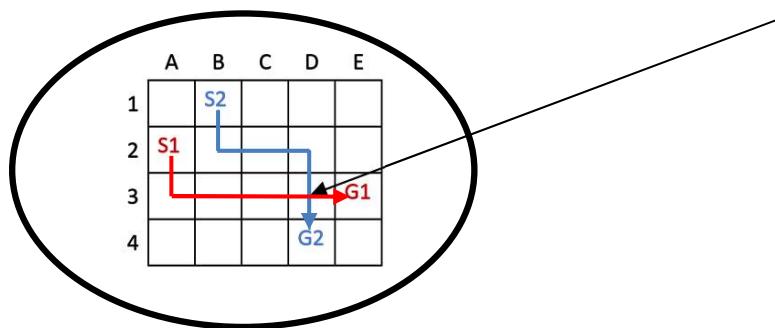
Then, find a time-minimal path for the agent with priority 2 that does not collide with the paths of higher-priority agents.

Conflict-Based Search

	A	B	C	D	E
1		S2			
2	S1				
3				G1	
4				G2	

- Conflict-based search [Sharon, Stern, Felner and Sturtevant]: Optimal (or bounded-suboptimal) MAPF solver that plans for each agent independently, if possible

Find time-minimal paths for all agents independently

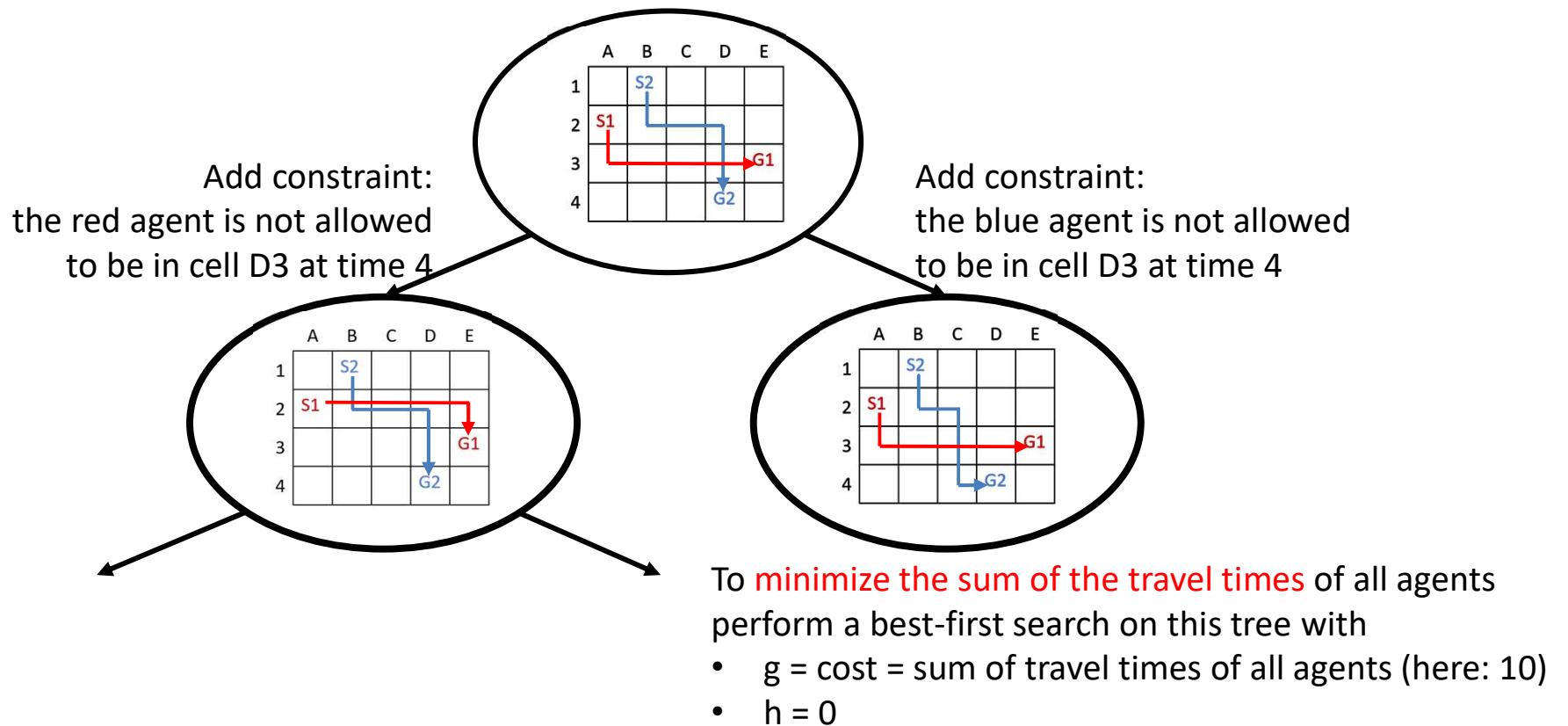


Conflict (here: vertex collision)

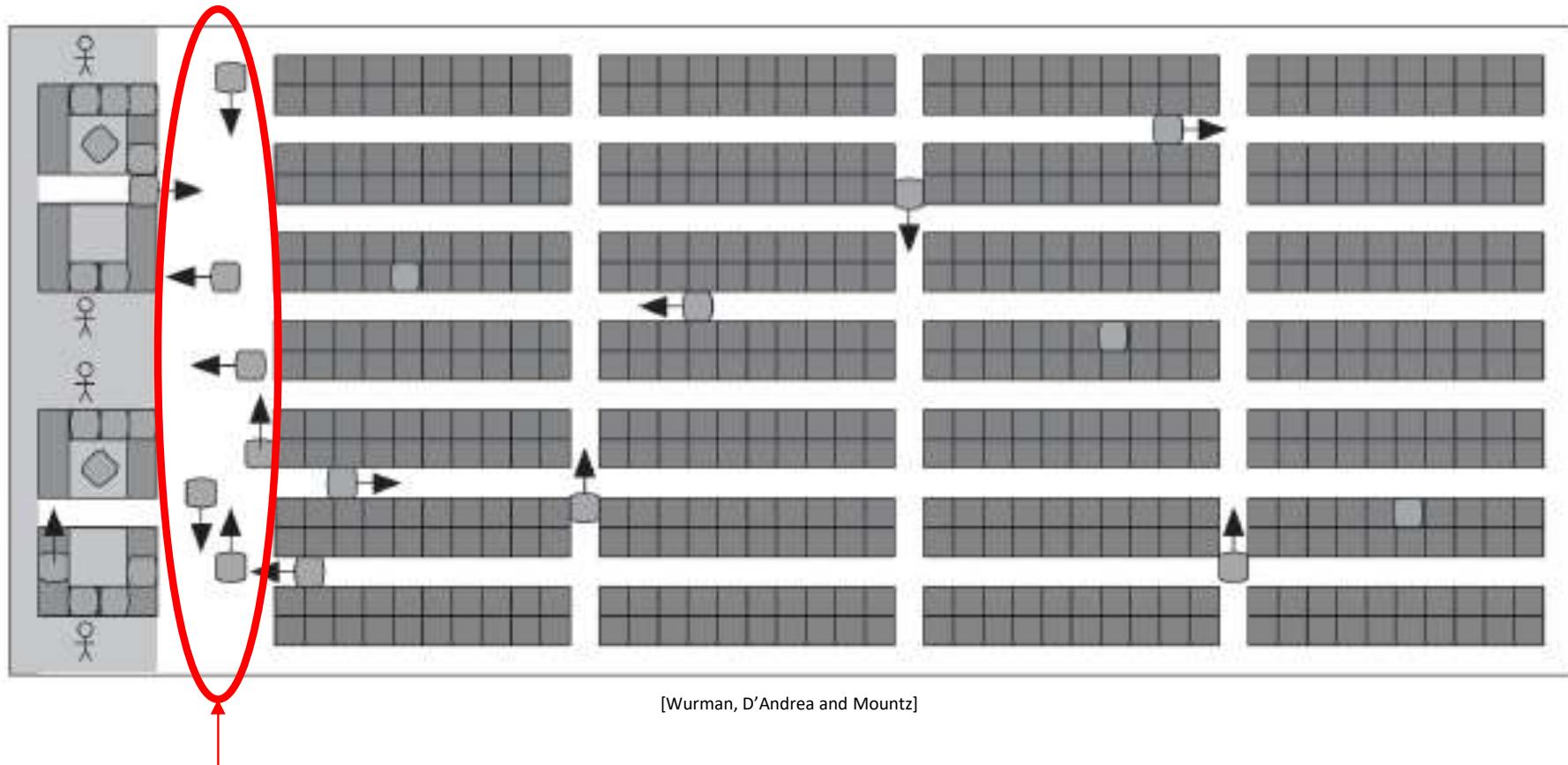
Conflict-Based Search

	A	B	C	D	E
1		S2			
2	S1				
3					G1
4				G2	

- Conflict-based search [Sharon, Stern, Felner and Sturtevant]: Optimal (or bounded-suboptimal) MAPF solver that plans for each agent independently, if possible



Execution of MAPF Plans



Use the MAPF methods here (in a small area of high congestion but with few agents) rather than over the whole fulfillment center

Conclusion

- Many options for multi-robot coordination
- Type of task makes a great difference in the performance of the coordination mechanism