Working Together

Working Together

- Why and how to agents work together?
- Since agents are autonomous, they have to make decisions at run-time, and be capable of dynamic coordination.
- Overall they will need to be able to share:
  - Tasks
  - Information
- If agents are designed by different individuals, they may not have common goals.

Benevolent Agents

- If we “own” the whole system, we can design agents to help each other whenever asked
- In this case, we can assume agents are *benevolent*: our best interest is their best interest
- Problem-solving in benevolent systems is *cooperative distributed problem solving* (CDPS)
- *Benevolence simplifies the system design task enormously!*

Self-Interested Agents

- If agents represent individuals or organizations, (the more general case), then we cannot make the benevolence assumption
- Agents will be assumed to act to further their own interests, possibly at expense of others
- Potential for *conflict*
- May complicate the design task enormously

Coherence and coordination

- Criteria for assessing an agent-based system.
- *Coherence*

how well the [multiagent] system behaves as a unit along some dimension of evaluation (Bond and Gasser).

- We can measure coherence in terms of solution quality, how effiently resources are used, conceptual clarity and so on.
- *Coordination*

the degree. . . to which [the agents]. . . can avoid “extraneous” activity [such as] . . . synchronizing and aligning their activities (Bond and Gasser).

If the system is perfefctly coordinated, agents will not get in each others’ way, in a physical or a metaphorical sense.

Cooperative Distributed Problem Solving (CDPS)

- Neither global control nor global data storage — no agent has sufficient information to solve entire problem
- Control and data are distributed

CDPS System Characteristics and Consequences

- Communication is slower than computation
  - loose coupling
  - efficient protocol
  - modular problems
  - problems with large grain size

More CDPS System Characteristics and Consequences

- Any unique node is a potential bottleneck
  - distribute data
  - distribute control
  - organized behavior is hard to guarantee (since no one node has complete picture)

Four Phases to Solution

1. Problem Decomposition
2. Sub-problem distribution
3. Sub-problem solution
4. Answer synthesis

The contract net protocol deals with phase 2.

Problem decomposition

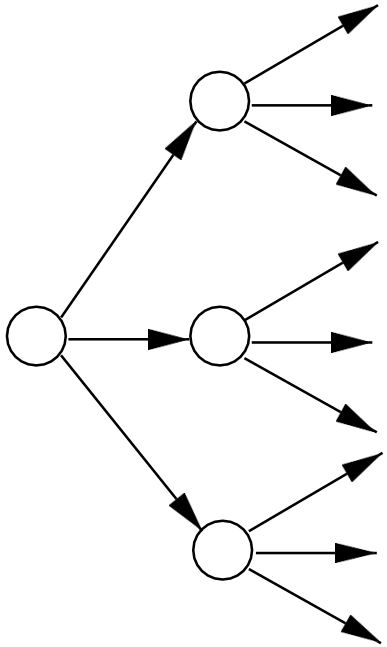
- The overall problem to be solved is divided into smaller sub-problems.
  - This is typically a recursive/hierarchical process.
    - Subproblems get divided up also.
    - In ACTORS, this is done until we are at the level of individual program instructions.
- Clearly there is some processing to do the division. How this is done is one design choice.
- Another choice is *who* does the division.
  - Is it centralized?
  - Which agents have knowledge of task structure?
  - Who is going to solve the sub-problems?

Sub-problem solution

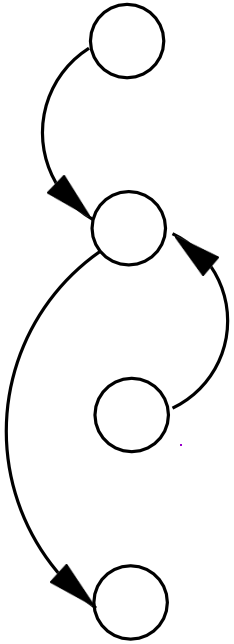
- The sub-problems derived in the previous stage are solved.
- Agents typically share some information during this process.
- A given step may involve two agents synchronizing their actions.

Solution synthesis

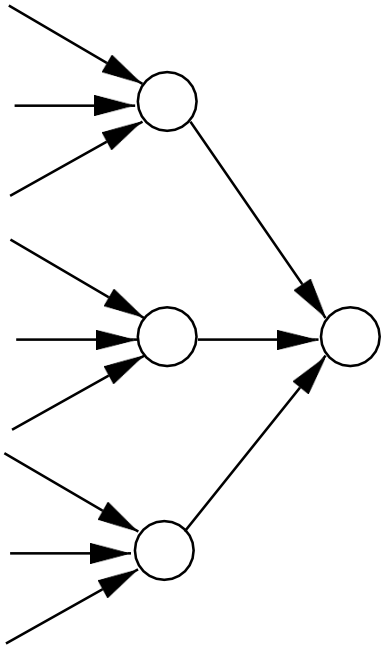
- In this stage solutions to sub-problems are integrated.
- Again this may be hierarchical
  - Different solutions at different levels of abstraction.



decomposition



solution



synthesis

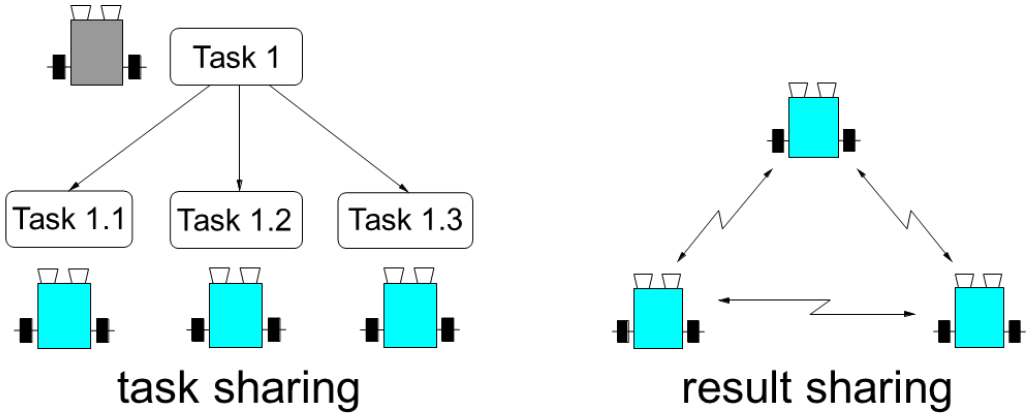
Task Sharing and Result Sharing

- Given this model of cooperative problem solving, we

have two activities that are likely to be present:

–*task sharing*:
components of a task are distributed to component agents;
how do we decide how to allocate tasks to agents?

–*result sharing*:
information (partial results etc) is distributed.
how do we assemble a complete solution from the parts?



The Contract Net

- A well known task-sharing protocol for *task allocation* is the *contract net*:
  1. Recognition
  2. Announcement
  3. Bidding
  4. Awarding
  5. Expediting

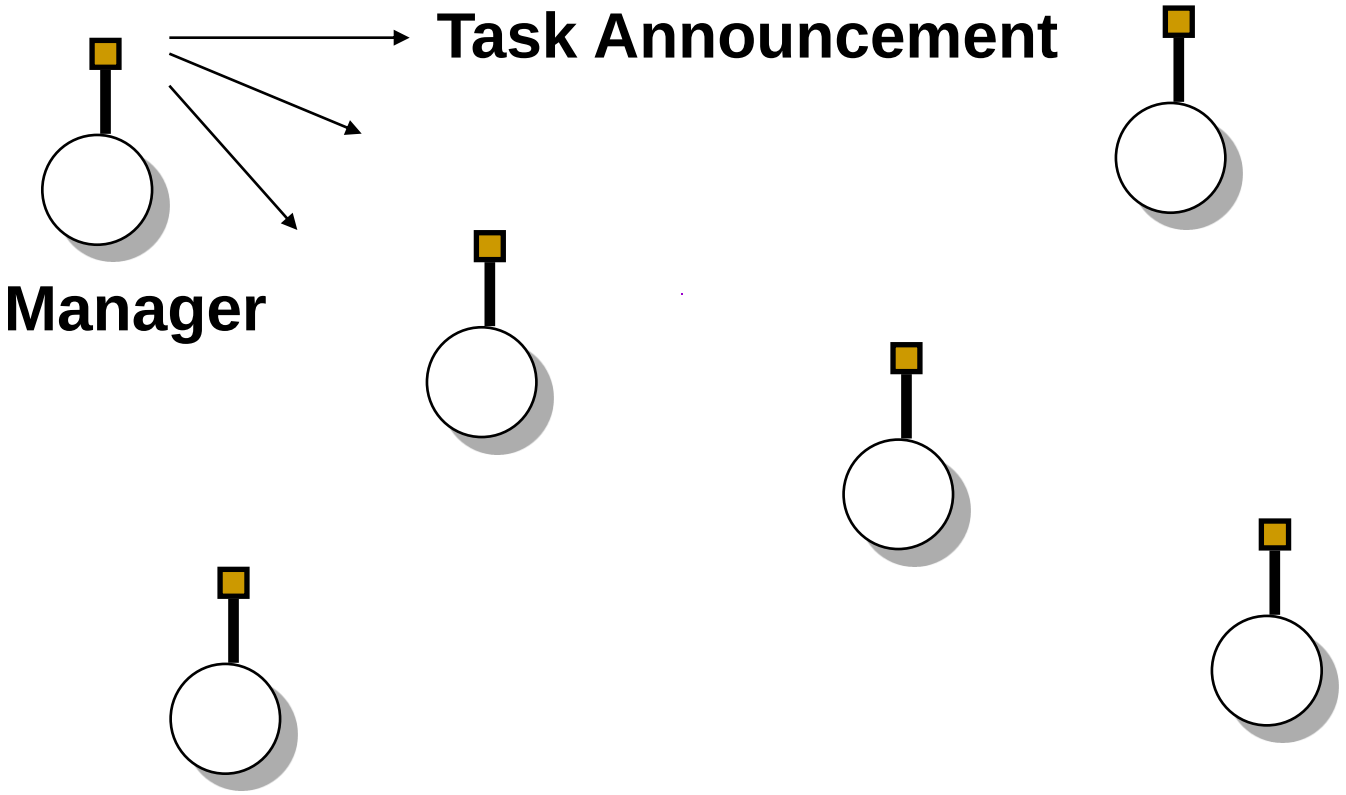
The Contract Net

- An approach to *distributed problem solving*, focusing on task distribution
- Task distribution viewed as a kind of contract negotiation
- "Protocol" specifies *content* of communication, not just form
- Two-way transfer of information is natural extension of transfer of control mechanisms

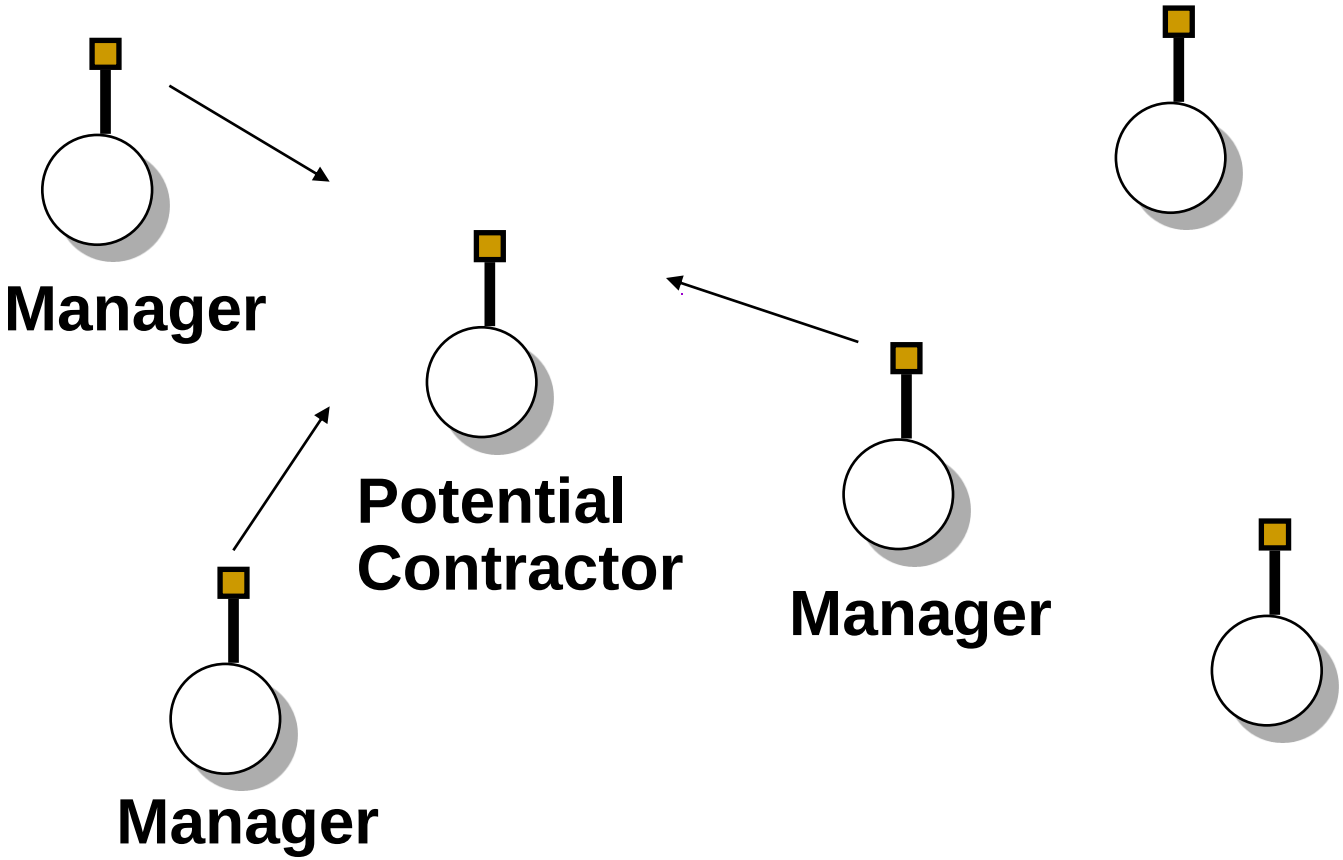
Contract Net

- The collection of nodes is the “contract net”
- Each node on the network can, at different times or for different tasks, be a manager or a contractor
- When a node gets a composite task (or for any reason can’t solve its present task), it breaks it into subtasks (if possible) and announces them (acting as a manager), receives bids from potential contractors, then awards the job (example domain: network resource management, printers, …)

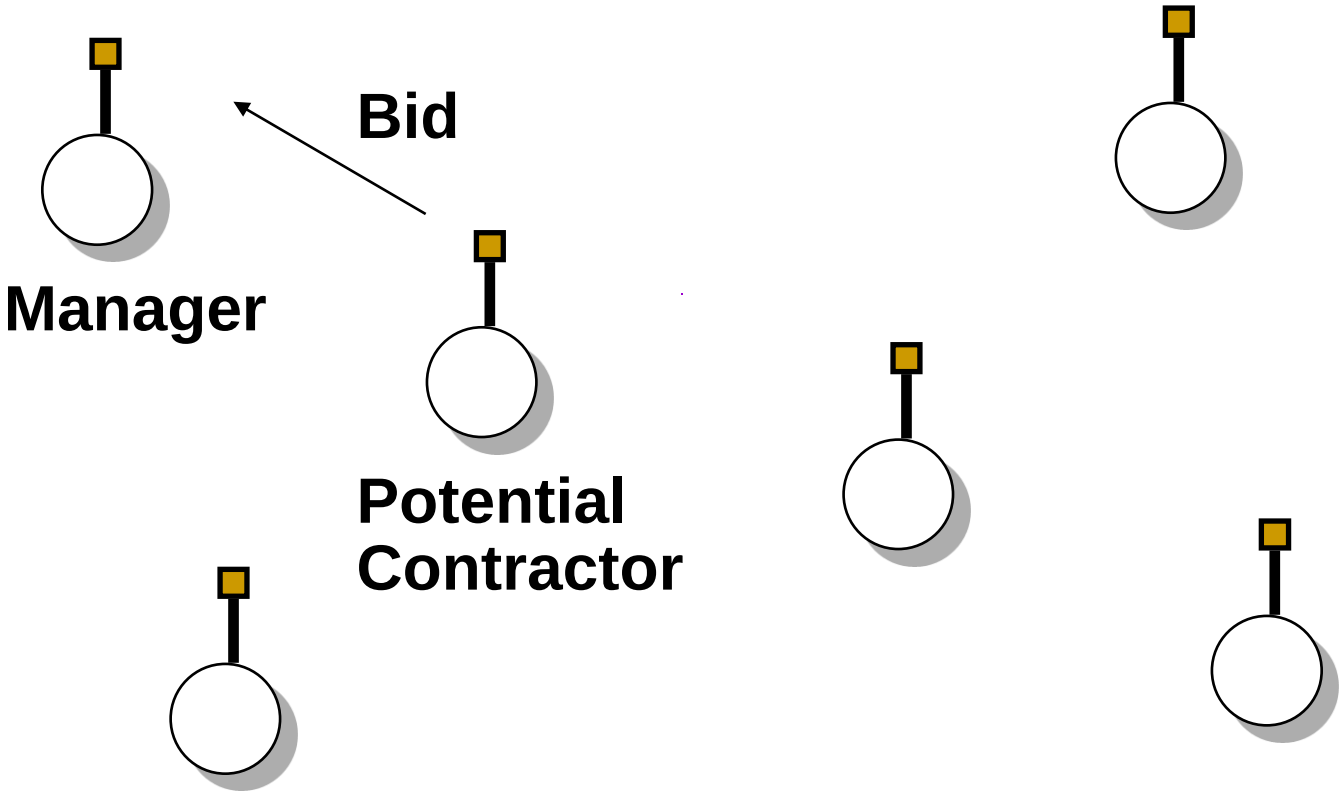
Node Issues Task Announcement



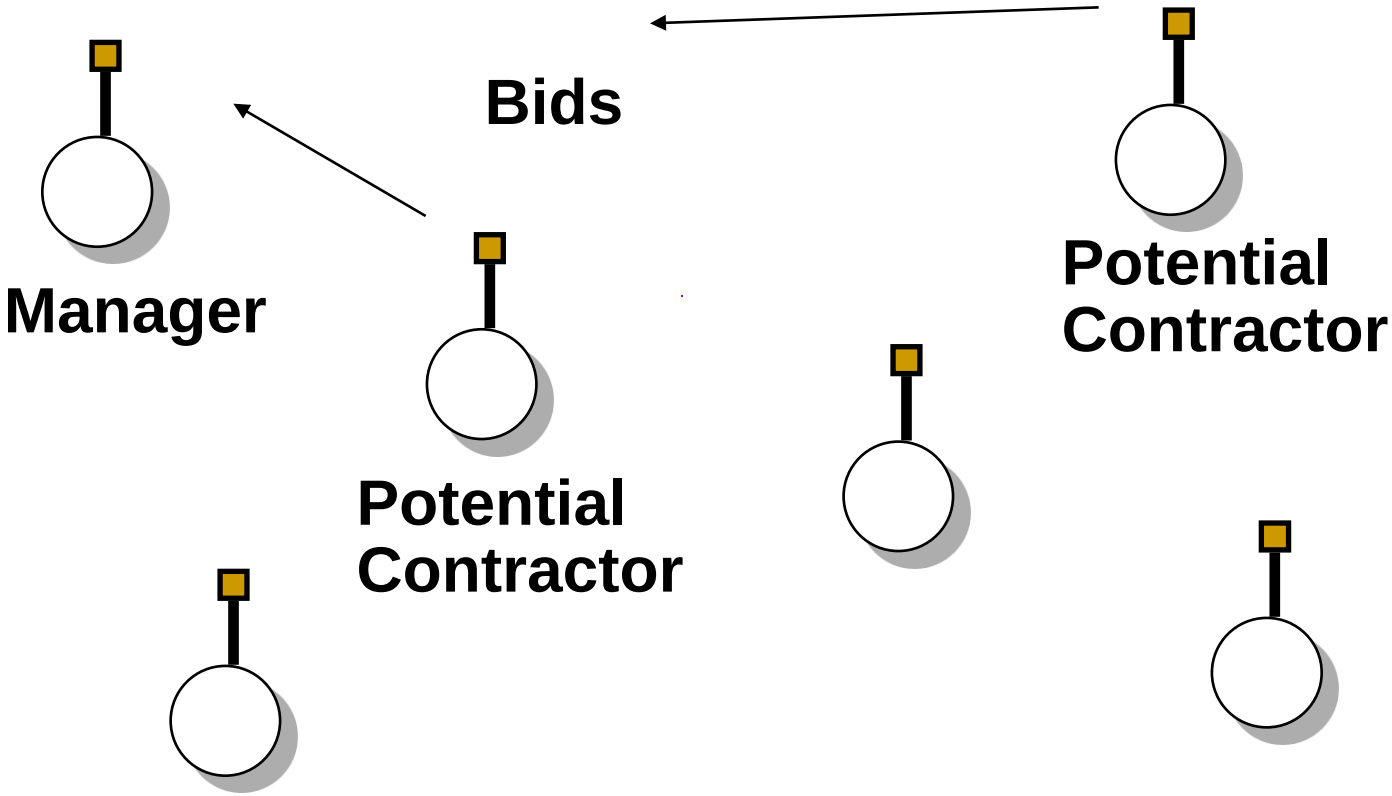
Idle Node Listening to Task Announcements



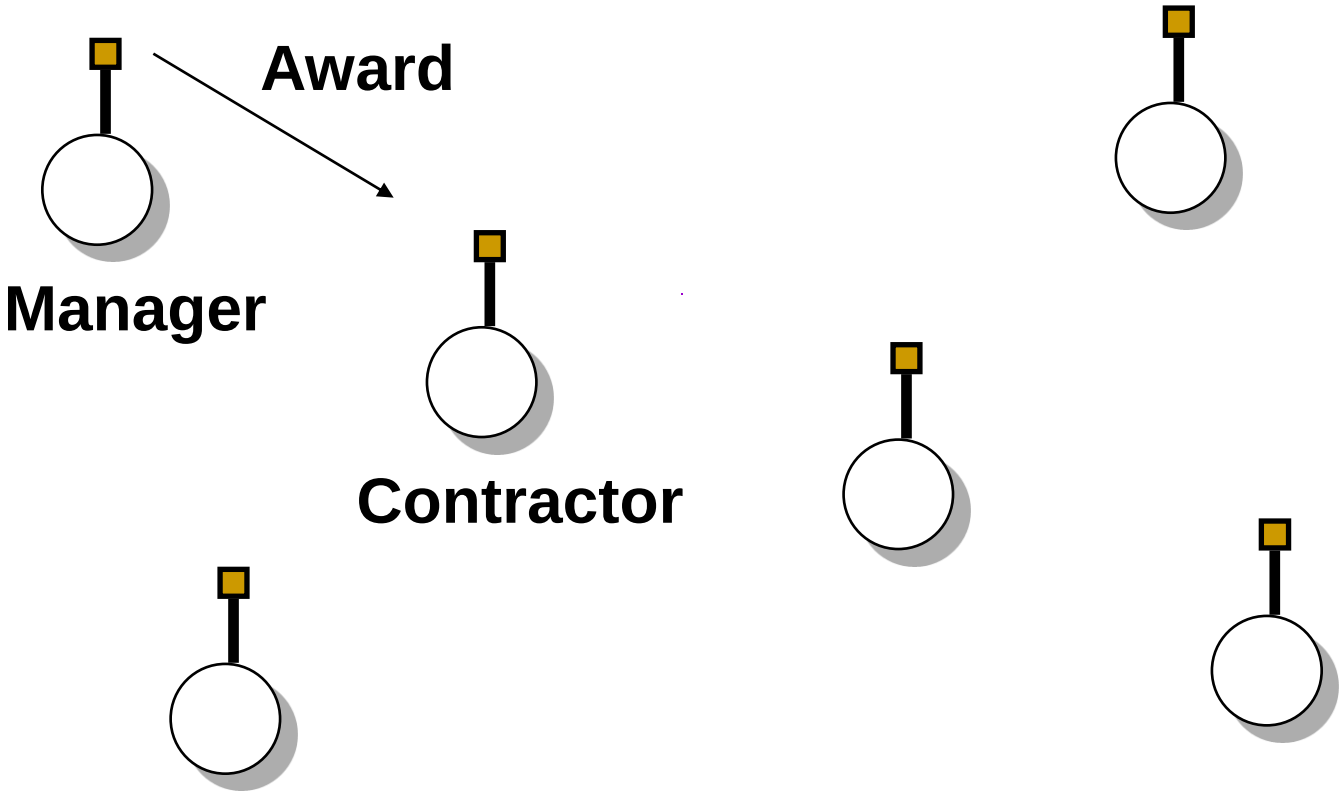
Node Submitting a Bid



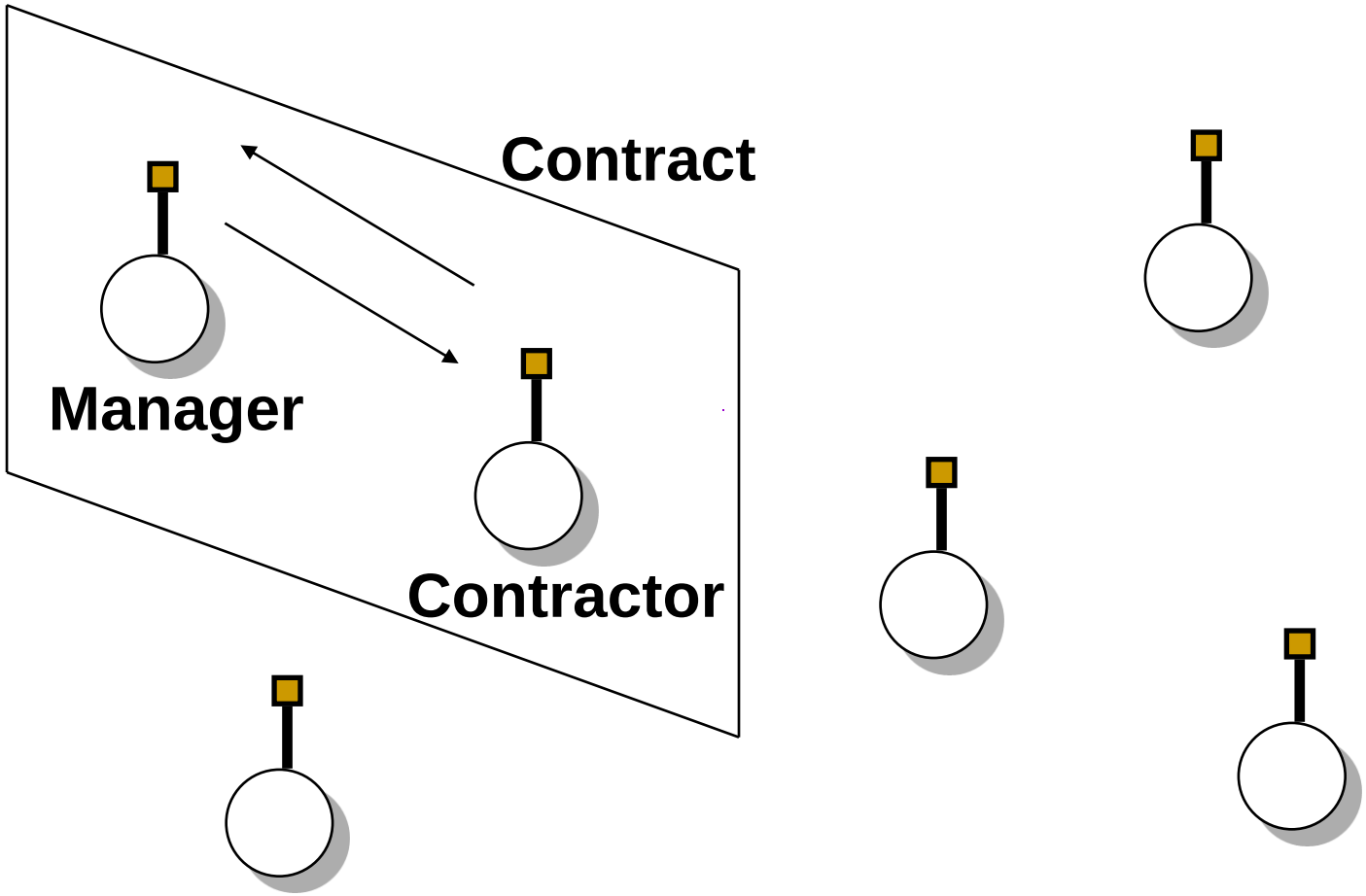
Manager listening to bids



Manager Making an Award



Contract Established



Recognition

- In this stage, an agent recognizes it has a problem it wants help with.
  Agent has a goal, and either…
  - realizes it cannot achieve the goal in isolation — does not have capability
  - realizes it would prefer not to achieve the goal in isolation (typically because of solution quality, deadline, etc.)

Announcement

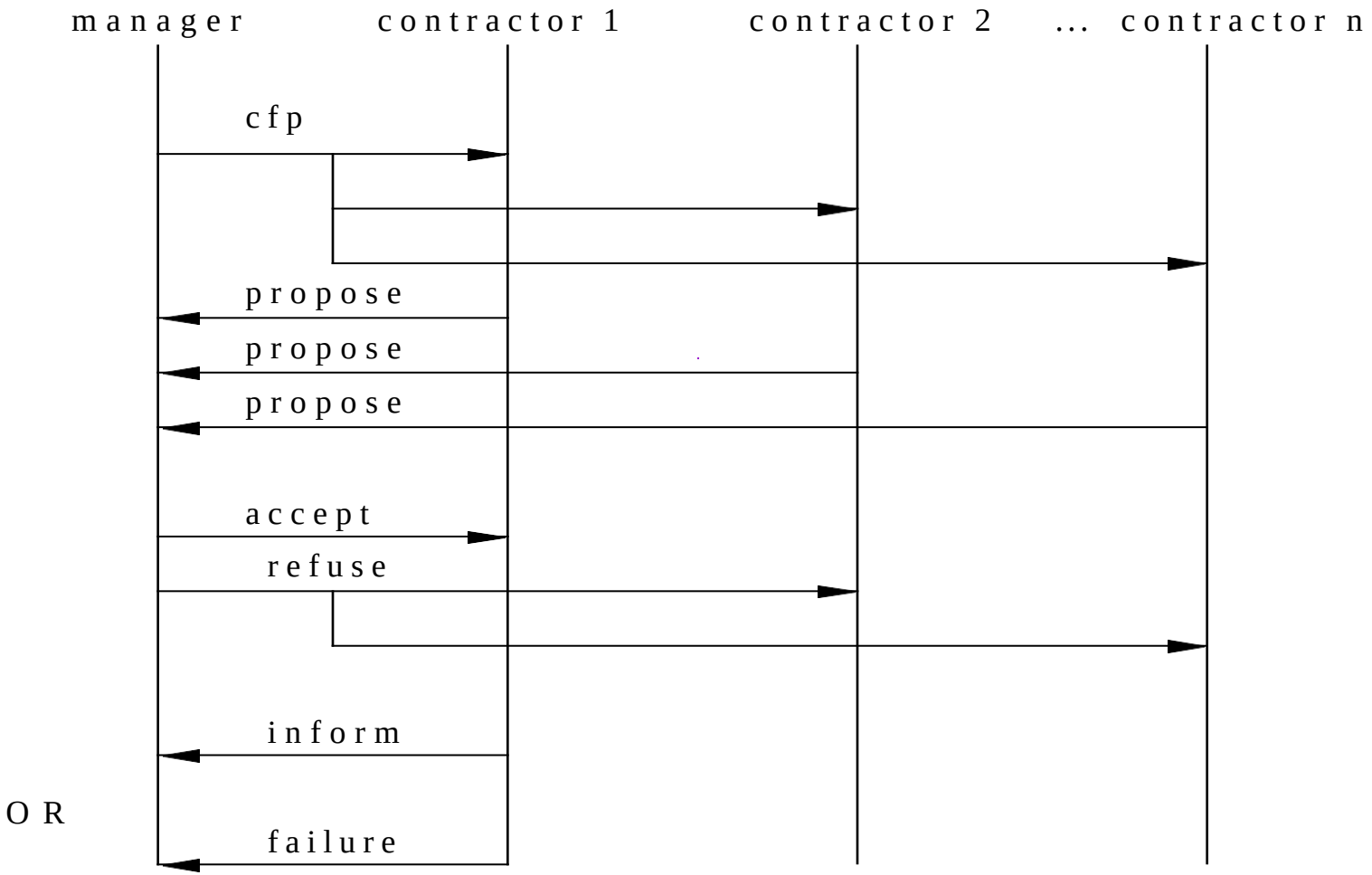
- In this stage, the agent with the task sends out an *announcement* of the task which includes a *specification* of the task to be achieved
- Specification must encode:
  - description of task itself (maybe executable)
  - any constraints (e.g., deadlines, quality constraints)
  - meta-task information (e.g., “bids must be submitted by…”)
- The announcement is then *broadcast*

Bidding

- Agents that receive the announcement decide for themselves whether they wish to *bid* for the task
- Factors:
  - agent must decide whether it is capable of expediting task
  - agent must determine quality constraints & price information (if relevant)
- If they do choose to bid, then they submit a *tender*

Awarding & Expediting

- Agent that sent task announcement must choose between bids & decide who to “award the contract” to
- The result of this process is communicated to agents that submitted a bid
- The successful *contractor* then expedites the task
- May involve generating further manager-contractor relationships: *sub-contracting*


manager
contractor 1
contractor 2
... contractor n
cfp
propose
propose
propose
accept
refuse
inform
O R
failure

Issues for Implementing Contract Net

- How to…
  - …specify *tasks*?
  - …specify *quality of service*?
  - …select between competing offers?
  - …differentiate between offers based on multiple criteria?

Deciding how to bid

- At time t a contractor i is scheduled to carry out τ_i^t.
- Contractor i also has resources e_i.
- Then i receives an annoucement of task specification ts, which is for a set of tasks $\tau(ts)$.
- These will cost i $c_i^t(\tau)$ to carry out.
- The *marginal cost* of carrying out τ will be:

$$\mu_i(\tau(ts) \mid \tau_i^t) = c_i(\tau(ts) \cup \tau_i^t) - c_i(\tau_i^t)$$

that is the difference between carrying out what it has already agreed to do and what it has already agreed plus the new tasks.

Deciding how to bid

- Due to synergies, this is often not just:

$$c_i(\tau(ts))$$

in fact, it can be zero — the additional tasks can be done for free.

- Think of the cost of giving another person a ride to work.

- As long as $\mu_i(\tau(ts) \mid \tau_i^t) < e$ then the agent can afford to do the new work, then it is rational for the agent to bid for the work.

  - Otherwise not.

Domain-Specific Evaluation

- Task announcement message prompts potential contractors to use domain specific task evaluation procedures; there is deliberation going on, not just selection — perhaps no tasks are suitable at present
- Manager considers submitted bids using domain specific bid evaluation procedure

Types of Messages

- Task announcement
- Bid
- Award
- Interim report (on progress)
- Final report (including result description)
- Termination message (if manager wants to terminate contract)

Efficiency Modifications

- Focused addressing — when general broadcast isn't required
- Directed contracts — when manager already knows which node is appropriate
- Request-response mechanism — for simple transfer of information without overhead of contracting
- Node-available message — reverses initiative of negotiation process

Message Format

- Task Announcement Slots:
  - Eligibility specification
  - Task abstraction
  - Bid specification
  - Expiration time
- The existence of a common internode language allows new nodes to be added to the system modularly, without the need for explicit linking to others in the network (e.g., as needed in standard procedure calling) or object awareness (as in OOP)

Task Announcement Example

(common internode language)

To: *
From: 25
Type: Task Announcement
Contract: 43–6
Eligibility Specification: Must-Have FFTBOX
Task Abstraction:
Task Type Fourier Transform
Number-Points 1024
Node Name 25
Position LAT 64N LONG 10W
Bid Specification: Completion-Time
Expiration Time: 29 1645Z NOV 1980

Result Sharing

- In results sharing, agents provide each other with information as they work towards a solution.
- It is generally accepted that results sharing improves problem solving by:
  - –Independent pieces of a solution can be cross-checked. - Confidence
  - –Combining local views can achieve a better overall view. - Completeness
  - –Shared results can improve the accuracy of results. - Precision
  - –Sharing results allows the use of parallel resources on a problem. - Timeliness

Result Sharing in Blackboard Systems

- The first scheme for cooperative problem solving: the *blackboard system*
- Results shared via shared data structure (BB)
- Multiple agents (KSs/KAs) can read and write to BB
- Agents write partial solutions to BB
- BB may be structured into hierarchy
- Mutual exclusion over BB required $\Rightarrow$ bottleneck
- Not concurrent activity
- Compare: LINDA tuple spaces, JAVASPACES

Result Sharing in Subscribe/Notify Pattern

- Common design pattern in OO systems: *subscribe/notify*
- An object *subscribes* to another object, saying "tell me when event e happens"
- When event e happens, original object is notified
- Information pro-actively *shared* between objects
- Objects required to know about the *interests* of other objects $\Rightarrow$ inform objects when relevant information arises

Handling inconsistency

- A group of agents may have inconsistencies in their:
  - Beliefs
  - Goals or intentions
- Inconsistent beliefs arise because agents have different views of the world.
  - May be due to sensor faults or noise or just because they can't see everything.
- Inconsistent goals may arise because agents are built by different people with different objectives.

Handling inconsistency

- Three ways to handle inconsistency (Durfee at al.)
- Do not allow it
  - For example, in the contract net the only view that matters is that of the manager agent.
- Resolve inconsistency
  - Agents discuss the inconsistent information/goals until the inconsistency goes away. Argumentation.
- Build systems that degrade gracefully in the face of inconsistency. – Most desirable
  - Functionally accurate/cooperative systems: Nodes do the best they can with their current information, but their solutions to their local sub-problems may be only partial, tentative, and incorrect.

Handling inconsistency -Functionally accurate/cooperative systems

- Problem solving is not tightly constrained to a particular sequence of events - it progresses opportunistically (i.e. not in a strict predetermined order, but taking advantage of whatever opportunities arise) and incrementally (i.e. by gradually piecing together solutions to sub-problems).
- Agents communicate by exchanging high-level intermediate results, rather than by exchanging raw data.
- Uncertainty and inconsistency is implicitly resolved when partial results are exchanged and compared with other partial solutions. Thus inconsistency and uncertainty is resolved as problem solving progresses, rather than at the beginning or end of problem solving.
- The solution is not constrained to a single solution route: there are many possible ways of arriving at a solution, so that if one fails, there are other ways of achieving the same end. This makes the system robust against localized failures and bottlenecks in problem solving.

Coordination

- Coordination is managing dependencies between agents

- Example
  - We both want to leave the room through the same door. We are walking such that we will arrive at the door at the same time. What do we do to ensure we can both get through the door?
  - We both arrive at the copy room with a stack of paper to photocopy. Who gets to use the machine first?

Coordination

- Von Martial (1990) suggested suggested a typology for coordination relationships. He suggested that, broadly, relationships betwwn activities could he either positive or negative.
- *Positive* coordination is 'are all those relationships between two plans from which some benefit can be derived, for one or both of the agents plans, by combining them'
  - Requested (explicit)
  - Non-requested (implicit) – pareto optimality
    - Action equality: we both plan to do something, and by recognizing this one of us can be saved the effort.
    - Consequence: What I plan to do will have the side-effect of achieving something you want to do.
    - Favor: What I plan to do will make it easier for you to do what you want to do.

Coordination by Joint intentions

- Practical reasoning and intentions
- Intentions also play a critical role in coordination: they provide both the stability and predictability that is necessary for social interaction, and the flexibility and reactivity that is necessary to cope with a changing environment.
- Just as we have individual intentions, we can have joint intentions for a team of agents.
- Coordinated action that is not cooperative vs coordinated cooperative action
- Levesque defined the idea of a *joint persistent goal* (JPG).
- A group of agents have a collective commitment to bring about some goal φ, "move the couch". – responsibility
- Also have motivation ϕ, "Simon wants the couch moved".
- The mental states of agents mirror those in BDI agents.
- Agents don't believe that φ is satisfied, but believe it is possible.
- Agents maintain the goal φ until a termination condition is reached.

Coordination by Joint intentions

- The terminations condition is that it is *mutually believed* that:
  - goal φ is satisfied; or
  - goal φ is impossible; or
  - the motivation ϕ is no longer present.
- You and I have a mutual belief that p if I believe p and you believe p and I believe that you believe p and I believe that you believe that I believe p and . . . .

Coordination by Joint intentions

- The termination condition is achieved when an agent realises that the goal is satisfied, impossible and so on.
- But it doesn't drop the goal right away.
- Instead it adopts a new goal — to make this new
- knowledge mutually believed.
- This ensures that the agents are coordinated.
- They don't stop working towards the goal until they are all appraised of the situation.
- Mutual belief is achieved by communication.

Coordination by norms and social laws

- Societies are often regulated by (often unwritten) rules of behavior – conventions vs agent actions
- Example:
  - A group of people is waiting at the bus stop. The bus arrives. Who gets on the bus first?
- Another example:
  - On 34th Street, which side of the sidewalk do you walk along?
- In an agent system, we can design the norms and program agents to follow them, or let norms evolve.

Offline design

- Recall how we described agents before:

$$Ag : R^E \rightarrow Ac$$

a function which, given a run ending in a state, gives us an action.

- A *constraint* is then a pair:

$$(E^t, \alpha)$$

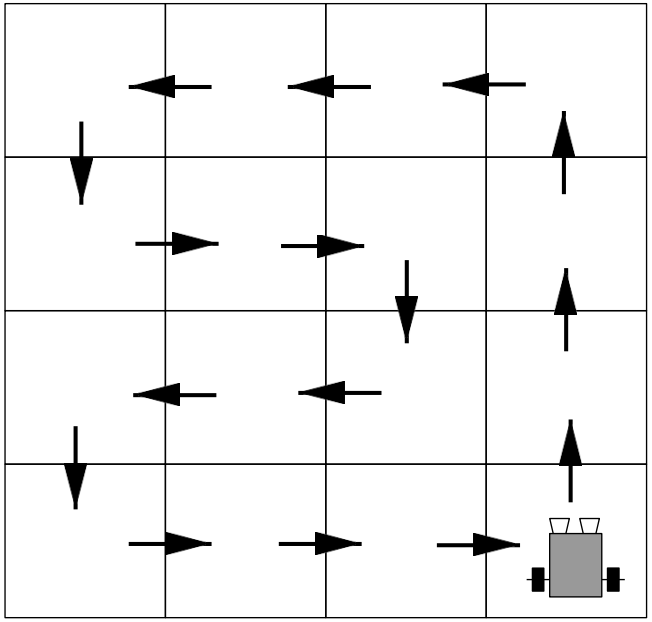
where $E^t \subseteq E$ and $\alpha \in Ac$.

- This constraint says that α cannot be done in any state in E^t.
- A *social law* is *a set of constraints*.

Offline design

- We can refine our view of an environment.
- *Focal* states, $F \subseteq E$ are the states we want our agent to be able to get to.
- From any focal state $e \in F$ it should be possible to get to any other focal state $e^{t} \in F$ (though not necessarily right away).
- A *useful* social law is then one that does not prevent agents from getting from one focal state to another.

- A useful social law that prevents collisions :



- Not necessarily efficient (On^2 steps to get to a specific square).

Emergence

- We can also design systems in which social laws emerge.

  Conventions should be global

  - T-shirt game (Shoham and Tennenholtz):

  Agents have both a red t-shirt and a blue t-shirt and wear one. Goal is for everyone to end up with the same color on. In each round, each agent meets one other agent, and decides whether or not to change their shirt. During the round they only see the shirt their pair is wearing — they don't get any other information.

  - What *strategy update function* should they use?

Strategy Update Functions

- *Simple majority*:

Agents pick the shirt they have seen the most.

- *Simple majority with types*:

Agents come in two types. When they meet an agent of the same type, agents pass their memories.
Otherwise they act as simple majority.

- *Highest cumulative reward*:

Agents can “see” how often other agents (some subset of all the agents) have matched their pair. They pick the shirt with the largest number of matches.

Multiagent planning

- Another approach to coordinate is to explicitly plan what all the agents do.
- For example, come up with a large STRIPS plan for all the agents in a system.
  - Could have:
    - Centralized planning for distributed plans
    - One agent comes up with a plan for everybody
    - Distributed planning
    - A group of agents come up with a centralized plan for another group of agents.
- Distributed planning for distributed plans
- Agents build up plans for themselves, but take into account the actions of others.

Multiagent planning

- In general, the more decentralized it is, the harder it is.
- Georgeff proposed a distributed version of STRIPS.
- New list: *during:* Specifies what must be true while the action is carried out.
- This list contains a set of conditions which must hold while the action is being carried out. A plan is seen as a set of states; an action is seen as a function which maps the set onto itself. The precondition of an action specifies the domain of the action; the add and delete lists specify the range.
- This places constraints on when other agents can do things.

Multiagent planning

- Different agents plan to achieve their goals using these operators and then do:
  - Interaction analysis: do different plans affect one another. Describe goal interactions
    - Satisfiability: Two actions are said to be satisfiable if there is some sequence in which they may be executed without invalidating the preconditions of one or both.
    - Commutativity: a restricted case of satisfiability: if two actions may be executed in parallel, then they are said to be commutative. It follows that if two actions are commutative, then either they do not interact, or any interactions are harmless
    - Precedence: the sequence in which actions may be executed; if action a1 has precedence over action a2, then the preconditions of a2 are met by the postconditions of a1. That is not to say that a1must be executed before a2; it is possible for two actions to have precedence over each other.

Multiagent planning

- (Interaction analysis)
- Safety analysis: which interactions are problematic? which of these interactions are unsafe. Safeness of pairs of actions in terms of the precedence and commutativity of the pair. Safety analysis involves two stages. First, all actions which are harmless (i.e. where there is no interaction, or the actions commute) are removed from the plan. This is known as simplification. Secondly, the set of all harmful interactions is generated. This stage also involves searching; a rule known as the commutativity theorem is applied to reduce the search space. All harmful interactions have then been identified.
- Interaction resolution: treat the problematic interactions as *critical sections* and enforce mutual exclusion.