Nested *ABM*s

When Certain Agents Use *ABM*s to Make Decisions

Rob Axtell and Scott Eldridge
George Mason University

Methodological Individualism

Social Science Models Represent the Actions of Individuals

- 19th C idea, characteristic of economics, game theory, finance
- It denies that distinct human actions can be aggregated into summary mathematical representations, in general
- A high standard and often not met (e.g., representative agent macroeconomics)
- Agent-based modeling (*ABM*) is methodologically individualist (*MI*) and can serve as a methodology in economics, say, when analytical difficulties arise
- However, agents in *MI* models are *not* themselves *MI*, i.e., they rely on summary mathematical representations to make decisions

ABM Provides a Workaround

Shallow vs Deep Methodological Individualism

- If agents had methodologically individualist internal models in conventional analytical social science models, it is not clear how to 'solve' them
  - What is the appropriate solution concept?
  - How hard would it be to solve such models?
- Representing the internal deliberations of even one additional agent can be challenging analytically (e.g., *k*-level rationality in game theory)
- But with *ABM* 'solving' a model really just means 'running' it, marching it fwd in time
- Shallow methodological individualism: *MI* models where the agents are *not MI*
- Deep methodological individualism: *MI* models in which the agents are *MI*

Prior Art

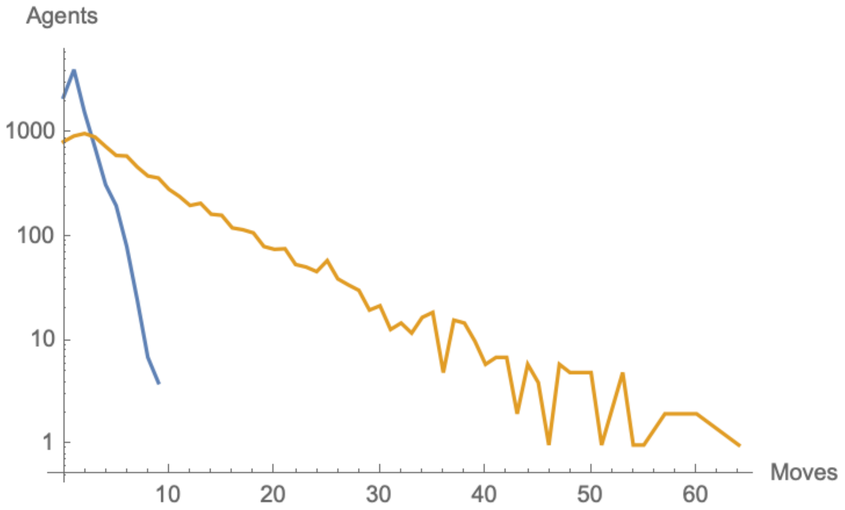
An Idea Often Discussed, Rarely Implemented

- *Theory of Mind* in philosophy considers the importance of individuals having an internal representation of those with whom they are interacting
- Early *ABM*: in *SWARM* it was possible to instantiate an *ABM* in which internal objects were also *SWARM* objects, including other *ABM*s; never much utilized
- In *NetLogo* similar ideas have been played around with, such models run very slowly
- To some extent possible in *RePast* and *Mason*, often simply based on recursion
- What has been little explored is the possibility of *many level ABM*s in which all agents above the lowest level utilize *ABMs*
- Focus first on *ABM*s with 2 levels...

Schelling-type Segregation Models

Unhappy Agents Utilize their Own *ABMs*

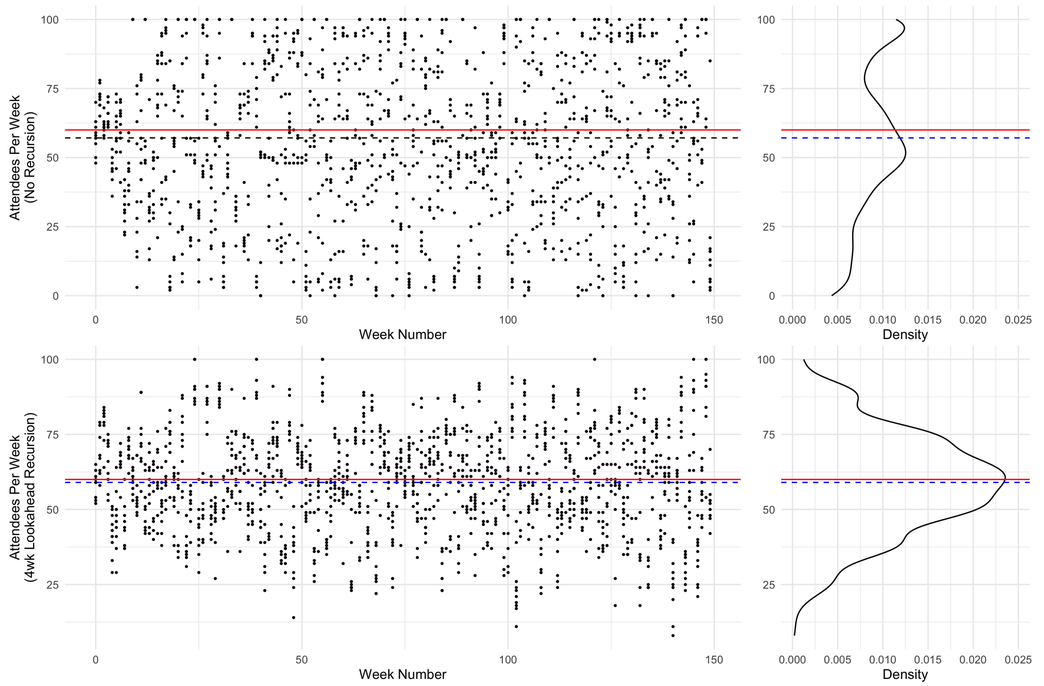
- In usual models, unhappy agents select places at random where they will be happy...
- *...without considering whether those places are likely to keep them happy in the future*
- A 2 level *ABM* gives each unhappy agent the ability to conduct its own 'simulations' of candidate locations
- Such 'candidates' can be based either on the actual model or an idiosyncratic (local) version having incomplete information



El Farol *ABM*

Results of Offering Agents an Additional Decision Function that is an *ABM*

- In the usual *El Farol ABM*, each agent has a set of decision rules, keeps track of their performance, and chooses the highest performing one
- Here, each set of decision rules is augmented with an *ABM* that also makes a forecast and the agents either use that information or not
- The result of having this option leads to more probability mass around the desired outcome



Nested *ABM*s Multiple Levels Deep

Combinatorial Growth of the Number of Agents and Models

- 1 *ABM* with *A* agents
- Give every agent an *A* - 1 agent *ABM*: *A*+1 *ABM*s and *A*+*A*(*A*-1) = *A*^2 agents
- Give every one of the agents in the *A* - 1 agent level models a *A* - 2 agent *ABM*...
- ...
- Give every one of the agents in the 3 agent level models a 2 agent *ABM*
- Overall, the number of:
  - *ABM*s: $1 + ePI(P, 1) - 2I(P + 1)$
  - Agents needed: $ePI(P, 1) - I(P + 1)$

$$I(k, x) = \int_x^\infty t^{k+1} exp(-t) dt$$

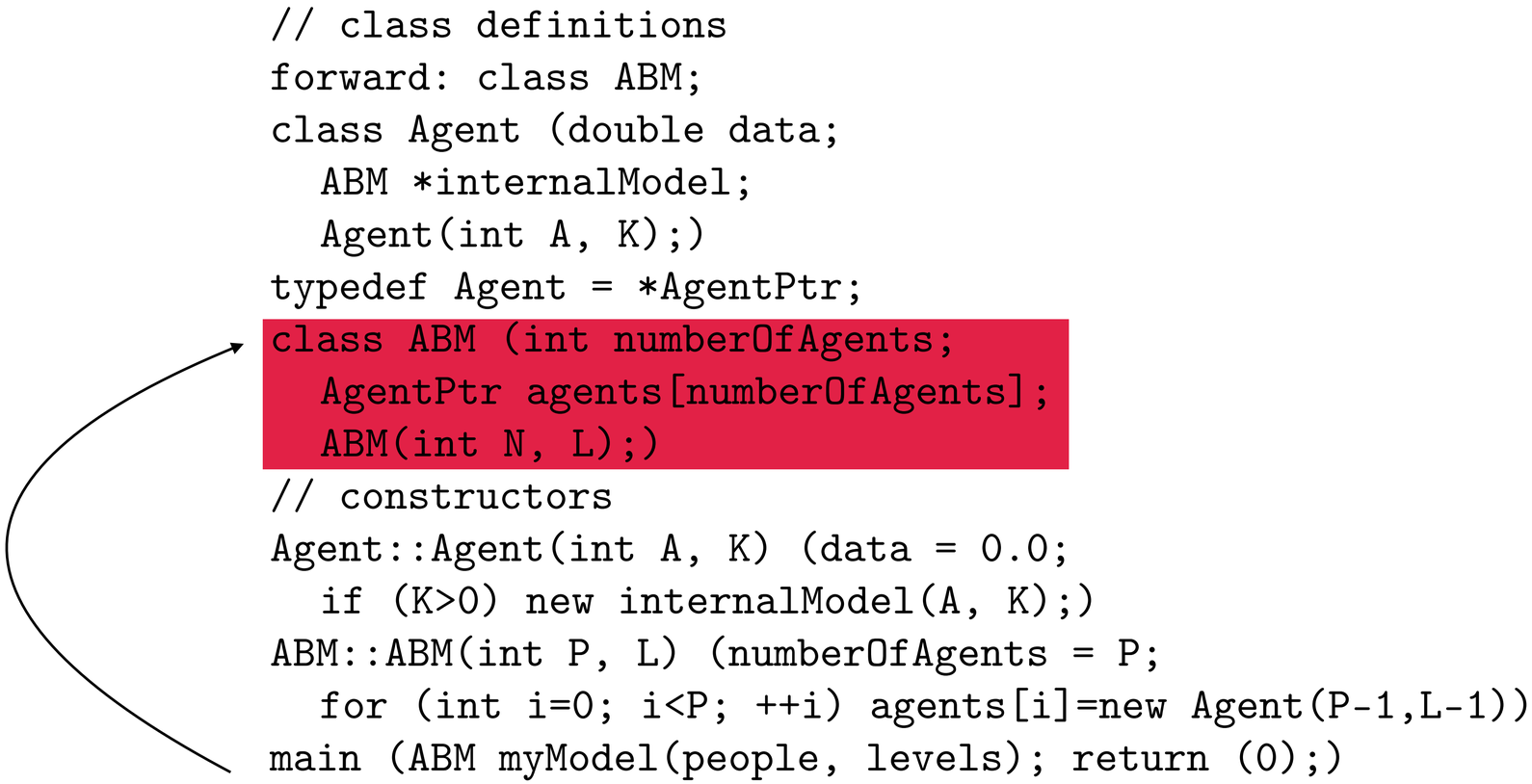
Pseudo-Code

Instantiates *ABMs* for Arbitrary Number of People Down any Number of Levels

```
// class definitions
forward: class ABM;
class Agent (double data;
  ABM *internalModel;
  Agent(int A, K);)
typedef Agent = *AgentPtr;
class ABM (int numberOfAgents;
  AgentPtr agents[numberOfAgents];
  ABM(int N, L);)
// constructors
Agent::Agent(int A, K) (data = 0.0;
  if (K>0) new internalModel(A, K);)
ABM::ABM(int P, L) (numberOfAgents = P;
  for (int i=0; i<P; ++i) agents[i]=new Agent(P-1,L-1))
main (ABM myModel(people, levels); return (0);)
```

Pseudo-Code

Instantiates *ABMs* for Arbitrary Number of People Down any Number of Levels

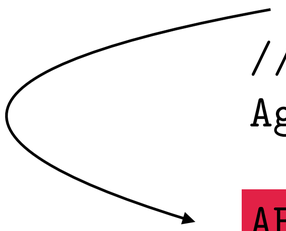


```
// class definitions
forward: class ABM;
class Agent (double data;
  ABM *internalModel;
  Agent(int A, K);)
typedef Agent = *AgentPtr;
class ABM (int numberOfAgents;
  AgentPtr agents[numberOfAgents];
  ABM(int N, L);)
// constructors
Agent::Agent(int A, K) (data = 0.0;
  if (K>0) new internalModel(A, K);)
ABM::ABM(int P, L) (numberOfAgents = P;
  for (int i=0; i<P; ++i) agents[i]=new Agent(P-1,L-1))
main (ABM myModel(people, levels); return (0);)
```

Pseudo-Code

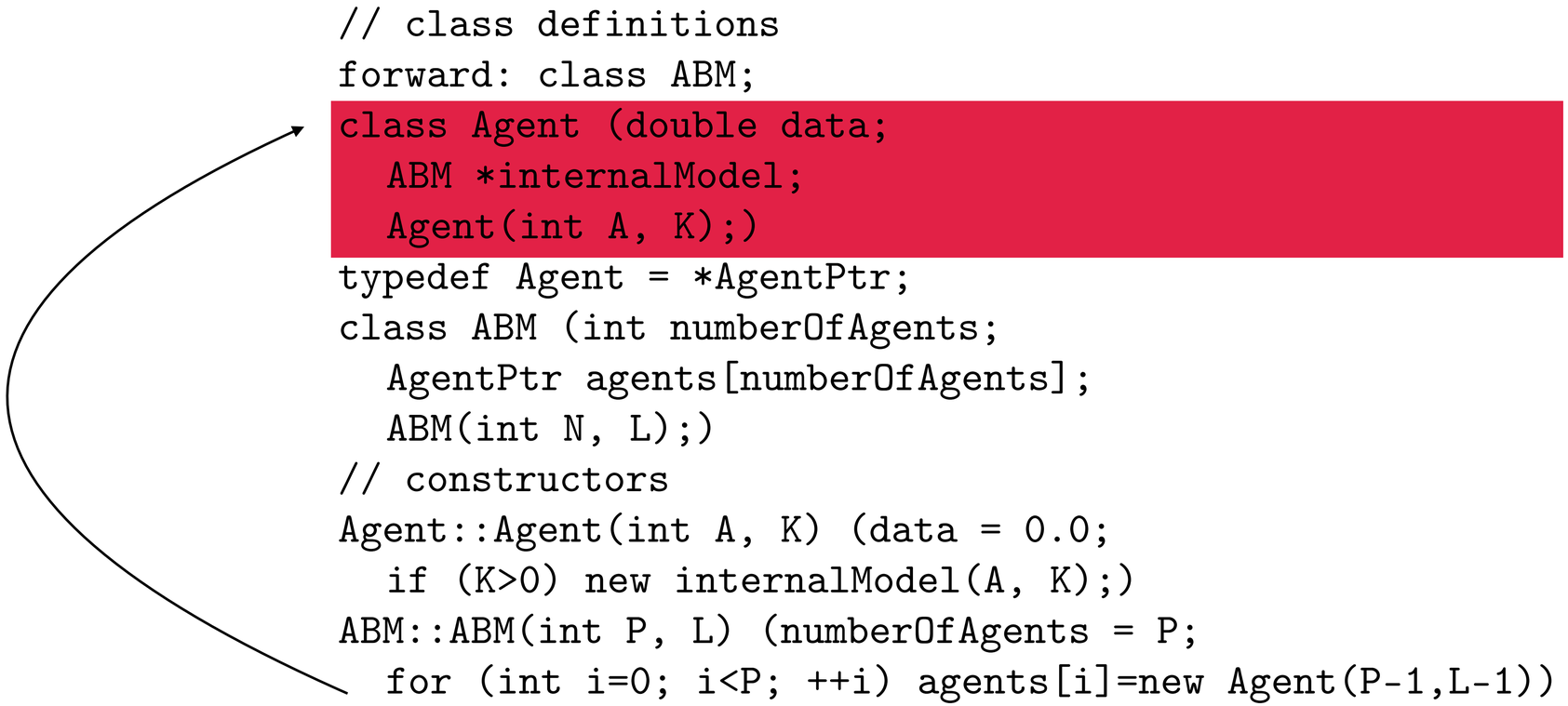
Instantiates *ABMs* for Arbitrary Number of People Down any Number of Levels

```
// class definitions
forward: class ABM;
class Agent (double data;
  ABM *internalModel;
  Agent(int A, K);)
typedef Agent = *AgentPtr;
class ABM (int numberOfAgents;
  AgentPtr agents[numberOfAgents];
  ABM(int N, L);)
// constructors
Agent::Agent(int A, K) (data = 0.0;
  if (K>0) new internalModel(A, K);)
ABM::ABM(int P, L) (numberOfAgents = P;
  for (int i=0; i<P; ++i) agents[i]=new Agent(P-1,L-1))
main (ABM myModel(people, levels); return (0);)
```



Pseudo-Code

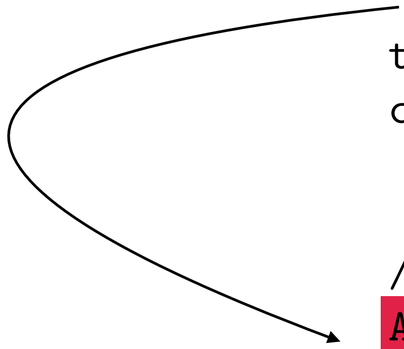
Instantiates *ABMs* for Arbitrary Number of People Down any Number of Levels



```
// class definitions
forward: class ABM;
class Agent (double data;
  ABM *internalModel;
  Agent(int A, K);)
typedef Agent = *AgentPtr;
class ABM (int numberOfAgents;
  AgentPtr agents[numberOfAgents];
  ABM(int N, L);)
// constructors
Agent::Agent(int A, K) (data = 0.0;
  if (K>0) new internalModel(A, K);)
ABM::ABM(int P, L) (numberOfAgents = P;
  for (int i=0; i<P; ++i) agents[i]=new Agent(P-1,L-1))
main (ABM myModel(people, levels); return (0);)
```

Pseudo-Code

Instantiates *ABMs* for Arbitrary Number of People Down any Number of Levels

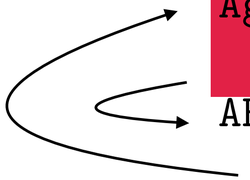


```
// class definitions
forward: class ABM;
class Agent (double data;
  ABM *internalModel;
  Agent(int A, K);)
typedef Agent = *AgentPtr;
class ABM (int numberOfAgents;
  AgentPtr agents[numberOfAgents];
  ABM(int N, L);)
// constructors
Agent::Agent(int A, K) (data = 0.0;
  if (K>0) new internalModel(A, K);)
ABM::ABM(int P, L) (numberOfAgents = P;
  for (int i=0; i<P; ++i) agents[i]=new Agent(P-1,L-1))
main (ABM myModel(people, levels); return (0);)
```

Pseudo-Code

Instantiates *ABMs* for Arbitrary Number of People Down any Number of Levels

```
// class definitions
forward: class ABM;
class Agent (double data;
  ABM *internalModel;
  Agent(int A, K);)
typedef Agent = *AgentPtr;
class ABM (int numberOfAgents;
  AgentPtr agents[numberOfAgents];
  ABM(int N, L);)
// constructors
Agent::Agent(int A, K) (data = 0.0;
  if (K>0) new internalModel(A, K);)
ABM::ABM(int P, L) (numberOfAgents = P;
  for (int i=0; i<P; ++i) agents[i]=new Agent(P-1,L-1))
main (ABM myModel(people, levels); return (0);)
```



Nested *ABM*s

A Conceptual Idea that is Realizable using Modern Computing

- Extant methodologically individualist models are *shallow*
- It is possible to make *MI* models in which the agents are themselves *MI*
- Having 2 levels—the original level and one lower level—seems reasonable given discussions in game theory (e.g., *k*-level rationality) and philosophy (*ToM*)
- Going to arbitrary levels is possible but consumes a lot of computing resources
- Many open questions:
  - Can fully-rational behavior be seen as the limit of all agents having all models?
  - Agents do *not* have perfect representations of other agents: we need to use this!