ABIDES Learning Summary

What Oracles Represent

Oracles are the sandbox "physics" for price discovery. Rather than pulling historical data, a config specifies a stochastic process that plays the role of hidden fundamental value. Agents can ping the oracle for noisy observations, mimicking proprietary research signals or imperfect valuation models. Because the oracle is seeded, every run with the same seed produces the same latent path, letting us compare agents head-to-head under identical information.

- Think of the oracle as a signal generator: it defines the true value path r_t that no agent fully observes. Agents only see noisy samples $o_t = r_t + \varepsilon_t$, where ε_t captures modeling error, execution noise, or misestimation.
- Using oracles lets us design controlled experiments: "If everyone had access to noisy fundamentals of variance σ_n^2, which strategy wins?" It is intentionally idealized versus real markets where only public order flows are observable.
- Exchanges also rely on the oracle to seed opening prices or log the true fundamental path (e.g., `fundamental_JPM.bz2`) so we can evaluate P&L and tracking error ex post.

Oracle

- **SparseMeanRevertingOracle:** Generates a latent fundamental series via an Ornstein–Uhlenbeck (OU) process

  $$dr_t = \kappa(\bar{r} - r_t)dt + \sigma\, dW_t$$

  per symbol, seeded with configurable $\bar{r}$, κ, and volatility parameters. To keep prices from hugging the mean, it inserts Poisson-distributed "megashocks" with spacing parameter λ_a and bimodal normal magnitudes. This combination produces realistic drifts punctuated by jumps. Agents call `observePrice(symbol, t, sigma_n)` to receive $o_t \sim \mathcal{N}(r_t, \sigma_n)$ using their own random states, so two agents observing simultaneously get different signals. The exchange stores the exact r_t series for later analysis.

Configs

- **sparse_zi_100 / sparse_zi_1000:** Minimal lab with one exchange and only ZeroIntelligence agents (100 or 1000). Useful to study the kernel lifecycle, latency matrix, and how ZI behavior alone shapes order books. Because every agent shares the same oracle but randomizes trade direction via a coin flip, we can inspect how surplus thresholds R and η affect inventory swings and P&L.
- **value_noise:** Adds two archetypes—Noise agents (one-shot random liquidity) and Value agents (systematic fundamental traders). No market maker is instantiated, so spreads are entirely determined by these two populations. This config highlights how a mix of informed and uninformed flow moves prices relative to the latent fundamental.
- **rmsc01:** First "reference market simulation" setup with a richer ecology: one exchange, a market maker, 50 ZI agents, 25 heuristic-belief-learning agents, and 24 momentum agents. Demonstrates how multiple strategy families (fundamental, technical, liquidity providers) interact within the same session.
- **rmsc02:** Same agent mix as rmsc01 but showcases the data subscription mechanism and the deterministic latency model. The market maker and momentum agents subscribe to the book instead of polling, and a line-based latency model places agents between NYC and Seattle to simulate realistic communication delays. Also serves as a template for plugging in your own agent via the `--agent_name` flag.

- **rmsc03:** Shifts to a different ecosystem: two adaptive POV market makers, 5,000 noise agents, 100 value agents, 25 momentum agents, plus an optional POV execution agent. Configurable CLI flags let you tune the market makers' participation rate, adaptive spread, skewing, etc., and an optional execution bot slices a single order by participation-of-volume. Useful for studying adaptive liquidity provision and execution in a busier market.

Agents

- **ZeroIntelligenceAgent (ZI):**
  - **Purpose:** A baseline agent from experimental econ literature that knows its private valuation but otherwise acts randomly. It helps measure how much of the market dynamics come from simple surplus-seeking behavior under latency constraints.
  - **Wakeups:** Inter-arrival times are exponential with mean $1/\lambda_a$. On each wake, it cancels resting orders and requests the current spread (logged as `BID_DEPTH`/`ASK_DEPTH`/`IMBALANCE`). Those entries reflect the information snapshot; an actual trade occurs only if the subsequent limit order gets filled.
  - **Signal processing:** Receives a noisy oracle observation o_t, updates posterior mean/variance (r_t, σ_t) via Bayesian weighting, then projects to r_T (expected close) using OU dynamics.
  - **Decision rule:**
    - Flips a coin to choose long vs. short (bounded by $q_{\max}$ holdings). Despite the "zero intelligence" label, it still respects the oracle signal when pricing.
    - Draws surplus $R \in [R_{\min}, R_{\max}]$. Proposed limit price is $p = r_T \mp R$ (minus for buys, plus for sells).
    - If the inside quote already offers surplus $\geq \eta R$, it takes liquidity immediately; otherwise it posts a 100-share limit at p and waits for execution (triggering `ORDER_EXECUTED`/`HOLDINGS_UPDATED` only when filled).
  - **Use cases:** Stress-test matching engine, latency models, and queue dynamics under many simple agents. Compare different R buckets to see how aggressiveness affects ending cash.
- **NoiseAgent:**
  - **Purpose:** Provide background order flow that mimics retail or random liquidity. Helps other strategies encounter realistic noise and ensures there are prints even when informed agents wait.
  - **Wakeups:** Each agent gets a single timestamp t_w drawn uniformly between open and close. Until t_w, it keeps rescheduling itself—so it trades exactly once.
  - **Action:** At t_w it requests `QUERY_SPREAD`, receives bid/ask ladders, flips a coin, and submits a 20–50 share limit order at the inside quote (buy at best ask, sell at best bid). If the opposing side has depth, it fills instantly; otherwise it becomes the new inside quote. If the book is empty, it does nothing.
  - **Use cases:** Inject randomness, test resiliency of market makers or value agents, and study how one-shot traders impact spreads.
- **ValueAgent:**
  - **Purpose:** Prototype fundamental investor. Unlike ZI, direction is deterministic: always buy when projected fundamental exceeds midprice and sell otherwise.
  - **Wakeups:** Exponential inter-arrival with rate λ_a. Each wake cancels resting orders, requests `QUERY_SPREAD`, and waits for the exchange reply.
  - **Signal processing:** Same Bayesian update as ZI, producing r_T (expected close).
  - **Decision rule:**
    - If $r_T > m$ (midprice), places a buy; if $r_T < m$, places a sell. No coin flips.
    - Chooses aggressiveness with probability p_{aggr}: either crosses/quotes inside the spread or posts passively up to `depth_spread` ticks away. Order size drawn from $[20, 50]$ shares.

– **Use cases:** Study how informed flow tracks the latent fundamental, analyze P&L relative to oracle truth, and compare against ZI/Noise populations.

- **HeuristicBeliefLearningAgent (HBL):**
  - **Purpose:** Builds on ZI by learning from recent order stream history. Inspired by heuristic belief learning models where agents estimate fill probabilities from observed orders.
  - **Wakeups:** Same as ZI, but after the base wake it requests an order stream of length L (logged as `QUERY_ORDER_STREAM`) before requesting the spread.
  - **Signal processing:** Uses the same oracle observation and Bayesian update as ZI to determine direction and total valuation v.
  - **Decision rule:**
    * Analyzes the last L trades/orders, aggregating counts of successful and unsuccessful buy/sell orders at each price level.
    * Computes estimated execution probability $\Pr(\text{fill at } p)$ and expected surplus $E_s(p)$ for all prices between the observed min/max.
    * Chooses the price $p^\star$ with the highest positive expected surplus and submits a 100-share limit order there; if $E_s(p^\star) \leq 0$, it can abstain or fall back to ZI behavior.
  - **Use cases:** Examine how learning from microstructure data improves over pure ZI behavior, and compare expected vs. realized surplus by price level.

- **MomentumAgent:**
  - **Purpose:** Simple order-book-driven trend follower. Compares short vs. long moving averages of the mid-price to decide trade direction without any oracle signal.
  - **Wakeups:** Either polls the spread every `wake_up_freq` or subscribes to market data. Upon receiving bid/ask info, updates its mid-price history.
  - **Decision rule:**
    * Maintains rolling 20-period and 50-period moving averages of the mid-price.
    * If the 20-period average is greater than or equal to the 50-period average, places a buy order of random size (between `min_size` and `max_size`) at the current ask; otherwise sells at the current bid.
  - **Use cases:** Provides a purely technical baseline to contrast with oracle-driven agents; useful for studying how trend-following interacts with other strategies.

- **MarketMakerAgent:**
  - **Purpose:** Provide symmetric liquidity around the mid-price. It does not consult the oracle or manage inventory; it simply keeps bids and asks at multiple price levels.
  - **Wakeups:** Either polls every `wake_up_freq` or subscribes to market data. Each wake it cancels existing quotes before placing new ones.
  - **Decision rule:**
    * Polling mode (default in `rmsc01`): after `QUERY_SPREAD`, computes $mid = \lfloor (bid + ask)/2 \rfloor$ and half-spread $s = \lfloor |ask - bid|/2 \rfloor$. With `subscribe_num_levels`=5 it builds 10 levels, posting bids at $mid - s - i$ and asks at $mid + s + i$ for $i = 0 \ldots 9$. Even if $s = 0$ (1-tick spread), the ladder ensures depth beyond the inside; orders only execute when price moves or incoming marketable flow consumes that level.
    * Subscription mode: receives multi-level book updates, splits its size according to `levels_quote_dict`, and places quotes at the actual bid/ask prices provided (or one tick beyond the deepest level if needed). For example, with `subscribe_num_levels`=3 and a split $[0.5, 0.5, 0, 0, 0]$, a 400-share size would be split into two 200-share orders at the top two bid levels and similarly on the ask side. Each new snapshot triggers cancellation of prior quotes and placement of fresh ones.

- **Use cases:** Ensures there's always liquidity on both sides so other agents can trade; great for testing how informed strategies interact with passive liquidity.

- **AdaptiveMarketMakerAgent (POV MM):**
  - **Purpose:** Ladder-style market maker that sizes orders based on a target participation-of-volume (pov) and can adapt its spread window to observed market conditions. Used in rmsc03.
  - **Wakeups:** Supports polling or subscription similar to other agents. Polling mode cancels orders, waits `cancel_limit_delay`, requests spread and transacted volume over `wake_up_freq`; subscription mode requests `MARKET_DATA` and volume updates at `subscribe_freq`.
  - **Order sizing:** Computes recent transacted volume and sets per-side size to $\max(\text{pov} \times \text{volume}, \text{min\_order\_size})$. If `skew_beta` ¿ 0 it skews size via a sigmoid of current inventory so surplus inventory gets quoted more aggressively.
  - **Adaptive spread:** If `window_size='adaptive'`, maintains an EWMA of the spread (controlled by `spread_alpha`) and updates the ladder width/tick spacing accordingly. Anchors (`top`, `bottom`, `middle`) determine how the window is centered around mid.
  - **Ladder placement:** Computes bid/ask price levels spanning `num_ticks` on each side with spacing `level_spacing` $\times$ spread. Optional `backstop_quantity` places larger orders at the extremes to prevent liquidity dropouts. Then submits symmetric buy/sell orders at each price with the computed sizes.
  - **Use cases:** Models a more realistic liquidity provider that reacts to actual volume and spread changes, making it a good testbed for adaptive market-making strategies and execution agents.

- **POVExecutionAgent:**
  - **Purpose:** Execute a single parent order by trading a fixed participation-of-volume (POV) rate. You specify direction, total quantity, start/end times, and the agent slices the order accordingly.
  - **Workflow:** Every `freq` (e.g., 1 minute) it measures total transacted volume over the last `lookback_period`, cancels any resting orders, and submits a market order of size $\text{pov} \times \text{volume}$ in the chosen direction—until the parent order quantity is filled or the end time passes.
  - **Use cases:** Benchmark execution algorithm for stress-testing adaptive market makers and examining slippage vs. participation rates.