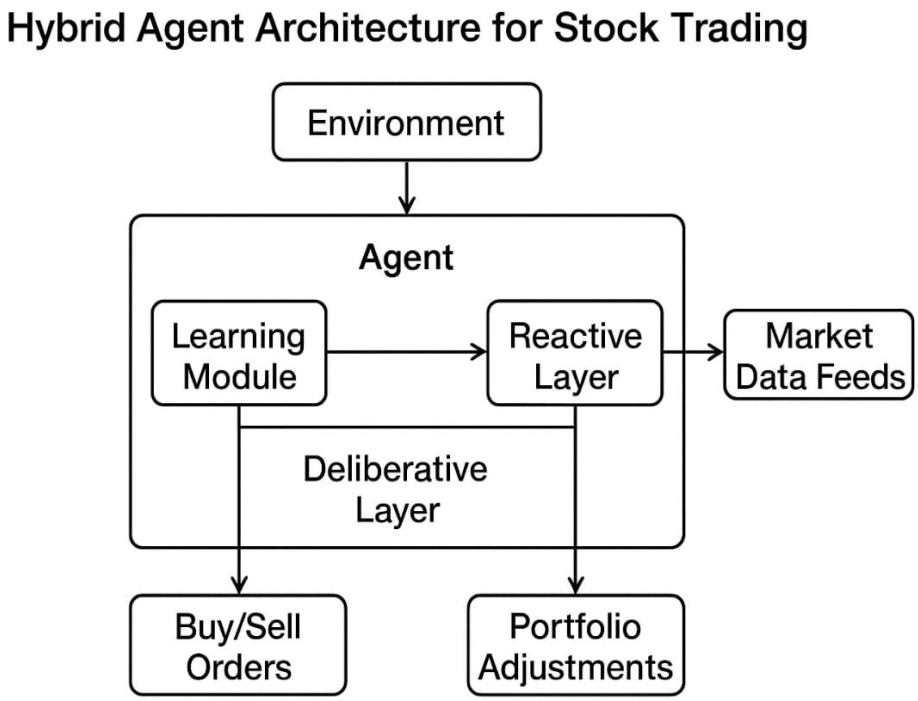
Problem Statement: Identify agent architecture for Stock trading systems.



Pseudo Code:

```
AGENT HybridTradingAgent
INPUTS:
  market_stream          // ticks: prices, volumes, news signals
  account_interface      // to place/cancel orders, query positions/cash
  clock                  // market time utils

STATE (Beliefs):
  B.prices, B.features, B.orderbook, B.signals
  B.positions, B.cash, B.pnl, B.unrealized_risk
  B.model_params         // forecasting / policy params
  B.world_uncertainty    // e.g., volatility, regime
  B.last_plan            // most recent target portfolio / orders

GOALS:
  maximize risk-adjusted return subject to constraints

CONSTRAINTS / RISK LIMITS:
  max_drawdown, max_position, max_leverage, max_var, max_order_rate, etc.

--------------------------------------------------------------------------
INIT():
  LOAD_STRATEGY_LIBRARY()          // mean-rev, momentum, stat-arb, news, etc.
  INIT_FORECAST_MODELS()           // e.g., ML predictors, RL policy, filters
  INIT_RISK_ENGINE()               // VaR/ES, volatility, stress, exposures
  INIT_EXECUTION_ENGINE()          // order slicer, smart router, TCA
  SET_REACTIVE_RULES()             // hard safety rules (stop-loss, halts…)
```

```
  B <- INITIAL_BELIEFS()
  LOG("agent ready")

--------------------------------------------------------------------------
MAIN_LOOP():
  WHILE clock.MARKET_IS_OPEN():
    obs <- RECEIVE(market_stream)                    // --- Perception ---
    B <- UPDATE_BELIEFS(B, obs)

    // ------------ Safety First (Reactive overrides everything) -------------
    if RISK_BREACH(B, CONSTRAINTS) then
       EMERGENCY_ACTIONS <- FLATTEN_OR_HEDGE(B)
       EXECUTE(EMERGENCY_ACTIONS, account_interface)
       CONTINUE   // skip planning; keep loop tight
    end if

    REACTIVE_ACTIONS <- REACTIVE_LAYER(B)
    if NOT EMPTY(REACTIVE_ACTIONS) then
       EXECUTE(REACTIVE_ACTIONS, account_interface)
    end if

    // ----------------- Deliberative plan on schedule / events --------------
    if SHOULD_PLAN(clock, B, obs) then
       FORECASTS <- FORECAST_RETURNS_AND_RISK(B)       // models -> μ, Σ, probs
       CANDIDATES <- GENERATE_CANDIDATE_PLANS(FORECASTS,
STRATEGY_LIBRARY)
       BEST_PLAN  <- ARGMAX_OVER_PLANS(UTILITY(plan, B, CONSTRAINTS),
CANDIDATES)
       B.last_plan <- BEST_PLAN

       PLANNED_ORDERS <- TRANSLATE_PLAN_TO_ORDERS(BEST_PLAN, B,
account_interface)
    else
       PLANNED_ORDERS <- {}
    end if

    // ----------------------- Arbitration / Fusion --------------------------
    ACTIONS <- ARBITRATE(REACTIVE_ACTIONS, PLANNED_ORDERS,
priority="safety>reactive>plan")

    // --------------------------- Execution -------------------------------
    EXECUTE(ACTIONS, account_interface)
    MONITOR_FILLS_AND_SLIPPAGE(ACTIONS, account_interface)

    // ----------------------------- Learning ------------------------------
    REWARD <- COMPUTE_REWARD(B)                      // e.g., PnL net of costs, risk
    B.model_params <- LEARNING_UPDATE(B.model_params, B, ACTIONS, REWARD)
    ADAPT_RISK_LIMITS_IF_NEEDED(B, performance_window)

    LOG_STEP(B, ACTIONS, REWARD)
  END WHILE
```

```
  CLOSE_OUT_POSITIONS_IF_REQUIRED()
  LOG("market closed")

-------------------------------------------------------------------------------
FUNCTION UPDATE_BELIEFS(B, obs):
  B.prices      <- obs.prices
  B.orderbook   <- obs.orderbook_snapshot
  B.signals     <- EXTRACT_SIGNALS(B.prices, news=obs.news, tech_indicators(...))
  B.features    <- FEATURE_PIPELINE(B)                // lags, regimes, embeddings
  B.positions   <- QUERY_POSITIONS(account_interface)
  B.cash, B.pnl <- QUERY_ACCOUNT(account_interface)
  B.unrealized_risk <- RISK_ENGINE(B.positions, B.prices, covariances(...))
  B.world_uncertainty <- ESTIMATE_VOLATILITY(B.prices)
  return B

-------------------------------------------------------------------------------
FUNCTION REACTIVE_LAYER(B):
  actions <- {}
  for each pos in B.positions:
     if HARD_STOP_LOSS_TRIGGERED(pos, B) then
        actions += CREATE_CLOSE_ORDER(pos)
     end if
     if TAKE_PROFIT_TRIGGERED(pos, B) then
        actions += PARTIAL_CLOSE(pos, tranche=rule.tp_tranche)
     end if
  end for
  if NEWS_HALT_OR_CIRCUIT_BREAKER(obs) then
     actions += CANCEL_OPEN_ORDERS()
  end if
  if POSITION_LIMIT_EXCEEDED(B) then
     actions += TRIM_TO_LIMITS(B)
  end if
  return actions

-------------------------------------------------------------------------------
FUNCTION SHOULD_PLAN(clock, B, obs):
  return clock.ON_REBALANCE_INTERVAL(5min)
        OR REGIME_SHIFT_DETECTED(B)
        OR LARGE_PRICE_JUMP(obs)
        OR HIGH_UNCERTAINTY_CHANGE(B)

-------------------------------------------------------------------------------
FUNCTION FORECAST_RETURNS_AND_RISK(B):
  μ <- PREDICT_EXPECTED_RETURNS(B.features, params=B.model_params)
  Σ <- ESTIMATE_COVARIANCE(B.prices, horizon=H)
  costs <- ESTIMATE_TC_IMPACT(B.last_plan, B.positions)
  return {mu: μ, cov: Σ, costs: costs}

-------------------------------------------------------------------------------
FUNCTION GENERATE_CANDIDATE_PLANS(F, LIB):
  plans <- {}
  plans += MEAN_VARIANCE_OPTIMIZER(F.mu, F.cov, constraints=CONSTRAINTS)
```

```
  plans += RISK_PARITY_TARGETS(F.cov)
  plans += SIGNAL_DRIVEN_TARGETS(LIB, F.mu, B.features)
  plans += DO_NOTHING_PLAN()
  return plans

--------------------------------------------------------------------------------
FUNCTION UTILITY(plan, B, CONSTRAINTS):
  exp_ret <- EXPECTED_RETURN(plan)
  risk    <- EXPECTED_RISK(plan)           // e.g., variance, drawdown, VaR/ES
  costs   <- TRANSACTION_COSTS(plan, B.positions)
  penalty <- CONSTRAINT_PENALTY(plan, CONSTRAINTS)
  return exp_ret - λ*risk - costs - penalty

--------------------------------------------------------------------------------
FUNCTION TRANSLATE_PLAN_TO_ORDERS(plan, B, api):
  target_positions <- PLAN_TO_TARGET_POSITIONS(plan, B.cash, price=B.prices)
  deltas <- target_positions - B.positions
  orders <- EXECUTION_ENGINE_CREATE_ORDERS(deltas, participation_cap,
child_slicer="TWAP/VWAP/POV")
  return orders

--------------------------------------------------------------------------------
FUNCTION ARBITRATE(reactive_orders, planned_orders, priority):
  if priority == "safety>reactive>plan":
     // safety already enforced; merge reactive first, then compatible plan orders
     actions <- reactive_orders
     actions += REMOVE_CONFLICTS(planned_orders, reactive_orders)
     return actions
  end if

--------------------------------------------------------------------------------
FUNCTION EXECUTE(orders, api):
  for o in orders:
     SUBMIT_ORDER(api, o)
  TRACK_AND_ADAPT(order_ids=orders, rules={max_slippage, timeout_cancel, reprice})
  return
```