מצוין, זוהי בדיוק השאלה הנכונה והחשובה ביותר בשלב זה. בניית רכיבים מתוחכמים היא דבר אחד; חיבורם למערכת חיה, נושמת ורובסטית שפועלת באופן עקבי הוא אתגר אחר לגמרי.

בוא נפרק את התהליך הזה לתוכנית בנייה מפורטת, שלב אחר שלב. התוכנית הזו תסביר **איך המערכת פועלת כלולאה אחת שלמה** וחשוב מכל, **איך ולמה הלוגיקה הזו נשארת זהה לחלוטין בין סביבת האימון למסחר החי**.

### **תוכנית תפעולית מאוחדת (SOP): מסחר חי ואימון**

#### **1. עקרון הזהב: שכבת ההפשטה (Abstraction Layer)**

לפני שנצלול לשלבים, חייבים להבין את עקרון הליבה שיאפשר לקוד להיות זהה. אנחנו מפרידים בין **"מה" המערכת עושה** (הלוגיקה) לבין **"איך" היא מקבלת קלט ומוציאה פלט** (החיבור לעולם החיצון).

* **דוגמה 1: DataHandler (מטפל הנתונים)**
  + אנחנו נגדיר ממשק (Interface) אחיד בשם AbstractDataHandler שיש לו פונקציה אחת: start\_streaming\_data().
  + ניצור שתי מחלקות שמממשות את הממשק הזה:
    1. RithmicDataHandler: מממשת את הפונקציה על ידי חיבור ל-API של Rithmic והאזנה לעדכוני מחיר בזמן אמת.
    2. BacktestDataHandler: מממשת את הפונקציה על ידי פתיחת קובץ נתונים היסטורי, קריאת שורה-אחר-שורה, והפקת עדכוני מחיר מדומים.
  + **הקוד הראשי של המערכת לא יודע ולא אכפת לו איזו מהן פועלת.** הוא פשוט מפעיל את start\_streaming\_data(). הבחירה בין השתיים תקבע בקובץ תצורה פשוט לפני הרצת המערכת.
* **דוגמה 2: ExecutionHandler (מטפל הביצוע)**
  + אותו עיקרון בדיוק. ממשק AbstractExecutionHandler עם פונקציה execute\_trade(trade\_plan).
  + LiveExecutionHandler יממש אותה על ידי שליחת פקודה ל-API של הברוקר.
  + BacktestExecutionHandler יממש אותה על ידי רישום העסקה בסימולטור וחישוב רווח/הפסד.

**עיקרון זה הוא הבסיס לכל מה שיבוא. הוא מבטיח שהמנוע והלוגיקה הפנימית זהים ב-100%**.

#### **2. ארכיטקטורה מונעת-אירועים: תוכנית בנייה מפורטת**

כעת נבנה את המערכת, שלב אחר שלב, ונראה איך כל הרכיבים מתחברים ופועלים כלולאה אחת.

##### **שלב 1: איסוף וחישוב נתונים (The "Refinery")**

* **1.1. אתחול:** המערכת מופעלת. מודול DataHandler (הגרסה החיה או הבדיקה) מתחיל לפעול.
* **1.2. בניית נרות:** מודול BarGenerator מאזין לנתונים הגולמיים מה-DataHandler. הוא צובר אותם ובונה נרות של 5 ו-30 דקות. ברגע שנר נסגר, הוא יורה אירוע NEW\_BAR עם נתוני הנר המלאים (OHLCV).
* **1.3. חישוב אינדיקטורים מרכזי:** מודול ה-IndicatorEngine מאזין לאירועי NEW\_BAR.
  + **פעולה:** הוא לוקח את הנר החדש, מבצע את כל החישובים הנדרשים (HA, FVG, MLMI, NW-RQK, LVN, MMD), ומעדכן את הערכים האחרונים ב-"מחסן מאפיינים" (Feature Store) מרכזי בזיכרון. זה מבטיח שכל חישוב מתבצע פעם אחת בלבד.
  + **הוא לא עושה שום דבר מעבר לחישוב.**

##### **שלב 2: הכנת המצב לסוכנים (The "Pipelines to the Engine")**

* **2.1. הרכבת המטריצות:** לאחר שה-IndicatorEngine סיים, מופעלים ה-MatrixAssemblers.
  + Assembler\_30m ניגש ל-Feature Store, שולף את ערכי 30 הדקות הרלוונטיים, ומרכיב מהם שורת מאפיינים חדשה. הוא דוחף את השורה החדשה לתחתית המטריצה N x F של Agent\_30m וזורק את השורה הישנה ביותר מלמעלה.
  + תהליך זהה קורה עבור Assembler\_5m ו-Assembler\_Regime.
* **בשלב זה, כל המטריצות של הסוכנים מעודכנות ומוכנות.**

##### **שלב 3: מחזור ההחלטה של ה-MARL (The "Engine Cycle")**

זהו לב המערכת כפי שתיארת, והוא פועל כלולאה לאחר ששלב 2 הסתיים.

* **3.1. הטמעה וקטורית:** המערכת מעבירה כל מטריצה מעודכנת לרשת ה-NN הייעודית שלה.
  + Matrix\_30m -> NN\_Embedder\_30m -> Vector\_30m
  + Matrix\_5m -> NN\_Embedder\_5m -> Vector\_5m
  + הפלט הוא שלושה וקטורים דחוסים המייצגים את "הבנת המצב" של כל סוכן.
* **3.2. הפעלת תתי-המערכות המומחים:**
  + **סוכן מצב שוק (Agent\_Regime):** המערכת מפעילה את תת-המערכת שלו עם המטריצה והווקטור הרלוונטיים. תת-המערכת מחזירה את החלטתה הסופית: Regime\_Decision (למשל, "מגמה שורית חזקה").
  + שאר הסוכנים (Agent\_30m, Agent\_5m) מחזיקים כעת את ה"דעה" שלהם המיוצגת בווקטורים שלהם.
* **3.3. בדיקת סינרגיה והחלטת כניסה ראשונית:**
  + רשת המדיניות המשותפת מקבלת כקלט את Vector\_30m, Vector\_5m, ואת Regime\_Decision כהקשר.
  + היא בודקת אם התנאים בשלים לאחת מ-4 סינרגיות הכניסה.
  + **אם לא,** הלולאה מסתיימת. המערכת חוזרת להמתין לעדכון הנתונים הבא.
  + **אם כן,** המערכת ממשיכה לשלב הבא ומפעילה את המומחה האחרון.
* **3.4. הפעלת סוכן ניהול הסיכונים (Agent\_Risk):**
  + המערכת קוראת לפונקציה של Agent\_Risk, ומעבירה לו את פרטי הסינרגיה שהתגלתה ואת מצב החשבון העדכני.
  + Agent\_Risk מחזיר את המלצתו: Risk\_Proposal\_Vector.
* **3.5. החלטה סופית באמצעות סופרפוזיציה:**
  + כעת יש לנו את כל ארבע הדעות: הדעה מ-Agent\_30m/5m/Regime (שהובילה להחלטת הכניסה), והדעה של Agent\_Risk.
  + מנגנון ההחלטה הסופי (שכבת Attention או סוכן-על) מקבל את כל הווקטורים האלה כקלט.
  + הוא מבצע את השקלול הסופי ומקבל את ההחלטה המערכתית: **EXECUTE** או **REJECT**.

##### **שלב 4: ביצוע הפעולה**

* **4.1. שליחת פקודה:** אם ההחלטה היא EXECUTE, המערכת קוראת ל-ExecutionHandler ומעבירה לו את פרטי העסקה מתוך Risk\_Proposal\_Vector.
* **4.2. המתנה:** המערכת חוזרת למצב האזנה, ממתינה לעדכון הנתונים הבא כדי להתחיל את הלולאה מחדש.

#### **דוגמה קונקרטית עם זמנים**

בוא נראה איך זה עובד על שעון:

* **10:29:59:** המערכת במצב המתנה.
* **10:30:00:** מגיע טיק אחרון. BarGenerator מזהה סגירה של נר 5 דקות וגם נר 30 דקות. הוא יורה שני אירועי NEW\_BAR.
* **10:30:00.010 (10 אלפיות השנייה):** IndicatorEngine תופס את האירועים, מחשב את כל האינדיקטורים ל-5 ו-30 דקות ומעדכן את ה-Feature Store.
* **10:30:00.020:** ה-MatrixAssemblers מופעלים ומעדכנים את כל מטריצות ה-N x F.
* **10:30:00.025:** לולאת ה-MARL הראשית מתחילה.
* **10:30:00.050:** כל המטריצות הומרו לווקטורים. הסוכנים מקבלים את המצב המעודכן.
* **10:30:00.060:** המדיניות המשותפת מזהה סינרגיה.
* **10:30:00.080:** סוכן הסיכונים מופעל ומחזיר המלצה.
* **10:30:00.090:** מנגנון הסופרפוזיציה מקבל החלטה סופית: EXECUTE.
* **10:30:00.100:** ה-ExecutionHandler שולח את הפקודה לברוקר.

**כל התהליך, מרגע סגירת הנר ועד שליחת הפקודה, חייב להסתיים בתוך שבריר שנייה, וזו הסיבה שנשתמש ב-TensorRT כדי להאיץ את שלב 3.1 ו-3.5.**

זוהי התוכנית המפורטת לאיך החלקים מתחברים למערכת אחת. היא מבוססת על הפרדת אחריות, זרימת נתונים לוגית, וממשקים ברורים, מה שהופך אותה לרובסטית, ניתנת לבדיקה, וחשוב מכל - זהה לחלוטין באופן פעולתה בין אימון למסחר חי.