GRID TRADING MASTERY · PART 7B

Grid-Framework + Indicator Execution Styles

Grid เป็น Framework (Zone/Step/Capital/Risk) · Indicator เป็น Execution Engine

4 Execution Styles: Mean-Reversion · Trend · Volume · Composite

การเลือก Style ตาม Market Condition

แก่นของ PART นี้

"Grid" ไม่ใช่ strategy เดียว — มันคือ **framework** ที่กำหนด zone, step, capital และ risk — Indicator เป็น "execution engine" ที่ตัดสินใจว่าจะใช้ grid framework นั้น *อย่างไร* ในแต่ละช่วงเวลา ทั้ง 4 styles ใช้ grid math เหมือนกัน แต่ execution logic ต่างกัน

```
Grid Math (zone/step/Q/risk) = คงที่ · Execution Style =
เปลี่ยนตาม regime
```

7B.1 Grid เป็น Framework — ไม่ใช่ Strategy

ความเข้าใจผิดทั่วไป: "Grid trading" = strategy หนึ่ง ที่มี rule ชัดเจนว่า "ซื้อทุก $2k ลง ขายทุก $2k ขึ้น"

ความจริง: Grid คือ **framework สำหรับ organize capital และ risk** — ส่วน execution (ว่าจะ deploy เมื่อไหร่, ขนาดไหน, เพิ่ม/ลดอย่างไร) ขึ้นอยู่กับ indicator execution style

GRID FRAMEWORK

(structure — คำนวณครั้งเดียว ไม่เปลี่ยน)

[Z] Zone [Low, High]
Donchian(20d) + Hurst filter

[S] Step Size
ATR × k (adaptive)

[K] Capital per Level
Kelly f* × capital / N

[R] Risk Budget
Max DD = 8% total capital

[N] N Orders
= (Zone High – Zone Low) / Step

Grid math
defines WHAT

⇄

Indicator
decides WHEN

INDICATOR EXECUTION

(timing — เปลี่ยนตาม regime)

Style 1: Mean-Reversion
RSI extreme + BB near edge → deploy

Style 2: Trend-Adaptive
H rising → step ใหญ่ขึ้น, Q ลด

Style 3: Volume-Adaptive
Volume สูง → Q เพิ่ม 30%

Style 4: Composite Score
Score 0–100 → deploy % และ size mult

ผลลัพธ์ที่ได้

Framework-only: Sharpe 0.65 · DD –4.2% + ADX filter: Sharpe 0.72 · DD –2.8% improved

P&L รวมใกล้เคียงกัน แต่ risk-adjusted return ดีขึ้น — Framework คือ alpha หลัก, Indicator ปรับ Sharpe

ซ้าย: Framework กำหนดโครงสร้าง (zone/step/capital/risk) · ขวา: 4 Execution Styles ใช้ framework เดียวกัน ได้ผลต่างกัน

```
Grid Framework Components (คงที่ ไม่เปลี่ยน):
  1. Zone: [zone_low, zone_high] — จาก Donchian/Bootstrap (Part 3)
  2. Step: grid spacing — จาก ATR/IV (Part 3, 7)
  3. Capital: grid_capital, Q per level — จาก Kelly (Part 5)
  4. Risk: DD halt, position limits — จาก Part 5

Indicator Execution Engine (เปลี่ยนตาม style/regime):
```

```
- เมื่อไหร่ deploy capital (timing)
- ขนาดเท่าไหร่ต่อ level (size)
- เพิ่ม Q ลด Q ตาม signal (adaptive)
- Stop/resume ตาม regime (state machine)
```

7B.2 Style 1 — Mean-Reversion Execution

Style 1: Mean-Reversion — เหมาะกับ H < 0.50, RSI extreme, BB near edge

Deploy ทุก level เท่ากัน, เน้น cashflow ทุกรอบ

```
class MeanReversionExecution:
    """
    Style 1: Deploy grid ทั้งหมดเมื่อ mean-reversion signal ชัด
    เน้น: ทุก level fill บ่อย, cashflow สม่ำเสมอ
    """
    def __init__(self, grid_framework):
        self.grid = grid_framework

    def should_deploy(self, indicators):
        score, _ = composite_grid_score(**indicators)
        return score > 60  # deploy เมื่อ score > 60

    def get_level_sizes(self, n_levels):
        # Equal weight ทุก level
        return [self.grid.base_q] * n_levels

    def should_adjust_size(self, indicators):
        # ไม่ปรับ size — คงที่ตาม Kelly
        return 1.0

    def get_execution_params(self, indicators):
        deploy = self.should_deploy(indicators)
        sizes = self.get_level_sizes(self.grid.n_levels)
        return {"deploy": deploy, "sizes": sizes, "style":
"mean_reversion"}

เหมาะกับ: Beginner → Advanced
ข้อดี: ง่าย, predictable, cashflow สม่ำเสมอ
ข้อเสีย: ไม่ adapt ตาม volatility regime
```

7B.3 Style 2 — Trend-Adaptive Execution

Style 2: Trend-Adaptive — ปรับ Step และ Q ตาม Hurst + ADX

ใช้ Step Pyramid (Part 2) อัตโนมัติ เพิ่ม step เมื่อ H สูง

```
class TrendAdaptiveExecution:
    """
    Style 2: ปรับ step + size ตาม trend strength
    H ต่ำ = step เล็ก Q เต็ม | H สูง = step ใหญ่ Q ลด
    """
    def get_adaptive_step(self, base_step, hurst, adx):
        # Step Pyramid (Part 2)
        hurst_mult = 1 + max(0, (hurst - 0.45) / 0.15) * 0.5
        adx_mult   = 1 + max(0, (adx - 20) / 30) * 0.3
        return base_step * hurst_mult * adx_mult

    def get_adaptive_size_mult(self, hurst, adx):
        # ลด Q เมื่อ trending มากขึ้น
        if hurst > 0.60 or adx > 35:
            return 0.4
        elif hurst > 0.55 or adx > 28:
            return 0.7
        return 1.0

    def get_execution_params(self, indicators):
        hurst = indicators["hurst30"]
        adx   = indicators["adx14"]

        step = self.get_adaptive_step(self.grid.base_step, hurst, adx)
        size_mult = self.get_adaptive_size_mult(hurst, adx)
        sizes = [self.grid.base_q * size_mult] * self.grid.n_levels

        return {
            "deploy": True,  # ไม่หยุด เพียงแต่ปรับ
            "step": step,
            "sizes": sizes,
            "style": "trend_adaptive"
        }
```

```
เหมาะกับ: ผู้เข้าใจ Hurst + Step Pyramid
ข้อดี: ไม่ต้อง pause grid เมื่อ trend เริ่ม → ลดทอน trend ผ่าน step ใหญ่
ข้อเสีย: ถ้า trend แรงมาก (H>0.65) ก็ยังขาดทุน unrealized
```

7B.4 Style 3 — Volume-Adaptive Execution

Style 3: Volume-Adaptive — เพิ่ม Q เมื่อ volume สูง (liquidity ดี), ลด Q เมื่อ volume ต่ำ

เน้น execution quality ใน high-liquidity windows

```
class VolumeAdaptiveExecution:
    """
    Style 3: ปรับ Q ตาม volume profile
    Volume สูง = fill ง่าย, spread แคบ → เพิ่ม Q
    Volume ต่ำ = fill ยาก, spread กว้าง → ลด Q
    """
    def __init__(self, grid_framework, volume_history=None):
        self.grid = grid_framework
        volume_history = volume_history or []
        self.avg_volume = sum(volume_history[-20:]) / 20 if
volume_history else 0
        self.volume_by_hour =
self._compute_hourly_profile(volume_history) if volume_history else {}

    def _compute_hourly_profile(self, volume_history):
        # คำนวณ avg volume ต่อชั่วโมง (0-23)
        hourly = {}
        for vol, timestamp in volume_history:
            hour = timestamp.hour
            hourly.setdefault(hour, []).append(vol)
        return {h: sum(v)/len(v) for h, v in hourly.items()}

    def get_size_multiplier(self, current_volume, current_hour):
        # Volume ratio vs historical average
        vol_ratio = current_volume / self.avg_volume

        # Hourly profile bonus
        hourly_avg = self.volume_by_hour.get(current_hour,
self.avg_volume)
        hourly_ratio = hourly_avg / self.avg_volume

        if vol_ratio > 1.5 and hourly_ratio > 1.2:
            return 1.3  # เพิ่ม 30% ในช่วง high-liquidity
```

```
        elif vol_ratio < 0.5 or hourly_ratio < 0.6:
            return 0.6  # ลด 40% ในช่วง low-liquidity
        return 1.0

# ช่วงเวลา High Volume สำหรับ BTC:
# 13:00–15:00 UTC (Europe afternoon + US morning overlap)
# 20:00–22:00 UTC (US afternoon peak)
# หลีกเลี่ยง: 03:00–06:00 UTC (Asia night, US sleep)
```

7B.5 Style 4 — Composite Score Execution

Style 4: Composite Score — รวม RSI + BB + Volume + Hurst เป็น score 0–100

ใช้ score กำหนด deploy % และ size multiplier (จาก Part 3B)

```
class CompositeScoreExecution:
    """
    Style 4: Composite Score จาก Part 3B กำหนดทุกอย่าง
    """
    def get_execution_params(self, indicators):
        score, sub_scores = composite_grid_score(**indicators)

        # Deploy percentage
        if score >= 75:
            deploy_pct = 1.00
        elif score >= 55:
            deploy_pct = 0.60
        elif score >= 35:
            deploy_pct = 0.30
        else:
            deploy_pct = 0.00

        # Size multiplier (สูงกว่าก็ deploy per level มากขึ้น)
        size_mult = 0.5 + (score / 100) * 0.5  # 0.5× ถึง 1.0×

        # Level allocation (emphasis on most signal levels)
        rsi_score = sub_scores["rsi"]
        if rsi_score > 70:  # RSI extreme → เน้น lower levels
            level_weights = "bottom_heavy"
        elif sub_scores["bb"] > 70:  # Near BB → เน้น near-edge
            level_weights = "bell"
        else:
            level_weights = "equal"

        sizes = self.grid.allocate_by_weight(level_weights, size_mult)

        return {
            "deploy": deploy_pct > 0,
```

```
            "deploy_pct": deploy_pct,
            "sizes": sizes,
            "size_mult": size_mult,
            "score": score,
            "style": "composite"
        }

เหมาะกับ: Quant / systematic trader
ข้อดี: รวม signal ทุกตัว ไม่มี blind spot
ข้อเสีย: ซับซ้อน ต้อง calibrate weights
```

7B.6 เลือก Execution Style ตาม Trader Profile

Style	Trader Profile	Time Commitment	Key Metric	Expected Return
Mean-Reversion	Beginner / Long-term	ต่ำ (weekly check)	Cashflow/day	0.20–0.35%/วัน
Trend-Adaptive	Intermediate	กลาง (daily check)	Cashflow + less DD	0.18–0.30%/วัน
Volume-Adaptive	Active Trader	สูง (hourly)	Execution quality	0.22–0.38%/วัน
Composite Score	Quant / Advanced	Automated (real-time)	Risk-adjusted return	0.25–0.45%/วัน

Style ไม่ใช่สิ่งที่เลือกครั้งเดียว

ใน practice: ใช้ Mean-Reversion เป็น baseline | เพิ่ม Trend-Adaptive เมื่อ Hurst เริ่มขึ้น | เพิ่ม Volume-Adaptive เมื่อเป็น professional automated grid

Composite Score เป็น superset — รวมทุก style ไว้ด้วยกัน แต่ต้องการ infrastructure ที่ดีกว่า

7B.7 Unified Grid Controller — รวมทุก Style

```
class UnifiedGridController:
    """
    Grid controller ที่รองรับทุก execution style
    ใช้ config เพื่อเลือก style
    """
    STYLES = {
        "mean_reversion": MeanReversionExecution,
        "trend_adaptive": TrendAdaptiveExecution,
        "volume_adaptive": VolumeAdaptiveExecution,
        "composite":      CompositeScoreExecution,
    }

    def __init__(self, grid_framework, style="mean_reversion"):
        self.grid = grid_framework
        self.execution = self.STYLES[style](grid_framework)
        self.state_machine = GridStateMachine()  # Part 4
        self.dd_monitor = DrawdownMonitor()      # Part 5

    def run_cycle(self, market_data):
        """
        ทำงาน 1 รอบ (ทุก 1 ชั่วโมง):
        1. ตรวจ DD halt
        2. ตรวจ regime state
        3. รับ execution params จาก style
        4. Execute orders
        """
        # Step 1: DD Check (Part 5)
        dd_status, dd_value, _ = self.dd_monitor.check(
            market_data["portfolio_history"]
        )
        if dd_status == "HALT":
            return {"action": "HALT", "reason": f"DD={dd_value:.1%}"}

        # Step 2: Regime Check (Part 4)
        state, signals = self.state_machine.evaluate(
            market_data["hurst30"],
            market_data["adx14"],
            market_data["bb_width"]
```

```
        )
        if state == "GRID_OFF":
            return {"action": "PAUSE", "state": state}

        # Step 3: Execution Style
        exec_params = self.execution.get_execution_params(market_data)

        # Merge regime multiplier
        size_mult = self.state_machine.get_size_multiplier()
        exec_params["sizes"] = [q * size_mult for q in
exec_params["sizes"]]

        # Step 4: Place orders
        orders = self.grid.create_orders(exec_params)
        return {"action": "EXECUTE", "orders": orders, "params":
exec_params}
```

7B.8 แบบฝึกหัด

แบบฝึกหัด Part 7B

▶ ข้อ 1: เหตุใด Grid Framework (zone/step/risk) ถึงไม่ใช่ส่วนที่ "ปรับตาม market" แต่ Execution Style ควรเปลี่ยนได้?

▶ ข้อ 2: Composite Score = 48 · Style = Mean-Reversion — deploy กี่เปอร์เซ็นต์? Style = Composite — deploy กี่เปอร์เซ็นต์?

▶ ข้อ 3: Volume-Adaptive Style ทำไมถึงเหมาะกับ "Active Trader" มากกว่า Beginner?

▶ ข้อ 4: UnifiedGridController ตรวจ DD Halt ก่อน Regime State — ทำไมลำดับนี้สำคัญ?