```
// This Pine Script® code is subject to the terms of the Mozilla Public License 2.0 at https://mozilla.org/MPL/2.0/

//@version=6

indicator("MFB Key Level Response", "MFB Key Level Response", overlay = true, max_lines_count = 200, max_labels_count = 200, max_boxes_count = 500)


// --- Constants ---

const string TIMEZONE = "America/New_York"

const color BULL_COLOR = #089981

const color BEAR_COLOR = #f23645

const color NEUTRAL_COLOR = #5b9cf6

const color PANEL_BACKGROUND = #161616

const int MAX_FOOTPRINT_HEATMAP_BOXES = 450


// --- Inputs ---

string toleranceModeInput = input.string("Ticks", "Tolerance measured in", options = ["Ticks", "ATR"], group = "Key Levels", tooltip = "Choose a fixed tick tolerance or an ATR-scaled tolerance for key-level interaction.")

int toleranceTicksInput = input.int(4, "Touch tolerance (ticks)", minval = 0, maxval = 1000, group = "Key Levels", tooltip = "Distance from a key level that counts as a touch when tick mode is selected.")

float toleranceAtrInput = input.float(0.10, "Touch tolerance (ATR)", minval = 0.0, maxval = 5.0, step = 0.05, group = "Key Levels", tooltip = "Distance from a key level that counts as a touch when ATR mode is selected.")
```

```
bool useDailyLevelsInput = input.bool(true, "Previous day high / low", group = "Key Levels", tooltip = "Include the prior completed daily candle's high and low.")

bool useWeeklyLevelsInput = input.bool(true, "Previous week high / low", group = "Key Levels", tooltip = "Include the prior completed weekly candle's high and low.")

bool useAsiaLevelsInput = input.bool(true, "Completed Asia high / low", group = "Key Levels", tooltip = "Include the completed 18:00–00:00 New York session range.")

bool useLondonLevelsInput = input.bool(true, "Completed London high / low", group = "Key Levels", tooltip = "Include the completed 00:00–06:00 New York session range.")

bool useOvernightLevelsInput = input.bool(true, "Completed overnight high / low", group = "Key Levels", tooltip = "Include the completed 18:00–09:30 New York overnight range.")

bool useInitialBalanceInput = input.bool(true, "Completed initial balance high / low", group = "Key Levels", tooltip = "Include the completed 09:30–10:30 New York initial balance.")

float manualPocInput = input.float(0.0, "Manual prior-session POC (0 = off)", minval = 0.0, group = "Manual Profile Levels", tooltip = "Optional manually supplied POC. This is not read from another indicator or represented as an exact profile calculation.")

float manualVahInput = input.float(0.0, "Manual prior-session VAH (0 = off)", minval = 0.0, group = "Manual Profile Levels", tooltip = "Optional manually supplied value-area high. Enter 0 to disable.")

float manualValInput = input.float(0.0, "Manual prior-session VAL (0 = off)", minval = 0.0, group = "Manual Profile Levels", tooltip = "Optional manually supplied value-area low. Enter 0 to disable.")
```

float manualHvnOneInput = input.float(0.0, "Manual HVN 1 (0 = off)", minval = 0.0, group = "Manual Profile Levels", tooltip = "Optional manually supplied high-volume node. Enter 0 to disable.")

float manualHvnTwoInput = input.float(0.0, "Manual HVN 2 (0 = off)", minval = 0.0, group = "Manual Profile Levels", tooltip = "Optional second manually supplied high-volume node. Enter 0 to disable.")

int trendWindowInput = input.int(12, "Completed 5-minute trend bars", minval = 4, maxval = 60, group = "Trend Context", tooltip = "Number of completed 5-minute bars used for net move and directional efficiency.")

int pivotLengthInput = input.int(2, "Confirmed swing pivot length", minval = 1, maxval = 10, group = "Trend Context", tooltip = "Bars on each side required to confirm a 5-minute swing high or low; pivots are never used before confirmation.")

float efficiencyThresholdInput = input.float(35.0, "Minimum directional efficiency (%)", minval = 0.0, maxval = 100.0, step = 1.0, group = "Trend Context", tooltip = "Minimum absolute net movement divided by total close-to-close travel over the trend window.")

int structurePivotLengthInput = input.int(3, "CISD swing pivot length", minval = 1, maxval = 10, group = "Structural Evidence", tooltip = "Confirmed chart-timeframe swing length used by the ported MFB CISD/CHoCH evidence logic.")

int minimumSweepTicksInput = input.int(2, "Minimum failed-auction sweep (ticks)", minval = 1, maxval = 1000, group = "Structural Evidence", tooltip = "Price must pierce beyond the key level by at least this distance before a reclaim can become a failed-auction candidate.")

int confirmationWindowFiveMinuteInput = input.int(6, "Candidate confirmation window (5m bars)", minval = 1, maxval = 30, group = "Structural Evidence", tooltip = "Maximum time after a candidate begins to receive confirmed structural evidence, expressed in five-minute bars.")

int fvgMaximumAgeInput = input.int(100, "FVG-123 maximum age (chart bars)", minval = 1, maxval = 1000, group = "Structural Evidence", tooltip = "Maximum age of the three-candle FVG pattern when its later retest-and-breakout confirms.")

float fvgMinimumWidthTicksInput = input.float(0.0, "FVG-123 minimum width (ticks)", minval = 0.0, maxval = 1000.0, group = "Structural Evidence", tooltip = "Optional minimum three-candle gap width; zero accepts any positive-width gap.")

float rewardRiskInput = input.float(2.0, "Guide target (R)", minval = 0.25, maxval = 10.0, step = 0.25, group = "Signal Guides", tooltip = "Target guide distance as a multiple of the entry-to-stop risk.")

int guideProjectionInput = input.int(15, "Entry guide projection (bars)", minval = 1, maxval = 100, group = "Signal Guides", tooltip = "Number of chart bars to project entry, stop, and target guides after a confirmed signal.")

int maximumGuideSetsInput = input.int(8, "Maximum recent guide sets", minval = 1, maxval = 20, group = "Signal Guides", tooltip = "Limits retained entry/stop/target line sets to keep the chart uncluttered.")

bool showSignalMarkersInput = input.bool(true, "Show confirmed signal markers", group = "Display", tooltip = "Show one marker for each confirmed failed-auction or continuation event.")

bool showSignalPriceLabelsInput = input.bool(false, "Show exact signal entry price", group = "Display", tooltip = "Label each confirmed signal at its exact signal-bar close, formatted to the symbol's minimum tick.")

bool showTradeGuidePriceTagsInput = input.bool(false, "Show entry / stop / target price tags", group = "Display", tooltip = "Show tick-precise labels at the ends of projected entry, stop, and target lines.")

bool showTradeGuidesInput = input.bool(true, "Show entry / stop / target guides", group = "Display", tooltip = "Draw one projected entry, stop, and configured R-multiple target set for each confirmed signal.")

```
bool showDashboardInput = input.bool(true, "Show status dashboard", group = "Display", tooltip = "Show trend state, active level event, most recent signal, and Stage 1 scope.")

color bullishColorInput = input.color(BULL_COLOR, "Bullish", group = "Style", tooltip = "Color for bullish signals and targets.")

color bearishColorInput = input.color(BEAR_COLOR, "Bearish", group = "Style", tooltip = "Color for bearish signals and stop guides.")

color entryColorInput = input.color(NEUTRAL_COLOR, "Entry / neutral", group = "Style", tooltip = "Color for entry and neutral status visuals.")

bool showFootprintHeatmapInput = input.bool(true, "Show key-level footprint heatmap", group = "Footprint Heatmap", tooltip = "Draw actual footprint rows near the active key-level interaction. The heatmap is visual context only and does not qualify signals.")

int footprintTicksPerRowInput = input.int(4, "Ticks per footprint row", minval = 1, maxval = 100, group = "Footprint Heatmap", tooltip = "Price height of each footprint row, expressed in symbol ticks.")

int footprintImbalancePercentInput = input.int(300, "Imbalance threshold (%)", minval = 1, maxval = 1000, group = "Footprint Heatmap", tooltip = "Buy or sell imbalance threshold used to mark footprint rows.")

int heatmapRadiusTicksInput = input.int(24, "Radius around active level (ticks)", minval = 1, maxval = 1000, group = "Footprint Heatmap", tooltip = "Show only footprint rows within this distance above or below the active key level.")

int heatmapBarsInput = input.int(20, "Maximum bars per interaction", minval = 1, maxval = 100, group = "Footprint Heatmap", tooltip = "Limit how many bars from one active key-level interaction receive heatmap boxes.")


// --- Functions ---
```

```
trackCompletedSession(string sessionSpec) =>
    bool inSession = not na(time(timeframe.period, sessionSpec, TIMEZONE))
    bool sessionStarts = inSession and not inSession[1]
    bool sessionEnds = not inSession and inSession[1]
    var float buildingHigh = na
    var float buildingLow = na
    var float completedHigh = na
    var float completedLow = na
    if sessionStarts
        buildingHigh := high
        buildingLow := low
    else if inSession
        buildingHigh := math.max(nz(buildingHigh, high), high)
        buildingLow := math.min(nz(buildingLow, low), low)
    if sessionEnds
        completedHigh := buildingHigh
        completedLow := buildingLow
    [completedHigh, completedLow]


levelIsTouched(float levelPrice, float tolerancePrice) =>
    not na(levelPrice) and levelPrice > 0.0 and high >= levelPrice -
tolerancePrice and low <= levelPrice + tolerancePrice
```

considerTouchedLevel(float bestPrice, string bestName, float candidatePrice, string candidateName, float tolerancePrice) =>

  bool candidateTouched = levelIsTouched(candidatePrice, tolerancePrice)

  float candidateDistance = math.abs(close - candidatePrice)

  float bestDistance = na(bestPrice) ? na : math.abs(close - bestPrice)

  bool candidateWins = candidateTouched and (na(bestPrice) or candidateDistance < bestDistance - syminfo.mintick * 0.5)

  bool tied = candidateTouched and not na(bestPrice) and math.abs(candidateDistance - bestDistance) <= syminfo.mintick * 0.5

  float chosenPrice = candidateWins ? candidatePrice : bestPrice

  string chosenName = candidateWins ? candidateName : tied and not str.contains(bestName, candidateName) ? bestName + " / " + candidateName : bestName

  [chosenPrice, chosenName]

// --- Automatic key levels ---

[previousDayHigh, previousDayLow] = request.security(syminfo.tickerid, "D", [high[1], low[1]], gaps = barmerge.gaps_off, lookahead = barmerge.lookahead_on)

[previousWeekHigh, previousWeekLow] = request.security(syminfo.tickerid, "W", [high[1], low[1]], gaps = barmerge.gaps_off, lookahead = barmerge.lookahead_on)

[asiaHigh, asiaLow] = trackCompletedSession("1800-0000:1234567")

[londonHigh, londonLow] = trackCompletedSession("0000-0600:1234567")

[overnightHigh, overnightLow] = trackCompletedSession("1800-0930:1234567")

```
[initialBalanceHigh, initialBalanceLow] = trackCompletedSession("0930-1030:23456")

float atrForTolerance = ta.atr(14)

float tolerancePrice = toleranceModeInput == "Ticks" ? toleranceTicksInput * syminfo.mintick : math.max(nz(atrForTolerance, syminfo.mintick * 10.0) * toleranceAtrInput, syminfo.mintick)


// --- Select the closest level touched by this bar ---

[level1, name1] = considerTouchedLevel(na, "", useDailyLevelsInput ? previousDayHigh : na, "PDH", tolerancePrice)

[level2, name2] = considerTouchedLevel(level1, name1, useDailyLevelsInput ? previousDayLow : na, "PDL", tolerancePrice)

[level3, name3] = considerTouchedLevel(level2, name2, useWeeklyLevelsInput ? previousWeekHigh : na, "PWH", tolerancePrice)

[level4, name4] = considerTouchedLevel(level3, name3, useWeeklyLevelsInput ? previousWeekLow : na, "PWL", tolerancePrice)

[level5, name5] = considerTouchedLevel(level4, name4, useAsiaLevelsInput ? asiaHigh : na, "Asia High", tolerancePrice)

[level6, name6] = considerTouchedLevel(level5, name5, useAsiaLevelsInput ? asiaLow : na, "Asia Low", tolerancePrice)

[level7, name7] = considerTouchedLevel(level6, name6, useLondonLevelsInput ? londonHigh : na, "London High", tolerancePrice)

[level8, name8] = considerTouchedLevel(level7, name7, useLondonLevelsInput ? londonLow : na, "London Low", tolerancePrice)

[level9, name9] = considerTouchedLevel(level8, name8, useOvernightLevelsInput ? overnightHigh : na, "Overnight High", tolerancePrice)
```

```
[level10, name10] = considerTouchedLevel(level9, name9,
useOvernightLevelsInput ? overnightLow : na, "Overnight Low",
tolerancePrice)

[level11, name11] = considerTouchedLevel(level10, name10,
useInitialBalanceInput ? initialBalanceHigh : na, "IB High", tolerancePrice)

[level12, name12] = considerTouchedLevel(level11, name11,
useInitialBalanceInput ? initialBalanceLow : na, "IB Low", tolerancePrice)

[level13, name13] = considerTouchedLevel(level12, name12, manualPocInput
> 0.0 ? manualPocInput : na, "Manual POC", tolerancePrice)

[level14, name14] = considerTouchedLevel(level13, name13, manualVahInput
> 0.0 ? manualVahInput : na, "Manual VAH", tolerancePrice)

[level15, name15] = considerTouchedLevel(level14, name14, manualValInput
> 0.0 ? manualValInput : na, "Manual VAL", tolerancePrice)

[level16, name16] = considerTouchedLevel(level15, name15,
manualHvnOneInput > 0.0 ? manualHvnOneInput : na, "Manual HVN 1",
tolerancePrice)

[selectedLevel, selectedLevelName] = considerTouchedLevel(level16,
name16, manualHvnTwoInput > 0.0 ? manualHvnTwoInput : na, "Manual HVN
2", tolerancePrice)

bool keyLevelTouched = not na(selectedLevel)

footprint footprintData = request.footprint(footprintTicksPerRowInput, 70,
footprintImbalancePercentInput)

float footprintDelta = not na(footprintData) ? footprint.delta(footprintData) :
na

float footprintBuyVolume = not na(footprintData) ?
footprint.buy_volume(footprintData) : na

float footprintSellVolume = not na(footprintData) ?
footprint.sell_volume(footprintData) : na
```

```
// --- Confirmed 5-minute trend measurements ---

[closedFiveMinuteTime, fiveMinuteClose, fiveMinuteNetMove,
fiveMinuteAverageTravel, fiveMinutePivotHigh, fiveMinutePivotHighTime,
fiveMinutePivotLow, fiveMinutePivotLowTime] =
request.security(syminfo.tickerid, "5", [time[1], close[1], close[1] -
close[trendWindowInput + 1], ta.sma(math.abs(close - close[1]),
trendWindowInput)[1] * trendWindowInput, ta.pivothigh(high,
pivotLengthInput, pivotLengthInput)[1], not na(ta.pivothigh(high,
pivotLengthInput, pivotLengthInput)[1]) ? time[pivotLengthInput + 1] : na,
ta.pivotlow(low, pivotLengthInput, pivotLengthInput)[1], not
na(ta.pivotlow(low, pivotLengthInput, pivotLengthInput)[1]) ?
time[pivotLengthInput + 1] : na], gaps = barmerge.gaps_off, lookahead =
barmerge.lookahead_on)

float fiveMinuteEfficiency = not na(fiveMinuteAverageTravel) and
fiveMinuteAverageTravel > 0.0 ? math.abs(fiveMinuteNetMove) /
fiveMinuteAverageTravel * 100.0 : 0.0

bool newFiveMinuteSample = not na(closedFiveMinuteTime) and
ta.change(closedFiveMinuteTime) != 0


// Store only confirmed pivots and count consecutive higher-high / higher-low
or lower-high / lower-low sequences.

var float priorSwingHigh = na

var float latestSwingHigh = na

var int priorSwingHighTime = na

var int latestSwingHighTime = na

var int processedSwingHighTime = na

var int higherHighCount = 0
```

```
var int lowerHighCount = 0

var float priorSwingLow = na

var float latestSwingLow = na

var int priorSwingLowTime = na

var int latestSwingLowTime = na

var int processedSwingLowTime = na

var int higherLowCount = 0

var int lowerLowCount = 0


if barstate.isconfirmed and not na(fiveMinutePivotHigh) and
(na(processedSwingHighTime) or fiveMinutePivotHighTime !=
processedSwingHighTime)

   priorSwingHigh := latestSwingHigh

   priorSwingHighTime := latestSwingHighTime

   latestSwingHigh := fiveMinutePivotHigh

   latestSwingHighTime := fiveMinutePivotHighTime

   higherHighCount := not na(priorSwingHigh) and latestSwingHigh >
priorSwingHigh ? math.min(higherHighCount + 1, 10) : 0

   lowerHighCount := not na(priorSwingHigh) and latestSwingHigh <
priorSwingHigh ? math.min(lowerHighCount + 1, 10) : 0

   processedSwingHighTime := fiveMinutePivotHighTime


if barstate.isconfirmed and not na(fiveMinutePivotLow) and
(na(processedSwingLowTime) or fiveMinutePivotLowTime !=
processedSwingLowTime)
```

```
    priorSwingLow := latestSwingLow

    priorSwingLowTime := latestSwingLowTime

    latestSwingLow := fiveMinutePivotLow

    latestSwingLowTime := fiveMinutePivotLowTime

    higherLowCount := not na(priorSwingLow) and latestSwingLow >
priorSwingLow ? math.min(higherLowCount + 1, 10) : 0

    lowerLowCount := not na(priorSwingLow) and latestSwingLow <
priorSwingLow ? math.min(lowerLowCount + 1, 10) : 0

    processedSwingLowTime := fiveMinutePivotLowTime


int trendWindowMilliseconds = trendWindowInput * 5 * 60 * 1000

bool highPivotsRecent = not na(latestSwingHighTime) and not
na(closedFiveMinuteTime) and closedFiveMinuteTime - latestSwingHighTime
<= trendWindowMilliseconds

bool lowPivotsRecent = not na(latestSwingLowTime) and not
na(closedFiveMinuteTime) and closedFiveMinuteTime - latestSwingLowTime
<= trendWindowMilliseconds

bool bullishTrendCandidate = higherHighCount >= 2 and higherLowCount >= 2
and highPivotsRecent and lowPivotsRecent and fiveMinuteNetMove > 0.0 and
fiveMinuteEfficiency >= efficiencyThresholdInput

bool bearishTrendCandidate = lowerHighCount >= 2 and lowerLowCount >= 2
and highPivotsRecent and lowPivotsRecent and fiveMinuteNetMove < 0.0 and
fiveMinuteEfficiency >= efficiencyThresholdInput

var int establishedTrend = 0

if barstate.isconfirmed and newFiveMinuteSample

    if bullishTrendCandidate
```

```
    establishedTrend := 1
  else if bearishTrendCandidate
    establishedTrend := -1
  else if establishedTrend == 1 and not na(latestSwingLow) and
fiveMinuteClose < latestSwingLow
    establishedTrend := 0
  else if establishedTrend == -1 and not na(latestSwingHigh) and
fiveMinuteClose > latestSwingHigh
    establishedTrend := 0
string trendStatus = establishedTrend == 1 ? "Bullish" : establishedTrend == -1
? "Bearish" : "Neutral"


var string lastSignalClassification = "No confirmed setup"
var string lastSignalLevelName = ""
var string lastSignalEvidence = ""
var float lastSignalLevel = na
var float lastSignalEntry = na
var float lastSignalStop = na
var float lastSignalTarget = na
var int lastSignalBar = na
// --- Canonical MFB CISD / CHoCH evidence (confirmed chart-timeframe
bars) ---
float structuralPivotHigh = ta.pivothigh(high, structurePivotLengthInput,
structurePivotLengthInput)
```

```
float structuralPivotLow = ta.pivotlow(low, structurePivotLengthInput, structurePivotLengthInput)

var float lastStructuralSwingHigh = na

var float lastStructuralSwingLow = na

var float lastBearFvgTop = na

var float lastBullFvgBottom = na

if not na(structuralPivotHigh)

   lastStructuralSwingHigh := structuralPivotHigh

if not na(structuralPivotLow)

   lastStructuralSwingLow := structuralPivotLow

if bar_index >= 2 and high < low[2]

   lastBearFvgTop := low[2]

if bar_index >= 2 and low > high[2]

   lastBullFvgBottom := high[2]

bool bullishChochEvent = ta.crossover(close, lastStructuralSwingHigh)

bool bearishChochEvent = ta.crossunder(close, lastStructuralSwingLow)

bool bullishCisdEvent = ta.crossover(close, lastBearFvgTop)

bool bearishCisdEvent = ta.crossunder(close, lastBullFvgBottom)

bool bullishCisdEvidence = barstate.isconfirmed and (bullishChochEvent or bullishCisdEvent) and not na(lastStructuralSwingHigh) and not na(lastBearFvgTop) and close > lastStructuralSwingHigh and close > lastBearFvgTop

bool bearishCisdEvidence = barstate.isconfirmed and (bearishChochEvent or bearishCisdEvent) and not na(lastStructuralSwingLow) and not
```

```
na(lastBullFvgBottom) and close < lastStructuralSwingLow and close <
lastBullFvgBottom


// --- Single-event auction state machine ---

int confirmationChartBars = math.max(1,
int(math.ceil(confirmationWindowFiveMinuteInput * 300.0 /
math.max(nz(timeframe.in_seconds(), 60), 60))))

float minimumSweepDistance = minimumSweepTicksInput * syminfo.mintick

var int eventState = 0

var int eventClass = 0

var int eventDirection = 0

var bool conflictBullPossible = false

var bool conflictBearPossible = false

var bool conflictBullFailedAuction = false

var bool conflictBearFailedAuction = false

var float eventLevel = na

var string eventLevelName = ""

var int eventStartBar = na

var int candidateBar = na

var int trendAtTouch = 0

var float eventExtremeHigh = na

var float eventExtremeLow = na

var float lastConsumedLevel = na

var bool eventArmed = true
```

```
var int lastKeyLevelTouchBar = na
var float lastKeyLevelTouchPrice = na

bool bullishFailedAuctionSignal = false
bool bearishFailedAuctionSignal = false
bool bullishContinuationSignal = false
bool bearishContinuationSignal = false
bool bullishFvg123Event = false
bool bearishFvg123Event = false

if barstate.isconfirmed and not na(lastConsumedLevel) and math.abs(close - lastConsumedLevel) > tolerancePrice * 2.0
    eventArmed := true
if barstate.isconfirmed and timeframe.isminutes and timeframe.multiplier <= 5 and eventState == 0 and eventArmed and keyLevelTouched
    eventState := 1
    eventClass := 0
    eventDirection := 0
    conflictBullPossible := false
    conflictBearPossible := false
    conflictBullFailedAuction := false
    conflictBearFailedAuction := false
    eventLevel := selectedLevel
    eventLevelName := selectedLevelName
```

```
    eventStartBar := bar_index
    candidateBar := na
    trendAtTouch := establishedTrend
    eventExtremeHigh := high
    eventExtremeLow := low
    lastConsumedLevel := selectedLevel
    eventArmed := false
if barstate.isconfirmed and eventState > 0
    eventExtremeHigh := math.max(nz(eventExtremeHigh, high), high)
    eventExtremeLow := math.min(nz(eventExtremeLow, low), low)
    if keyLevelTouched
        lastKeyLevelTouchBar := bar_index
        lastKeyLevelTouchPrice := selectedLevel


// --- Ported FVG-123 sequence: three-candle gap, later retest, then close beyond the gap ---
var float bullishFvgTop = na
var float bullishFvgBottom = na
var int bullishFvgCreatedBar = na
var int bullishFvgTouchBar = na
var int bullishFvgKeyTouchBar = na
var float bullishFvgKeyLevel = na
var float bearishFvgTop = na
```

```
var float bearishFvgBottom = na
var int bearishFvgCreatedBar = na
var int bearishFvgTouchBar = na
var int bearishFvgKeyTouchBar = na
var float bearishFvgKeyLevel = na
if barstate.isconfirmed
   if not na(bullishFvgTop)
      bool bullFvgExpired = bar_index - bullishFvgCreatedBar > fvgMaximumAgeInput
      bool bullFvgFilled = low <= bullishFvgBottom
      if bullFvgExpired or bullFvgFilled
         bullishFvgTop := na
         bullishFvgBottom := na
         bullishFvgCreatedBar := na
         bullishFvgTouchBar := na
         bullishFvgKeyTouchBar := na
         bullishFvgKeyLevel := na
      else if not na(bullishFvgTouchBar) and bar_index > bullishFvgTouchBar and close > bullishFvgTop
         bool bullFvgKeyValid = not na(bullishFvgKeyTouchBar) and bullishFvgKeyTouchBar < bullishFvgCreatedBar and bullishFvgCreatedBar - bullishFvgKeyTouchBar <= confirmationChartBars
         bool bullFvgSameEvent = eventState > 0 and not na(eventLevel) and not na(bullishFvgKeyLevel) and math.abs(eventLevel - bullishFvgKeyLevel) <= tolerancePrice
```

```
        bool bullFvgWidthValid = (bullishFvgTop - bullishFvgBottom) /
syminfo.mintick >= fvgMinimumWidthTicksInput

        bullishFvg123Event := bullFvgKeyValid and bullFvgSameEvent and
bullFvgWidthValid

        bullishFvgTop := na

        bullishFvgBottom := na

        bullishFvgCreatedBar := na

        bullishFvgTouchBar := na

        bullishFvgKeyTouchBar := na

        bullishFvgKeyLevel := na

    else if bar_index > bullishFvgCreatedBar and high >= bullishFvgBottom
and low <= bullishFvgTop and close >= bullishFvgBottom and
na(bullishFvgTouchBar)

        bullishFvgTouchBar := bar_index

  if not na(bearishFvgTop)

    bool bearFvgExpired = bar_index - bearishFvgCreatedBar >
fvgMaximumAgeInput

    bool bearFvgFilled = high >= bearishFvgTop

    if bearFvgExpired or bearFvgFilled

        bearishFvgTop := na

        bearishFvgBottom := na

        bearishFvgCreatedBar := na

        bearishFvgTouchBar := na

        bearishFvgKeyTouchBar := na

        bearishFvgKeyLevel := na
```

```
    else if not na(bearishFvgTouchBar) and bar_index > bearishFvgTouchBar and close < bearishFvgBottom

        bool bearFvgKeyValid = not na(bearishFvgKeyTouchBar) and bearishFvgKeyTouchBar < bearishFvgCreatedBar and bearishFvgCreatedBar - bearishFvgKeyTouchBar <= confirmationChartBars

        bool bearFvgSameEvent = eventState > 0 and not na(eventLevel) and not na(bearishFvgKeyLevel) and math.abs(eventLevel - bearishFvgKeyLevel) <= tolerancePrice

        bool bearFvgWidthValid = (bearishFvgTop - bearishFvgBottom) / syminfo.mintick >= fvgMinimumWidthTicksInput

        bearishFvg123Event := bearFvgKeyValid and bearFvgSameEvent and bearFvgWidthValid

        bearishFvgTop := na

        bearishFvgBottom := na

        bearishFvgCreatedBar := na

        bearishFvgTouchBar := na

        bearishFvgKeyTouchBar := na

        bearishFvgKeyLevel := na

    else if bar_index > bearishFvgCreatedBar and high >= bearishFvgBottom and low <= bearishFvgTop and close <= bearishFvgTop and na(bearishFvgTouchBar)

        bearishFvgTouchBar := bar_index

  if bar_index >= 2 and low > high[2] and (low - high[2]) / syminfo.mintick >= fvgMinimumWidthTicksInput

    bullishFvgTop := low

    bullishFvgBottom := high[2]
```

```
        bullishFvgCreatedBar := bar_index
        bullishFvgTouchBar := na
        bullishFvgKeyTouchBar := lastKeyLevelTouchBar
        bullishFvgKeyLevel := lastKeyLevelTouchPrice
    if bar_index >= 2 and high < low[2] and (low[2] - high) / syminfo.mintick >= fvgMinimumWidthTicksInput
        bearishFvgTop := low[2]
        bearishFvgBottom := high
        bearishFvgCreatedBar := bar_index
        bearishFvgTouchBar := na
        bearishFvgKeyTouchBar := lastKeyLevelTouchBar
        bearishFvgKeyLevel := lastKeyLevelTouchPrice


// Expire an interaction that never forms a valid candidate within the event window.
if barstate.isconfirmed and eventState == 1 and not na(eventStartBar) and bar_index - eventStartBar > confirmationChartBars
    eventState := 0
    eventClass := 0
    eventDirection := 0
    eventLevel := na
    eventLevelName := ""
    eventStartBar := na
    candidateBar := na
```

```
    trendAtTouch := 0
    eventExtremeHigh := na
    eventExtremeLow := na

// --- Candidate creation, expiry, invalidation, and one-shot confirmation ---
if barstate.isconfirmed and eventState == 1 and not na(eventLevel)
    bool bullishFailedCandidate = low < eventLevel - minimumSweepDistance and close > eventLevel
    bool bearishFailedCandidate = high > eventLevel + minimumSweepDistance and close < eventLevel
    bool bullishContinuationCandidate = trendAtTouch == 1 and close[1] > eventLevel and low <= eventLevel + tolerancePrice and close >= eventLevel
    bool bearishContinuationCandidate = trendAtTouch == -1 and close[1] < eventLevel and high >= eventLevel - tolerancePrice and close <= eventLevel
    bool bullishCandidate = bullishFailedCandidate or bullishContinuationCandidate
    bool bearishCandidate = bearishFailedCandidate or bearishContinuationCandidate
    if bullishCandidate and bearishCandidate
        eventState := 2
        eventClass := 3
        eventDirection := 0
        conflictBullPossible := true
        conflictBearPossible := true
        conflictBullFailedAuction := bullishFailedCandidate
```

```
        conflictBearFailedAuction := bearishFailedCandidate
        candidateBar := bar_index
    else if bullishCandidate
        eventState := 2
        eventClass := bullishFailedCandidate ? 1 : 2
        eventDirection := 1
        candidateBar := bar_index
    else if bearishCandidate
        eventState := 2
        eventClass := bearishFailedCandidate ? 1 : 2
        eventDirection := -1
        candidateBar := bar_index

if barstate.isconfirmed and eventState == 2 and not na(candidateBar)
    bool candidateExpired = bar_index - candidateBar > confirmationChartBars
    bool bullStructureNow = bullishCisdEvidence or bullishFvg123Event
    bool bearStructureNow = bearishCisdEvidence or bearishFvg123Event
    bool trendInvalidated = eventClass == 2 and ((eventDirection == 1 and
establishedTrend != 1) or (eventDirection == -1 and establishedTrend != -1))
    bool failedAuctionInvalidated = eventClass == 1 and ((eventDirection == 1
and close < eventLevel) or (eventDirection == -1 and close > eventLevel))
    bool conflictInvalidated = eventClass == 3 and ((close > eventLevel and
conflictBearPossible) or (close < eventLevel and conflictBullPossible))
```

```
    bool opposingStructure = (eventClass == 1 or eventClass == 2) ?
((eventDirection == 1 and bearStructureNow) or (eventDirection == -1 and
bullStructureNow)) : eventClass == 3 and ((conflictBullPossible and
bearStructureNow) or (conflictBearPossible and bullStructureNow))

    if candidateExpired or trendInvalidated or failedAuctionInvalidated or
conflictInvalidated or opposingStructure

        eventState := 0

        eventClass := 0

        eventDirection := 0

        eventLevel := na

        eventLevelName := ""

        eventStartBar := na

        candidateBar := na

        trendAtTouch := 0

        eventExtremeHigh := na

        eventExtremeLow := na

        conflictBullPossible := false

        conflictBearPossible := false

        conflictBullFailedAuction := false

        conflictBearFailedAuction := false

    else if bar_index > candidateBar

        int confirmedDirection = eventDirection

        int confirmedClass = eventClass

        if eventClass == 3 and bullStructureNow != bearStructureNow
```

```
    confirmedDirection := bullStructureNow and conflictBullPossible ? 1 :
bearStructureNow and conflictBearPossible ? -1 : 0

    confirmedClass := confirmedDirection == 1 ? (conflictBullFailedAuction
? 1 : 2) : confirmedDirection == -1 ? (conflictBearFailedAuction ? 1 : 2) : 3

  bool confirmedBull = confirmedDirection == 1 and bullStructureNow and
not bearStructureNow

  bool confirmedBear = confirmedDirection == -1 and bearStructureNow
and not bullStructureNow

  bool selectedCisd = confirmedBull ? bullishCisdEvidence : confirmedBear
? bearishCisdEvidence : false

  bool selectedFvg123 = confirmedBull ? bullishFvg123Event :
confirmedBear ? bearishFvg123Event : false

  string evidenceText = selectedCisd and selectedFvg123 ? "CISD + FVG-
123" : selectedCisd ? "CISD" : "FVG-123"

  if confirmedBull or confirmedBear

    float entryValue = close

    float stopValue = confirmedBull ? eventExtremeLow - syminfo.mintick :
eventExtremeHigh + syminfo.mintick

    float riskValue = confirmedBull ? entryValue - stopValue : stopValue -
entryValue

    if riskValue > 0.0

      bullishFailedAuctionSignal := confirmedBull and confirmedClass == 1

      bearishFailedAuctionSignal := confirmedBear and confirmedClass ==
1

      bullishContinuationSignal := confirmedBull and confirmedClass == 2

      bearishContinuationSignal := confirmedBear and confirmedClass == 2
```

```
            lastSignalLevelName := eventLevelName
            lastSignalLevel := eventLevel
            lastSignalEvidence := evidenceText
            lastSignalClassification := confirmedClass == 1 ? (confirmedBull ? "Bullish Failed Auction" : "Bearish Failed Auction") : (confirmedBull ? "Bullish Continuation" : "Bearish Continuation")
            lastSignalEntry := entryValue
            lastSignalStop := stopValue
            lastSignalTarget := confirmedBull ? entryValue + riskValue * rewardRiskInput : entryValue - riskValue * rewardRiskInput
            lastSignalBar := bar_index
        eventState := 0
        eventClass := 0
        eventDirection := 0
        eventLevel := na
        eventLevelName := ""
        eventStartBar := na
        candidateBar := na
        trendAtTouch := 0
        eventExtremeHigh := na
        eventExtremeLow := na
        conflictBullPossible := false
        conflictBearPossible := false
        conflictBullFailedAuction := false
```

```
    conflictBearFailedAuction := false


// --- Signal visuals: one marker and one guide set per resolved event ---
bool anyBullishSignal = bullishFailedAuctionSignal or
bullishContinuationSignal
bool anyBearishSignal = bearishFailedAuctionSignal or
bearishContinuationSignal
bool anyConfirmedSignal = anyBullishSignal or anyBearishSignal
var array<line> guideLines = array.new_line()
var array<label> guideLabels = array.new_label()
var array<label> signalPriceLabels = array.new_label()
var table statusTable = table.new(position.middle_right, 2, 7, bgcolor =
PANEL_BACKGROUND, frame_color = #2E2E2E, frame_width = 1,
border_color = #2E2E2E, border_width = 1)
var array<box> footprintHeatmapBoxes = array.new_box()


plotshape(showSignalMarkersInput and bullishFailedAuctionSignal, title =
"Bullish Failed Auction", style = shape.labelup, location = location.belowbar,
color = bullishColorInput, text = "Rev", textcolor = color.white, size = size.tiny)
plotshape(showSignalMarkersInput and bearishFailedAuctionSignal, title =
"Bearish Failed Auction", style = shape.labeldown, location =
location.abovebar, color = bearishColorInput, text = "Rev", textcolor =
color.white, size = size.tiny)
plotshape(showSignalMarkersInput and bullishContinuationSignal, title =
"Bullish Trend Continuation", style = shape.labelup, location =
location.belowbar, color = bullishColorInput, text = "CN", textcolor =
color.white, size = size.tiny)
```

```
plotshape(showSignalMarkersInput and bearishContinuationSignal, title =
"Bearish Trend Continuation", style = shape.labeldown, location =
location.abovebar, color = bearishColorInput, text = "CN", textcolor =
color.white, size = size.tiny)


if not showSignalPriceLabelsInput and array.size(signalPriceLabels) > 0
   while array.size(signalPriceLabels) > 0
      label staleSignalPriceLabel = array.pop(signalPriceLabels)
      label.delete(staleSignalPriceLabel)
if showSignalPriceLabelsInput and anyConfirmedSignal and not
na(lastSignalEntry)
   while array.size(signalPriceLabels) >= maximumGuideSetsInput
      label oldSignalPriceLabel = array.shift(signalPriceLabels)
      label.delete(oldSignalPriceLabel)
   string signalPricePrefix = bullishFailedAuctionSignal or
bearishFailedAuctionSignal ? "Rev Entry " : "CN Entry "
   color signalPriceColor = anyBullishSignal ? bullishColorInput :
bearishColorInput
   label exactSignalPriceLabel = label.new(bar_index, lastSignalEntry,
signalPricePrefix + str.tostring(lastSignalEntry, format.mintick), xloc =
xloc.bar_index, yloc = yloc.price, color = color.new(signalPriceColor, 10), style
= label.style_label_left, textcolor = color.white, size = size.tiny)
   array.push(signalPriceLabels, exactSignalPriceLabel)


if not showTradeGuidePriceTagsInput and array.size(guideLabels) > 0
   while array.size(guideLabels) > 0
```

```
        label staleGuideLabel = array.pop(guideLabels)
        label.delete(staleGuideLabel)
if showTradeGuidesInput and anyConfirmedSignal and not na(lastSignalEntry) and not na(lastSignalStop) and not na(lastSignalTarget)
    int guideLimit = maximumGuideSetsInput * 3
    while array.size(guideLines) + 3 > guideLimit
        line oldLine = array.shift(guideLines)
        line.delete(oldLine)
        if array.size(guideLabels) > 0
            label oldLabel = array.shift(guideLabels)
            label.delete(oldLabel)
    int guideEndBar = bar_index + guideProjectionInput
    color directionColor = anyBullishSignal ? bullishColorInput : bearishColorInput
    line entryGuide = line.new(bar_index, lastSignalEntry, guideEndBar, lastSignalEntry, xloc = xloc.bar_index, color = entryColorInput, style = line.style_dotted, width = 2)
    line stopGuide = line.new(bar_index, lastSignalStop, guideEndBar, lastSignalStop, xloc = xloc.bar_index, color = bearishColorInput, style = line.style_dotted, width = 2)
    line targetGuide = line.new(bar_index, lastSignalTarget, guideEndBar, lastSignalTarget, xloc = xloc.bar_index, color = directionColor, style = line.style_dotted, width = 2)
    array.push(guideLines, entryGuide)
    array.push(guideLines, stopGuide)
    array.push(guideLines, targetGuide)
```

```
    if showTradeGuidePriceTagsInput
        label entryTag = label.new(guideEndBar, lastSignalEntry, "Entry " + str.tostring(lastSignalEntry, format.mintick), xloc = xloc.bar_index, yloc = yloc.price, color = color.new(entryColorInput, 15), style = label.style_label_left, textcolor = color.white, size = size.tiny)
        label stopTag = label.new(guideEndBar, lastSignalStop, "SL " + str.tostring(lastSignalStop, format.mintick), xloc = xloc.bar_index, yloc = yloc.price, color = color.new(bearishColorInput, 15), style = label.style_label_left, textcolor = color.white, size = size.tiny)
        label targetTag = label.new(guideEndBar, lastSignalTarget, "TP " + str.tostring(rewardRiskInput, "#.##") + "R " + str.tostring(lastSignalTarget, format.mintick), xloc = xloc.bar_index, yloc = yloc.price, color = color.new(directionColor, 15), style = label.style_label_left, textcolor = color.white, size = size.tiny)
        array.push(guideLabels, entryTag)
        array.push(guideLabels, stopTag)
        array.push(guideLabels, targetTag)

// --- Footprint heatmap around the active auction level ---
float heatmapRadiusPrice = heatmapRadiusTicksInput * syminfo.mintick
bool activeHeatmapEvent = eventState > 0 and not na(eventLevel) and not na(eventStartBar) and bar_index - eventStartBar <= heatmapBarsInput
if not showFootprintHeatmapInput and array.size(footprintHeatmapBoxes) > 0
    while array.size(footprintHeatmapBoxes) > 0
        box staleHeatmapBox = array.pop(footprintHeatmapBoxes)
        box.delete(staleHeatmapBox)
```

```
if barstate.isconfirmed and showFootprintHeatmapInput and
activeHeatmapEvent and not na(footprintData)
    array<volume_row> footprintRows = footprint.rows(footprintData)
    int footprintRowCount = array.size(footprintRows)
    float levelTop = eventLevel + heatmapRadiusPrice
    float levelBottom = eventLevel - heatmapRadiusPrice
    float maxVolumeNearLevel = 0.0
    if footprintRowCount > 0
        for rowIndex = 0 to footprintRowCount - 1
            volume_row footprintRow = array.get(footprintRows, rowIndex)
            float rowTop = volume_row.up_price(footprintRow)
            float rowBottom = volume_row.down_price(footprintRow)
            float rowVolume = volume_row.total_volume(footprintRow)
            bool overlapsActiveLevel = not na(rowTop) and not na(rowBottom) and
rowTop >= levelBottom and rowBottom <= levelTop
            if overlapsActiveLevel
                maxVolumeNearLevel := math.max(maxVolumeNearLevel,
nz(rowVolume))
        if maxVolumeNearLevel > 0.0
            for rowIndex = 0 to footprintRowCount - 1
                volume_row footprintRow = array.get(footprintRows, rowIndex)
                float rowTop = volume_row.up_price(footprintRow)
                float rowBottom = volume_row.down_price(footprintRow)
                float rowVolume = volume_row.total_volume(footprintRow)
```

```
            bool overlapsActiveLevel = not na(rowTop) and not na(rowBottom) and
rowTop >= levelBottom and rowBottom <= levelTop and rowVolume > 0.0

            if overlapsActiveLevel

               float clippedTop = math.min(rowTop, levelTop)

               float clippedBottom = math.max(rowBottom, levelBottom)

               float rowDelta = volume_row.delta(footprintRow)

               bool rowBuyImbalance =
volume_row.has_buy_imbalance(footprintRow)

               bool rowSellImbalance =
volume_row.has_sell_imbalance(footprintRow)

               bool rowImbalance = rowBuyImbalance or rowSellImbalance

               float normalizedVolume = rowVolume / maxVolumeNearLevel

               int boxTransparency = math.max(8, math.min(92, 92 -
int(math.round(normalizedVolume * 60.0)) - (rowImbalance ? 15 : 0)))

               color rowColor = rowBuyImbalance and not rowSellImbalance ?
bullishColorInput : rowSellImbalance and not rowBuyImbalance ?
bearishColorInput : rowDelta > 0.0 ? bullishColorInput : rowDelta < 0.0 ?
bearishColorInput : entryColorInput

               if clippedTop > clippedBottom

                  while array.size(footprintHeatmapBoxes) >=
MAX_FOOTPRINT_HEATMAP_BOXES

                     box oldHeatmapBox = array.shift(footprintHeatmapBoxes)

                     box.delete(oldHeatmapBox)

                  box heatmapBox = box.new(bar_index, clippedTop, bar_index + 1,
clippedBottom, xloc = xloc.bar_index, bgcolor = color.new(rowColor,
```

```
boxTransparency), border_color = color.new(rowColor,
math.min(boxTransparency + 8, 100)))

                array.push(footprintHeatmapBoxes, heatmapBox)


// --- Status dashboard ---

string activeLevelText = eventState > 0 and not na(eventLevel) ?
eventLevelName + " @ " + str.tostring(eventLevel, format.mintick) :
keyLevelTouched ? selectedLevelName + " @ " + str.tostring(selectedLevel,
format.mintick) : "Waiting for touch"

string eventStatusText = eventState == 0 ? "Waiting for response" : eventState
== 1 ? "Level touched — monitoring" : eventClass == 1 ? "Failed Auction
candidate" : eventClass == 2 ? "Continuation candidate" : "NEUTRAL /
CONFLICT"

string evidenceStatusText = eventState != 2 ? "Awaiting candidate" :
eventClass == 3 ? "CISD / FVG-123 resolves conflict" : "Waiting: CISD or FVG-
123"

string latestSignalText = lastSignalBar == bar_index ? lastSignalClassification
+ " | " + lastSignalLevelName + " | " + lastSignalEvidence : lastSignalBar >= 0 ?
lastSignalClassification + " | " + lastSignalLevelName + " | " +
lastSignalEvidence : "No confirmed signal yet"

color latestSignalColor = str.contains(lastSignalClassification, "Bullish") ?
bullishColorInput : str.contains(lastSignalClassification, "Bearish") ?
bearishColorInput : chart.fg_color

bool supportedChartTimeframe = timeframe.isminutes and
timeframe.multiplier <= 5

if barstate.islast

   if showDashboardInput
```

table.cell(statusTable, 0, 0, "MFB KEY LEVEL RESPONSE", text_color = #DBDBDB, bgcolor = PANEL_BACKGROUND, text_size = size.small)

table.cell(statusTable, 1, 0, "HEATMAP", text_color = entryColorInput, bgcolor = PANEL_BACKGROUND, text_size = size.small)

table.cell(statusTable, 0, 1, "5m trend", text_color = #DBDBDB, bgcolor = PANEL_BACKGROUND)

table.cell(statusTable, 1, 1, trendStatus + " | Eff. " + str.tostring(fiveMinuteEfficiency, "#.0") + "%", text_color = establishedTrend == 1 ? bullishColorInput : establishedTrend == -1 ? bearishColorInput : #808080, bgcolor = PANEL_BACKGROUND)

table.cell(statusTable, 0, 2, "Active level", text_color = #DBDBDB, bgcolor = PANEL_BACKGROUND)

table.cell(statusTable, 1, 2, activeLevelText, text_color = eventState > 0 ? #f2c744 : #DBDBDB, bgcolor = PANEL_BACKGROUND)

table.cell(statusTable, 0, 3, "Response", text_color = #DBDBDB, bgcolor = PANEL_BACKGROUND)

table.cell(statusTable, 1, 3, eventStatusText, text_color = eventClass == 3 ? NEUTRAL_COLOR : eventClass == 1 ? (eventDirection == 1 ? bullishColorInput : bearishColorInput) : eventClass == 2 ? (eventDirection == 1 ? bullishColorInput : bearishColorInput) : #808080, bgcolor = PANEL_BACKGROUND)

table.cell(statusTable, 0, 4, "Structural evidence", text_color = #DBDBDB, bgcolor = PANEL_BACKGROUND)

table.cell(statusTable, 1, 4, evidenceStatusText, text_color = #DBDBDB, bgcolor = PANEL_BACKGROUND)

table.cell(statusTable, 0, 5, "Latest signal", text_color = #DBDBDB, bgcolor = PANEL_BACKGROUND)

```
    table.cell(statusTable, 1, 5, latestSignalText, text_color =
latestSignalColor, bgcolor = PANEL_BACKGROUND)

    table.cell(statusTable, 0, 6, "Scope", text_color = #808080, bgcolor =
PANEL_BACKGROUND, text_size = size.tiny)

    table.cell(statusTable, 1, 6, not na(footprintData) ? "Footprint heatmap
only | signals unchanged" : "No footprint data | signals unchanged", text_color
= #808080, bgcolor = PANEL_BACKGROUND, text_size = size.tiny)

  else

    table.clear(statusTable, 0, 0, 1, 6)


// --- Alerts ---

alertcondition(bullishFailedAuctionSignal, "MFB: Bullish Failed Auction",
"Bullish Failed Auction confirmed on {{ticker}} {{interval}}. See the indicator
dashboard for level and structural evidence.")

alertcondition(bearishFailedAuctionSignal, "MFB: Bearish Failed Auction",
"Bearish Failed Auction confirmed on {{ticker}} {{interval}}. See the indicator
dashboard for level and structural evidence.")

alertcondition(bullishContinuationSignal, "MFB: Bullish Trend Continuation",
"Bullish Trend Continuation confirmed on {{ticker}} {{interval}}. See the
indicator dashboard for level and structural evidence.")

alertcondition(bearishContinuationSignal, "MFB: Bearish Trend Continuation",
"Bearish Trend Continuation confirmed on {{ticker}} {{interval}}. See the
indicator dashboard for level and structural evidence.")

if anyConfirmedSignal and barstate.isconfirmed

  string dynamicAlertMessage = lastSignalClassification + " | " + syminfo.ticker
+ " " + timeframe.period + " | " + lastSignalLevelName + " @ " +
str.tostring(lastSignalLevel, format.mintick) + " | Evidence: " +
```

```
lastSignalEvidence + " | Footprint heatmap is visual only; signal remains
price/structure based"

    alert(dynamicAlertMessage, alert.freq_once_per_bar_close)


plot(eventState > 0 ? eventLevel : keyLevelTouched ? selectedLevel : na,
"Active auction level", color = #f2c744, linewidth = 2, style = plot.style_linebr)
```