

# PLAN\_DCA\_STATE\_RECONCILE

**Status:** planned (backlog, awaits operator GO) **Priority:** P1.5 **Erstellt:** 2026-05-17 22:30 UTC **Quelle:** Operator-Backlog-Spec 2026-05-17 — DCA-Inkonsistenz zwischen `bot_stdout.log`, `dca_log.json`, `live_portfolio.json`, `position_snapshots`, `trade_logs`  
**Roadmap-ID:** DCA-STATE-RECONCILE **Voraussetzung:** laufendes Monitoring-Fenster (C1-Shadow + 12h-Audit) ist abgeschlossen. **Folgearbeit:** DCA-STATE-SAVE-RACE-FIX (eigene Roadmap-ID, P1, code-fix nach Forensik).

## 1. Ausgangslage

Aus `bot_stdout.log` sind **4 DCA-Events** für aktuell offene Positionen sichtbar:

Datum	Symbol	DCA-Event
2026-05-16 07:14:48	XLM/USDT	1 × DCA-Rettung @ 0.1510
2026-05-16 07:14:49	ENA/USDT	1 × DCA-Rettung @ 0.1067
2026-05-16 10:08:00	ENA/USDT	1 × DCA-Rettung @ 0.1058
2026-05-17 01:27:10	SHIB/USDT	1 × DCA-Rettung @ 0.0000

Aktuell offene Positionen mit DCA-Historie: **ENA · XLM · SHIB** = 3 Positionen / 4 Events.

(HUMA hatte keinen DCA — und wurde am 2026-05-17 19:45 UTC via SL geschlossen, RISK-GUARD blockierte den DCA-Rescue korrekt mit `reason=dca_blocked_bear_regime`.)

## 2. Defects

### DCA-1: State-Persistence-Gap für ENA / XLM

`live_portfolio.json` zeigt für die DCA-Positionen `avg_price ≈ entry_price`. Zu prüfen: - ist `entry_price` nach DCA als Average überschrieben worden (möglich, war bei T-SPLIT-2-State so dokumentiert) - oder fehlt `avg_price` / `tranches` komplett

### DCA-2: `dca_log.json` inkonsistent für XLM

Bot-Log zeigt XLM-DCA 2026-05-16 07:14:48, `dca_log.json` zeigt jedoch `dca_count=0`, `tranches=['tranche_1']`.

### DCA-3: `trade_logs` nicht geeignet für offene-DCA-Auswertung

`trade_logs` enthält nur `closed_trades`. Für offene Positionen keine `status=open` Rows verfügbar. By-design — muss aber für GUI / Monitoring beachtet werden.

## 3. Scope (read-only Forensik)

- Read-only Analyse, keine State-Mutation
- Forensische Rekonstruktion aus 6 Quellen
- Abgleich der 3 offenen DCA-Positionen
- Dokumentation der korrekten DCA-Historie je Position
- Optionaler manueller File-Recovery-Plan, nur mit Operator-GO

## 4. Non-Scope / Boundaries

- 0x **Bot-Touch** während Monitoring
- 0x Restart
- 0x Trading-Logik ändern
- 0x DB-Migration
- 0x automatisierte Reparatur
- 0x DB-Korrekturen ohne separate Freigabe
- 0x Änderung an Risk-Guard, C1, Binance-Execution

## 5. Analysequellen

#	Quelle	Pfad	Charakter
1	bot_stdout.log	/home/node/.openclaw/workspace/trading/logs/bot_stdout.log	Event-Log (ground truth)
2	position_snapshots	DB Tabelle	Time-Series, gut für Cost/Qty-Verlauf
3	dca_log.json	/home/node/.openclaw/workspace/trading/logs/dca_log.json	DCA-Manager-State
4	live_portfolio.json	/home/node/.openclaw/workspace/trading/logs/live_portfolio.json	Bot-Runtime-State
5	trade_logs	DB Tabelle	Realisierte Closes (nicht open)
6	File mtimes	stat ...	State-Save-Zeitpunkte für Race-Detection

## 6. Methodik

Pro offene Position (ENA, XLM, SHIB):

1. **Event-Liste aus bot\_stdout.log** chronologisch ziehen `bash grep "DCA-Rettung: <SYMBOL>\|tranche_2\|tranche_3" bot_stdout.log` Erfassen: timestamp, price, neue avg, neuer SL.
2. **Cost / Qty Verlauf aus position\_snapshots** rekonstruieren `sql SELECT created_at, ROUND(position_value,2), quantity, stop_loss, entry_price FROM position_snapshots WHERE symbol=:sym AND status='open' ORDER BY created_at;` Sprünge in `quantity` und `position_value` markieren DCA-Zeitpunkte.
3. **dca\_log.json** Eintrag pro Symbol auslesen: `dca_count`, `tranches`, `total_invested`, `avg_price`.
4. **live\_portfolio.json** Eintrag pro Symbol auslesen: `entry_price`, `quantity`, `cost`, `stop_loss`, `_dca_rescued_*` - Marker.
5. **Cross-Check 4 Quellen** in Tabelle:
6. source-of-truth = bot\_stdout.log (Event-Log)
7. state-files sollten konsistent sein
8. Abweichungen werden flagged mit Confidence-Level

## 7. Deliverables

### A. Pro-Position-Forensik-Tabelle

Symbol	Initial Entry	Akt. Qty	Rek. Tranchen	Rek. Avg	Akt. SL	Quelle-je-Feld	Confidence
ENA/USDT	...	...	3 (1+2 DCA)	...	...	log/snapshots/portfolio	high/med/low
XLM/USDT	...	...	2 (1+1 DCA)	...	...	...	...
SHIB/USDT	...	...	2 (1+1 DCA)	...	...	...	...

### B. State-Abweichungen-Liste

```
{symbol, source_file, field, expected_from_log, actual_in_file, gap_severity}
```

### C. Entscheidungsvorlage Operator

Option	Wirkung	Risk
1 Nur dokumentieren	Inkonsistenz bleibt; Operator weiß Bescheid	0
2 State-Files manuell korrigieren	live_portfolio.json + dca_log.json patchen, Bot-Restart nötig	mittel (HISTORY-1-Fix-C als Präzedenz)
3 Bot-Code-Fix vorbereiten	DCA-STATE-SAVE-RACE-FIX kicken	mittel
4 GUI nur mit Source-Warnung erweitern	„DCA-historisch nicht in State" Hinweis im View-Position	gering, GUI-only

### D. Final-Report-PDF mit Timeline pro Position + Empfehlung

## 8. Acceptance Criteria

- Für ENA, XLM, SHIB ist nachvollziehbar dokumentiert welche DCA-Events tatsächlich stattgefunden haben.
- Es ist klar, ob `live_portfolio.json.entry_price` aktuell Entry oder Average bedeutet.
- Es ist klar, ob `dca_log.json` als Quelle zuverlässig genug ist.

- Es gibt **keine** Änderungen am laufenden Bot während des Monitorings.
- Ein späterer Bot-Fix (DCA-STATE-SAVE-RACE-FIX) ist getrennt beschrieben.

## 9. Folgearbeit: DCA-STATE-SAVE-RACE-FIX (separater Plan)

---

Nach der Forensik kann ein gezielter Code-Fix entworfen werden. Mögliche Fokus-Punkte:

- `paper_trade._save_state()` unmittelbar nach erfolgreicher DCA-Rettung absichern
- atomare Persistenz von `live_portfolio.json` und `dca_log.json` (z. B. write+rename Pattern)
- Reload-Race durch mtime-cookie (HISTORY-1 Fix B) zwischen Bot und Worker prüfen
- DCA-Tranchen als first-class state behandeln (eigene Spalte / klares Schema)

Eigener Roadmap-Eintrag DCA-STATE-SAVE-RACE-FIX P1 , eigener Plan, eigener Cutover.

## 10. STOP

---

Kein Code, kein State-Touch vor Operator- GO DCA-STATE-RECONCILE .

Bis dahin: \* Monitoring läuft ununterbrochen weiter \* Operator sieht in `/admin/positions` weiter den (möglicherweise inkonsistenten) `entry_price` \* Forensik-Aufwand ist niedrig: ~1-2h Analyse-Arbeit, keine Live-Aktion