STATE-CROSS-PROCESS-RECONCILE-1.1 + 1.4 — Plan-Only

Datum: 2026-06-04 (post-cutover Commit `c4651dd`) **Modus:** READ-ONLY · keine Code/State/Env-Änderungen **Re-Scope:** ersetzt MS-LOCK-STATE-1 (Thread-Lock-Annahme war falsch)

---

1. Root Cause final

Aspekt	Klärung
Bot-interne Concurrency	**keine** Thread-Race auf `live_trader.state`. Scan-Loop ist single-threaded; MS-Runner synchron innerhalb des Scan-Cycles
Pump.fun-WS-Thread	Schreibt **eigene** State-Datei (`tier3/pump_fun_monitor.py::STATE_FILE`), nie `live_portfolio.json`
Wallet-Tracker	Eigene Datei — kein Conflict
Echter Race-Window	**Cross-Process Bot↔Worker** auf `live_portfolio.json`
Bot-Mitigation existierend	`_reload_state_if_externally_changed` (trader_core.py:1288) prüft mtime-Cookie nur **1× pro Tick** in `update_prices` (line 1346)
Lücke	Worker kann **nach** dem `_reload`-Check und **vor** einem späteren Bot-`_save_state` zwischenspeichern → der Bot-Save überschreibt Worker-Updates

Konkrete Race-Sequenz

```
t=0  Bot _reload_state_if_externally_changed → state X eingelesen, cookie=mtime_X
t=1  Bot mutiert state in RAM (z.B. pos['BTC'].current_price=...)
t=2  Worker update_stop_loss → liest Disk X → setzt pos['BTC'].stop_loss=NEU
                            → _save_state → Disk wird X', mtime=mtime_X'
t=3  Bot _save_state (z.B. nach execute_buy) → schreibt RAM (mit STALE SL!)
                            → Disk wird X'' — Worker's SL VERLOREN
                            → cookie=mtime_X''
```

---

2. Betroffene Dateien

Datei	Zweck der Plan-Änderung
`trading/execution/trader_core.py`	Erweiterung `_save_state` um Pre-Save-mtime-Recheck + neue `_merge_disk_into_ram_for_save` Helper
`trading/execution/live_trade.py`	KEIN direkter Touch (erbt `_save_state`) — aber `STATE_FILE`-Pfad bleibt LIVE_STATE_FILE
`trading/main.py`	KEIN Touch — alle bestehenden `live_trader._save_state()` calls bleiben
`trading/strategies/multi_strategy_runner.py`	Optional: `persist_state_silent` ruft Save mit `allow_abort=True` (cooldown loss ist akzeptabel)
`trading/command_worker.py`	KEIN Touch — Worker liest Disk vor jeder Mutation (LiveTrader fresh per call)
`trading/db_emitter.py`	KEIN Touch — `bot_statuses.metadata_json` schon vorhanden
`trading/tests/test_state_cross_process_reconcile_1_1.py`	NEU — 20 Tests
`trading/tests/test_state_observability_1_4.py`	NEU — 10 Tests

0× DB-Migration · 0× GUI · 0× env · 0× CommandBus · 0× Strategy-Param · 0× Mainnet.

---

3. Aktuelle `_save_state` / `_load_state` Call-Sites

`_save_state` (27 Aufrufer im Bot-Prozess + 2 im Worker)

Datei	Line	Pfad / Kontext	Tier
`trader_core.py`	616	`execute_buy` nach pos eingefügt	T1-must-merge
`trader_core.py`	753	`_record_close_cooldown` (REPEAT-COOLDOWN-1)	T2-may-abort

`trader_core.py`	865	`execute_sell` nach cash+closed_trade	T1-must-merge
`trader_core.py`	966	T3 `execute_buy_tier3`	T1-must-merge
`trader_core.py`	1037	T3 mark_pending	T2-may-abort
`trader_core.py`	1104	P0-RUNTIME-SAFETY-FIX-1 (pending cleanup)	T2-may-abort
`trader_core.py`	1188	`partial_sell`	T1-must-merge
`trader_core.py`	1283	DCA-Rescue path	T1-must-merge
`trader_core.py`	1439	update_prices SL trigger → sell loop	T1-must-merge
`trader_core.py`	1469	update_prices final (post-tick)	T1-must-merge
`live_trade.py`	436	`_remove_position_on_exchange_only`	T1-must-merge
`live_trade.py`	501	SYNC-BALANCE-SANITY-1 Save	T2-may-abort
`live_trade.py`	525	`_sync_balance` post-cash-update	T1-must-merge
`live_trade.py`	1229	execute_sell pending=True	T1-must-merge
`live_trade.py`	1238	execute_sell pending=False	T1-must-merge
`live_trade.py`	1335	partial pending cleanup	T1-must-merge
`main.py`	576	Re-entry Cleanup	T1-must-merge
`main.py`	822, 834	Recovery-Skip-Cooldown writes	T2-may-abort
`main.py`	1767	Scan-Cycle end	T1-must-merge
`multi_strategy_runner.py`	via `_persist_state_silent`	MS-cooldown record	T2-may-abort
`command_worker.py`	622	update_stop_loss	Worker-only
`command_worker.py`	687	update_take_profit	Worker-only
`baseline_bootstrap.py`	283	boot-time auto-import	T2-may-abort

`_load_state` (1 Aufrufer)

- `trader_core.py:264` — `__init__` einmalig

`_reload_state_if_externally_changed`

- Definiert: `trader_core.py:1288`
- Aufgerufen: `trader_core.py:1346` (`update_prices` , 1× pro Tick)

---

4. Minimal-Fixplan 1.1 (PRE-SAVE-MTIME-RECHECK-1)

4.1 Neue Save-API

```
def _save_state(self, *, allow_abort: bool = False, source: str = 'unknown') -> bool:
    """Returns True on success, False on aborted-save."""
```

Backward-kompatibel: bestehende `live_trader._save_state()` calls behalten Default `allow_abort=False, source='unknown'` . Migration der labels passiert in einem Folge-Commit (P3).

4.2 Pre-Save-Recheck-Logik (pseudo)

```
def _save_state(allow_abort=False, source='unknown'):
    try:
        disk_mtime_ns = STATE_FILE.stat().st_mtime_ns
    except OSError:
        disk_mtime_ns = None

    if disk_mtime_ns and self._state_mtime_ns and disk_mtime_ns > self._state_mtime_ns:
        # Conflict detected
        self._state_coherence['conflict_count'] += 1
        self._state_coherence['last_conflict_at'] = utcnow()
        self._state_coherence['last_external_mtime_ns'] = disk_mtime_ns

        if allow_abort:
            logger.warning(f"[STATE-COHERENCE] save aborted, source={source}")
            self._state_coherence['retry_count'] += 1
            return False

        # must-merge path
        logger.warning(f"[STATE-COHERENCE] pre-save mtime conflict, source={source}")
```

```
        merged = self._merge_disk_into_ram_for_save(disk_mtime_ns)
        if merged is None:
            logger.error(f"[STATE-COHERENCE] merge failed — falling back to blind save")
        else:
            self.state = merged
            self._state_coherence['merged_count'] += 1

    # ── existing atomic write path ──────────────────────────────
    STATE_FILE.parent.mkdir(exist_ok=True)
    tmp_file = STATE_FILE.with_suffix('.tmp')
    with open(tmp_file, 'w') as f:
        json.dump(self.state, f, indent=2)
    tmp_file.rename(STATE_FILE)
    self._state_mtime_ns = STATE_FILE.stat().st_mtime_ns
    return True
```

4.3 Section-Ownership-Matrix

State-Section	Schreibt	Merge-Regel
`cash`	**Nur Bot**	RAM gewinnt
`closed_trades`	Bot (sells), Worker (close_position)	**UNION by `position_id`** (concat dedupliziert)
`positions[sym].stop_loss`	Bot (update_prices SL-adjust), Worker (update_stop_loss)	**Disk gewinnt** wenn Disk ≠ RAM-load-snapshot
`positions[sym].take_profit`	Bot, Worker (update_take_profit)	gleiche Regel wie stop_loss
`positions[sym].current_price`	Bot	RAM gewinnt
`positions[sym].pending`	Bot	RAM gewinnt
`positions[sym]` Existenz	Bot adds, Worker deletes	**3-Wege-Merge** (siehe 4.4)
`sl_cooldowns`	**Nur Bot**	RAM gewinnt
`repeat_loss_count`	**Nur Bot**	RAM gewinnt
`tier3_used/positions/budget`	**Nur Bot**	RAM gewinnt
`ms_candidate_cooldowns`	**Nur Bot** (MS-Runner)	RAM gewinnt
`starting_capital`	Bot (boot only)	RAM gewinnt
`created_at`	Bot (boot only)	RAM gewinnt

4.4 `_merge_disk_into_ram_for_save` Helper (pseudo)

```
def _merge_disk_into_ram_for_save(disk_mtime_ns) -> Optional[dict]:
    try:
        with open(STATE_FILE, 'r') as f:
            D = json.load(f)
    except Exception as e:
        return None

    R = self.state
    merged = dict(R)  # Bot-only sections → R wins

    # closed_trades: union by position_id
    seen_ids = set()
    union = []
    for t in D.get('closed_trades', []) + R.get('closed_trades', []):
        tid = t.get('position_id')
        if tid and tid in seen_ids:
            continue
        seen_ids.add(tid)
        union.append(t)
    merged['closed_trades'] = union[-500:]

    # positions: 3-way merge
    R_pos = R.get('positions', {})
    D_pos = D.get('positions', {})
    merged_pos = {}
    for sym in set(R_pos.keys()) | set(D_pos.keys()):
        if sym in R_pos and sym not in D_pos:
            merged_pos[sym] = R_pos[sym]                # Bot added → keep
        elif sym in D_pos and sym not in R_pos:
            continue                                     # Worker removed → drop
        else:
            p = dict(R_pos[sym])
            for field in ('stop_loss', 'take_profit'):
                if D_pos[sym].get(field) != R_pos[sym].get(field):
```

```
                    p[field] = D_pos[sym].get(field)
            merged_pos[sym] = p
    merged['positions'] = merged_pos

    return merged
```

4.5 Antworten auf die 8 Operator-Fragen

#	Frage	Antwort
1	Wo liegt der aktuelle mtime-Cookie?	`self._state_mtime_ns` in `TraderCore.__init__` (line 171); aktualisiert von `_load_state` (277), `_save_state` (312), `_reload_state_if_externally_changed` (1326)
2	Welche `_save_state` Call-Sites?	27 im Bot, 2 im Worker — Tabelle §3
3	Kritische Sections?	`positions` (3-Wege-Merge), `closed_trades` (union), `cash` (Bot), `ms_candidate_cooldowns` (Bot), `sl_cooldowns` (Bot), `repeat_loss_count` (Bot), `tier3_*` (Bot), pending-Felder in positions[sym] (Bot)
4	Minimal sichere Merge-Strategie?	Section-Ownership-Matrix §4.3 + 3-Wege-Position-Merge §4.4
5	Wann abbrechen statt mergen?	`allow_abort=True` für Cooldown/Loss-Count/Pending-Marker-Saves (T2). Default-Pfad (T1) merged immer.
6	Wie verhindern wir Verlust neuer Bot-Positionen?	3-Wege-Merge: in RAM aber nicht in Disk → **immer keep RAM**
7	Wie verhindern wir Verlust Worker-SL/TP-Updates?	Per-Feld-Merge in positions[sym]: wenn Disk-SL/TP ≠ Load-SL/TP → Disk gewinnt
8	Partial-Pending-Marker?	`pending` ist Bot-only-Feld in positions[sym] → RAM gewinnt; Position-Existenz folgt 3-Wege-Regel

---

5. Observability-Plan 1.4 (REGRESSION-OBSERVABILITY-1)

5.1 In-Memory Counter (auf TraderCore-Instanz)

```
self._state_coherence = {
    'reload_mtime_jump_count':      0,
    'save_precheck_conflict_count': 0,
    'save_precheck_retry_count':    0,
    'save_precheck_merged_count':   0,
    'last_conflict_at':           None,
    'last_external_mtime_ns':     None,
    'last_merge_source':          None,
    'last_merge_summary':         None,
}
```

5.2 Log-Messages (level=WARNING/INFO, kein DEBUG)

Trigger	Level	Format
`_reload_state_if_externally_changed` returns True	INFO (existing)	unchanged
Pre-save conflict, allow_abort=False	WARNING	`[STATE-COHERENCE] pre-save mtime conflict, source={source} — merging disk into RAM (old={old_ns}, new={new_ns})`
Pre-save conflict, allow_abort=True	WARNING	`[STATE-COHERENCE] save aborted, source={source} — external mtime jumped`
Merge executed	INFO	`[STATE-COHERENCE] merged external update, source={source} summary={...}`
Merge failed (fallback)	ERROR	`[STATE-COHERENCE] merge failed, source={source} — falling back to blind save`

5.3 `bot_statuses.metadata_json` Erweiterung

`trading/main.py:1917` `emit_bot_status` bekommt zusätzlichen Metadata-Key:

```
metadata={
    ...existing keys...,
    'state_coherence': {
        'reload_mtime_jump_count':    live_trader._state_coherence['reload_mtime_jump_count'],
        'save_precheck_conflict_count': live_trader._state_coherence['save_precheck_conflict_count'],
```

```
        'save_precheck_retry_count':    live_trader._state_coherence['save_precheck_retry_count'],
        'save_precheck_merged_count':   live_trader._state_coherence['save_precheck_merged_count'],
        'last_conflict_at':           live_trader._state_coherence['last_conflict_at'],
        'last_merge_source':          live_trader._state_coherence['last_merge_source'],
    },
}
```

Keine DB-Migration nötig — `metadata_json` ist JSONB.

5.4 Optional: Tagespanik-Schwelle

`save_precheck_conflict_count > 10 / 24 h` → Telegram-Alert. Für Phase 1.4 **erstmal nur Log + bot_statuses**, kein Telegram.

---

6. Tests

`test_state_cross_process_reconcile_1_1.py` (20 Tests)

#	Test	Erwartung
1	`test_save_without_external_change_writes_normally`	Cookie sync → normal save
2	`test_save_detects_external_mtime_bump`	Mock stat → conflict_count += 1
3	`test_must_merge_preserves_worker_sl_update`	Worker setzt pos['BTC'].SL=12; Bot merged → final SL=12
4	`test_must_merge_preserves_worker_tp_update`	analog für TP
5	`test_must_merge_preserves_bot_new_position`	RAM hat pos['ETH'] (neu); Disk hat es nicht → final hat ETH
6	`test_must_merge_drops_worker_closed_position`	RAM hat pos['BTC']; Disk hat es NICHT (Worker close) → final hat BTC NICHT
7	`test_must_merge_preserves_ms_candidate_cooldowns`	RAM hat cooldown, Disk hat keine → final state hat cooldown
8	`test_must_merge_preserves_sl_cooldowns`	analog `sl_cooldowns`
9	`test_must_merge_unions_closed_trades`	Disk hat trade_A; RAM hat trade_B → final hat beide
10	`test_must_merge_dedups_closed_trades_by_position_id`	gleiche position_id → 1×
11	`test_must_merge_position_simultaneous_buy_close_collision`	RAM neue pos['ETH'] + Disk gelöschte pos['BTC'] → final hat ETH, kein BTC
12	`test_allow_abort_skips_write_and_increments_retry`	`_save_state(allow_abort=True)` mit Conflict → kein write, retry_count += 1
13	`test_allow_abort_path_does_not_lose_disk_state`	Disk-State bleibt unverändert nach abort
14	`test_no_conflict_no_merge_call`	Normalfall: merge-Helper nicht gerufen
15	`test_atomic_tmp_rename_preserved`	tmp + rename Pattern unverändert
16	`test_json_validity_post_merge`	merged JSON ist valides JSON
17	`test_merge_failure_falls_back_to_blind_save`	json.load wirft → log ERROR, blind save
18	`test_worker_save_round_trip_unchanged`	Worker `_save_state` direkt → kein neuer Code-Path
19	`test_partial_pending_marker_preserved`	RAM pos['BTC'].pending=True, Disk pos['BTC'] ohne pending → final hat pending=True
20	`test_source_label_logged`	source='ms_runner' im Log-Output sichtbar

`test_state_observability_1_4.py` (10 Tests)

#	Test	Erwartung
1	`test_coherence_dict_initialized`	`_state_coherence` dict mit 0-Werten nach `__init__`
2	`test_reload_increments_jump_count`	`_reload_state_if_externally_changed` True → counter +1
3	`test_conflict_increments_count`	pre-save conflict → counter +1
4	`test_retry_increments_on_abort`	allow_abort=True conflict → retry +1
5	`test_merged_count_increments`	must-merge path → merged_count +1
6	`test_last_conflict_timestamp_updates`	last_conflict_at sets to UTC ISO
7	`test_last_merge_summary_populated`	merge writes summary {"positions_merged": n, ...}
8	`test_bot_status_metadata_contains_state_coherence`	main.py emit_bot_status payload check (Mock)
9	`test_no_log_spam_when_no_conflict`	Normaler save → 0 STATE-COHERENCE log lines

10	`test_log_message_format_includes_source`	WARNING log enthält `source=`

Regression-Sweep (P0-Tests aus History)

- `test_sync_balance_sanity_1` (20)
- `test_exit_reason_fix_1` (20)
- `test_ms_candidate_dedup_1` (20)
- `test_ms_stablecoin_block_1` (12)
- `test_phase_n7` + `test_phase_n8` (cumulative)
- `test_data_link_1` + `test_data_link_1_fu2`
- `test_t_split_2_emitter_wiring`

Ziel: 0 neue Failures (pre-existing N8-test bleibt erwartet failing).

---

7. Cutover-Plan (SOT-1d-Pattern)

Step	Action
1	Crontab `bot_watchdog.sh` UND `worker_watchdog.sh` freeze via Marker `# CUTOVER_FREEZE_RECON1`
2	Backup: `/root/recon-1-backup-{TS}/` mit `live_portfolio.json` + Code-Files
3	`docker compose build clawbot` + `docker compose build clawbot-worker` (beide!)
4	`docker compose up -d --force-recreate clawbot clawbot-worker`
5	3-Way MD5: `trader_core.py` Repo == Image (clawbot) == Image (worker) == Container (clawbot) == Container (worker)
6	Bot host-PID + Worker host-PID neu, beide healthy
7	Crontab freeze entfernen
8	30-min Window: `[STATE-COHERENCE]` Log + `bot_statuses.metadata_json.state_coherence` Counter beobachten
9	Operator-Functional-Test: GUI close_position während Bot-Scan läuft → erwarteter Pfad: merged, Counter +1
10	24-h Monitor: `state_save_precheck_conflict_count > 0` belegt echte Race-Hits

Rollback

- `git revert <commit>` → Build + Recreate clawbot + clawbot-worker
- Backup-Datei restore (live_portfolio.json) wenn merged-state inkonsistent wirkt

---

8. Risiken

#	Risiko	Severity	Mitigation
1	Merge-Helper liefert beschädigten State (false-positive Field-Konflikt)	Mittel	Per-Feld-Merge nur für SL/TP; Tests #19 (pending) + #11 (collision)
2	Worker schreibt zwischen Bot-Merge und Bot-Save (2.-Generation Race)	Niedrig	Counter zählt; 2 Conflicts in Folge → Log ERROR. Phase 1.2 (FCNTL-FLOCK) wäre der echte Fix
3	Aggressive Abort verhindert legitime Saves	Niedrig	`allow_abort=True` nur für Cooldown/Loss-Marker (T2). T1 nie abort.
4	Counter-Inflation (Bot saved sich selbst inkrementiert)	Niedrig	Cookie wird nach jedem eigenen Save resync'd
5	Performance-Cost (extra stat() pro Save)	sehr niedrig	1 stat() ≤ 50 µs
6	Worker-Side bekommt Recreate	Niedrig	Sauberer Recreate, keine offenen Long-Running Commands
7	Falsche Reihenfolge der Merge-Helper (Bot-positions vs Worker-deletes)	Mittel	Tests #5 + #6 + #11 spezifisch
8	`closed_trades` Union ohne `position_id` → Duplicates	Niedrig	Test #10 dedupliziert; legacy ohne position_id by-index akzeptabel
9	Pre-existing N8-Test failure	nicht-related	bleibt unverändert

---

9. Bundle vs Split — Empfehlung

→ Empfehlung: BUNDLE 1.1+1.4 in EINEM Commit + EINEM Cutover.

Begründung

Argument	Wirkung
1.4 ist nur sinnvoll mit 1.1 (Counter zählen ohne neue Save-Logik wäre sinnlos)	Split erzwingt 1.1 vor 1.4 → 1.4 alleine bringt nichts
Beide treffen `trader_core._save_state`	Doppel-Cutover = doppeltes Risiko-Fenster auf clawbot+clawbot-worker
1.4 ist strikt additiv (Counter + Log + 1 metadata-Key)	Low-Risk-Add
Tests sind disjunkt aber Code-Touch überlappt	Bundle vermeidet 2-fache Edit-Rounds
Observability AB Tag 1 ist wertvoller als nachgereicht	Operator sieht Conflicts sofort beim ersten echten Race

Alternative — Split (nicht empfohlen)

- Phase 1.1 zuerst → 7 Tage Beobachtung ohne Counter (nur Log)
- Phase 1.4 danach
- Kostet 2 Cutover, längere Beobachtung mit weniger Daten

---

10. MS-Live-Readiness-Update

Bedingung	Status
REPEAT-CANDIDATE-DEDUP-1	live (commit c4651dd)
MS-STABLECOIN-BLOCK-1	live (commit c4651dd)
STATE-CROSS-PROCESS-RECONCILE-1.1+1.4	**dieser Plan — HARTE Vorbedingung vor MS-Live**
MS-LIVE-OHLCV-BACKTEST-1	offen P2

Sobald 1.1+1.4 live + 24 h Counter sauber → **MS-Live darf in Testnet-Execution geschaltet werden**.

---

11. Boundaries dieses Plans

0× Code-Touch · 0× State-Edit · 0× Bot-Recreate · 0× Worker-Recreate · 0× MS-Live-Aktivierung · 0× Env-Änderung · 0× DB-Migration · 0× Order-Logik · 0× Mainnet · 0× CommandBus · 0× Push.

---

12. Operator-Entscheidungen vor `GO EXECUTE`

Q	Frage	Default-Empfehlung
Q1	Bundle 1.1+1.4 oder Split?	**A** Bundle
Q2	Default `allow_abort` für MS-cooldown saves?	**A** True (cooldown loss akzeptabel)
Q3	Telegram-Alert bei > 10 conflicts/24h?	**B** Nein in dieser Phase — nur Log + bot_statuses
Q4	source-Label-Migration aller 27 Save-Sites?	**B** Out-of-Scope für 1.1 — `source='unknown'` default, später separater P3-Commit
Q5	`closed_trades` ohne `position_id` → dedup?	**B** Akzeptieren als-ist (legacy entries)
Q6	Worker-Recreate Teil dieses Cutovers?	**A** Ja — beide brauchen neuen Code (`trader_core.py` ist shared)
Q7	Backup-Pfad?	`/root/recon-1-backup-{TS}/`
Q8	Observability-Dashboard (GUI-Widget) als Follow-up?	**A** Ja als separater P3-Commit (außer Scope)

STOP

Plan abgeschlossen. Kein Code geändert. Warte auf Operator-Entscheidungen Q1–Q8 + `GO EXECUTE STATE-CROSS-PROCESS-RECONCILE-1` für Bundle 1.1+1.4.