04_Slices

Slice 0: Bootstrap & Contracts

- Goal: Turn the plan/ADRs into importable code contracts and a runnable skeleton so every later slice builds on stable interfaces.
- Scope
    - Repo scaffold and packaging (one top-level Python package, e.g., `bt/`).
    - CLI stub with subcommands: `bt backtest`, `bt shadow`, `bt paper`, `bt live`.
    - Ports/contracts as code (ABC/Protocol):
        - `Clock`, `EventBus` (in-process), `SnapshotStore`, `RunManifestStore`
        - `MarketDataProvider` (history + live bars), `OrderRouter` (paper, real),
        - `RiskEngine` (pre-trade), `PortfolioStore` (positions, cash, P&L),
        - `Logger/Telemetry` (structured logs + metrics),
        - `ConfigProvider` and `SecretsProvider`.
    - CI: unit tests + type checks + lint; build and run on push.
    - ADR links/docstrings on each port; versioned with the repo.
- Out of scope:
    - Concrete exchange adapters, full paper/live implementations, non-trivial metrics/alerts.
- Repo structure

```
backtester/
  backtester/core/        # event bus, app services
  backtester/ports/       # interfaces: broker, datafeed, storage, clock
  backtester/adapters/    # paper broker, parquet store, REST polling feed
  backtester/strategy/    # strategy template(s)
  backtester/cli/         # CLI entrypoints
  backtester/sim/         # fill models, slippage, fees
  backtester/risk/
  backtester/stores/      # snapshots, portfolio, parquet
  backtester/util/
tests/
  tests/contract/     # adapter contract tests
  tests/e2e/
sample_data/
ci/
docs/
adr/
scripts/
tickets/
```

- Done when:
    - `backtester --help` shows the four subcommands; each runs a no-op pipeline.
    - All ports compile and are importable; basic unit test per port passes.

    - CI is green: tests + types + lint.
    - A sample `backtest --noop` run writes a `run_manifest.json` and structured logs.

Slice 1: Config & Secrets

- Goal: Make runs reproducible, safe, and auditable via typed config + layered overrides and secret handling.
- **Scope**
    - Typed `Config` schema (e.g., dataclass) with sections: data paths, symbols, timeframes, costs, risk limits, runtime SLOs, live endpoints.
    - Precedence and merge order: **defaults < file (YAML/JSON) < env vars < CLI flags**.
    - `SecretsProvider`: never logs secrets; supports env vars and one local secure backend (dev).
    - `RunManifestStore`: persist config snapshot, Git SHA, start/end times, and environment (py version, OS).
    - CLI: `--config`, `--set key=value`, and `--env-prefix` support.
- **Out of scope**
    - Cloud secret managers, multi-env promotion flows.
- **Done when**
    - Any run emits a manifest including **full effective config hash** and Git SHA.
    - Missing/invalid config fails fast with a clear error.
    - Secrets never appear in logs or manifests.
    - Re-running the same config on the same data produces the same manifest hash.

Slice 2: Backtest core

- **Goal**: Deterministic backtest loop over **1-minute OHLCV** for one symbol with reliable accounting and auditable outputs. CSV trades; Parquet storage.
- **Scope**
    - **Clock & loop**: deterministic simulated clock stepping bar-by-bar (UTC).
    - **Data ingestion (history)**: read Parquet bars; schema: `ts_utc, symbol, open, high, low, close, volume[, ntrades], source`.
        - QC: strict monotonic timestamps, no overlaps; configurable gap handling (`--strict` fail vs `--fill forward/none`).
    - **Event pipeline**: `bar → strategy.on_bar → order intents → risk (minimal) → sim matcher → fills → accounting`.
    - **Matcher (bar-based)**:
        - Market fills at bar close ± slippage.
        - Limit fills on cross using side-consistent intra-bar price (min/max).
        - Aggregated fill record per order (MVP).
    - **Accounting**: FIFO lots; realized/unrealized P&L; equity curve; fees/commissions applied post-fill.
    - **Outputs**: `trades.csv`, `orders.csv`, `equity.csv`, `runs/<id>/logs.jsonl`; Parquet bar cache for reproducibility.

    - **CLI**: `bt backtest --config path.yml [--from 2024-01-01 --to 2024-03-01] [--strict]`.
    - **Perf guardrails**: single-symbol backtest ≥ 1 month completes in < 5s on laptop-class HW; memory bounded.
- **Out of scope**
    - Multi-symbol, tick-level, partial-fills fidelity, funding/fees beyond cash spot.
- **Done when**
    - Running the same config/data twice yields **identical file hashes** for `trades.csv` and `equity.csv`.
    - Gap/duplicate bars are **logged** and either **refused** (strict) or **tolerated** (fill mode) per config; choice is visible in the manifest.
    - All outputs are produced; a manifest captures Git SHA, effective config hash, runtime, and data sources.
    - Unit tests cover: bar ordering/QC, limit cross logic, P&L math (FIFO), and determinism (seeded).

Slice 3: Strategy plug + fixed slippage and fees + Minimal Risk in Sim

- **Goal:** Enable strategy development with realistic fills/costs and early pre-trade guardrails—**the same path** later used by paper/live.
  Scope
- Strategy API:
    - `on_start(ctx)`, `on_bar(ctx, symbol, bar) -> list[OrderIntent]`, `on_fill(ctx, fill)`, `on_stop(ctx)`.
    - Access to `ctx.state` (positions, cash), `ctx.now_utc`, and `ctx.submit(order)`.
- Fill/Execution policy (sim):
    - Market: fill at bar close ± slippage.
    - Limit: fill if price crosses; price = min/max within bar consistent with side.
    - Partial fills aggregated into a single fill record in sim (MVP simplification).
- Cost model: pluggable `CostModel` with commissions/fees, spread, slippage (bps or function), min notional.
- **Minimal Risk (enabled in sim now; reused later):**
  - Allowed symbols whitelist.
  - Per-order notional cap.
  - Max open positions count.
  - Daily loss limit (hard stop → block new orders).
  - All risk blocks are logged with reason codes.
  Out of scope
- Advanced exposure buckets, leverage/margin logic, borrow/short.
  Done when
- Deterministic backtest: given same data+config, orders and P&L are identical run-to-run.
- Costs are included and can be toggled/parameterized via config.
- Risk violations block orders in sim and are visible in logs with reason codes.

- Unit tests cover: limit-cross logic, slippage application, and each minimal risk rule.

Slice 4 Paper trading parity

- **Goal:** Run on **live 1-min bars** with an in-process paper broker using the same order path as sim, to validate parity before any real exchange.
- **Scope**
  - **LiveDataProvider (REST polling)**: UTC timestamps, retries/backoff, gap detection, lag metric.
  - **PaperOrderRouter**: same state machine as live (new/submitted/filled/canceled/rejected); idempotent `client_order_id`; restart-safe.
  - **Fill policy**: identical to sim (market @ close ± slippage; limit on valid cross).
  - **State**: `PortfolioStore` persisted; restart restores positions/cash/open orders.
  - **Controls**: global **halt/flatten**; dry-run mode logs would-be actions only.
  - **Telemetry**: `bars_lag`, `strategy_tick_latency`, `orders_submitted`, `orders_blocked_risk`.
- **Out of scope**:
  - Real exchange connectivity, advanced alerting dashboards
- **Done when**
  - Same-day **paper vs recorded sim** P&L difference **median ≤ 5 bps**.
  - **No missed bars** over a continuous **7-day** paper run (gaps detected & logged).
  - Idempotent order IDs prevent duplicates across restarts.
  - **Latency SLOs** met end-to-end: **p95 ≤ 500 ms**, **p99 ≤ 900 ms** from bar arrival to decision.

Slice 5 Shadow Mode

- **Purpose**: Soak-test the live runtime safely: compute signals and risk-check orders, but **do not submit** anywhere. Capture logs/metrics to validate stability and SLOs.
- **Scope**
  - `NullOrderRouter` that records proposed orders (with reasons if blocked) but never submits.
  - Full live pipeline: data → strategy → risk → (null) router → logs/metrics/snapshots.
  - Health and observability:
    - Metrics: `bars_lag`, `strategy_tick_latency`, `order_proposed_rate`, `risk_block_rate`, `error_rate`.
    - Alerts/warnings (log-level is fine for MVP): bar gap detected, error rate spike, daily loss threshold hit (from shadow P&L).
  - Artifacts: daily run manifests and compressed logs for replay.
- **Out of scope**: Exchange auth, order placement, reconciliation with real positions.
- **Done when**
  - **7-day** continuous shadow run completes without missed bars (gaps reported ≤ configured tolerance).
  - Latency SLOs met: **p95 ≤ 500 ms, p99 ≤ 900 ms** decision latency.
  - Metrics show non-zero proposed orders and risk blocks (i.e., strategy is “doing work”).

- Crash/restart resumes from the last snapshot with consistent internal state.

Slice 6 — Risk rails & safety

- Goal: Enforce per-order and portfolio guardrails pre-trade; provide an operator **kill-switch** to halt and flatten safely.
- **Scope**
  - **Rules (pre-trade)**:
    - Allowed symbols whitelist.
    - **Per-order notional cap** (quote currency).
    - **Max open positions** count.
    - **Daily realized loss limit** → hard stop (blocks new orders).
  - **Evaluation**: deterministic order; first failing rule stops evaluation; attach **stable reason code**.
  - **Operator controls**:
    - `bt halt` (block new orders immediately).
    - `bt flatten` (submit closes for all open positions, then halt).
    - Manual resume requires explicit flag.
  - **Observability**: per-rule counters, last block reason, rolling daily loss.
  - **Configuration**: thresholds in typed config; rules can be toggled per env (sim/paper/live).
- **Out of scope**
  - Leverage/margin, shorting/borrow, sector/bucket exposures (non-crypto).
- **Done when**
  - Any order breaching **notional**, **max positions**, or **daily loss** is blocked; an audit log entry with **reason code** is produced.
  - **Kill-switch** halts within **≤ 1s**; **flatten** closes positions then sets halt state; resume requires manual flag.
  - Shadow/paper runs show non-zero **risk_block_rate** under forced breach scenarios.

Slice 7 Binance REST live adapter (use Binance wrapper)

- **Goal**: Submit **tiny real orders** with the same semantics as paper; robust to restarts, rate limits, and transient failures. Use REST first (polling), then add WebSocket later (non-MVP).
- **Scope**
  - **Auth & signing**: API key/secret via `SecretsProvider`; HMAC signing; clock skew guard (time sync endpoint).
  - **Order placement**: `clientOrderId` idempotency; map venue filters (LOT_SIZE, MIN_NOTIONAL, PRICE_FILTER) at submit-time; reject early on violation.
  - **State & reconciliation**:
    - Poll open orders & account snapshot; reconcile against local state on startup and periodically.
    - Aggregate venue partial fills into **single logical fill** upstream (MVP policy).
  - **Error handling**: map HTTP/JSON errors to stable codes; classify **retryable vs terminal**; exponential backoff; respect rate-limit headers.

    - **Safety**: tiny size orders; configurable symbol allow-list; auto-halt on repeated failures.
    - **CLI**: `bt live --adapter binance --tiny --config path.yml`.
- **Out of scope**: WebSocket streaming, margin/derivatives, advanced order types beyond market/limit GTC.
- **Done when**
    - End-to-end tiny orders **fill** using the same state machine as paper.
    - **Startup reconciliation** yields correct positions; no duplicate live orders after process crash.
    - **Rate-limit handling** verified (throttled path exercised) with backoff; no hot loop.
    - Venue **filter violations** are blocked pre-submit with clear reason.
    - p95 order **round-trip ≤ 500 ms**, p99 ≤ 900 ms (from submit to ACK/terminal event) under light load; no missed live bars for a week.

Slice 8 Observability & recovery

- **Goal:** Metrics, structured logs, **snapshots/checkpoints**, and **crash-replay**; minimal alerts for bar gaps, order errors, and daily loss halts.
- **Scope**
    - **Structured logging**: JSONL with run id, ts_utc, component, level, msg, context (order id, symbol).
    - **Metrics**: `bars_lag`, `strategy_tick_latency`, `order_rtt`, `error_rate`, `risk_block_rate`, `pnl_intraday`.
    - **Snapshots**: state snapshot on position change and at fixed cadence; include open orders and cash.
    - **WAL**: append-only order/decision log enabling idempotent replay.
    - **Replay**: `bt replay --from snapshot --wal path` reconstructs state and verifies idempotency.
    - **Alerts** (log-level for MVP):
        - **Bar gap**: consecutive bar gap > tolerance.
        - **Order error rate**: rolling window above threshold.
        - **Daily loss halt**: triggered by RiskEngine.
- **Out of scope**
    - External monitoring stacks (Grafana/Prom), pager duty, rich dashboards.
- **Done when**
    - **Crash mid-submit** test: recovery results in correct state and **zero duplicate live orders**.
    - Metrics above are emitted; three alerts fire under induced conditions and are visible in logs.
    - Daily CSV/Parquet rollups generated; minimal summary printed at end of each run.

Contract Tests (Gating Before Real Orders)

- **Purpose**: Prove every adapter and store honors its port contract. These are **automated, repeatable** checks that gate enabling real trading.

- **Scope**
  - **MarketDataProvider contract tests**
  - History is chronological, **UTC**, complete for the requested window (or gaps are explicitly surfaced).
  - Live bars' timestamps are strictly increasing; lag metric reported.
  - **OrderRouter contract tests** (run against Paper and any real adapter sandbox/demo):
  - `submit` returns ACK with idempotent behavior on duplicate `client_order_id`.
  - `cancel` transitions correctly; cancel after fill is a no-op with stable status.
  - Rate-limit and transient errors map to **retryable** codes with backoff.
  - Partial fills (if any from venue) are **aggregated** into a single final fill event presented upstream (MVP policy).
  - **RiskEngine contract tests**
  - Each rule blocks when it should and emits a stable reason code.
  - **SnapshotStore / PortfolioStore contract tests**
  - Save/restore yields byte-for-byte equivalent trading state (positions, cash, open orders).
  - **Clock/EventBus**
  - Deterministic scheduling in sim; subscribers receive events in publish order (single-threaded MVP).

  Tooling & Run
  - CLI: `bt contract-tests [--adapter paper|binance-sandbox|...]`
  - Runs locally in CI; any failure blocks merging/enabling real orders.

  Done when
- All contract tests pass for: `MarketDataProvider` (history, live), `OrderRouter` (paper), `RiskEngine`, `SnapshotStore`, `PortfolioStore`, `Clock/EventBus`.
- A single summary report is produced (JUnit/JSON) and stored as an artifact in CI.
- Before enabling a real adapter, it must pass the same suite against its sandbox/demo environment.