In-Class Assignment AIF3: Kill-Switch Design

Digital Finance – BSc Course Prof. Dr. Joerg Osterrieder

Context. An autonomous trading agent at a Zurich prop shop manages a **$500M** equities book with a **$50M daily VaR** and a **$5M/minute** maximum burn rate. Knight Capital's 2012 incident destroyed $440M in 45 minutes because no layer of their stack could halt a runaway algorithm in time. Your task: design a **3-layer kill switch** (pre-trade / in-flight / post-trade) with measurable triggers and latencies that would have stopped Knight in **<10 seconds**.

Q1. Layer 1 – Pre-trade: specify 3 hard checks that fire *before* any order reaches the exchange. Give the numeric trigger and the target latency budget (must be < 1 ms to not break execution).

Solution. (1) **Notional limit**: reject any order $>$ $5M or bringing gross exposure $>$ $500M (hard-coded in the FIX gateway, $\sim$50 μs). (2) **Rate cap**: reject if $>$ 200 orders/second from a single strategy ID (Knight fired $\sim$4 million child orders in 45 min = 1,480/sec). (3) **Symbol allowlist**: reject any order outside the curated 2,000-ticker universe (Knight's bug routed to RETAIL flag symbols not meant for the algo). **Latency budget**: 3 checks $\times$ 50 μs = 150 μs, well under the 1 ms bound. Implementation: FPGA or C++ co-located with the exchange feed, not Python.

Q2. Layer 2 – In-flight: specify 2 checks that run *during* the trading day on aggregate position/PnL. Give triggers and latency budget (target < 1 second to catch the event but not false-positive on normal moves).

Solution. (1) **Intraday drawdown**: if realised+unrealised PnL $< -$$10M in any rolling 5-min window, cancel all open orders and flatten positions at VWAP over the next 15 min (not panic-sell). (2) **Position velocity**: if net delta changes by $>$ $50M (10% of AUM) in $<$ 60 seconds, halt new orders and page the risk desk. **Latency budget**: 1-second recompute is achievable on a standard kdb+ or Flink pipeline. Would have stopped Knight at the $50M mark, i.e. within $\sim$5 minutes, saving $390M. Second-order: these layers must themselves be killable – a corrupt layer-2 process cannot become a new failure mode.

Q3. Layer 3 – Post-trade: name 1 reconciliation that runs nightly (or T+5min in a real-time shop) and what it would have caught at Knight. Then give **one argument** why the 3-layer design is *still insufficient* on its own.

Solution. Reconciliation: positions-on-book vs. exchange drop-copy, flagging any ticker/quantity mismatch. Knight's dormant SMARS code was routing on the *old* order flag mapping; a T+5 drop-copy reconciliation would have shown a mismatch in the first reporting window (within minutes). **Still insufficient**: (i) **deployment discipline**: Knight's root cause was a 1-of-8-server config drift during a routine release – no kill switch fires on a pre-planned deployment with hard-coded limits that are themselves legacy code. You *also* need (a) an "is-this-deployment-legal" gate before go-live, (b) canary deploy to a single server with $100k cap for 15 min, and (c) one-click rollback with human approval. Kill switches stop run-time failures; process failures require process controls.