```
import { useState, useEffect, useRef, useCallback } from "react";

// ============================================================
// CONSTANTS & CONFIG
// ============================================================
const MODEL = "claude-sonnet-4-20250514";
const MAX_TOKENS = 1000;

const AGENTS = {
  MACRO: {
    id: "MACRO",
    name: "Macro Analysis",
    role: "macro_analyst",
    color: "#00ff88",
    icon: "◈",
    desc: "Regime detection & directional bias",
  },
  SENTIMENT: {
    id: "SENTIMENT",
    name: "Sentiment",
    role: "sentiment_analyst",
    color: "#ff9500",
    icon: "◎",
    desc: "News & social signal extraction",
  },
  TECHNICAL: {
    id: "TECHNICAL",
    name: "Technical",
    role: "technical_analyst",
    color: "#00cfff",
    icon: "◇",
    desc: "Pattern & indicator analysis",
  },
  RISK: {
    id: "RISK",
    name: "Risk Manager",
    role: "risk_manager",
    color: "#ff4466",
    icon: "◆",
    desc: "Drawdown, CVaR & position sizing",
  },
  DEBATE: {
    id: "DEBATE",
    name: "Debate Coordinator",
    role: "debate_coordinator",
    color: "#cc88ff",
    icon: "◉",
    desc: "Synthesizes agent disagreements",
  },
  DECISION: {
    id: "DECISION",
    name: "Decision Agent",
    role: "decision_agent",
    color: "#ffee00",
    icon: "★",
    desc: "Final execution signal",
  },
};

const AGENT_ORDER = ["MACRO", "SENTIMENT", "TECHNICAL", "RISK", "DEBATE", "DECISI

const STATUS = { IDLE: "IDLE", THINKING: "THINKING", DONE: "DONE", ERROR: "ERROR"

// ============================================================
// API LAYER
// ============================================================
async function callAgent(agentId, ticker, assetClass, context, memory) {
  const agent = AGENTS[agentId];
  const systemPrompts = {
    macro_analyst: `You are a Macro Analysis Agent in an institutional multi-agen
You specialize in market regime detection, trend identification, and directional
Analyze the asset from a macro perspective: current market regime (trending/rangi
key support/resistance levels, broader market context, and your directional bias
Be quantitative and specific. Mention specific price levels, percentages, and tim
Output: 3-4 sentences of analysis, then SIGNAL: [BULLISH/BEARISH/NEUTRAL] with co

    sentiment_analyst: `You are a Sentiment Analysis Agent in an institutional mu
You specialize in extracting signal from news flow, social sentiment, funding rat
Analyze current sentiment landscape for this asset: news sentiment, retail vs ins
fear/greed indicators, funding rates if crypto, options skew if equity.
Be specific about sentiment indicators and their implications.
Output: 3-4 sentences of analysis, then SIGNAL: [BULLISH/BEARISH/NEUTRAL] with co

    technical_analyst: `You are a Technical Analysis Agent in an institutional mu
You specialize in price action, technical indicators, and pattern recognition.
Analyze: key technical levels, momentum indicators (RSI, MACD), volume profile,
moving average structure, any classical patterns forming.
Be specific about price levels and indicator readings.
Output: 3-4 sentences of analysis, then SIGNAL: [BULLISH/BEARISH/NEUTRAL] with co

    risk_manager: `You are a Risk Management Agent in an institutional multi-agen
```

```
You specialize in position sizing, drawdown control, CVaR estimation, and stop-lo
Given the signals from other agents: ${context}
Assess: current risk environment, recommended position size (% of portfolio),
stop-loss level, take-profit target, max drawdown tolerance, and overall risk rat
Output: 3-4 sentences of risk assessment, then RISK_RATING: [LOW/MEDIUM/HIGH/EXTR

    debate_coordinator: `You are a Debate Coordinator Agent in an institutional m
Your role is to synthesize conflicting agent views and resolve disagreements thro
Agent signals received: ${context}
Identify the key points of agreement and disagreement between agents.
Weight each agent's signal by their confidence and the current market regime.
Resolve conflicts using regime-appropriate logic (e.g., in high volatility, weigh
Output: 3-4 sentences synthesizing the debate, then CONSENSUS: [BULLISH/BEARISH/N

    decision_agent: `You are the Final Decision Agent in an institutional multi-a
You receive the synthesized consensus and make the final executable trading decis
Full agent context: ${context}
Memory of recent decisions: ${memory}
Make a final, actionable trading decision. Specify: action (BUY/SELL/HOLD),
entry rationale, position size confirmation, stop-loss, take-profit, and time hor
Be decisive and specific. This is the final execution signal.
Output: 3-4 sentences of reasoning, then DECISION: [BUY/SELL/HOLD] with specific
  };

  const userMessage = `Asset: ${ticker} | Class: ${assetClass} | Timestamp: ${new
${agentId === "MACRO" || agentId === "SENTIMENT" || agentId === "TECHNICAL" ? "Pr
Provide your analysis now.`;

  const response = await fetch("https://api.anthropic.com/v1/messages", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      model: MODEL,
      max_tokens: MAX_TOKENS,
      system: systemPrompts[agent.role],
      messages: [{ role: "user", content: userMessage }],
    }),
  });

  if (!response.ok) throw new Error(`API error: ${response.status}`);
  const data = await response.json();
  return data.content[0].text;
}

// ============================================================
// MEMORY MODULE
// ============================================================
function MemoryModule({ memory }) {
  if (!memory.length) return null;
  return (
    <div style={styles.memoryPanel}>
      <div style={styles.memoryHeader}>
        <span style={styles.memoryTitle}>◈ LAYERED MEMORY</span>
        <span style={styles.memoryCount}>{memory.length} decisions</span>
      </div>
      <div style={styles.memoryLayers}>
        {["SHORT", "MID", "LONG"].map((layer, li) => {
          const layerMemory = memory.filter((_, i) => {
            if (layer === "SHORT") return i >= memory.length - 2;
            if (layer === "MID") return i >= memory.length - 5 && i < memory.leng
            return i < memory.length - 5;
          });
          const decay = layer === "SHORT" ? 1.0 : layer === "MID" ? 0.6 : 0.3;
          return (
            <div key={layer} style={styles.memoryLayer}>
              <div style={{ ...styles.memoryLayerLabel, opacity: decay }}>
                {layer}-TERM ({layerMemory.length})
              </div>
              {layerMemory.slice(-2).map((m, i) => (
                <div key={i} style={{ ...styles.memoryItem, opacity: decay }}>
                  <span style={{ color: m.decision === "BUY" ? "#00ff88" : m.deci
                    {m.decision}
                  </span>
                  <span style={styles.memoryTicker}>{m.ticker}</span>
                  <span style={styles.memoryTime}>{m.time}</span>
                </div>
              ))}
            </div>
          );
        })}
      </div>
    </div>
  );
}

// ============================================================
// AGENT CARD
// ============================================================
function AgentCard({ agentId, status, output, isActive }) {
  const agent = AGENTS[agentId];
  const cardRef = useRef(null);

  useEffect(() => {
    if (status === STATUS.DONE && cardRef.current) {
      cardRef.current.scrollIntoView({ behavior: "smooth", block: "nearest" });
    }
  }, [status]);

  const extractSignal = (text) => {
    const sigMatch = text?.match(/SIGNAL:\s*\[(BULLISH|BEARISH|NEUTRAL)\]/);
    const decMatch = text?.match(/DECISION:\s*\[(BUY|SELL|HOLD)\]/);
    const riskMatch = text?.match(/RISK_RATING:\s*\[(LOW|MEDIUM|HIGH|EXTREME)\]/)
    const consMatch = text?.match(/CONSENSUS:\s*\[(BULLISH|BEARISH|NEUTRAL)\]/);
    const confMatch = text?.match(/confidence\s+(\d+)/i);
    return {
      signal: sigMatch?.[1] || decMatch?.[1] || riskMatch?.[1] || consMatch?.[1],
      confidence: confMatch?.[1],
    };
  };

  const { signal, confidence } = output ? extractSignal(output) : {};

  const signalColor = {
    BULLISH: "#00ff88", BUY: "#00ff88",
    BEARISH: "#ff4466", SELL: "#ff4466",
    NEUTRAL: "#888888", HOLD: "#ffee00",
    LOW: "#00ff88", MEDIUM: "#ffee00",
    HIGH: "#ff9500", EXTREME: "#ff4466",
  }[signal] || agent.color;

  return (
    <div ref={cardRef} style={{
      ...styles.agentCard,
      borderColor: isActive ? agent.color : status === STATUS.DONE ? `${agent.col
      boxShadow: isActive ? `0 0 20px ${agent.color}33` : "none",
      opacity: status === STATUS.IDLE ? 0.4 : 1,
      transition: "all 0.3s ease",
    }}>
      <div style={styles.agentHeader}>
        <div style={styles.agentLeft}>
          <span style={{ ...styles.agentIcon, color: agent.color }}>{agent.icon}<
          <div>
            <div style={{ ...styles.agentName, color: agent.color }}>{agent.name}
            <div style={styles.agentDesc}>{agent.desc}</div>
          </div>
        </div>
        <div style={styles.agentRight}>
          {status === STATUS.THINKING && (
            <div style={styles.thinkingIndicator}>
              <span style={{ color: agent.color }}>PROCESSING</span>
              <ThinkingDots color={agent.color} />
            </div>
          )}
          {status === STATUS.DONE && signal && (
            <div style={{ ...styles.signalBadge, background: `${signalColor}22`,
              {signal} {confidence && `${confidence}%`}
            </div>
          )}
          {status === STATUS.ERROR && (
            <div style={{ ...styles.signalBadge, background: "#ff446622", borderC
              ERROR
            </div>
          )}
        </div>
      </div>
      {output && status === STATUS.DONE && (
        <div style={styles.agentOutput}>
          <div style={styles.outputText}>{cleanOutput(output)}</div>
        </div>
      )}
    </div>
  );
}

function cleanOutput(text) {
  return text.replace(/SIGNAL:\s*\[.*?\]/g, "").replace(/DECISION:\s*\[.*?\]/g, "
    .replace(/RISK_RATING:\s*\[.*?\]/g, "").replace(/CONSENSUS:\s*\[.*?\]/g, "").
}

function ThinkingDots({ color }) {
  const [frame, setFrame] = useState(0);
  useEffect(() => {
    const t = setInterval(() => setFrame(f => (f + 1) % 4), 300);
    return () => clearInterval(t);
  }, []);
  return <span style={{ color }}>{"●".repeat(frame)}{"○".repeat(3 - frame)}</span
}

// ============================================================
// DATA FLOW VISUALIZER
// ============================================================
function DataFlowBar({ activeAgent, agentStatuses }) {
  const nodes = AGENT_ORDER;
  return (
    <div style={styles.flowBar}>
      {nodes.map((id, i) => {
        const agent = AGENTS[id];
        const st = agentStatuses[id];
        const isActive = activeAgent === id;
        const isDone = st === STATUS.DONE;
        return (
          <div key={id} style={styles.flowNode}>
            <div style={{
              ...styles.flowDot,
              background: isDone ? agent.color : isActive ? agent.color : "#1a1a2
              borderColor: agent.color,
              boxShadow: isActive ? `0 0 12px ${agent.color}` : isDone ? `0 0 6px
              transform: isActive ? "scale(1.3)" : "scale(1)",
              transition: "all 0.3s ease",
            }}>
              {agent.icon}
            </div>
            <div style={{ ...styles.flowLabel, color: isDone ? agent.color : isAc
              {agent.name.split(" ")[0]}
            </div>
            {i < nodes.length - 1 && (
              <div style={{
                ...styles.flowArrow,
                color: isDone ? "#444" : "#222",
              }}>→</div>
            )}
          </div>
        );
      })}
    </div>
  );
}

// ============================================================
// FINAL DECISION DISPLAY
// ============================================================
function FinalDecision({ output, ticker }) {
  if (!output) return null;
  const decMatch = output.match(/DECISION:\s*\[(BUY|SELL|HOLD)\]/);
  const decision = decMatch?.[1];
  const color = { BUY: "#00ff88", SELL: "#ff4466", HOLD: "#ffee00" }[decision] ||

  return (
    <div style={{ ...styles.finalDecision, borderColor: color, boxShadow: `0 0 30
      <div style={styles.finalHeader}>
        <span style={styles.finalLabel}>EXECUTION SIGNAL</span>
        <span style={{ ...styles.finalTicker, color }}>{ticker}</span>
      </div>
      <div style={{ ...styles.finalDecisionText, color }}>
        {decision || "PROCESSING"}
      </div>
      <div style={styles.finalRationale}>{cleanOutput(output)}</div>
    </div>
  );
}

// ================================================================
// TRADE LOG
// ================================================================
function TradeLog({ trades }) {
  if (!trades.length) return null;
  return (
    <div style={styles.tradeLog}>
      <div style={styles.tradeLogHeader}>◈ DECISION LOG</div>
      <div style={styles.tradeLogList}>
        {[...trades].reverse().map((t, i) => (
          <div key={i} style={styles.tradeLogItem}>
            <span style={{ color: { BUY: "#00ff88", SELL: "#ff4466", HOLD: "#ffee
              {t.decision}
            </span>
            <span style={styles.tradeLogTicker}>{t.ticker}</span>
            <span style={styles.tradeLogClass}>{t.assetClass}</span>
            <span style={styles.tradeLogTime}>{t.time}</span>
          </div>
        ))}
      </div>
    </div>
  );
}

// ================================================================
// MAIN APP
// ================================================================
export default function TradingSystem() {
  const [ticker, setTicker] = useState("BTC/USD");
  const [assetClass, setAssetClass] = useState("Cryptocurrency");
  const [isRunning, setIsRunning] = useState(false);
  const [agentStatuses, setAgentStatuses] = useState(
    Object.fromEntries(AGENT_ORDER.map(id => [id, STATUS.IDLE]))
  );
  const [agentOutputs, setAgentOutputs] = useState(
    Object.fromEntries(AGENT_ORDER.map(id => [id, null]))
  );
  const [activeAgent, setActiveAgent] = useState(null);
  const [memory, setMemory] = useState([]);
  const [trades, setTrades] = useState([]);
  const [finalOutput, setFinalOutput] = useState(null);
  const [systemLog, setSystemLog] = useState([]);

  const log = useCallback((msg) => {
    setSystemLog(prev => [...prev.slice(-20), { time: new Date().toLocaleTimeStri
  }, []);

  const runSystem = useCallback(async () => {
    if (isRunning || !ticker.trim()) return;
    setIsRunning(true);
    setFinalOutput(null);
    setActiveAgent(null);

    const resetStatuses = Object.fromEntries(AGENT_ORDER.map(id => [id, STATUS.ID
    const resetOutputs = Object.fromEntries(AGENT_ORDER.map(id => [id, null]));
    setAgentStatuses(resetStatuses);
    setAgentOutputs(resetOutputs);

    log(`SYSTEM INIT: Analyzing ${ticker} [${assetClass}]`);

    const outputs = {};
    const memoryStr = memory.slice(-3).map(m => `${m.time}: ${m.decision} ${m.tic

    for (const agentId of AGENT_ORDER) {
      setActiveAgent(agentId);
      setAgentStatuses(prev => ({ ...prev, [agentId]: STATUS.THINKING }));
      log(`AGENT ${agentId}: Initializing...`);

      try {
        const contextStr = Object.entries(outputs)
          .map(([id, out]) => `[${AGENTS[id].name}]: ${out?.slice(0, 300)}`)
          .join("\n\n");

        const result = await callAgent(agentId, ticker, assetClass, contextStr, m
        outputs[agentId] = result;

        setAgentOutputs(prev => ({ ...prev, [agentId]: result }));
        setAgentStatuses(prev => ({ ...prev, [agentId]: STATUS.DONE }));
        log(`AGENT ${agentId}: Analysis complete`);

        if (agentId === "DECISION") {
          setFinalOutput(result);
          const decMatch = result.match(/DECISION:\s*\[(BUY|SELL|HOLD)\]/);
          const decision = decMatch?.[1] || "HOLD";
          const timestamp = new Date().toLocaleTimeString();
          const entry = { ticker, assetClass, decision, time: timestamp };
          setMemory(prev => [...prev.slice(-9), entry]);
          setTrades(prev => [...prev, entry]);
          log(`DECISION: ${decision} ${ticker} at ${timestamp}`);
        }

        await new Promise(r => setTimeout(r, 300));
      } catch (err) {
        setAgentStatuses(prev => ({ ...prev, [agentId]: STATUS.ERROR }));
        log(`AGENT ${agentId}: ERROR - ${err.message}`);
      }
    }

    setActiveAgent(null);
    setIsRunning(false);
    log("SYSTEM: Analysis cycle complete");
  }, [ticker, assetClass, isRunning, memory, log]);

  const logRef = useRef(null);
  useEffect(() => {
    if (logRef.current) logRef.current.scrollTop = logRef.current.scrollHeight;
  }, [systemLog]);

  return (
    <div style={styles.root}>
      {/* SCANLINE OVERLAY */}
      <div style={styles.scanlines} />

      {/* HEADER */}
      <div style={styles.header}>
        <div style={styles.headerLeft}>
          <div style={styles.logo}>◈ APEX</div>
          <div style={styles.logoSub}>MULTI-AGENT TRADING SYSTEM</div>
        </div>
        <div style={styles.headerCenter}>
          <div style={styles.statusLight} className={isRunning ? "pulse" : ""} />
          <span style={{ color: isRunning ? "#00ff88" : "#444", fontSize: 11, fon
            {isRunning ? "SYSTEM ACTIVE" : "STANDBY"}
          </span>
        </div>
        <div style={styles.headerRight}>
          <span style={styles.headerTime}>{new Date().toLocaleString()}</span>
        </div>
      </div>

      {/* FLOW BAR */}
      <DataFlowBar activeAgent={activeAgent} agentStatuses={agentStatuses} />

      {/* MAIN GRID */}
      <div style={styles.mainGrid}>
```

```
{/* LEFT COLUMN */}
<div style={styles.leftCol}>
  {/* CONTROL PANEL */}
  <div style={styles.controlPanel}>
    <div style={styles.controlHeader}>◈ ANALYSIS TARGET</div>
    <div style={styles.controlRow}>
      <div style={styles.inputGroup}>
        <label style={styles.inputLabel}>TICKER</label>
        <input
          style={styles.input}
          value={ticker}
          onChange={e => setTicker(e.target.value.toUpperCase())}
          placeholder="BTC/USD"
          disabled={isRunning}
        />
      </div>
      <div style={styles.inputGroup}>
        <label style={styles.inputLabel}>ASSET CLASS</label>
        <select
          style={styles.select}
          value={assetClass}
          onChange={e => setAssetClass(e.target.value)}
          disabled={isRunning}
        >
          <option>Cryptocurrency</option>
          <option>Equity</option>
          <option>Forex</option>
          <option>Commodity</option>
          <option>Index</option>
        </select>
      </div>
    </div>
    <button
      style={{
        ...styles.runButton,
        background: isRunning ? "#1a1a2e" : "#00ff8822",
        borderColor: isRunning ? "#333" : "#00ff88",
        color: isRunning ? "#444" : "#00ff88",
        cursor: isRunning ? "not-allowed" : "pointer",
      }}
      onClick={runSystem}
      disabled={isRunning}
    >
      {isRunning ? "◈ AGENTS ACTIVE..." : "◈ RUN ANALYSIS"}
    </button>
  </div>
```

```
        {/* MEMORY MODULE */}
        <MemoryModule memory={memory} />

        {/* SYSTEM LOG */}
        <div style={styles.sysLog}>
          <div style={styles.sysLogHeader}>◈ SYSTEM LOG</div>
          <div ref={logRef} style={styles.sysLogContent}>
            {systemLog.map((entry, i) => (
              <div key={i} style={styles.sysLogEntry}>
                <span style={styles.sysLogTime}>{entry.time}</span>
                <span style={styles.sysLogMsg}>{entry.msg}</span>
              </div>
            ))}
            {!systemLog.length && (
              <div style={styles.sysLogEmpty}>Awaiting system initialization...
            )}
          </div>
        </div>

        {/* TRADE LOG */}
        <TradeLog trades={trades} />
      </div>

      {/* RIGHT COLUMN - AGENTS */}
      <div style={styles.rightCol}>
        <div style={styles.agentsHeader}>◈ AGENT NETWORK</div>
        <div style={styles.agentGrid}>
          {AGENT_ORDER.map(id => (
            <AgentCard
              key={id}
              agentId={id}
              status={agentStatuses[id]}
              output={agentOutputs[id]}
              isActive={activeAgent === id}
            />
          ))}
        </div>

        {/* FINAL DECISION */}
        {finalOutput && (
          <FinalDecision output={finalOutput} ticker={ticker} />
        )}
      </div>
    </div>

    <style>{`
      @import url('https://fonts.googleapis.com/css2?family=Share+Tech+Mono&fam
        * { box-sizing: border-box; }
        ::-webkit-scrollbar { width: 4px; }
        ::-webkit-scrollbar-track { background: #0a0a12; }
        ::-webkit-scrollbar-thumb { background: #00ff8844; }
        @keyframes pulse { 0%,100%{opacity:1;box-shadow:0 0 8px #00ff88} 50%{opac
        @keyframes scanMove { 0%{transform:translateY(-100%)} 100%{transform:tran
        .pulse { animation: pulse 1.2s infinite; }
        select option { background: #0a0a12; color: #00ff88; }
      `}</style>
    </div>
  );
}

// ============================================================
// STYLES
// ============================================================
const styles = {
  root: {
    background: "#06060f",
    minHeight: "100vh",
    fontFamily: "'Barlow Condensed', sans-serif",
    color: "#c8c8d0",
    position: "relative",
    overflow: "hidden",
  },
  scanlines: {
    position: "fixed",
    top: 0, left: 0, right: 0, bottom: 0,
    backgroundImage: "repeating-linear-gradient(0deg, transparent, transparent 2p
    pointerEvents: "none",
    zIndex: 1000,
  },
  header: {
    display: "flex",
    alignItems: "center",
    justifyContent: "space-between",
    padding: "12px 24px",
    borderBottom: "1px solid #00ff8822",
    background: "#06060f",
    position: "sticky",
    top: 0,
    zIndex: 100,
  },
  headerLeft: { display: "flex", alignItems: "baseline", gap: 10 },
  logo: {
    fontFamily: "'Share Tech Mono', monospace",
    fontSize: 22,
    color: "#00ff88",
    letterSpacing: 4,
    fontWeight: 700,
  },
  logoSub: {
    fontFamily: "'Share Tech Mono', monospace",
    fontSize: 9,
    color: "#00ff8866",
    letterSpacing: 3,
  },
  headerCenter: { display: "flex", alignItems: "center", gap: 8 },
  statusLight: {
    width: 8, height: 8, borderRadius: "50%",
    background: "#00ff88",
    boxShadow: "0 0 8px #00ff88",
  },
  headerRight: {},
  headerTime: {
    fontFamily: "'Share Tech Mono', monospace",
    fontSize: 10,
    color: "#444",
  },
  flowBar: {
    display: "flex",
    alignItems: "center",
    justifyContent: "center",
    padding: "12px 24px",
    background: "#08080f",
    borderBottom: "1px solid #1a1a2e",
    gap: 0,
    overflowX: "auto",
  },
  flowNode: {
    display: "flex",
    alignItems: "center",
    gap: 4,
  },
  flowDot: {
    width: 32, height: 32,
    borderRadius: "50%",
    border: "1px solid",
    display: "flex",
    alignItems: "center",
    justifyContent: "center",
    fontFamily: "'Share Tech Mono', monospace",
    fontSize: 14,
    cursor: "default",
    },
    flowLabel: {
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 8,
      letterSpacing: 1,
      marginTop: 2,
      textAlign: "center",
      width: 32,
      display: "none",
    },
    flowArrow: {
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 16,
      padding: "0 8px",
      opacity: 0.4,
    },
    mainGrid: {
      display: "grid",
      gridTemplateColumns: "300px 1fr",
      gap: 0,
      minHeight: "calc(100vh - 100px)",
    },
    leftCol: {
      borderRight: "1px solid #1a1a2e",
      padding: 16,
      display: "flex",
      flexDirection: "column",
      gap: 12,
      overflowY: "auto",
      maxHeight: "calc(100vh - 100px)",
    },
    rightCol: {
      padding: 16,
      overflowY: "auto",
      maxHeight: "calc(100vh - 100px)",
    },
    controlPanel: {
      background: "#0a0a14",
      border: "1px solid #1a1a2e",
      borderRadius: 4,
      padding: 14,
    },
    controlHeader: {
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 10,
      color: "#00ff8888",
      letterSpacing: 2,
      marginBottom: 12,
    },
    controlRow: {
      display: "flex",
      gap: 8,
      marginBottom: 12,
    },
    inputGroup: { flex: 1 },
    inputLabel: {
      display: "block",
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 8,
      color: "#555",
      letterSpacing: 2,
      marginBottom: 4,
    },
    input: {
      width: "100%",
      background: "#06060f",
      border: "1px solid #1a1a2e",
      borderRadius: 2,
      color: "#00ff88",
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 13,
      padding: "6px 8px",
      outline: "none",
      letterSpacing: 1,
    },
    select: {
      width: "100%",
      background: "#06060f",
      border: "1px solid #1a1a2e",
      borderRadius: 2,
      color: "#00ff88",
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 11,
      padding: "6px 8px",
      outline: "none",
      letterSpacing: 1,
      cursor: "pointer",
    },
    runButton: {
      width: "100%",
      border: "1px solid",
      borderRadius: 2,
      padding: "10px 0",
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 12,
      letterSpacing: 3,
      transition: "all 0.2s ease",
    },
    memoryPanel: {
      background: "#0a0a14",
      border: "1px solid #1a1a2e",
      borderRadius: 4,
      padding: 12,
    },
    memoryHeader: {
      display: "flex",
      justifyContent: "space-between",
      alignItems: "center",
      marginBottom: 10,
    },
    memoryTitle: {
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 10,
      color: "#cc88ff88",
      letterSpacing: 2,
    },
    memoryCount: {
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 9,
      color: "#444",
    },
    memoryLayers: { display: "flex", flexDirection: "column", gap: 8 },
    memoryLayer: {
      borderLeft: "2px solid #1a1a2e",
      paddingLeft: 8,
    },
    memoryLayerLabel: {
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 8,
      color: "#cc88ff",
      letterSpacing: 2,
      marginBottom: 4,
    },
    memoryItem: {
      display: "flex",
      gap: 6,
      fontSize: 10,
      fontFamily: "'Share Tech Mono', monospace",
      alignItems: "center",
      marginBottom: 2,
    },
    memoryTicker: { color: "#888", fontSize: 9 },
    memoryTime: { color: "#444", fontSize: 9, marginLeft: "auto" },
    sysLog: {
      background: "#0a0a14",
      border: "1px solid #1a1a2e",
      borderRadius: 4,
      padding: 12,
      flex: 1,
      minHeight: 120,
      display: "flex",
      flexDirection: "column",
    },
    sysLogHeader: {
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 10,
      color: "#00ff8888",
      letterSpacing: 2,
      marginBottom: 8,
    },
    sysLogContent: {
      flex: 1,
      overflowY: "auto",
      maxHeight: 200,
    },
    sysLogEntry: {
      display: "flex",
      gap: 8,
      marginBottom: 3,
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 9,
    },
    sysLogTime: { color: "#444", flexShrink: 0 },
    sysLogMsg: { color: "#00ff8888" },
    sysLogEmpty: {
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 9,
      color: "#333",
      fontStyle: "italic",
    },
    tradeLog: {
      background: "#0a0a14",
      border: "1px solid #1a1a2e",
      borderRadius: 4,
      padding: 12,
    },
    tradeLogHeader: {
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 10,
      color: "#ffee0088",
      letterSpacing: 2,
      marginBottom: 8,
    },
    tradeLogList: { maxHeight: 150, overflowY: "auto" },
    tradeLogItem: {
      display: "flex",
      gap: 8,
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 10,
      alignItems: "center",
      marginBottom: 4,
      paddingBottom: 4,
      borderBottom: "1px solid #1a1a2e",
    },
    tradeLogTicker: { color: "#888" },
    tradeLogClass: { color: "#444", fontSize: 9 },
    tradeLogTime: { color: "#333", fontSize: 9, marginLeft: "auto" },
    agentsHeader: {
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 10,
      color: "#00ff8888",
      letterSpacing: 2,
      marginBottom: 12,
    },
    agentGrid: {
      display: "grid",
      gridTemplateColumns: "1fr 1fr",
      gap: 10,
      marginBottom: 16,
    },
    agentCard: {
      background: "#0a0a14",
      border: "1px solid",
      borderRadius: 4,
      padding: 12,
      transition: "all 0.3s ease",
    },
    agentHeader: {
      display: "flex",
      justifyContent: "space-between",
      alignItems: "flex-start",
      marginBottom: 6,
    },
    agentLeft: { display: "flex", gap: 8, alignItems: "flex-start" },
    agentIcon: {
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 18,
      lineHeight: 1,
    },
    agentName: {
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 11,
      letterSpacing: 1,
      fontWeight: 700,
    },
    agentDesc: {
      fontSize: 9,
      color: "#444",
      letterSpacing: 0.5,
      marginTop: 2,
      fontFamily: "'Share Tech Mono', monospace",
    },
    agentRight: { flexShrink: 0 },
    thinkingIndicator: {
      display: "flex",
      flexDirection: "column",
      alignItems: "flex-end",
      gap: 2,
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 8,
      letterSpacing: 1,
    },
    signalBadge: {
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 9,
      letterSpacing: 1,
      padding: "3px 6px",
      borderRadius: 2,
      border: "1px solid",
      fontWeight: 700,
    },
    agentOutput: {
      marginTop: 8,
      paddingTop: 8,
      borderTop: "1px solid #1a1a2e",
    },
    outputText: {
      fontSize: 10,
      lineHeight: 1.6,
      color: "#888",
      fontFamily: "'Share Tech Mono', monospace",
    },
    finalDecision: {
      border: "2px solid",
      borderRadius: 4,
      padding: 20,
      background: "#0a0a14",
      transition: "all 0.5s ease",
    },
    finalHeader: {
      display: "flex",
      justifyContent: "space-between",
      alignItems: "center",
      marginBottom: 8,
    },
    finalLabel: {
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 10,
      color: "#555",
      letterSpacing: 3,
    },
    finalTicker: {
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 13,
      letterSpacing: 2,
      fontWeight: 700,
    },
    finalDecisionText: {
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 36,
      fontWeight: 700,
      letterSpacing: 6,
      marginBottom: 12,
    },
    finalRationale: {
      fontFamily: "'Share Tech Mono', monospace",
      fontSize: 10,
      color: "#888",
      lineHeight: 1.7,
    },
  };
```