```
"use client";

import { useEffect, useMemo, useRef, useState } from "react";
import type { NodeId } from "../store/types";
import { useSureSailStore } from "../store/sureSailStore";

type Edge = { from: NodeId; to: NodeId; kind: "solid" | "dotted" };

export default function DependencyLinesLayer({
  orchestrator
}: {
  orchestrator: ReturnType<typeof import("../orchestrator/
useRunOrchestrator").useRunOrchestrator>;
}) {
  const dep = orchestrator.dependencyMap;
  const containerRef = useRef<HTMLDivElement | null>(null);
  const [rects, setRects] = useState<Record<string, DOMRect>>({});
  const [pulseKey, setPulseKey] = useState<string | null>(null);

  const stageIIUnlocked = useSureSailStore((s) => s.stageIIUnlocked);

  const edges: Edge[] = useMemo(() => {
    const list: Edge[] = [];
    (Object.keys(dep) as NodeId[]).forEach((from) => {
      dep[from].forEach((to) => {
        const kind: Edge["kind"] =
          from === "N_FOUNDATION_PROGRESS" ? "dotted" : "solid";
        list.push({ from, to, kind });
      });
    });
    return list;
  }, [dep]);

  // Register pulse emitter so orchestrator can pulse lines
  useEffect(() => {
    orchestrator.setPulseEdgeEmitter((from, to) => {
      setPulseKey(`${from}->${to}-${Date.now()}`);
      // clear after pulse
      setTimeout(() => setPulseKey(null), 480);
    });
  }, [orchestrator]);

  // Collect panel rects
  useEffect(() => {
    const collect = () => {
      const nodes = document.querySelectorAll("[data-nodeid]");
      const next: Record<string, DOMRect> = {};
      nodes.forEach((el) => {
        const id = (el as HTMLElement).dataset.nodeid!;
        next[id] = (el as HTMLElement).getBoundingClientRect();
      });
      setRects(next);
    };
    collect();
    const ro = new ResizeObserver(() => collect());
    ro.observe(document.body);
    window.addEventListener("scroll", collect, true);
    window.addEventListener("resize", collect);
    return () => {
      ro.disconnect();
      window.removeEventListener("scroll", collect, true);
      window.removeEventListener("resize", collect);
    };
  }, []);

  const svgPaths = edges
    .map((e) => {
      const a = rects[e.from];
      const b = rects[e.to];
      if (!a || !b) return null;

      // Convert to viewport-relative positions, then offset by scroll
      const ax = a.left + a.width * 0.98 + window.scrollX;
      const ay = a.top + a.height * 0.5 + window.scrollY;

      const bx = b.left + b.width * 0.02 + window.scrollX;
      const by = b.top + b.height * 0.5 + window.scrollY;

      const mx = (ax + bx) / 2;
      const path = `M ${ax} ${ay} C ${mx} ${ay}, ${mx} ${by}, ${bx} ${by}`;

      const isStageIIGate = e.from === "N_FOUNDATION_PROGRESS";
      const dimGate = isStageIIGate && !stageIIUnlocked;

      const isPulsing = pulseKey?.startsWith(`${e.from}->${e.to}`) ?? false;

      return { ...e, path, dimGate, isPulsing };
    })
    .filter(Boolean) as Array<Edge & { path: string; dimGate: boolean; isPulsing: boolean }>;

  return (
    <div ref={containerRef} className="depLayer">
      <svg width="100%" height="100%" style={{ position: "fixed", inset: 0 }}>
        {svgPaths.map((p) => {
          const stroke = p.dimGate ? "rgba(0,0,0,0.16)" : "rgba(0,0,0,0.20)";
          const pulseStroke = "rgba(0,255,255,0.55)";
          return (
            <path
              key={`${p.from}-${p.to}`}
              d={p.path}
              fill="none"
              stroke={p.isPulsing ? pulseStroke : stroke}
              strokeWidth={p.isPulsing ? 2.2 : 1.2}
              strokeDasharray={p.kind === "dotted" ? "6 6" : "0"}
              style={{ transition: "stroke 200ms ease, stroke-width 200ms ease", opacity: p.dimGate
? 0.6 : 0.9 }}
            />
          );
        })}
      </svg>
    </div>
  );
}
```