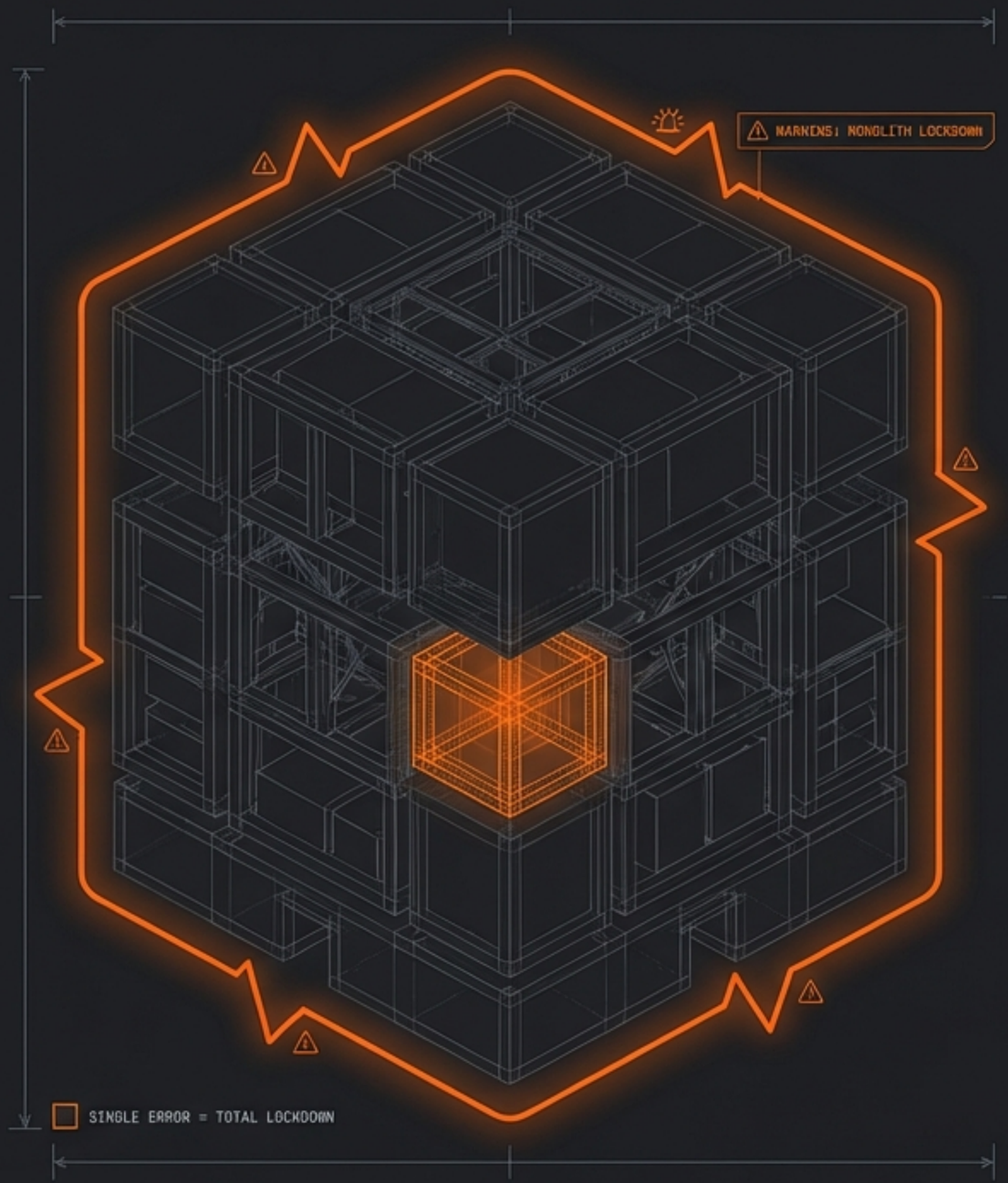
멀티레포(Multirepo) 기반 마이크로 프론트엔드 아키텍처

진정한 자율성과 확장성을 위한 엔지니어링 전략

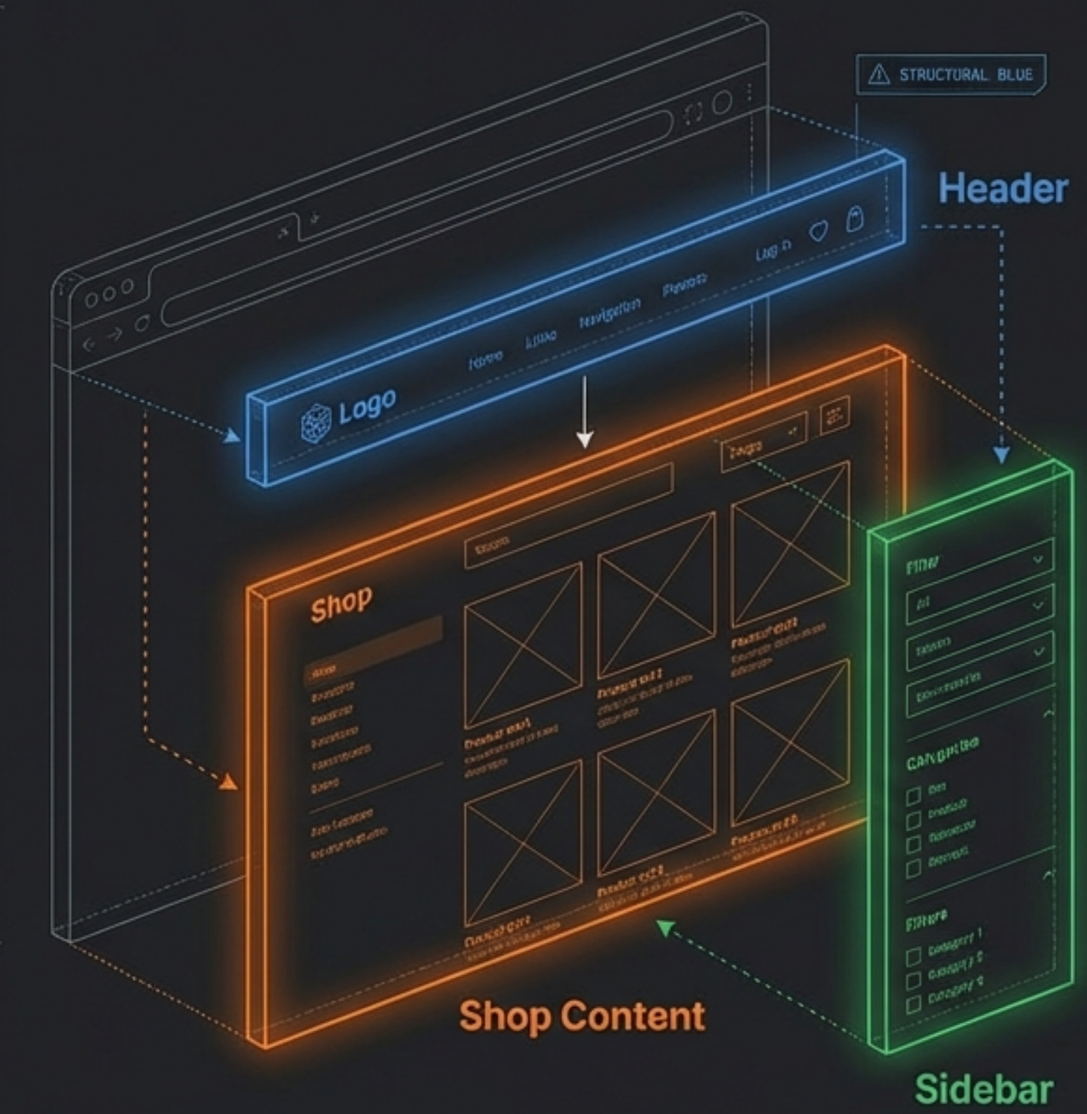
Architecture Strategy Series



모듈화된 코드, 그러나 결합된 프로세스

코드는 분리되었지만, 우리는 여전히 하나의 거대한 배에 타고 있습니다.

- **모놀리식(Monolithic)의 한계:** 작은 수정 사항도 전체 애플리케이션의 빌드와 배포를 요구함.
- **단일 오류 지점:** 하나의 컴포넌트 오류가 전체 시스템을 중단시킬 위험 (Single Point of Failure).
- **확장성 병목:** 서비스 규모가 커질수록 빌드 시간은 기하급수적으로 증가.



마이크로 프론트엔드: 독립적인 실행 단위

거대한 프론트엔드 애플리케이션을 비즈니스 도메인별로 나누어 독립적으로 개발, 배포, 동작하게 만드는 아키텍처.

- **Independent Deployment:** 배포의 독립성 보장
- **Technology Agnostic:** 팀별로 다른 프레임워크 사용 가능
- **Domain Driven:** 업무 영역별 분리

저장소 전략의 갈림길: 통합인가, 격리인가?

Monorepo

모노레포 (Monorepo): 따로 또 같이

- 하나의 저장소, 공유된 설정
- **단점:** 거대한 크기, 복잡한 권한

Multirepo

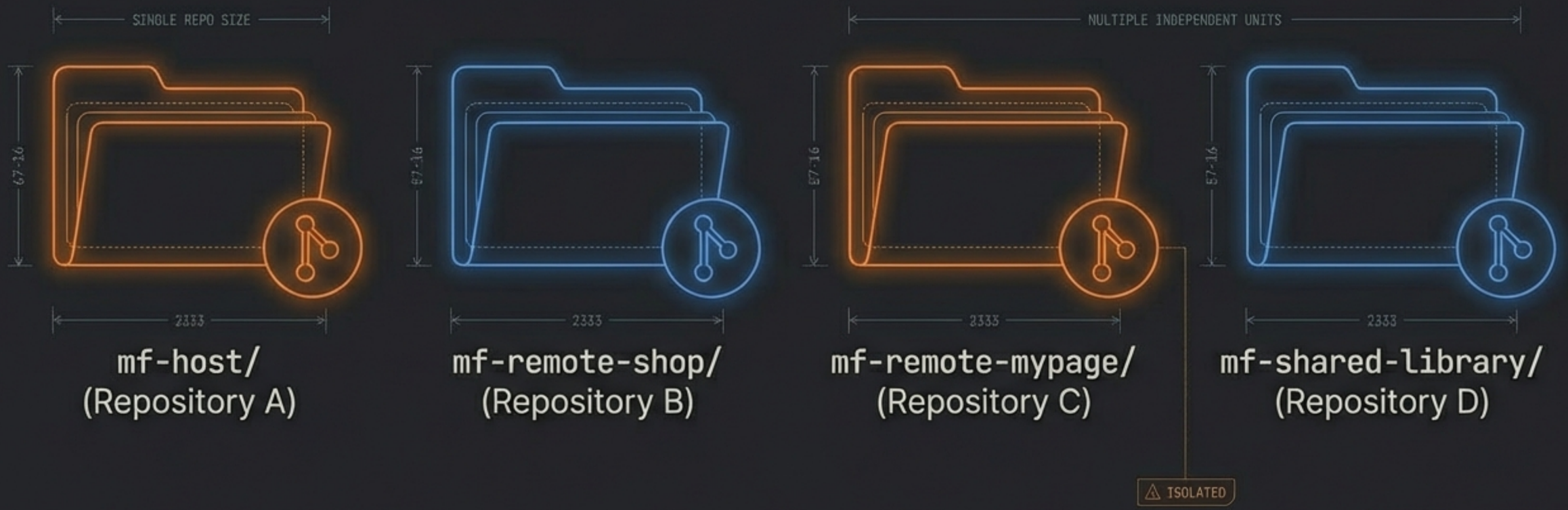
멀티레포 (Multirepo): 완전한 물리적 격리

- 서비스별 개별 저장소, 명확한 책임
- **단점:** 공통 라이브러리 관리 비용

ARCHITECTURAL PRECISION

멀티레포 구조의 물리적 아키텍처

각 프로젝트는 고유한 .git 디렉토리와 package.json을 가집니다.



mf-host/
(Repository A)

mf-remote-shop/
(Repository B)

mf-remote-mypage/
(Repository C)

mf-shared-library/
(Repository D)

ARCHITECTURAL
PRECISION

왜 멀티레포를 선택하는가? (핵심 가치)

완전한 독립성 (Complete Independence)

A팀의 커밋이 B팀의 빌드를 깨뜨릴 수 없음.
장애의 물리적 격리.

명확한 소유권 (Clear Ownership)

저장소 단위의 권한(Permission) 관리.
보안 및 접근 제어 용이.

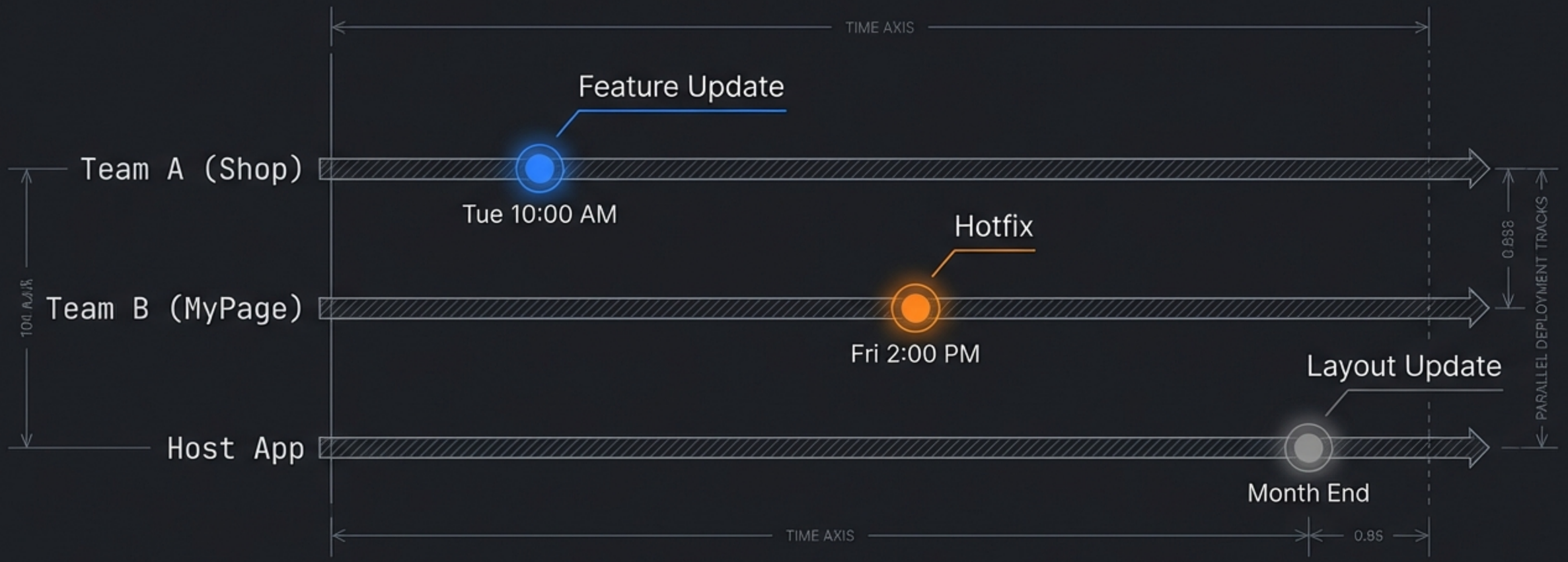
유연한 기술 스택

mf-remote-shop은 React 19를,
mf-remote-admin은 Vue를
사용할 수 있는 자유.

가벼운 저장소

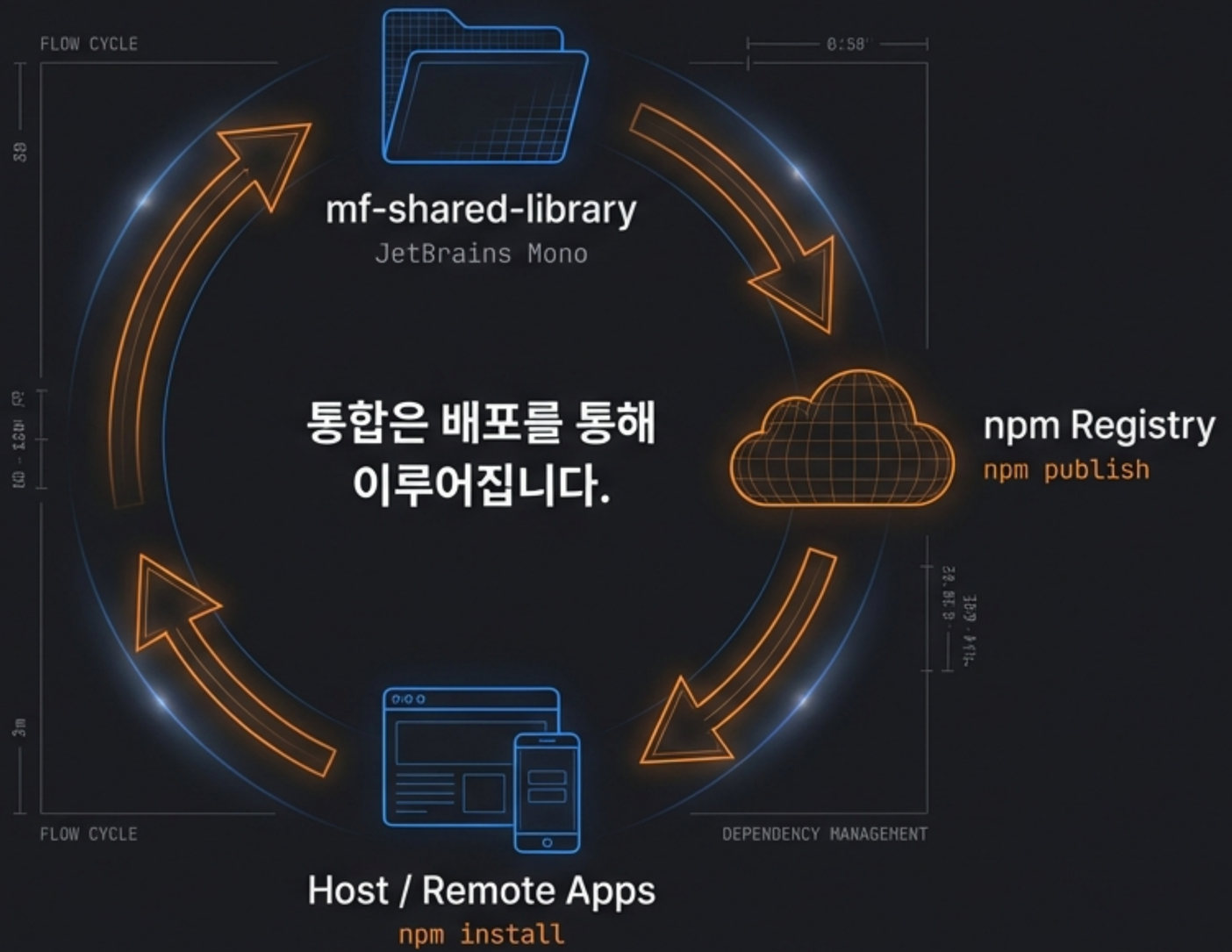
필요한 코드만 Clone하므로
Git 성능 이슈 없음.

자율성의 핵심: 독립적인 배포 파이프라인

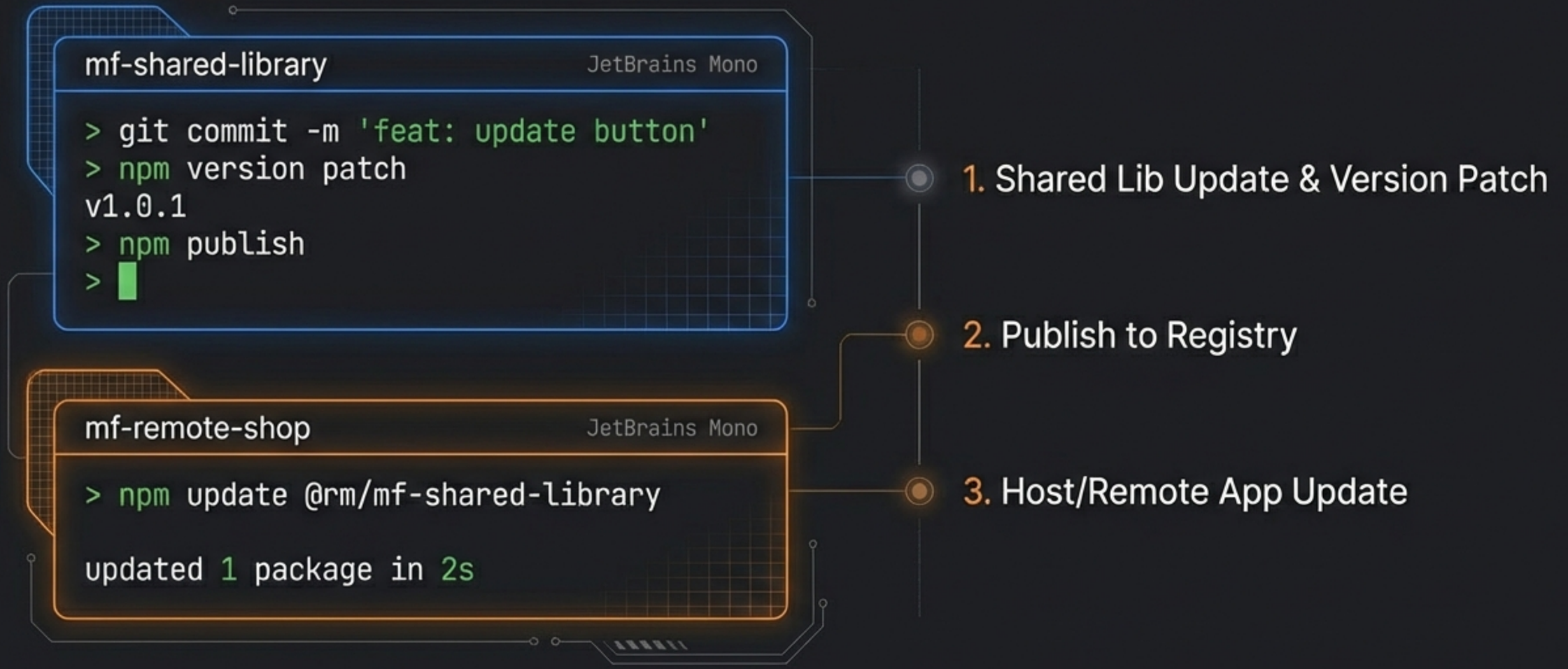


각 저장소는 별도의 CI/CD 파이프라인을 보유하며, 서로의 배포 일정에 영향을 주지 않습니다.

분리된 코드의 연결고리: npm 라이브러리 전략 (npm Library Strategy)



워크플로우: 라이브러리 배포와 소비



안전한 의존성 관리와 버전 전략

Major
(Breaking Changes
- 호환성 주의)

Minor
(New Features)

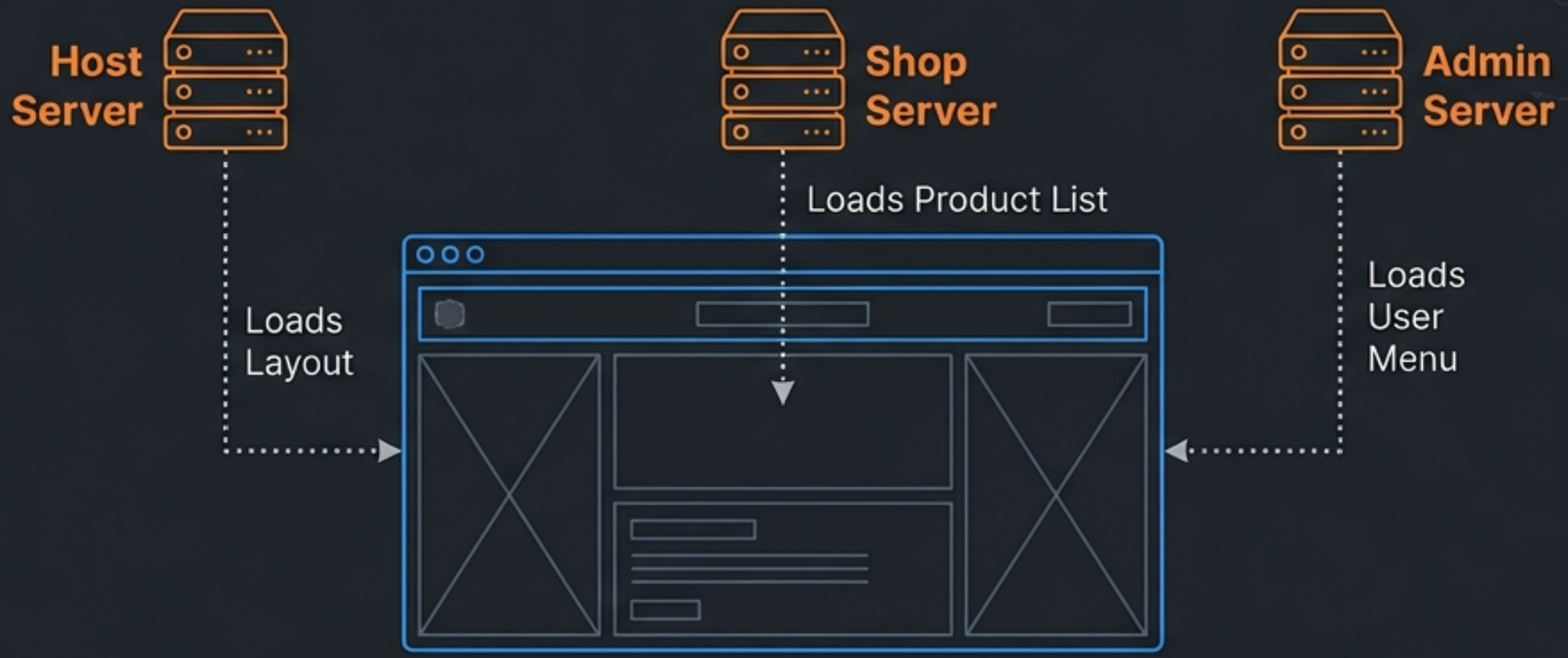
Patch
(Bug Fixes)

Semantic Versioning (SemVer) 준수

Peer Dependencies: react, react-dom 중복 로딩 방지 설정

Breaking Change 발생 시 Major 버전 업데이트 필수

런타임 통합: Module Federation




```
remotes: {
  shop: 'shop@http://.../remoteEntry.js',
  admin: 'admin@http://.../remoteEntry.js'
}
```

사용자는 단일 앱을 보지만, 브라우저는 런타임에 여러 원격 모듈을 조립합니다.

조직 구조가 아키텍처를 결정합니다 (Conway’s Law)

멀티레포가 적합한 조직 체크리스트

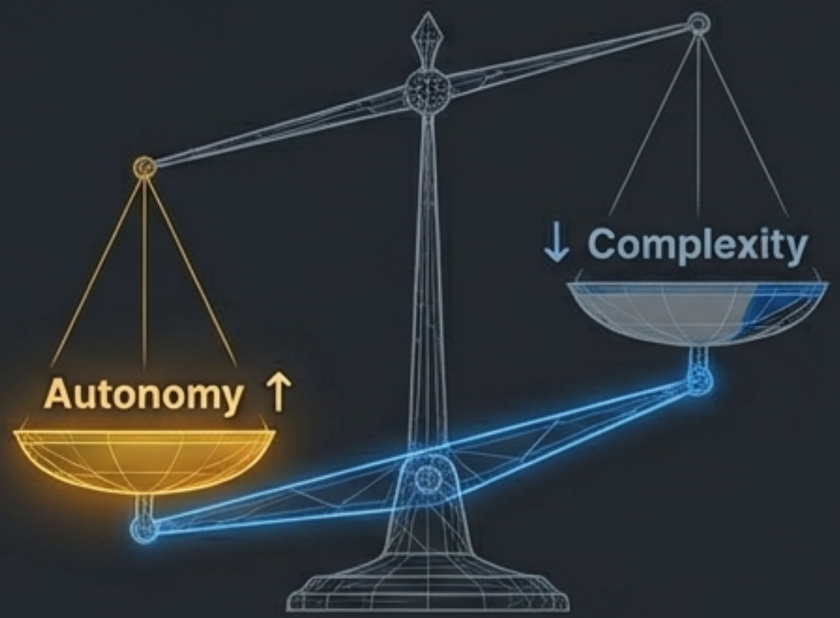
- ✔ **조직의 분리:** 부서나 자회사가 나뉘어 있어 소통 비용이 높은 경우
- ✔ **다양한 배포 주기:** 팀마다 배포 스케줄이 완전히 다른 경우
- ✔ **보안 요구사항:** 특정 소스코드의 접근 제어(Access Control)가 필요한 경우
- ✔ **기술 스택의 파편화:** 레거시와 모던 프레임워크가 공존해야 하는 경우
- ✔ **규모:** 개발자 **100**명 이상의 대규모 조직

비교 분석: 멀티레포 vs. 모노레포

Feature	멀티레포 (Multirepo)	모노레포 (Monorepo)
자율성 (Autonomy)	High (높음)	Mid (중간)
코드 일관성	Low (규약 필요)	High (강제 가능)
CI/CD	Independent (개별)	Unified (통합/복잡)
라이브러리 관리	npm Publish	Workspace Link

멀티레포는 '편의성'을 희생하여 '격리'와 '안정성'을 얻는 선택입니다.

도입 시 고려해야 할 트레이드오프 (Trade-offs)



- **프로젝트 생성 비용:** 저장소 및 파이프라인 초기 세팅
- **코드 중복:** 유틸리티 코드의 파편화 가능성
- **통합 테스트:** 분산된 환경에서의 E2E 테스트 복잡도
- **버전 파편화:** 런타임 의존성 충돌 위험

결론: 확장 가능한 미래를 위한 설계

멀티레포 MFE는 대규모 조직의 병목 현상을 제거하고
팀의 자율성을 극대화합니다.

당신의 조직은 **하나의 거대한 배입**니까, 아니면 **유연한 함대**입니까?