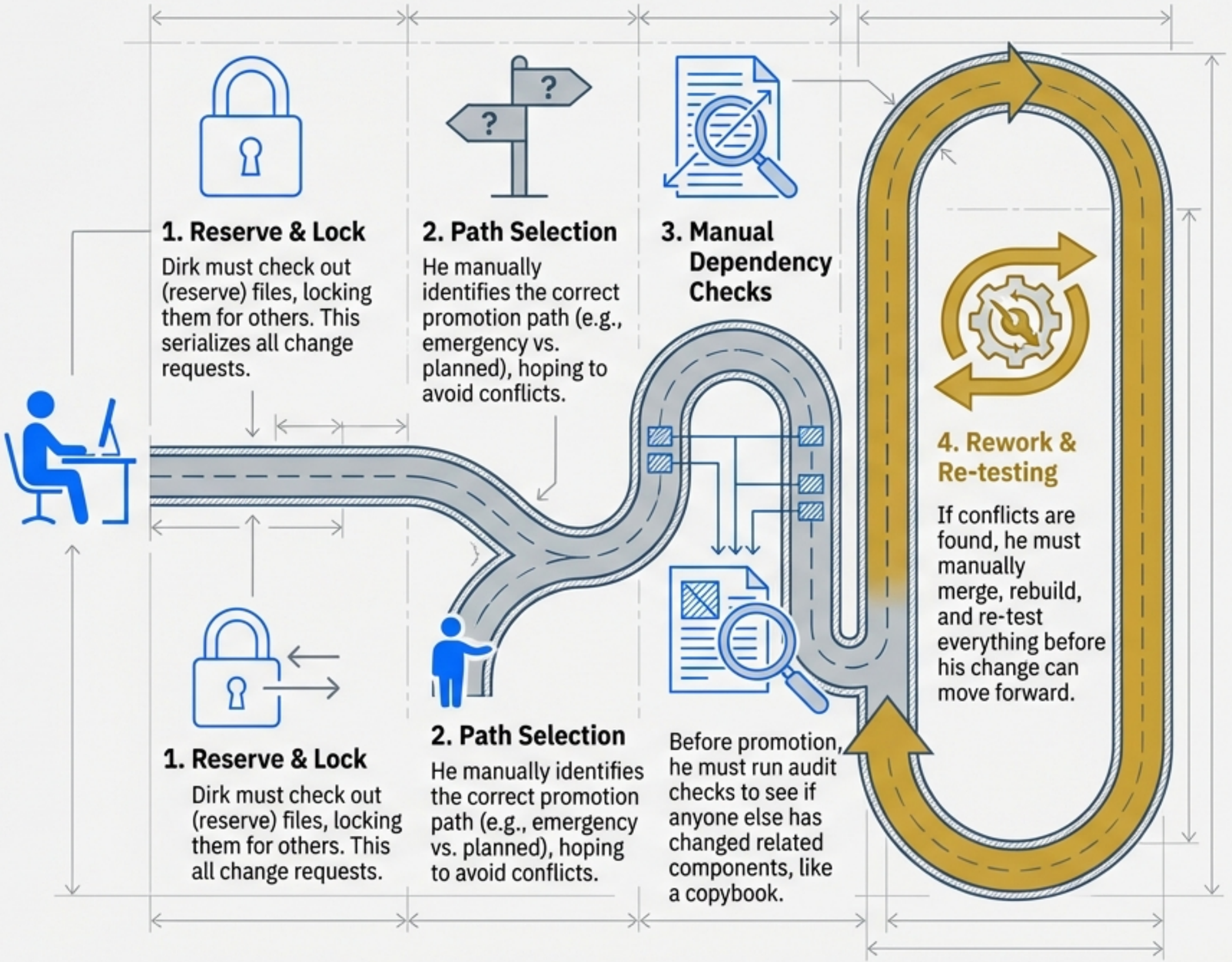
Modernizing Mainframe Development

From Serialized Workflows to Parallel DevOps

The Daily Struggle: A Developer's Life with Library Managers

Meet Dirk, a developer working with a traditional library manager. His workflow is serialized and manual, leading to frustration and delays.



The Hidden Costs of the Traditional Approach

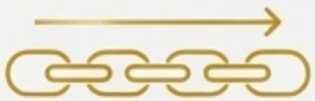
Why it persists:

- Mainframe development teams are accustomed to this approach.
- Does not require much disk space for developer workspaces.

The Real Impact:

Highly manual and repetitive tasks to check for dependencies.

Significant rework is required due to late decisions or conflicting changes.



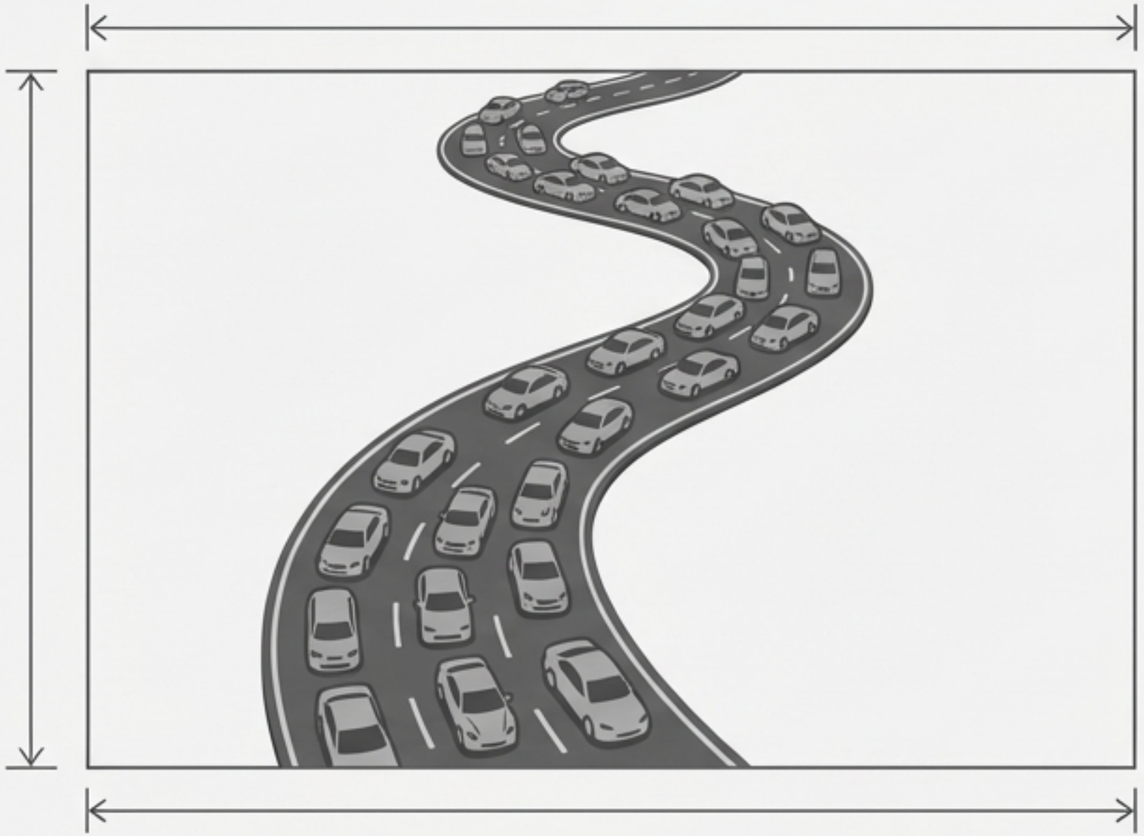
Serialization of work: Developers are forced to wait for each other.

Lack of true confutions and conflicting changes.

Dependencies to other activities in the staging hierarchy remain hidden until late in the process.

Lack of true parallel development limits agility and magnifies coordination overhead.

The Paradigm Shift: From Managing Files to Orchestrating Configurations



Old Way (File-Centric): Managing individual source files in libraries, relying on concatenation (SYSLIB) to find dependencies.

New Way (Configuration-Centric): Managing the *full application configuration*—a complete, versioned set of all necessary files—in isolated branches.

With a modern SCM, the entire configuration can be checked out together, supporting full isolation and true parallel development on the same artifacts.

The Foundation for Parallel Work: Git

Distributed Model

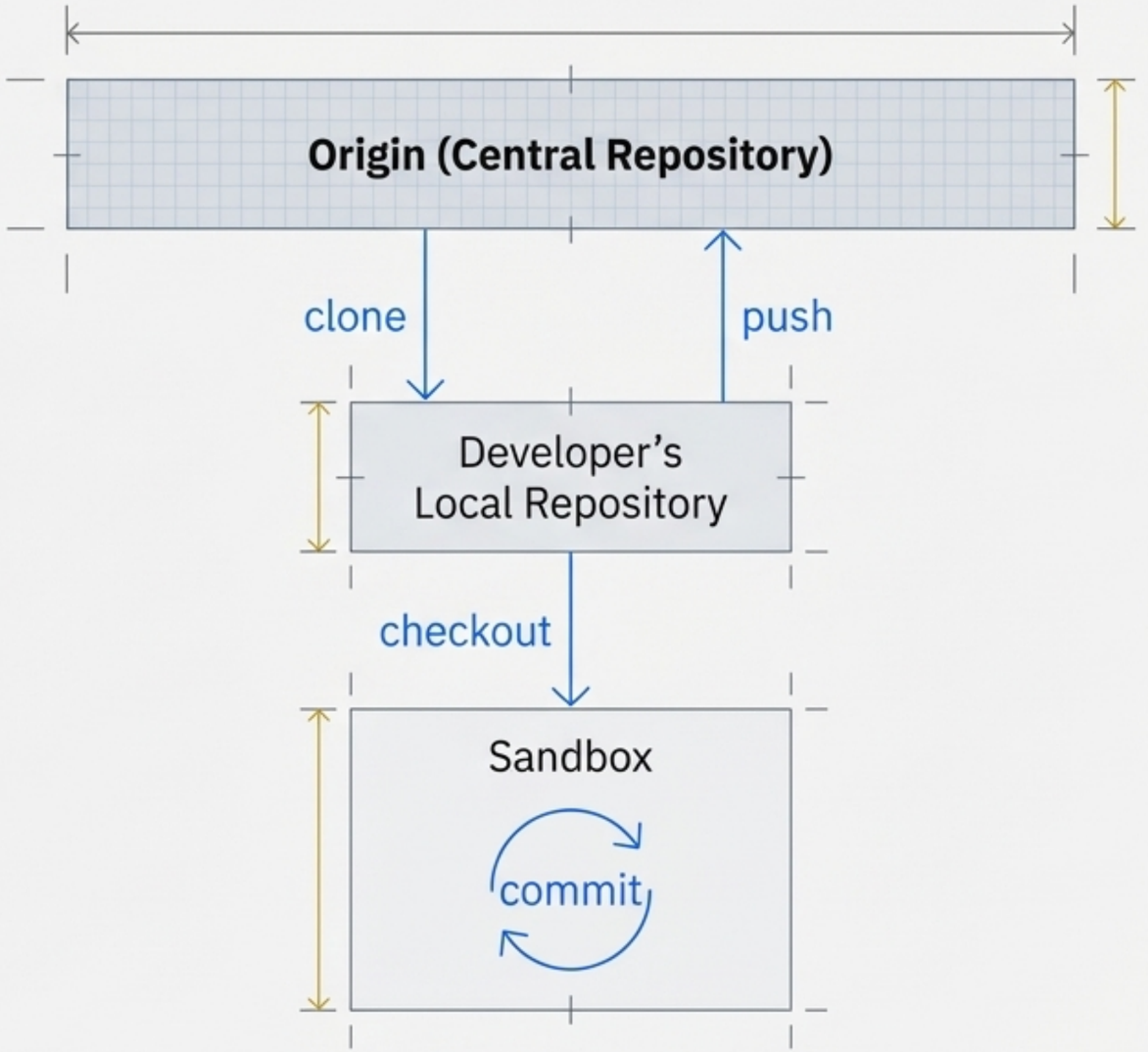
Every developer has a full copy of the repository, enabling offline work and reducing central server load. A central server (“origin”) acts as the single source of truth.

Branching

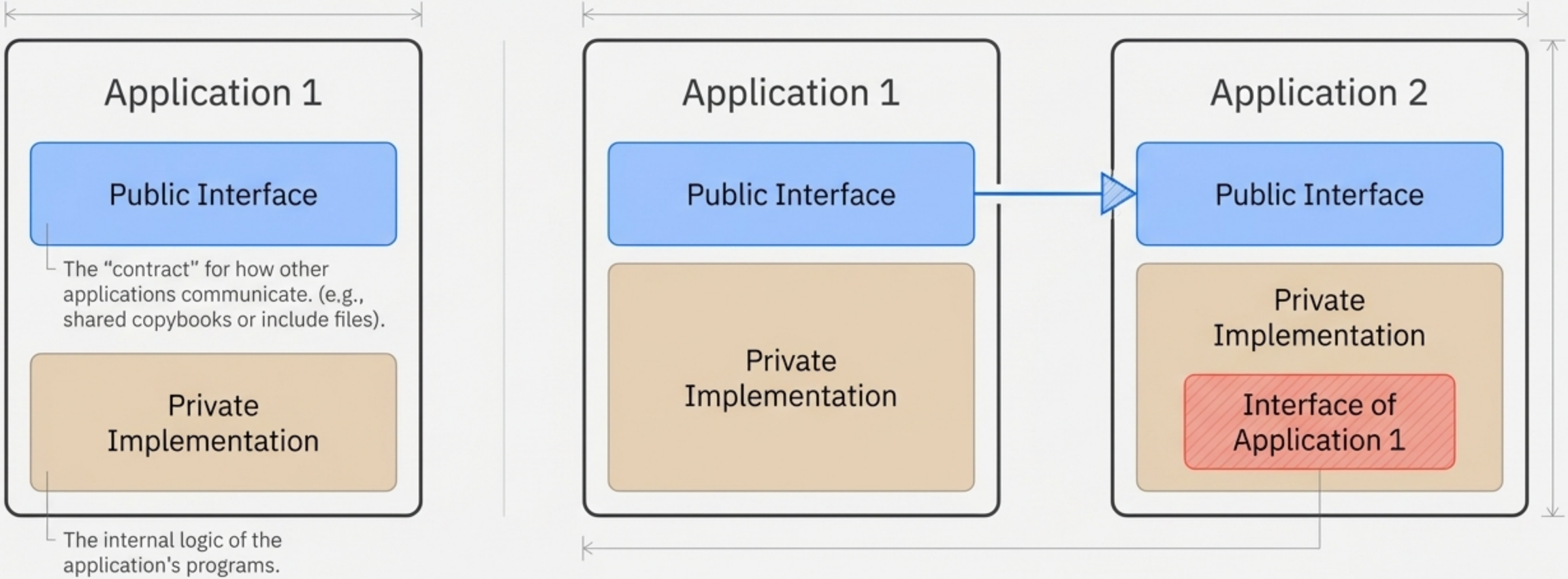
The core mechanism for isolation. Allows teams to work on maintenance, hotfixes, and new features simultaneously without interference. (e.g., Alice works on release 11 while Bob works on release 12).

Merging

The controlled process of integrating isolated changes back into the common codebase.

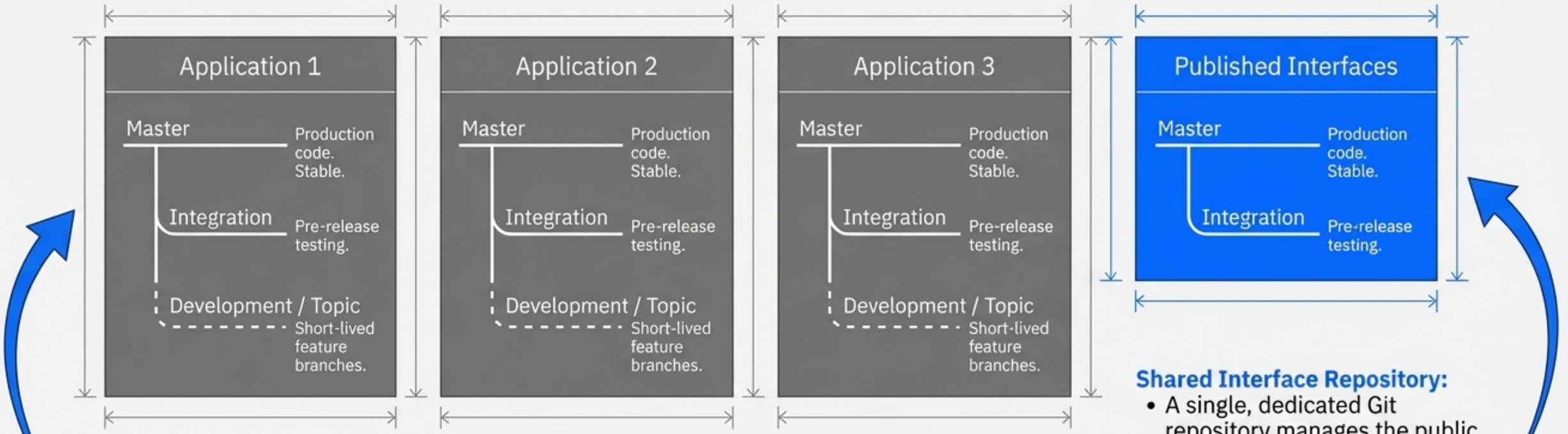


A Blueprint for Mainframe Applications in Git



Principle: An application consumes the public interfaces of the applications with which it interacts. This principle can be applied to existing COBOL and PL/I programs.

Recommended SCM Layout: One Repo Per Application



Application Repositories:

- Each application gets its own Git repository. This provides clear ownership, isolates lifecycles, and simplifies access control.

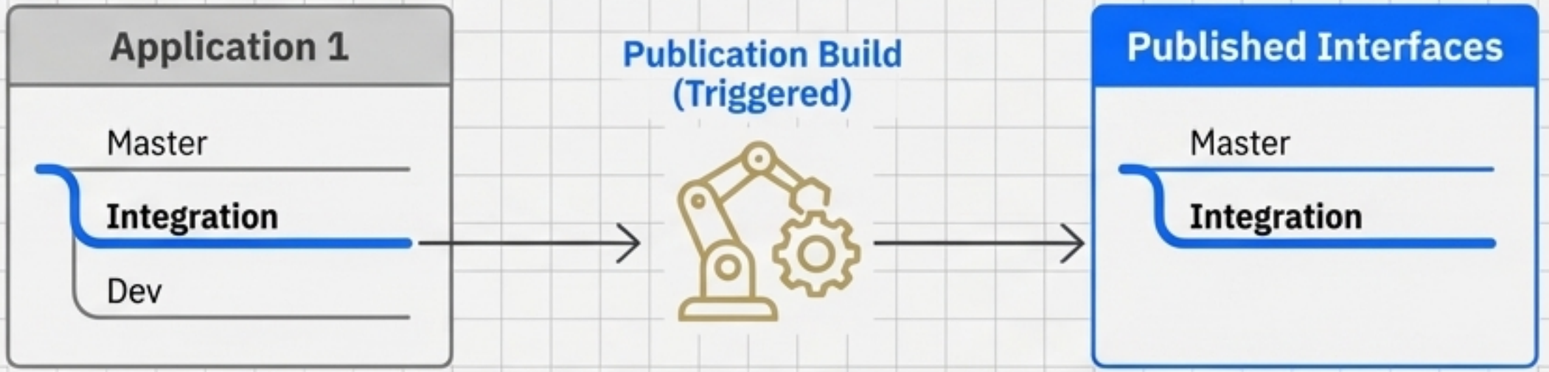
Shared Interface Repository:

- A single, dedicated Git repository manages the public interfaces (copybooks/ includes) of all appl applications, creating a central for managing cross-application dependencies.

The Adoption Process: Safely Managing Shared Interface Changes

The Challenge: How to handle changes to shared interfaces (e.g., a copybook) without breaking dependent applications.

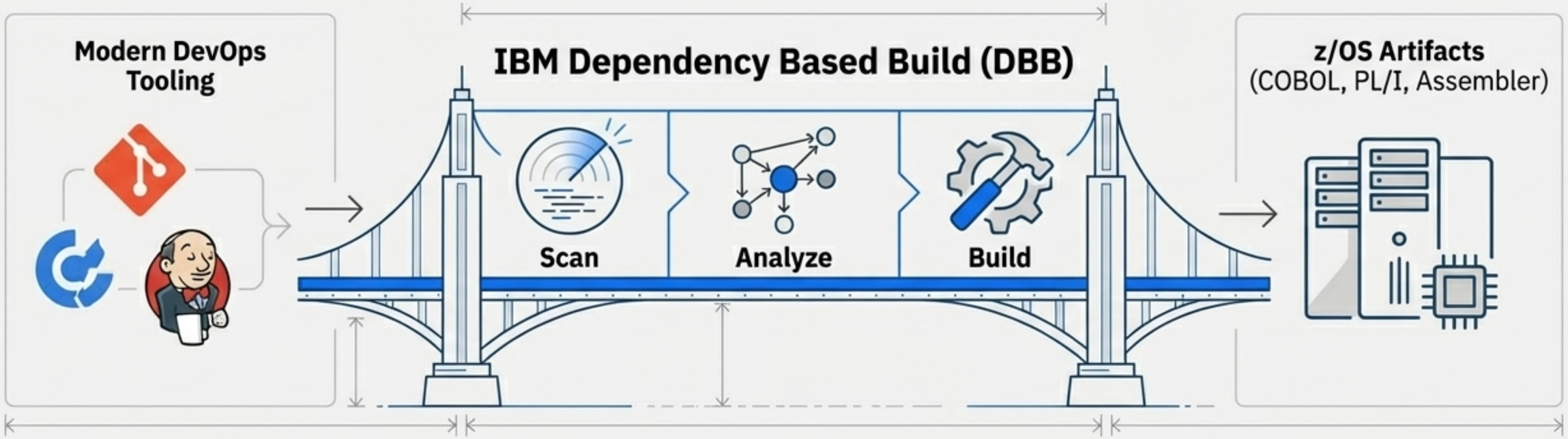
The Solution:
A controlled, automated publication workflow that differentiates between backward-compatible and breaking changes.



Technical divider

16:9 – ratio

The Intelligent Build Engine: IBM Dependency Based Build (DBB)



Dependency Analysis

DBB intelligently scans source files to understand their relationships.

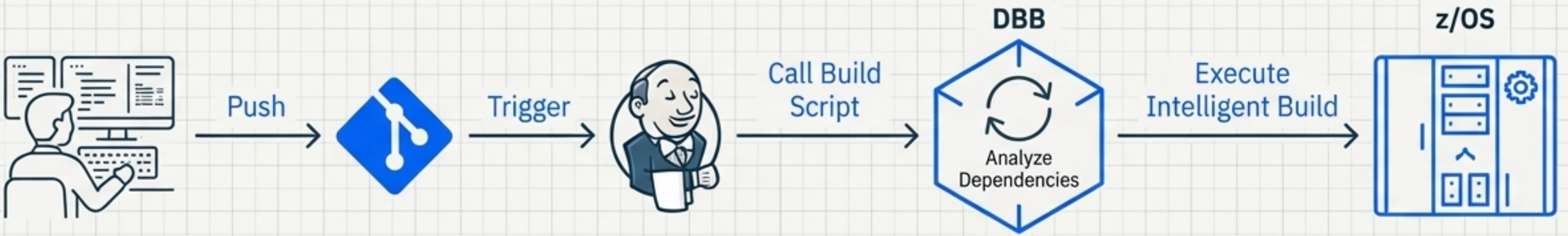
Impact Calculation

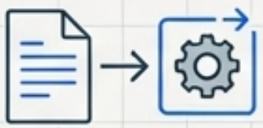
It knows that if a copybook changes, all programs that include it must be rebuilt. It understands direct, indirect, and link-time dependencies.

Efficient Builds

Instead of slow “full builds,” DBB enables fast, efficient “impact builds” that only compile and link what has actually changed.

Automating the Pipeline: From User Builds to Pipeline Builds

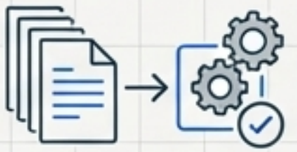




1. User Build (For Developers)

A fast build of a single program directly from the IDE (like IBM Developer for z/OS). Provides quick feedback without requiring a commit/push.

Technical divider



2. Pipeline Build (Automated & Auditable)

Triggered by a push to the Git server. Produces official, auditable binaries for deployment.

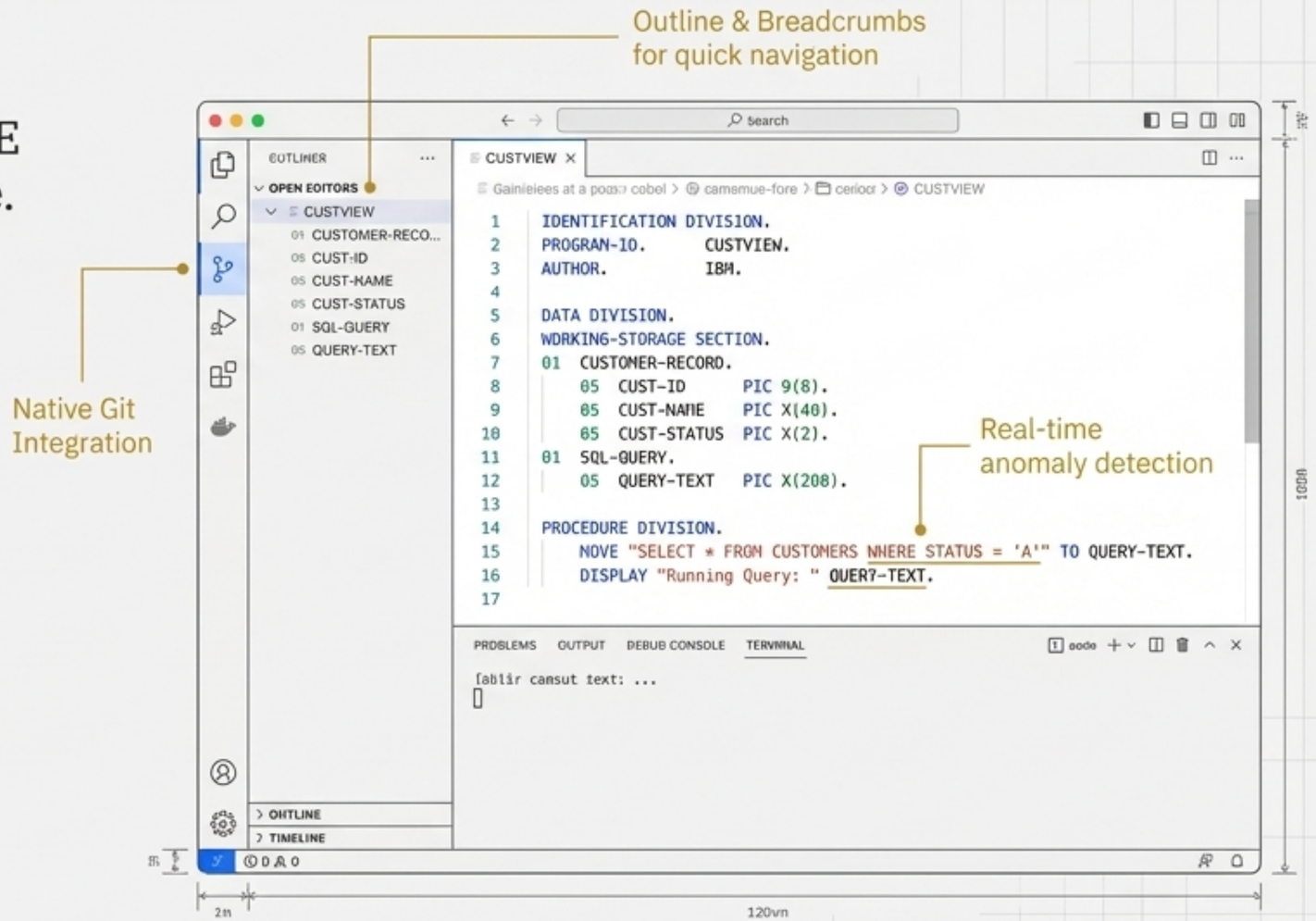
- **Full Build**: Compiles the entire application configuration from scratch.
- **Dependency-Based Build**: (Most common) Automatically builds only changed and impacted files using DBB's analysis.
- **Scoped Build**: Builds a user-defined list of files, often from a work item.

16:9 - ratio

The Modern Developer Cockpit: VS Code with Z Open Editor

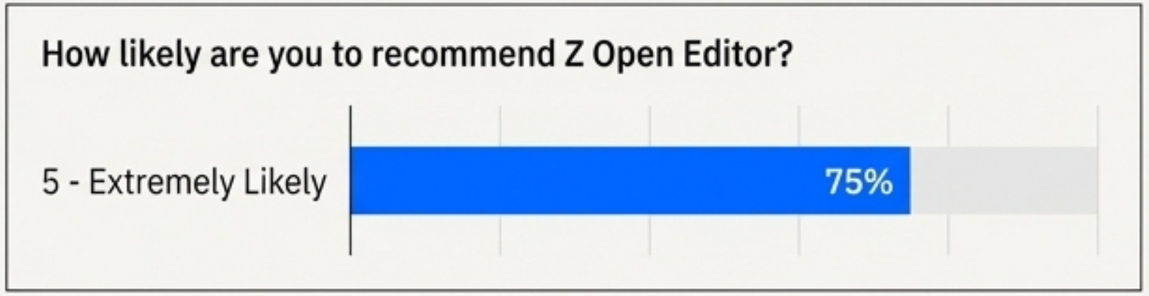
Mainframe development moves from the green screen to a modern, feature-rich IDE that new and experienced developers love.

- **Modern Editor Comforts:** Syntax highlighting, powerful search, and quick navigation.
- **Language Server Intelligence:** Real-time feedback, “look up all references,” and error checking without compiling.
- **Native Git Integration:** Seamlessly manage branches, commits, and pull requests from within the editor.
- **Unified Toolchain:** Mainframe developers use the same world-class tools as their distributed-systems colleagues.



The View from the Editor: What Developers are Saying

A recent user survey on GitHub highlights the transformative impact of modern tooling on mainframe development.



> “IBM Z Open Editor has become an essential element of our CI/CD chain modernization... allows us to integrate young developers more easily by avoiding the shock of the 24x80 green screen."
>
> - FALLAI-Denis

> "With IBM Z Open Editor in VSCode, I can work much faster than in ISPI in VSCode, I can work much faster than in ISPF... 95% of my work is in HLASM and this has simplified my life quite a bit."
>
> - mhammack

> "Once you start editing with VSCode you'll never go back to ISPF like editing. This was a big step forward... I can't imagine doing without it."
>
> - silvio14750

> "Using VSCode allows me to use my favorite editor mode (VIM) to write JCL and Assembler. This beats using ISPF hands-down."
>
> - dppduck

From Bottleneck to Superhighway: The Transformation Summarized

Before: The Library Manager Way

- ✗ Serialized work, file locking
- ✗ Manual dependency checks
- ✗ Late integration, frequent rework
- ✗ Rigid, sequential promotion paths
- ✗ ISPF-based ‘green screen’ editing

After: The Modern DevOps Way

- ✓ Parallel development in isolated branches
- ✓ Automated dependency-based builds (DBB)
- ✓ Early and continuous integration
- ✓ Flexible, automated pipelines
- ✓ VS Code + Z Open Editor experience

Results: **Increased Velocity** 🚀, **Improved Quality** 🛡, and a **Superior Developer Experience** ☆ that helps attract and retain talent.

Your Path Forward

This document provides a blueprint for modernizing mainframe development. To learn more and get started, explore these resources:

IBM Dependency Based Build Community
For questions, comments, and engaging with the authors and other users.
http://ibm.biz/dbb_community

DBB on GitHub
For sample scripts and additional help.
https://github.com/IBM/dbb

DBB Official Documentation
The landing page and Knowledge Center for in-depth information.
https://developer.ibm.com/mainframe/products/ibm-dependency-based-build

The transformation to parallel development with modern tools is not just about technology; it's about empowering your teams to build better software, faster.