

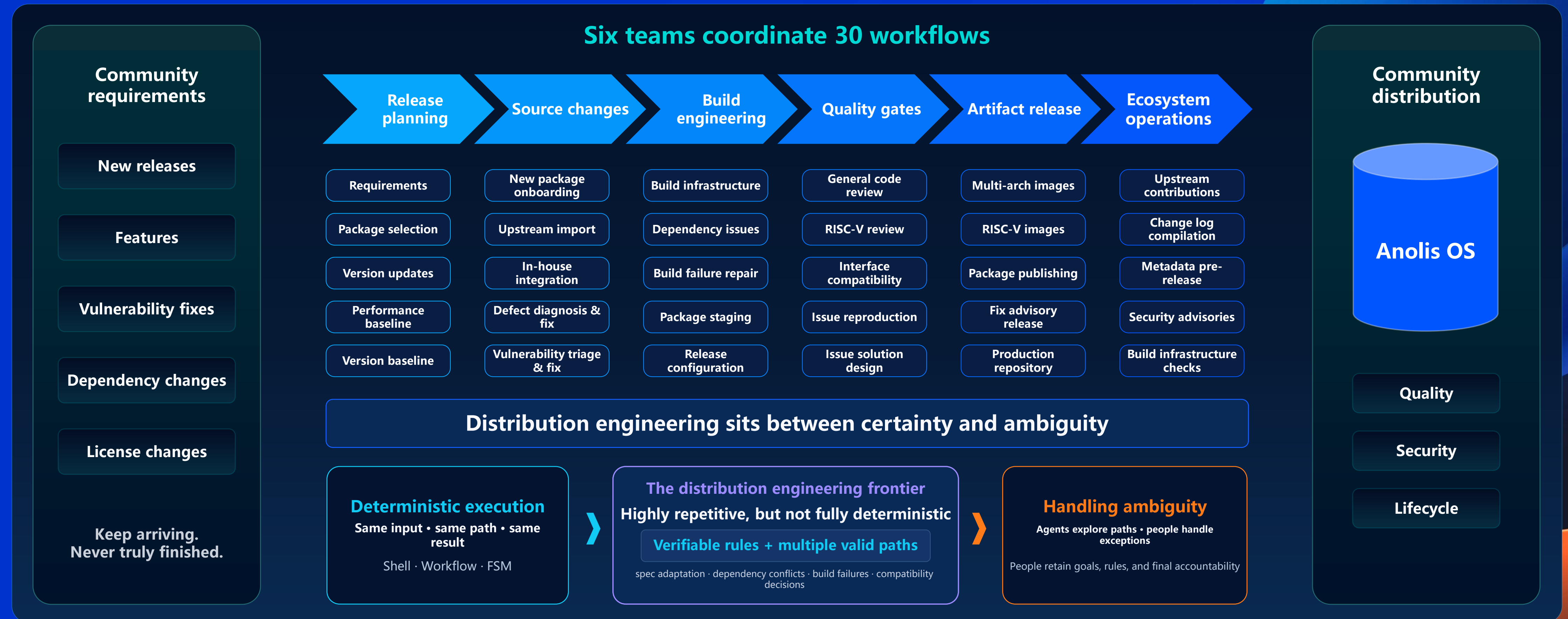
Toward AI Native A Paradigm Shift in Developing the Next-Generation Anolis OS

Shenglong Zhu

Operating Systems Expert, Alibaba Cloud
AI SIG Maintainer, OpenAnolis Community

A Community Distribution: 30 Workflows and Over 100 Tasks

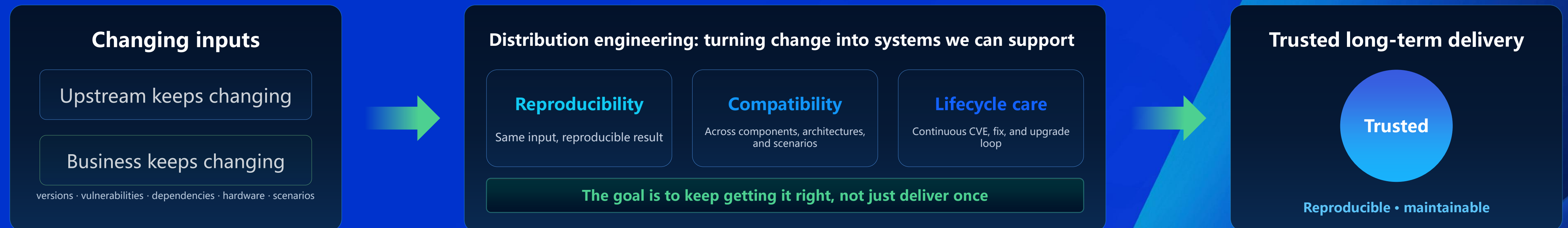
Automation is widespread. Repetitive but uncertain work still relies on people



What Is a Distribution? Trusted Delivery from Constant Change

Understanding what a distribution delivers shows which workflows AI Native should redesign

A distribution is a continuous production system, beyond a package collection



Three unavoidable physical constraints

Together, these constraints multiply complexity while human capacity grows only linearly



Conclusion: a distribution is a production machine that continuously reduces uncertainty

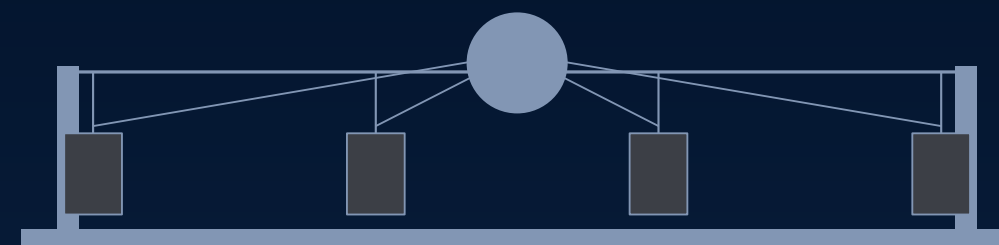
The development paradigm must decide what belongs to deterministic systems and what belongs to probabilistic intelligence

Paradigm 1: Workflows Handle Certainty, AI Handles Ambiguity

Technology creates productivity through organizational change. AI needs clear boundaries to scale

Change the tool or redesign the system?

1890s | Replace the motor



Old transmission · old workstations · old handoffs

The tool changed; the production logic did not.



30 years

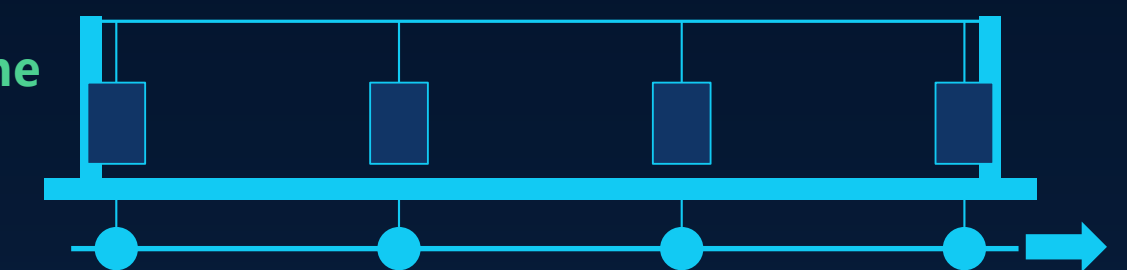
Little productivity gain
Tool swap ≠ system change



1920s | Redesign the production line

Reorder operations
Synchronize cycle times

Move quality gates upstream
Create continuous material flow



Productivity jumped only after the system was redesigned.

Today: rebuild workflows around AI instead of inserting AI into old processes

Paradigm 1 | Three Iterations Set the Boundary

Use explicit processes for deterministic work and AI for ambiguous work

01 Pure Agent

Giving everything to the model proved unstable



02 Plug-in architecture

More stable, but the framework was too heavy



03 Hybrid orchestration

Workflow + MCP + Skill



04 Clear boundaries

Systems handle certainty; AI handles ambiguity

Deterministic Work | Deterministic Tools

Scripts · CLI · Workflow

Inputs, paths, state, and retries are all verifiable

Uncertainty | The LLM's last mile

Semantics · uncertain decisions · long-tail cases

Use models only where exhaustive rules are impossible

Reusable Capabilities | Team Scale

Script capabilities · turn experience into Skills

Repeatable, portable, team-maintained

The LLM becomes a constrained explorer, rather than the judge of the entire process

Paradigm Shift: Engineers Step Out of the Execution Loop

Dual loops sustain execution and turn local gains into organizational capability



Paradigm 4: dist-git Manages Packages, Monorepo Manages the Next OS

Beyond repository consolidation: a computable, verifiable, traceable OS

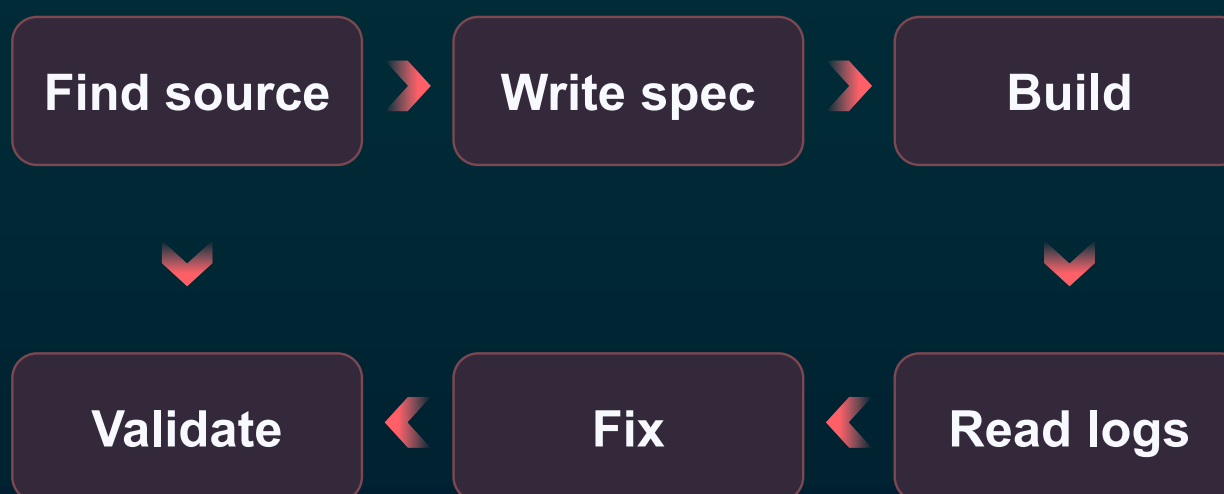


A Distribution-Wide Monorepo Makes Agent Development Auditable

The Monorepo holds engineering truth; PengLAI runs the loop; people own gates and accountability

Traditional | People Run the Chain

People carry context between steps



One failure

A person runs the loop again

The bottleneck is manual handoffs, not coding

Monorepo + PengLAI | Converge Within the Baseline

People own goals & gates

Monorepo baseline

packages · dependencies · rules · versions · evidence

API / MCP on demand | Results written back



Build failure → automatic diagnosis and repair

Validation failure → reselect with evidence; ask a person only if it cannot converge

Input | goals + acceptance
criteria



Agent | execute + validate



Output | PR + evidence

People move from step-by-step handoffs to defining goals, gates, and final accountability

Package Selection: Auditable Decisions from Constraints, Priorities, and Evidence

People set positioning, constraints and priorities. Agents gather evidence and make traceable decisions at each level

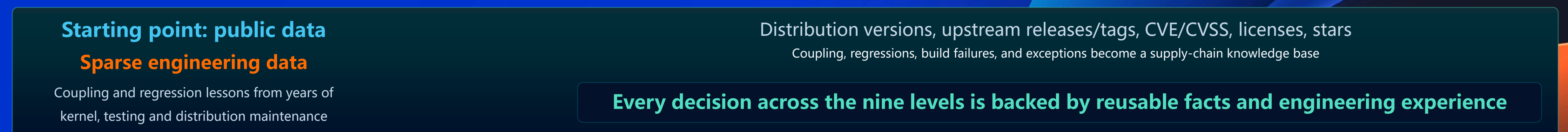
Upper layer • Auditable decision science

Create reusable community rules



AI orchestrates evidence collection, hard constraint defines the feasible set, and P1-P9 determine the conclusion; every level links back to evidence

Lower layer • Engineering evidence foundation



Next-generation Anolis OS package selection (2,800+ packages) | 1-2 days reduced to 30-40 minutes per package | ≥99% consistency across repeated runs

Distribution Build: From Manual Handoffs to a Learning System

People set goals and gates. Agents analyze, execute and verify. Delivery means a usable system, beyond a successful build

Start with a multiplication problem
beyond human capacity

2,000+ packages

× versions × architectures
× scenarios × lifecycle

≈8,000 hours
per release

30 workflows · 100+ tasks
Complexity multiplies faster than people scale

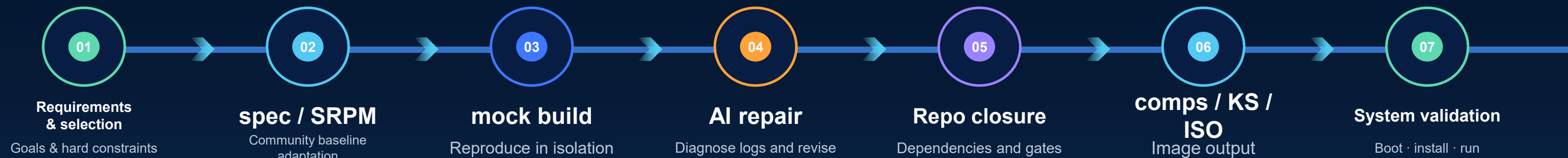
One continuous, verifiable distribution production chain

Evidence-driven decisions + deterministic execution + system acceptance

AI judgment: rules + private evidence

Deterministic build | Workflow / Script / Gate

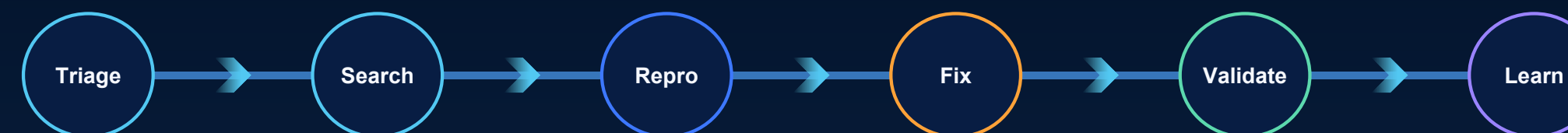
System acceptance | Is it deliverable?



Gates accept only: reproducible · bootable · installable · usable

Next Stage | Turn Failures into Rules

Move issue repair from one-off prompts to a traceable learning loop.
Every failure becomes a rule, test, or piece of knowledge.



Evidence loop: upstream patches, build logs, dependencies, and validation results feed back

Capability inferred from historical patterns

≈30% → ≈80%

Near-term floor

Long-term ceiling

One case end to end ≈20 min

CVE remediation: one patch may affect the entire system

180K+ historical commits

Review distribution history and analyze representative changes

The patch differs. Adaptation or missed fix?

Upstream and downstream interfaces and paths differ.
Judge against the target version.
Patch differences alone are insufficient.

Two-thirds

of representative changes involve
in-house patches or build adjustments

Upstream patch $\neq$ downstream-ready

Evidence: target-version code, interface context,
and spec build configuration

The patch applies. Is the fix complete?

Prerequisites or follow-up Fixes may be missing.
One commit rarely shows the full repair scope.

173

missing Fixes
found by Kitta and added

The repair chain extends beyond one patch

ANCK backport reviews
Trace upstream dependencies and related Fixes

The fix works. Did it break existing functionality?

A successful build is insufficient.
Verify the fix and compatibility
in the target environment.

80%

of representative changes involve
system-level components

One change can affect the whole system

Evidence: build, PoC, and regression results.
Conclusions are limited to existing test coverage.

Bring maintainers' judgment on correct adaptation, complete fixes, and regression avoidance into automated repair

Dual closed loops for CVE remediation and review

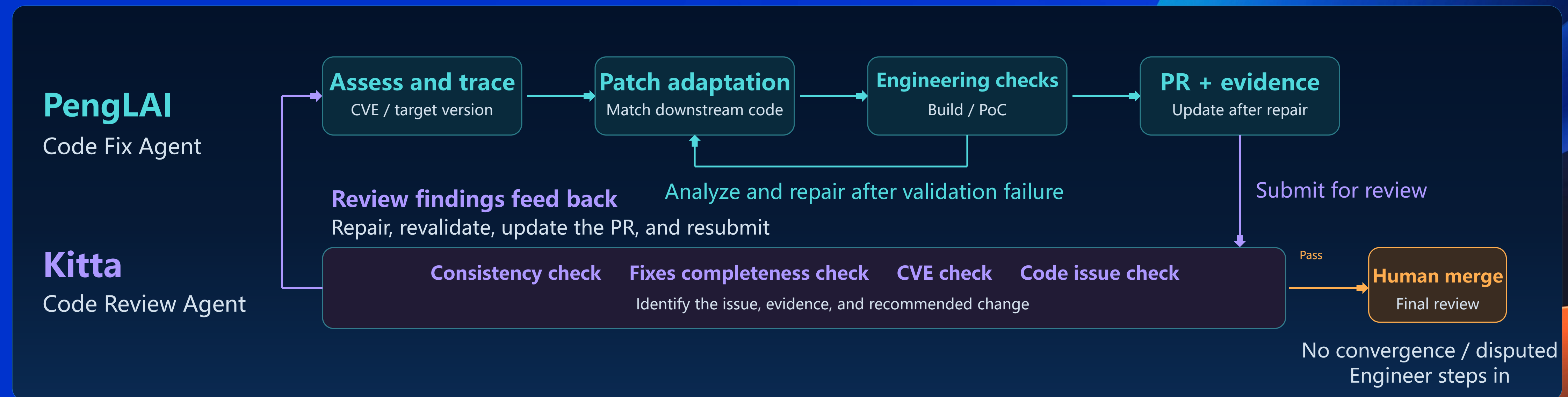
PengLAI Code repairs and provides evidence. Kitta reviews independently. Feedback drives revisions

Traditional process

Engineers follow execution, results, and review feedback



Now: the system keeps moving while people provide final review

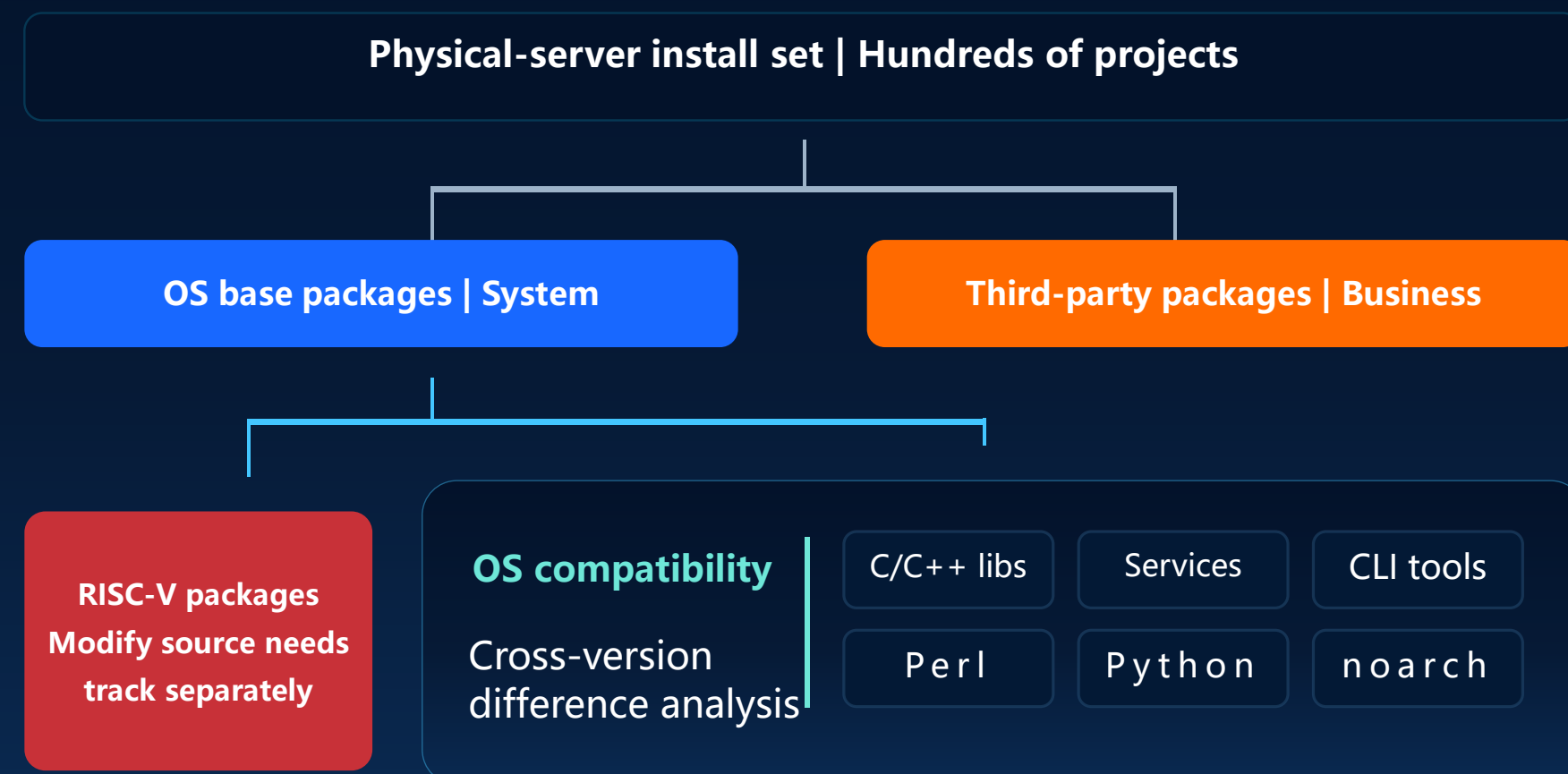


RISC-V Across OS Versions: Converge Scope, Prove Compatibility

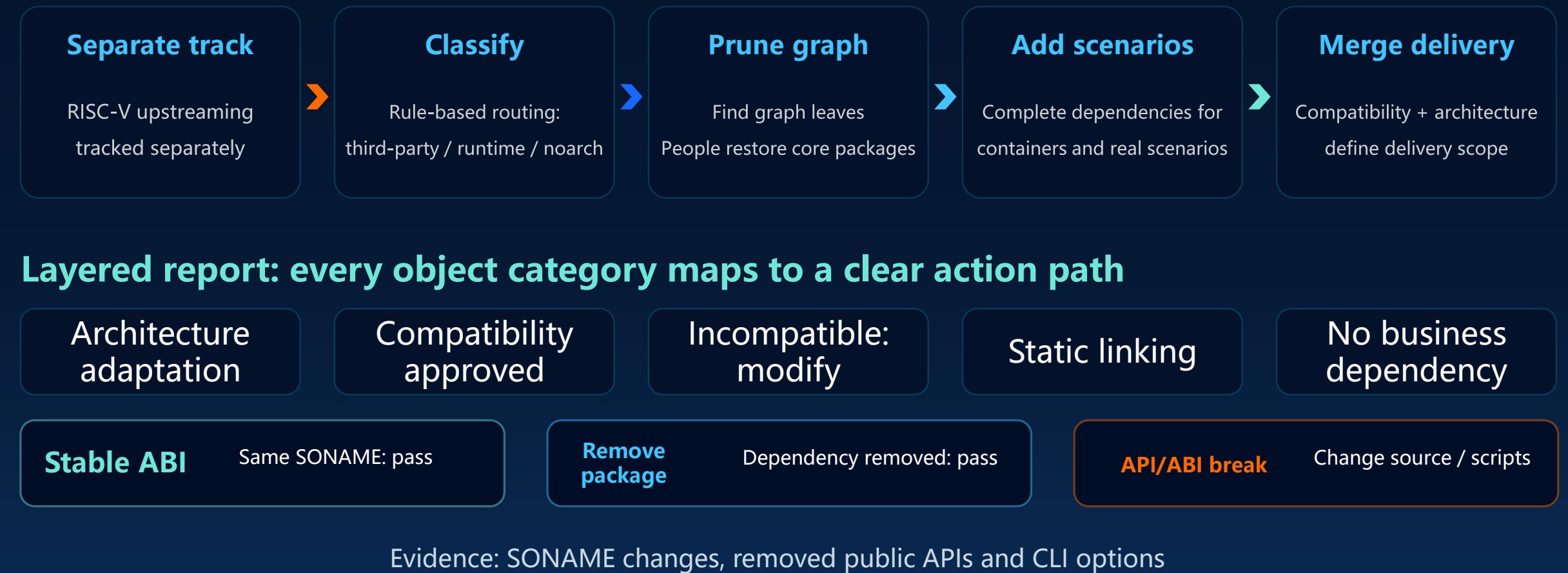
Alinux 3 → 4 | Scope and compatibility share one AI evidence engine

Dependency: full RISC-V support resides in the target OS → business upgrade becomes a prerequisite for the new architecture

Scope Convergence | Separate OS and Architecture



Compatibility | Narrow Scope, Prove Core Cases



AI evidence engine

Drives scope calculation and compatibility assessment

Scale evidence; people decide

Normalize inventories

Parse metadata
Map and deduplicate source

Intelligent routing

Classify ownership, architecture, and runtime automatically

Dependency graph

Compute reverse dependencies and identify leaves

Compare evidence

Compare SONAME, API, CLI, services, and configuration

Generate reports

Conclusions, evidence, and actions remain fully traceable

AI: extract, classify, compute, compare, draft. People: set scope, restore core packages, set risk thresholds, sign off

Business Images: Preserve Two Explicit Delta Stages

Bake in fixed changes, inject environment settings at first boot, and verify only the contracts workloads depend on

147 real image changes mapped by AI to two stages, one ownership boundary, and one runtime contract

Business image = BaseOS + build-time recipe + first-boot policy

Business delta | Varies by workload, hardware, and deployment environment

Build Time: Bake In Fixed Changes

Switch repos, add/remove/upgrade packages

Bake in common OS configuration

Repeatable · auditable · machine-independent

First Boot / Runtime: Environment-Specific Changes

**Hardware · region · security zone
drivers / network / storage / kernel / services**

Conditional injection · shared base unchanged

BaseOS: stable shared base

kernel · boot · systemd · network · storage · security

Maintain once, reuse across workloads; both stages follow one stable contract

Ownership, activation and repeatable checks define the boundary, beyond which files an image contains

AI Turns Scripts into Boundaries

Input | package operations · config differences · startup scripts

1 Who owns this change?

Shared → BaseOS | Specialized → business

2 When does it take effect?

Fixed → build time | Conditional → after boot

3 Could it create hidden risks?

Find duplication · detect conflicts

Output Minimal delta + executable contract

Clear ownership | Reusable tests | Traceable conclusions

An image delivers more than a disk: stable foundation + build-time recipe + startup policy + executable contract

AI Native: Expand Engineering Capacity and Distribution Scope

packages × versions × architectures × scenarios × lifecycle



Capability boundary

Deepen distribution engineering

Expand maintenance coverage and continuously improve distribution engineering

Extend into systems

Business runtime environments
System diagnosis and repair

People own goals, critical decisions, and final acceptance

Agents take on more analysis, execution, and validation

Try it yourself

Scan to join the Monorepo

Anolis OS 27 Monorepo

The OS as code: computable, verifiable, traceable



Scan for the full
Monorepo guide

Complete engineering structure

packages · groups · releases · tools · playbook

AI Agent

Deterministic tool orchestration Selection → build → repair → validation → release

PengLAI

Intelligent full-lifecycle platform Remote capability extensions

THANK YOU