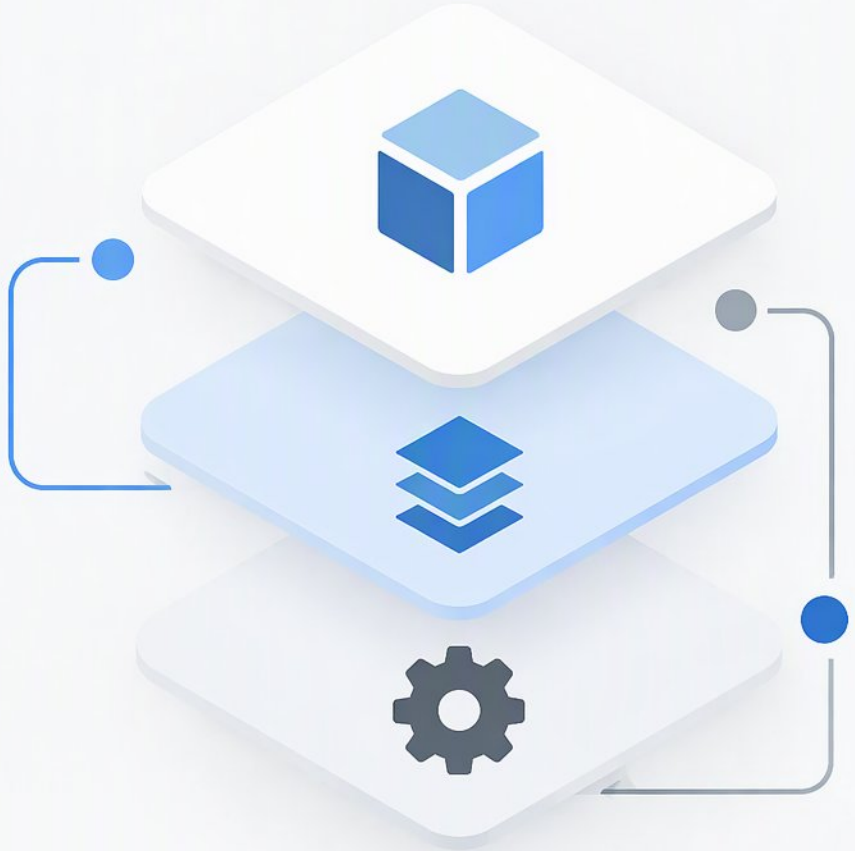
Introduction to
Examples of Design Systems

- Compare real system structures and governance choices, not visual styles
- Seven lenses: foundations, tokens, component API, documentation, contribution, release, adoption
- Mature systems diverge by org size, platform spread, accessibility duties, brand decentralization
- Read each case for reusable structural patterns and trade-offs, not component inventories
- Shared vocabulary: source of truth, contribution tiers, semantic versioning, deprecation windows, adoption telemetry



A Comparison Framework: Seven Layers of a Design System

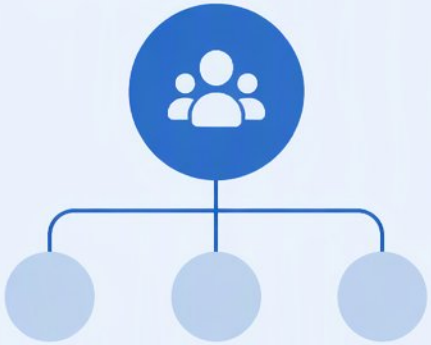
- Working architecture: foundations, tokens, components, patterns, guidelines, governance, tools
- Each layer answers a different product question; value comes from reinforcement
- Use the seven layers as an audit matrix — source each independently
- Governance can't be bought: decision rights and deprecation policy built locally
- This framework is the spine for every case study that follows



Three Governance Archetypes and When Each Breaks

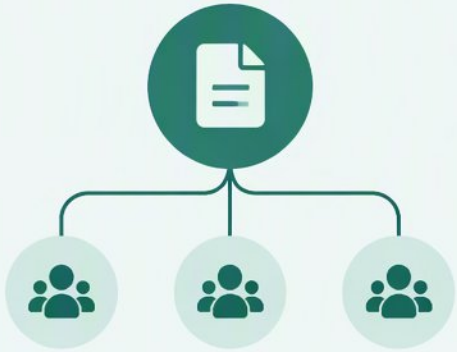
Centralized

- one core team owns all (Carbon, Lightning, Cloudscape) — high coherence, becomes a service org



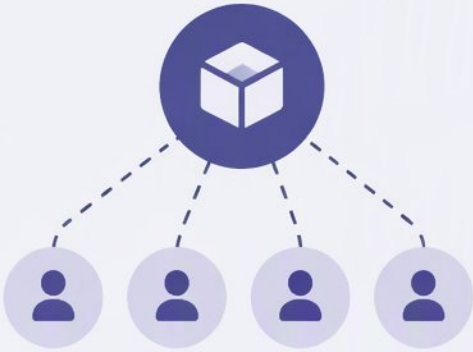
Federated

- core owns canon, product teams contribute via RFC (Polaris, Atlassian) — drift without RFC discipline



Open

- public components anyone can fork (shadcn, Radix) — broad adoption, every consumer maintains its version



Smaller org size
Fewer portfolios

Larger org size
More portfolios

Breaks here

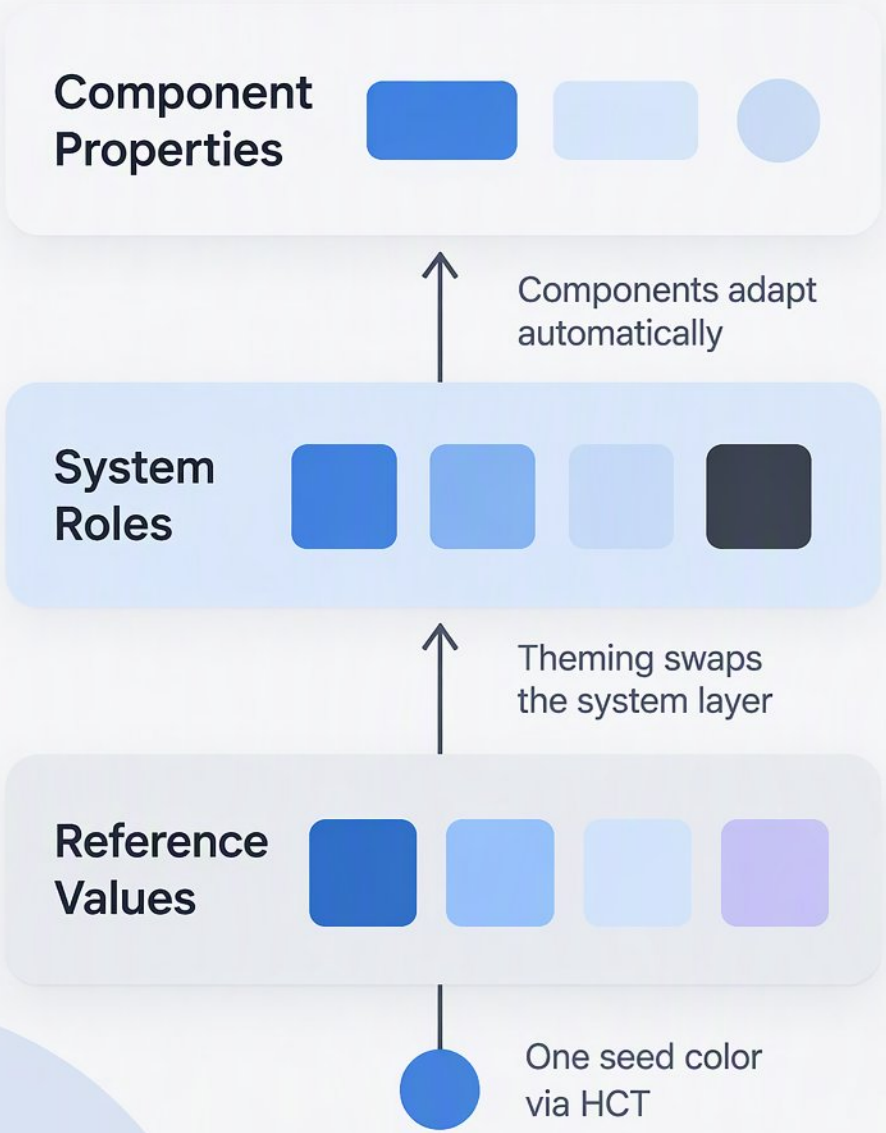
Breaks here

Breaks here

Hybrid is the most common 2026 shape:
foundations centralized, extensions distributed

Each model has a shelf life
set by org size and portfolio count

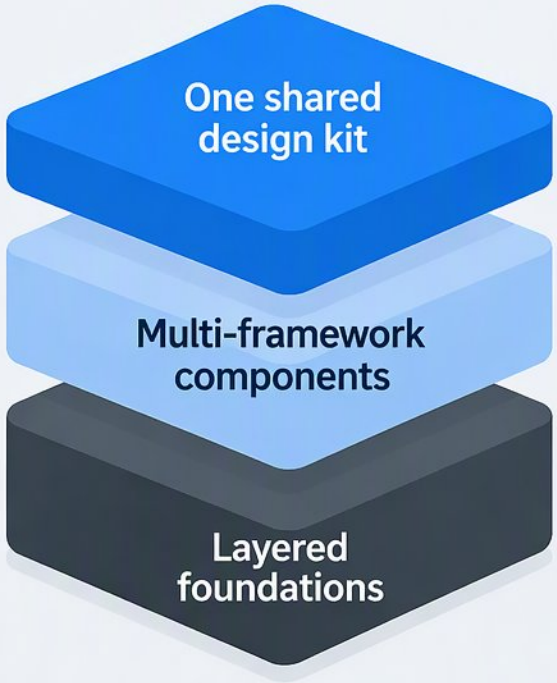
Case Study: Material Design 3 and Token-First Theming



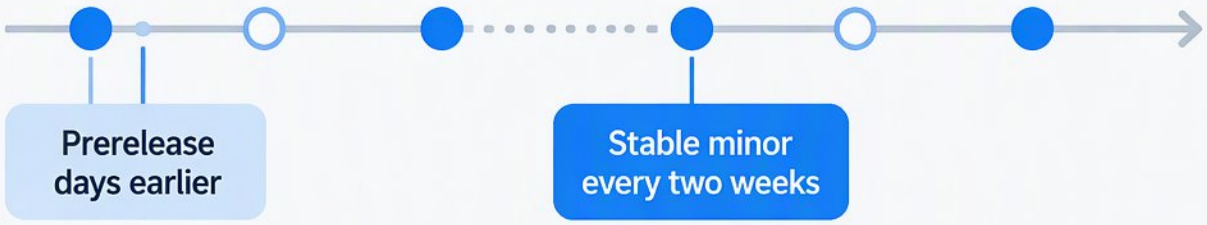
- Three token classes: reference values, system roles, component properties
- Theming swaps the system layer — components adapt automatically, no rebuild
- Dynamic color generates a full scheme from one seed via HCT
- Role pairs hit WCAG contrast by construction, not afterthought
- Trade-off: platform coherence versus limited brand differentiation without overrides
- Borrow: token-first architecture and contrast-safe role pairing as defaults

Case Study: IBM Carbon and Time-Based Releases

- Structure: layered foundations, multi-framevork components, one shared design kit
- Stable minor every two weeks; prerelease days earlier for product integration testing
- Patch releases ship as needed for security and bug fixes
- Major pipeline: preview flags (enable-v12-*), long prerelease, Active, Maintenance, EOL
- Migration designed in: enable all future-version flags before the major, expect no component changes
- What to borrow: predictable cadence plus a visible release-phases table



Time-Based Releases

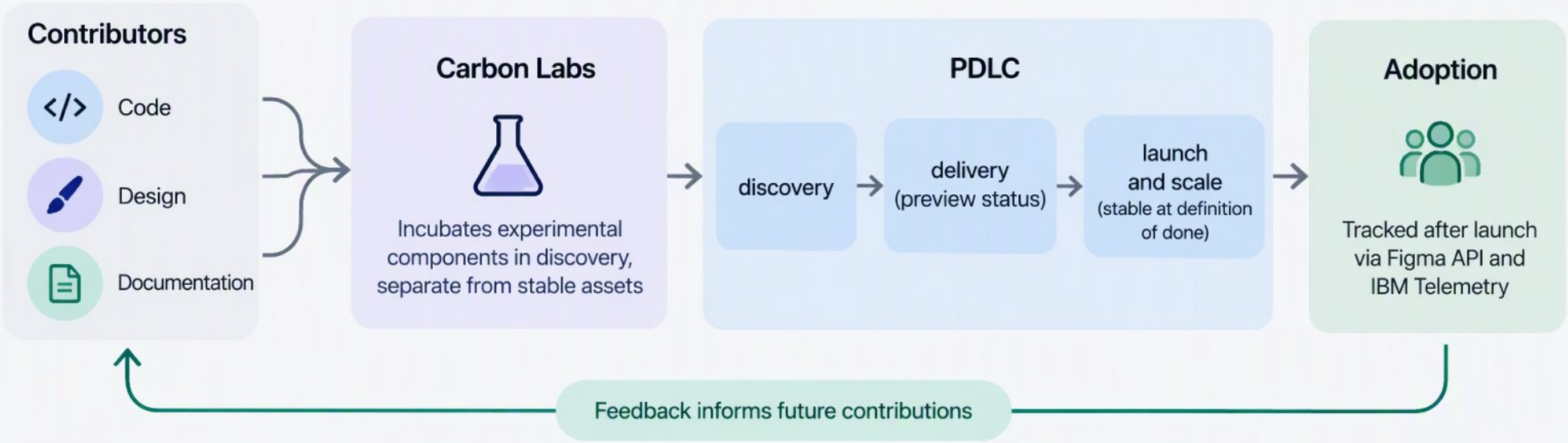


Major Release Pipeline



Case Study: Carbon's Contribution Lifecycle

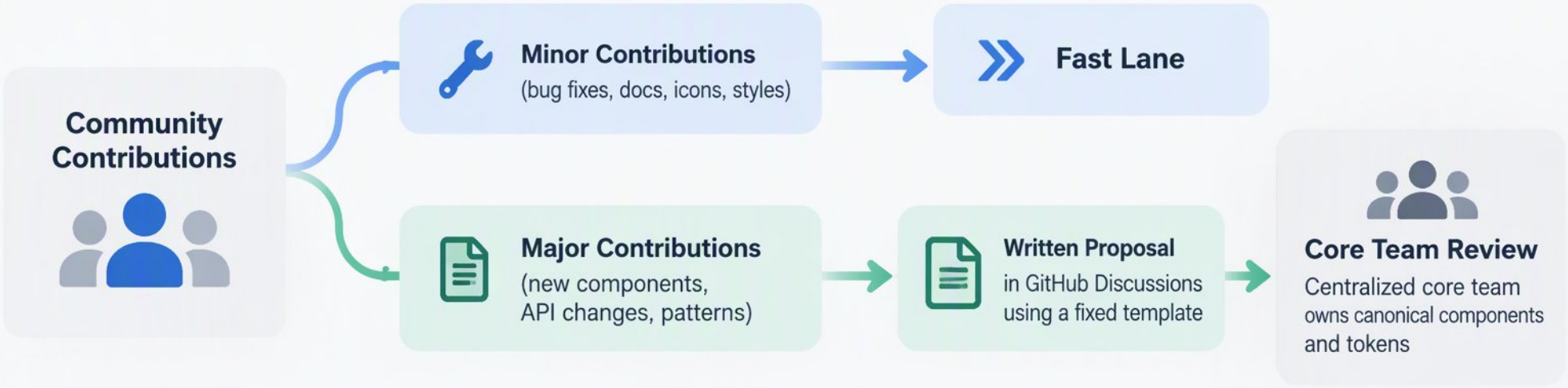
From open contributions to broad adoption



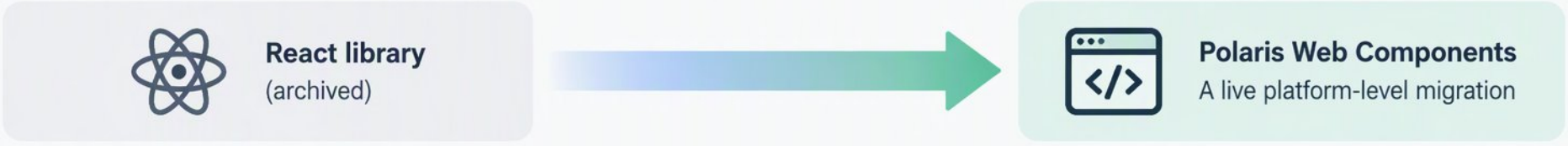
- Anyone can contribute code, design, or documentation — entry points map to each type
- Carbon Labs incubates experimental components in discovery, separate from stable assets
- PDLC phases: discovery, delivery (preview status), launch and scale (stable at definition of done)
- Prioritization weighs business impact — reuse potential and high-impact teams move work up
- Adoption tracked after launch via Figma API and IBM Telemetry

Case Study: Shopify Polaris and Tiered Contribution by Risk

A CLEAR PATH FROM IDEA TO IMPACT



- Contributions tiered into minor (bug fixes, docs, icons, styles) and major (new components, API changes, patterns)
- Major contributions start with a written proposal in GitHub Discussions using a fixed template
- A centralized core team owns canonical components and tokens; product teams contribute through the public model
- Polaris co-locates content and pattern guidance with components — voice and tone guidance is cited beyond Shopify
- Evolution signal: the React library was archived in favor of Polaris Web Components — a live platform-level migration



Case Study: Atlassian's Deliberately Narrowed Contribution Surface

- Contributions limited to Atlassians — external makers get a feedback collector instead
- Accepted: bug fixes and small enhancements; major changes and new components stay with the system team
- Stated rationale: avoid becoming a bottleneck for product teams shipping customer value
- Participation stays open: Slack help channel and fortnightly critique sessions
- Counter-narrative: bounded contribution is a deliberate strategy, not governance failure
- Compare with Polaris and Carbon — same archetype family, very different acceptance thresholds



Case Study: GitLab Pajamas and the Core / Extended Split

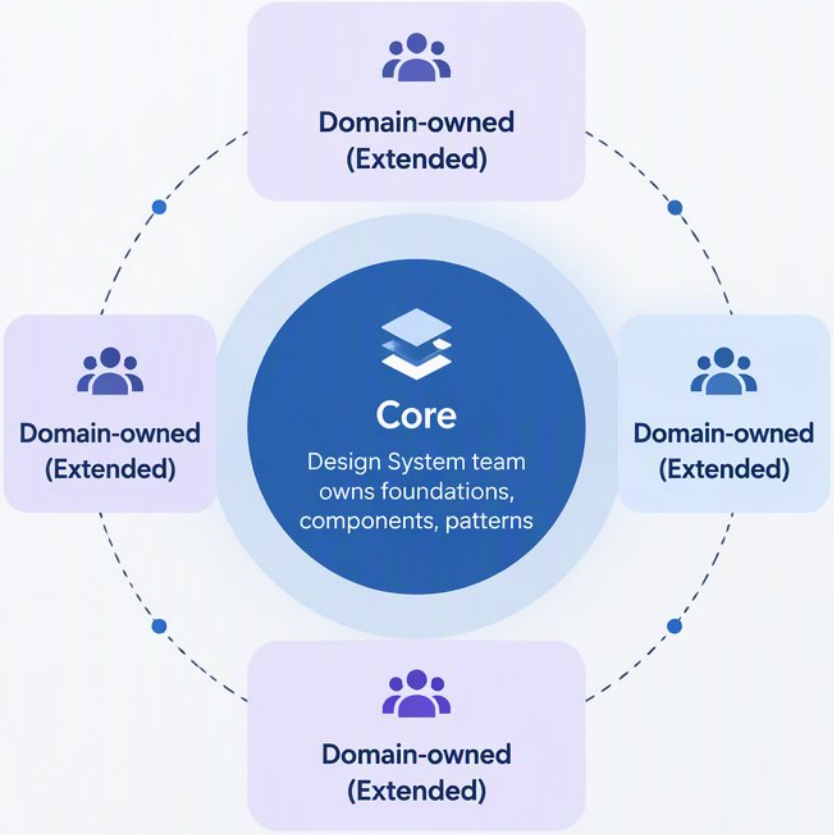
A clear structure for reuse, ownership, and support

Two layers

- Two layers: core (Design System team owns foundations, components, patterns) and extended (reusable, domain-owned)

Contribution classification

- Contributions classified as core, extended, out of scope, or pending classification



Four support levels

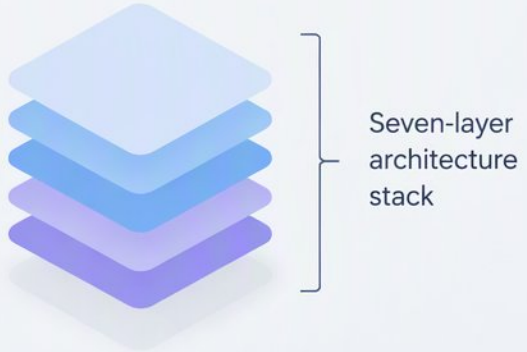
- No support
- Consult and review
- Task-specific
- End-to-end

Core change process

- Core changes require Design System team review, approval, and long-term maintenance

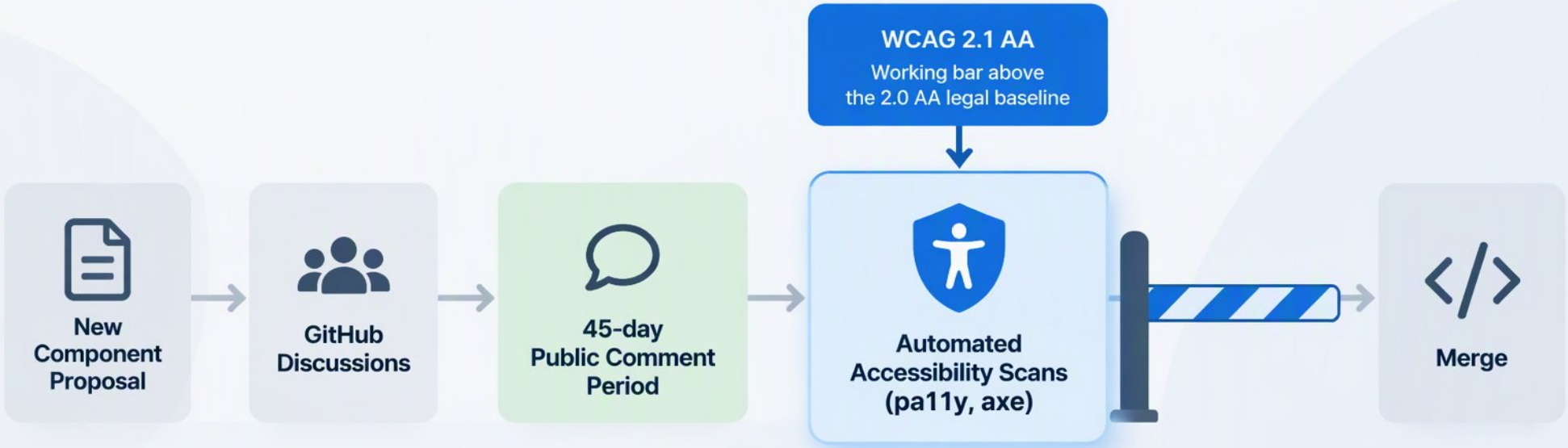
Service commitments

- 48-hour response
- one-week path forward
- two-week decisions



Case Study: USWDS and Accessibility as a Compliance Gate

OPEN CONTRIBUTION. ACCESSIBLE BY DESIGN. HIGHER STANDARDS FOR A MORE INCLUSIVE WEB.



Contributor Ladder

Contributor → Maintainer → Alumni

Each with explicit requirements and privileges.

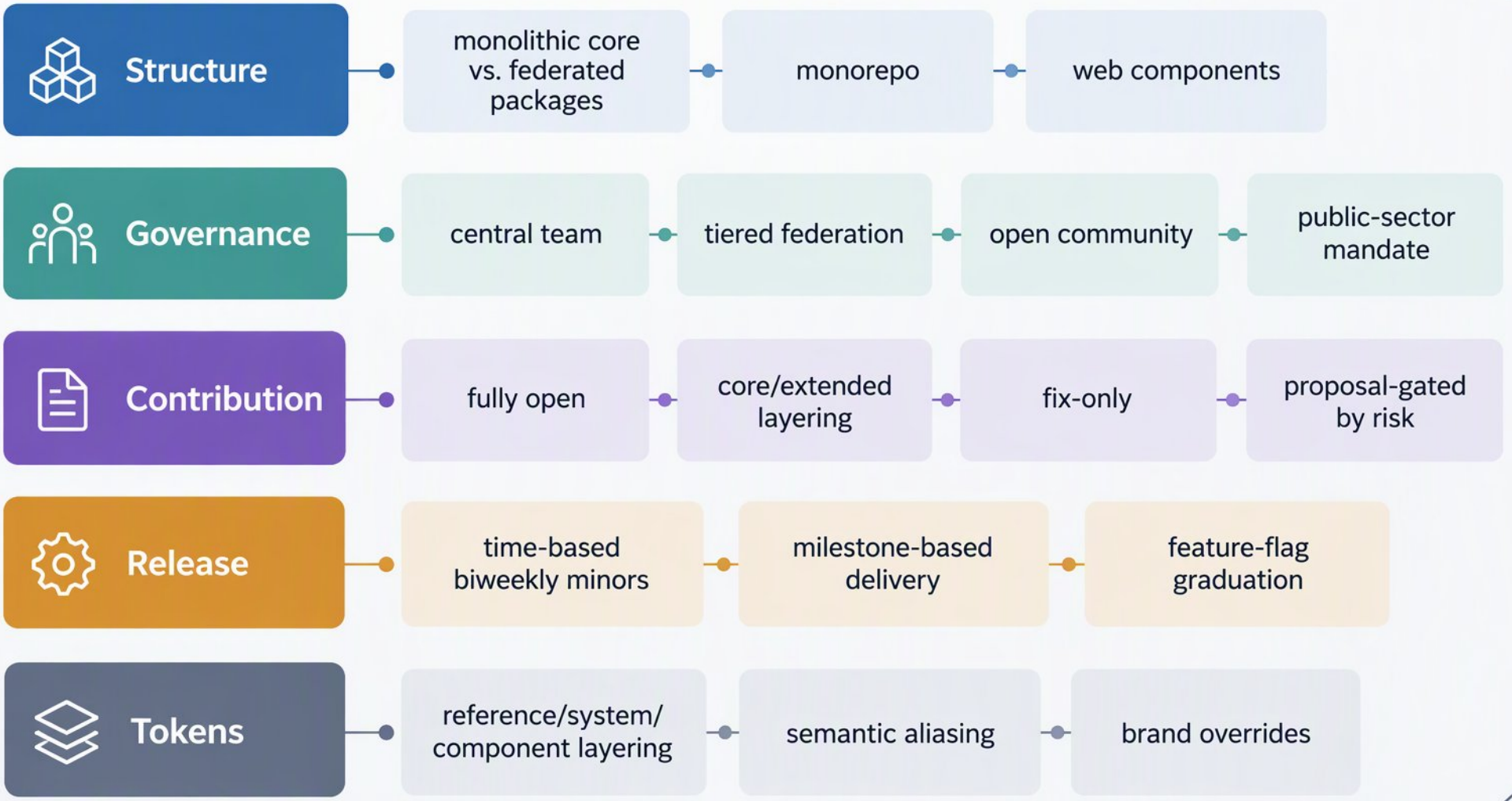
Adoption by Maturity Model

Principles → Guidance → Code

Not all-at-once migration.

1. Accessibility is structural: WCAG 2.1 AA working bar above the 2.0 AA legal baseline
2. Automated scans (pa11y, axe) run on every commit — no pass, no merge
3. New components begin in GitHub Discussions, then a 45-day public comment period
4. Contributor ladder: Contributor, Maintainer, Alumni — each with explicit requirements and privileges
5. Adoption paced by maturity model (Principles → Guidance → Code), not all-at-once migration

Pattern Matrix: Structures, Releases, and Decision Rights



Designing Your Decision Rights and RFC Process

Clear decisions. Healthier collaboration. A stronger, more sustainable project.

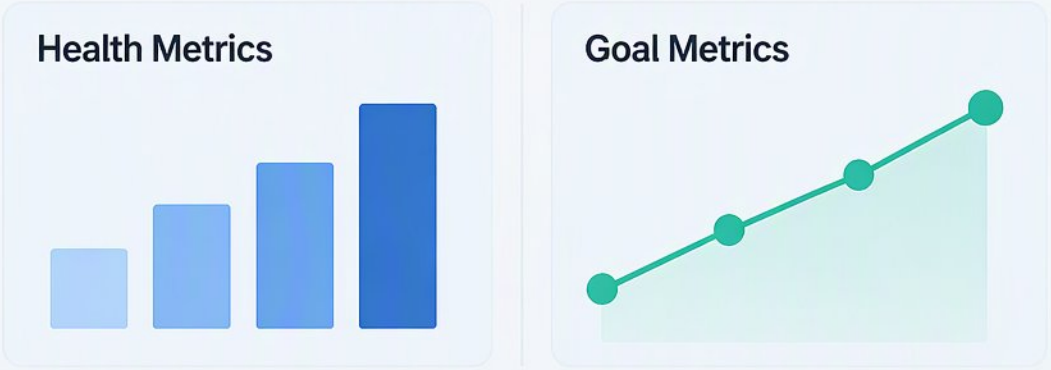
- Tier decisions by risk: core, product-specific, breaking changes, quality gates
- RFC covers problem, alternatives, affected surfaces, migration, open questions
- Reviewers comment on the document, not the pull request
- Core, Federated, Community tiers set review depth and name a maintenance owner
- Publish SLAs: response times, decision windows, escalation paths
- Track decision velocity, contribution rate, quality compliance, breaking-change frequency



Adoption Metrics and Deprecation Discipline

- Track health metrics continuously; reserve goal metrics for leadership accountability
- Measure coverage, token compliance, detach rate, contributions, accessibility pass rate
- Only ~38% of systems report wide adoption; weak governance is a top blocker
- Give deprecation a policy before a decision: notice, warning, lint rule, major removal
- Avoid the adoption trap: measuring what is easy, not what matters

Adoption Telemetry



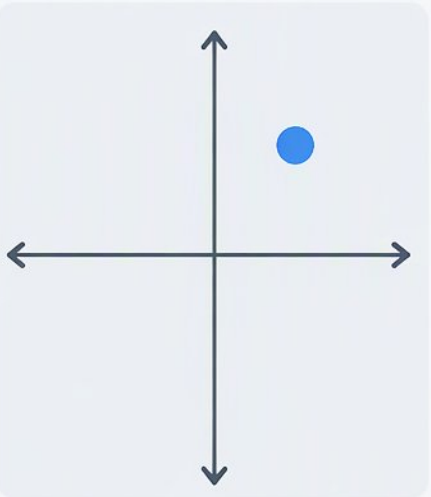
Deprecation Sequence



Workshop: Map Your System and Pick One Borrowable Practice

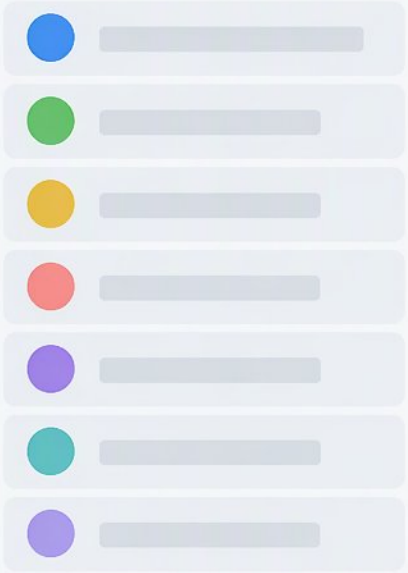
1 Step 1

Place your system on the structure and governance axes



2 Step 2

Score all seven layers — find the weakest layer and its failure mode



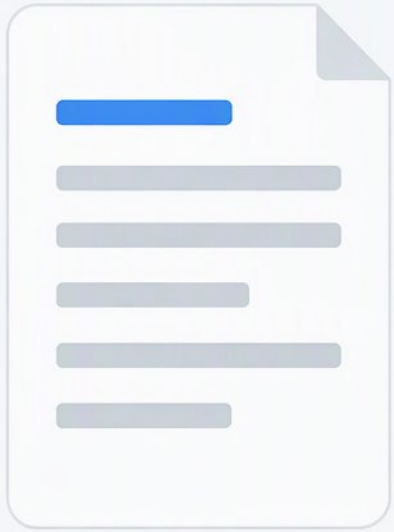
3 Step 3

Borrow one practice — Carbon's release phases, Pajamas' SLAs, USWDS' merge gate



4 Step 4

Write the first RFC or decision record with migration and measurement plans





Takeaway: Match structure and governance to org constraints, not popularity

PersonWise.

PersonWise • AI digital-human course platform

Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/ccdde6cc/public