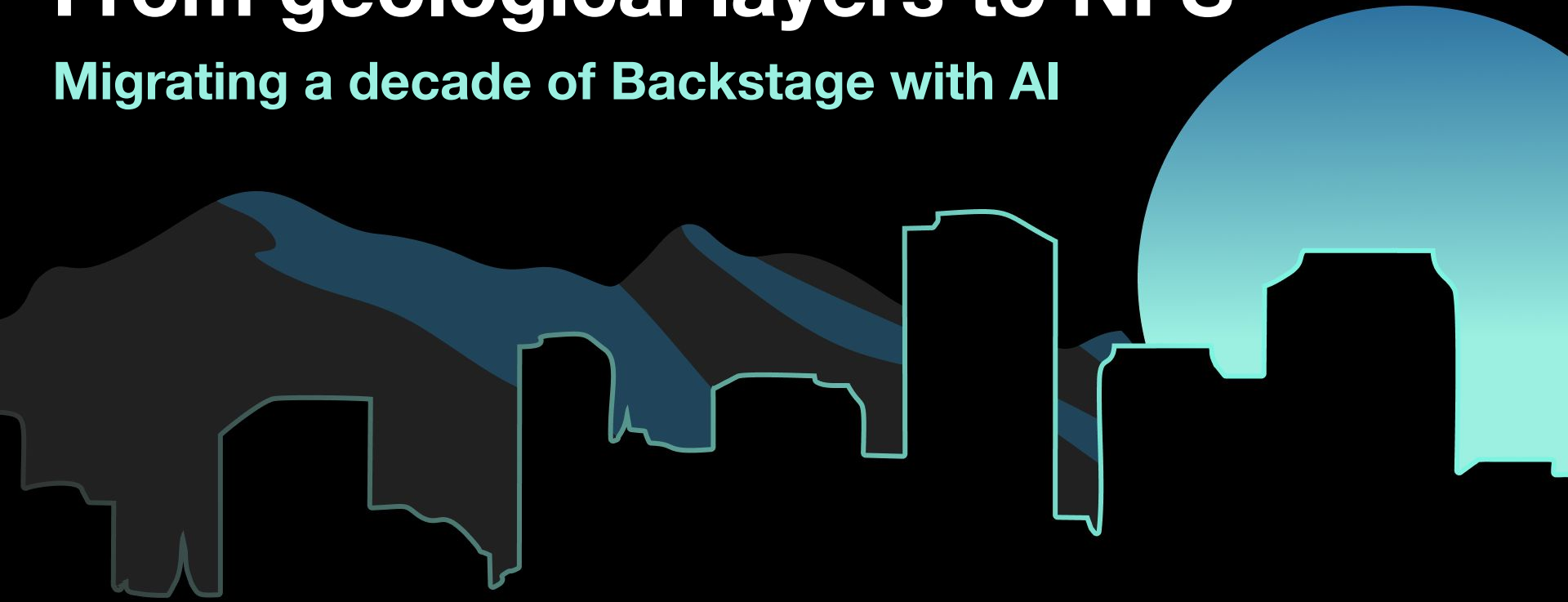
Backstage | CON

From geological layers to NFS

Migrating a decade of Backstage with AI



Speakers



Marat Dyatko

Dyatko

Senior Engineer @ Spotify
Contributor | Backstage



Patrik Oldsberg

Rugvip

Staff Engineer @ Spotify
Core Maintainer | Backstage

Agenda

01 Starting Point

02 Migration

03 Outcome

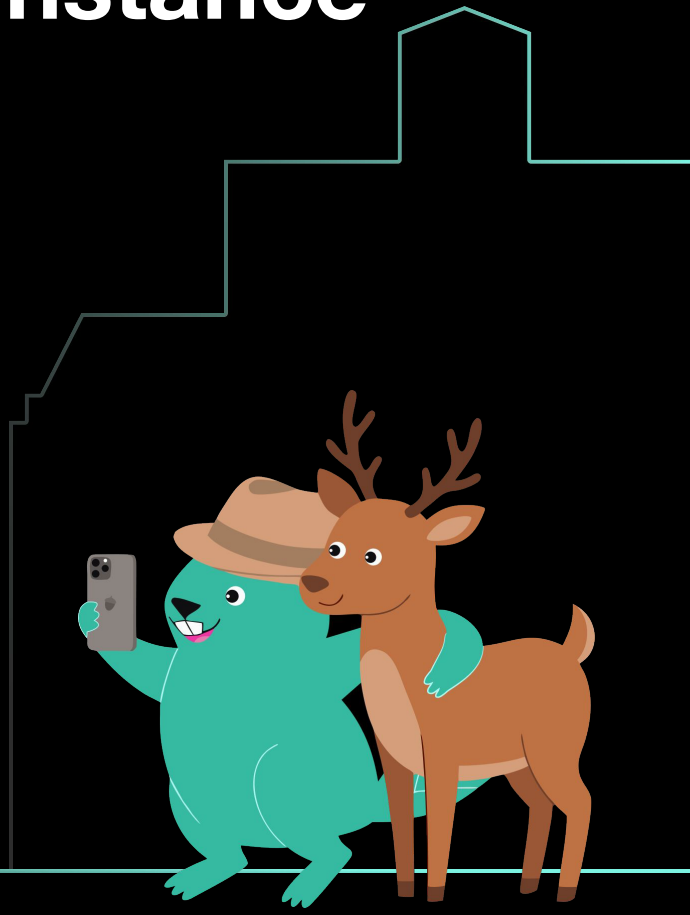




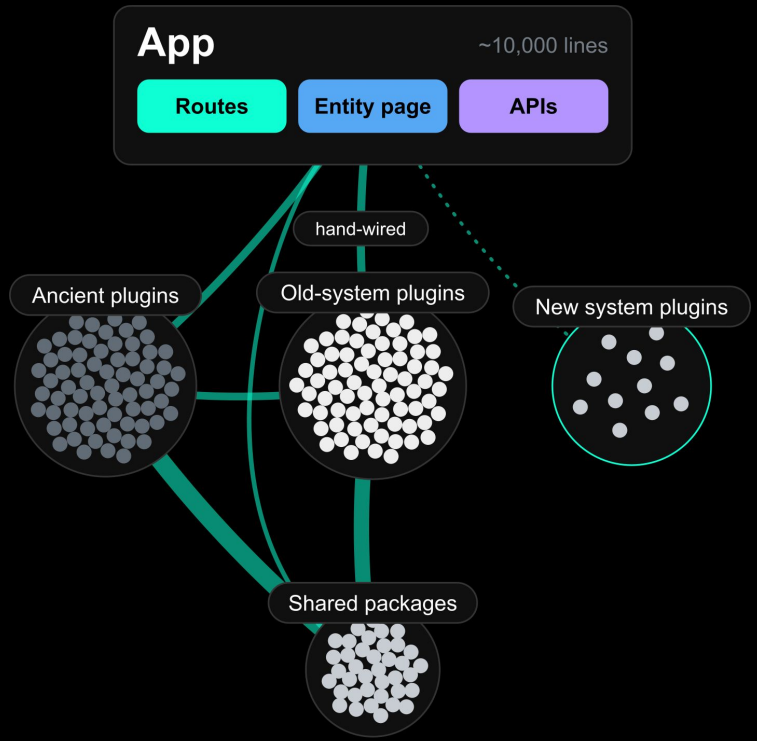
Starting Point

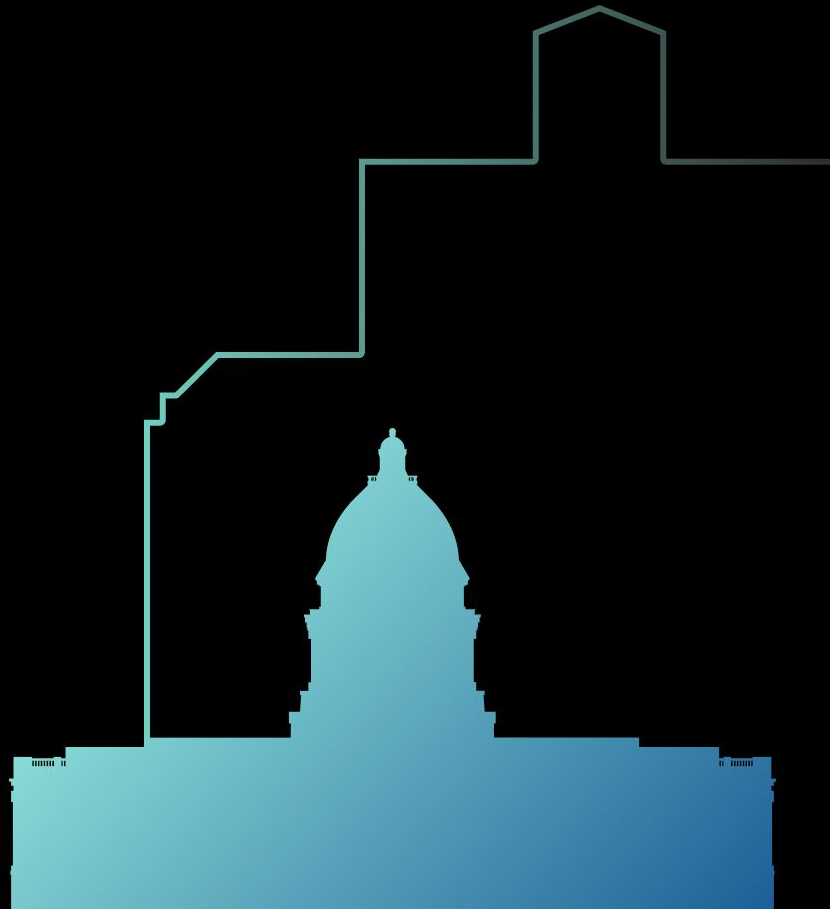
Starting point – Spotify's Instance

- Daily traffic: ~3 000 users
- Repository started in 2014
- ~150 plugins owned by ~70 teams
- Monthly ~200 builders, ~600 commits
- Maintained by central team of 4 engineers
- Frontend only, backends are separate

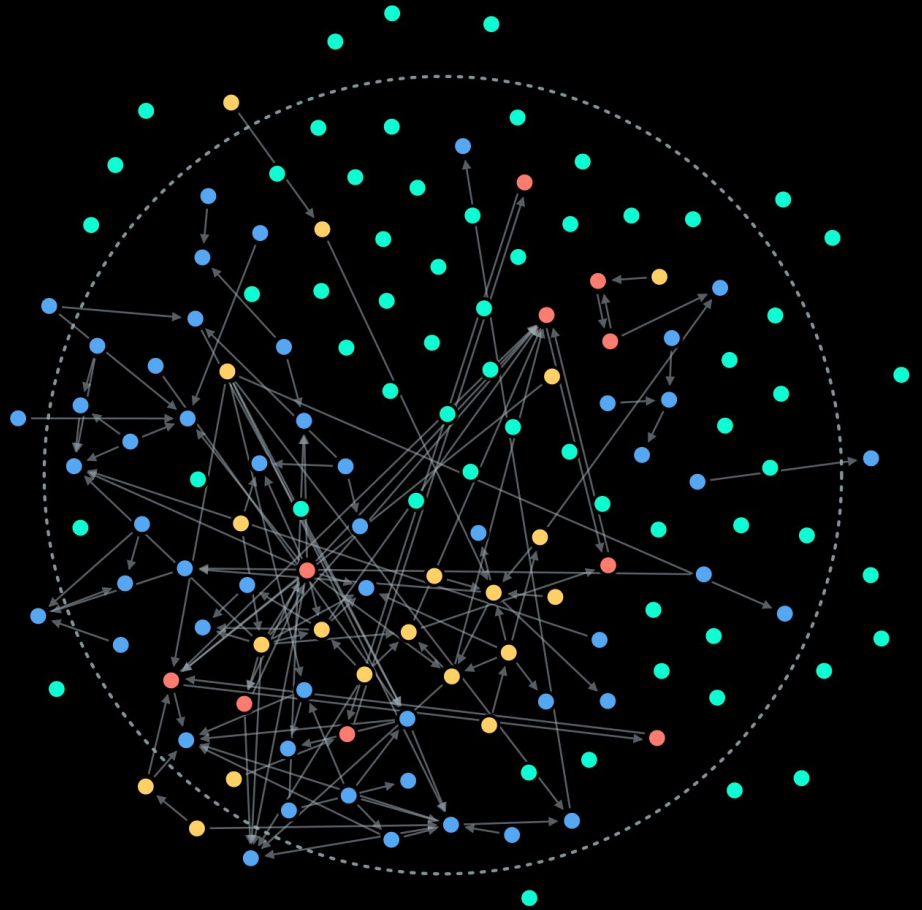


Starting point – Architecture







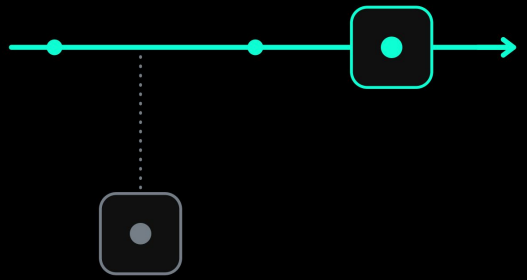


Backstage | CON

Self-contained Clean integration Tied to cycles Locked in a cycle

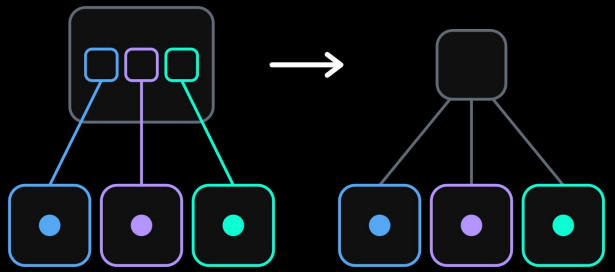
Starting point – Why migrate?

- Staying up to date



Starting point – Why migrate?

- Staying up to date
- Reduced centralized wiring
  - Less overhead



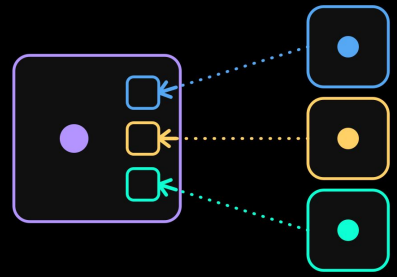
Starting point – Why migrate?

- Staying up to date
- Reduced centralized wiring
  - Less overhead
- Improved feature segregation
  - Faster and contained iteration
  - Failure isolation – improved reliability



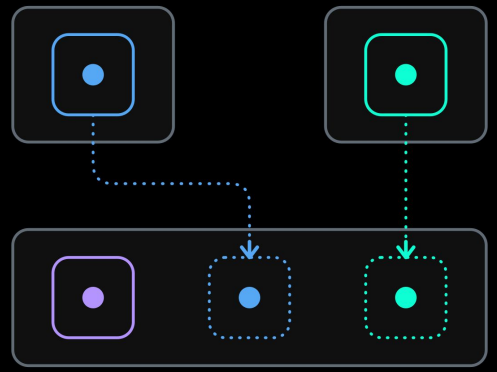
Starting point – Why migrate?

- Staying up to date
- Reduced centralized wiring
  - Less overhead
- Improved feature segregation
  - Faster and contained iteration
  - Failure isolation – improved reliability
- Internal extensibility
  - Cleaner architecture



Starting point – Why migrate?

- Staying up to date
- Reduced centralized wiring
  - Less overhead
- Improved feature segregation
  - Faster and contained iteration
  - Failure isolation – improved reliability
- Internal extensibility
  - Cleaner architecture
- Deployment flexibility
  - Dynamic plugins
  - Plugin owners own deployment lifecycle





Migration

Migration – Strategy

- Avoid a drawn out migration with a long tail
- Centralized – we did not ask teams to migrate themselves
- Complete migration vs dual support
  - Completely migrate simple plugins, e.g. single page
  - Dual support + feature flag complex plugins, e.g. entity pages
- Out of scope
  - BUI migration, ongoing at the same time but not connected
- Optimized for automation with coding agents



Migration – Preparation

- Enable NFS in the app
  - Enable partial migration of entity pages
- Update documentation and agentic instructions
  - That plugins must be built using NFS
  - That migration is ongoing
- Review plugin inventory
  - Validate owners + contact point
  - Remove stale plugins



Migration – Execution

Main migration skill

- Based on the OSS full-migration skill
- Defined Spotify-specific layers and decision tree: plain plugin, entity tab, etc
- 24 revisions: every surprise became a step, a pitfall or a check

How we ran agents

- 4 engineers, each with their own tools: Cursor, Claude Code, Codex
- We used regular coding models like Opus, not Fable-level ones
- The agent claims the ticket, follows the skill, opens the PR and cross-links

Migration – Communication

- We got mandate to force-merge and ship things quickly
- Merge deadline 2 working days out 😈
- PR template to help plugin owner to understand why they got this PR, with links to the docs and the review guide
- Preview build for each PR so they could simply open link and test

Migration Context

All plugins are being migrated to the New Frontend System for Portal. Your plugin's behavior stays the same — this is a wiring change, not a rewrite. See the review guide for what to check.

> ⏰ **Heads up:** We'd love your review, but to keep the migration moving, this PR will be merged on **Thursday, May 7**. If you need more time or have concerns, just leave a comment and we'll work with you.

Summary

Migrates `@internal/plugin-ships-log` to the New Frontend System (BEL-1202). This is a **Path C / component-only** migration: the plugin had no `plugin.ts`, no routes, and no API factories — it only exports `ShipsLogEventsCard` and

Filter changed files

- plugins/ships-log
  - package.json
  - src
    - index.ts
    - plugin.smoke.test.tsx
    - plugin.tsx
  - yarn.lock

Migration – Code changes

```ts
// Before · plugins/license-dash/src/plugin.ts

export const licenseDashPlugin = createPlugin({
  id: 'license-dash',
  routes: { root: rootRouteRef },
  featureFlags: [{ name: 'license-dash-rollout' }],
});

export const LicenseDashPage = licenseDashPlugin.provide(
  createRoutableExtension({
    name: 'LicenseDashPage',
    component: () => import('./components/Dashboard').then(m => m.Dashboard),
    mountPoint: rootRouteRef,
  }),
);

// Before · packages/app/src/app/plugins/index.ts

import { licenseDashPlugin } from '@internal/plugin-license-dash';

export const plugins: BackstagePlugin[] = [
  // …
  licenseDashPlugin,
];

// Before · packages/app/src/App.tsx

import { LicenseDashPage } from '@internal/plugin-license-dash';
// …
<Route path="/license-dash" element={<LicenseDashPage />} />
```

```ts
// After · plugins/license-dash/src/plugin.tsx: the only file

const licenseDashPage = PageBlueprint.make({
  params: {
    path: '/license-dash',
    routeRef: rootRouteRef,
    loader: () => import('./components/Dashboard').then(m => <m.Dashboard />),
  },
});

export default createFrontendPlugin({
  pluginId: 'license-dash',
  title: 'License Dash',
  icon: <RiKey2Line />,
  featureFlags: [{ name: 'license-dash-rollout' }],
  extensions: [licenseDashPage],
});
```

Migration – Code changes

```tsx
// Before · packages/app/src/components/EntityPage/kinds/component.tsx

import {
  ThreatModelingPageEntity,
  isThreatModelingApplicableToEntity as isThreatModelingAvailable,
} from '@internal/plugin-threat-modeling';
// … one import per tab

<EntityLayout.Group title="Observability">
  {/* … */}
  <EntityLayout.Route path="/threat-modeling" title="Threat Modeling" if=
{isThreatModelingAvailable}>
    <ThreatModelingPageEntity />
  </EntityLayout.Route>
</EntityLayout.Group>

// Before · packages/app/src/components/EntityPage/kinds/system.tsx: same again

import {
  ThreatModelingPageEntity,
  isThreatModelingApplicableToEntity as isThreatModelingAvailable,
} from '@internal/plugin-threat-modeling';

<EntityLayout.Group title="Threat Modeling">
  <EntityLayout.Route path="/threat-modeling" title="Threat Modeling" if=
{isThreatModelingAvailable}>
    <ThreatModelingPageEntity />
  </EntityLayout.Route>
</EntityLayout.Group>
```

```tsx
// After · plugins/threat-modeling/src/plugin.tsx

const threatModelingTab = EntityContentBlueprint.make({
  params: {
    path: '/threat-modeling',
    title: 'Threat Modeling',
    group: 'observability',
    filter: { kind: { $in: ['component', 'system'] } },
    loader: () => import('./components/ThreatModelingPage').then(m =>
<m.ThreatModelingPage />),
  },
});

export default createFrontendPlugin({
  pluginId: 'threat-modeling',
  extensions: [threatModelingTab],
});
```

Migration – Code changes

```tsx
// Before · packages/app/src/App.tsx

import { ScaffolderPage } from '@backstage/plugin-scaffolder';
import { ScaffolderFieldExtensions } from '@backstage/plugin-scaffolder-react';
import { RepoPickerExtension } from '@internal/plugin-templates';
import { UniqueComponentIdFieldExtension } from '@internal/plugin-scaffolder-
overrides';
// …

const fieldExtensions = (
  <ScaffolderFieldExtensions>
    <RepoPickerExtension />
    <UniqueComponentIdFieldExtension />
    {/* … 6 more */}
  </ScaffolderFieldExtensions>
);
// …
<Route path="/create" element={<ScaffolderPage /* … */ />}>
  {fieldExtensions}
</Route>
```

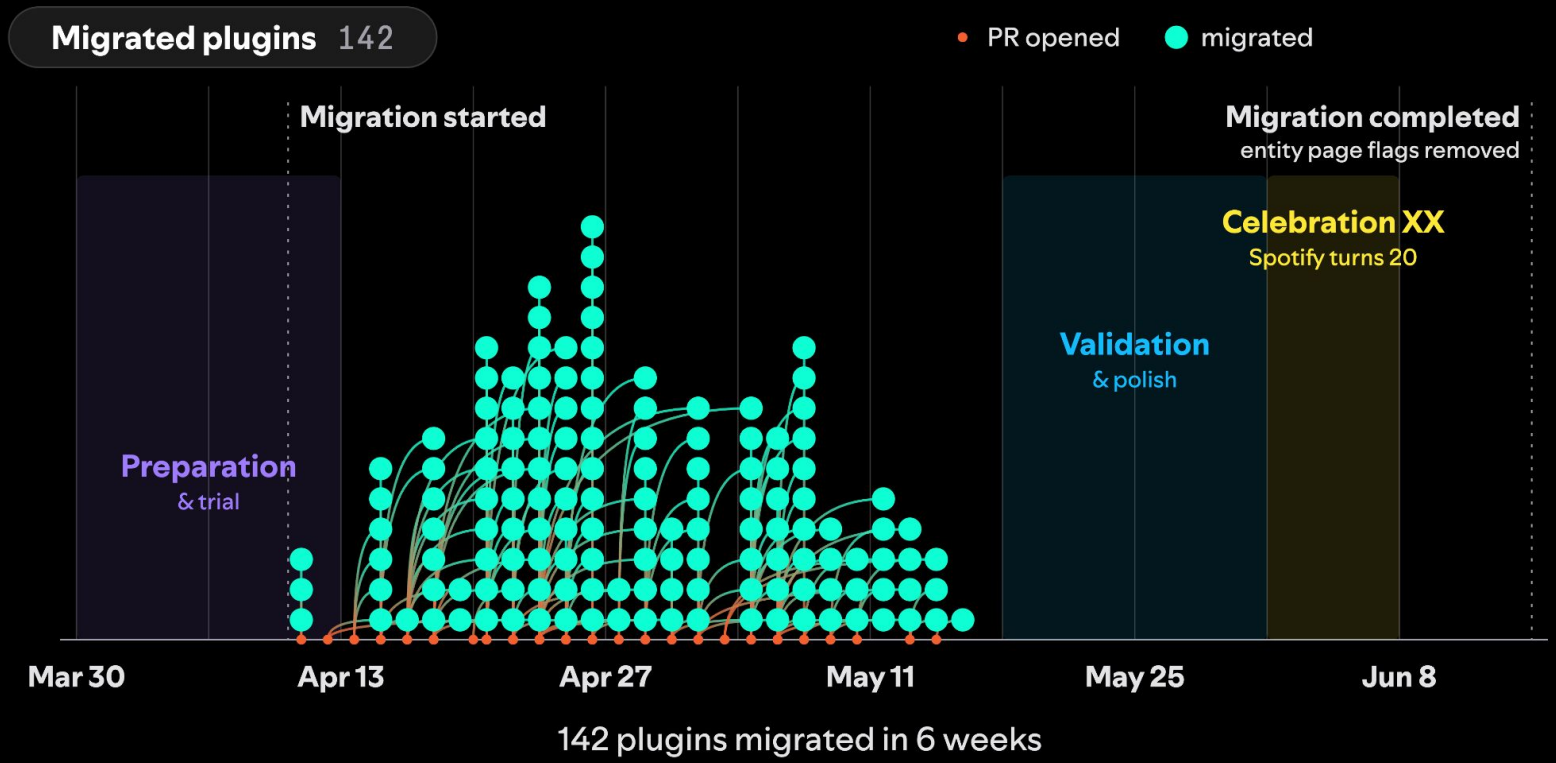
```tsx
// After · plugins/scaffolder-overrides/src/plugin.tsx

import scaffolderPlugin from '@backstage/plugin-scaffolder/alpha';
import { createFormField, FormFieldBlueprint } from '@backstage/plugin-
scaffolder-react/alpha';

const repoPickerField = FormFieldBlueprint.make({
  name: 'repo-picker',
  params: {
    field: () =>
      import('./extensions/RepoPicker').then(m =>
        createFormField({
          name: 'RepoPickerExtension',
          component: m.RepoPicker,
          schema: m.schema
        }),
      ),
  },
});

export default scaffolderPlugin.withOverrides({
  extensions: [repoPickerField /* , … 8 more */],
});
```

Migration – Timeline



Migra Completion



```tsx
// After · packages/app/src/App.ts (10 lines)

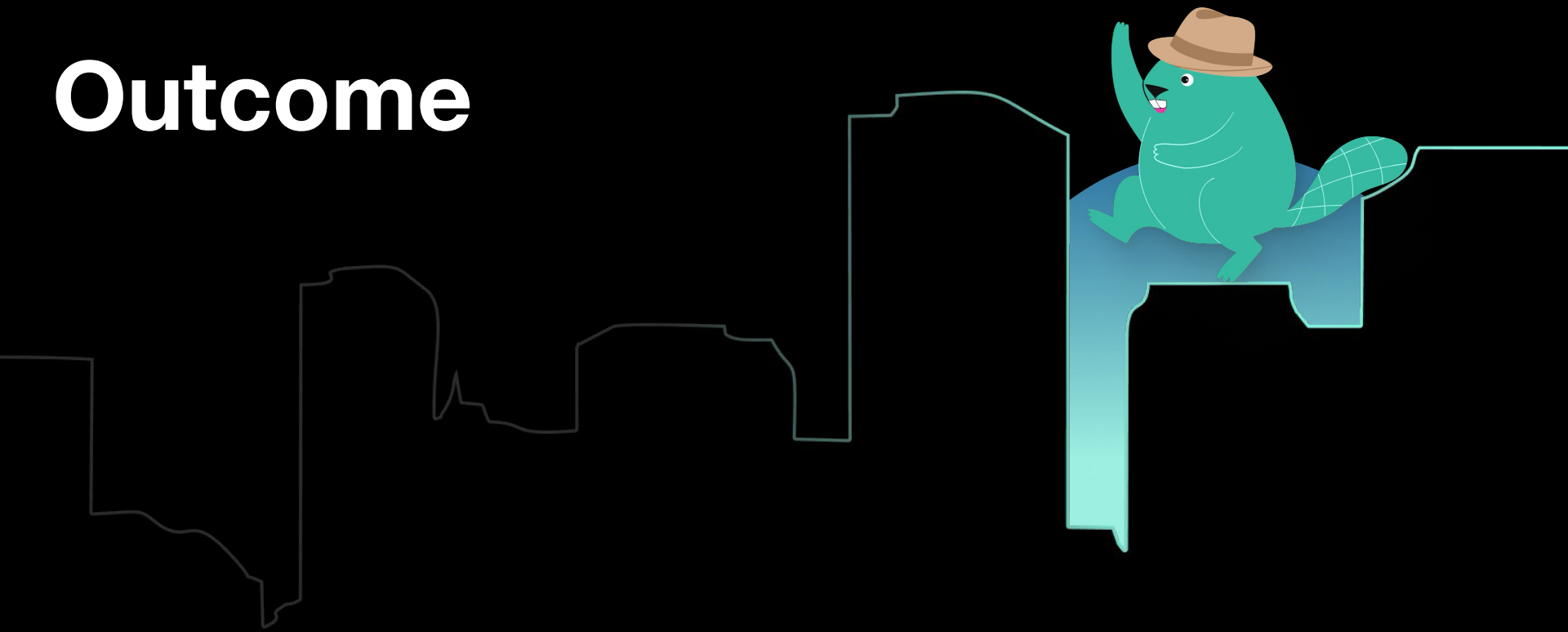
import { createApp } from '@backstage/frontend-defaults';

const app = createApp();

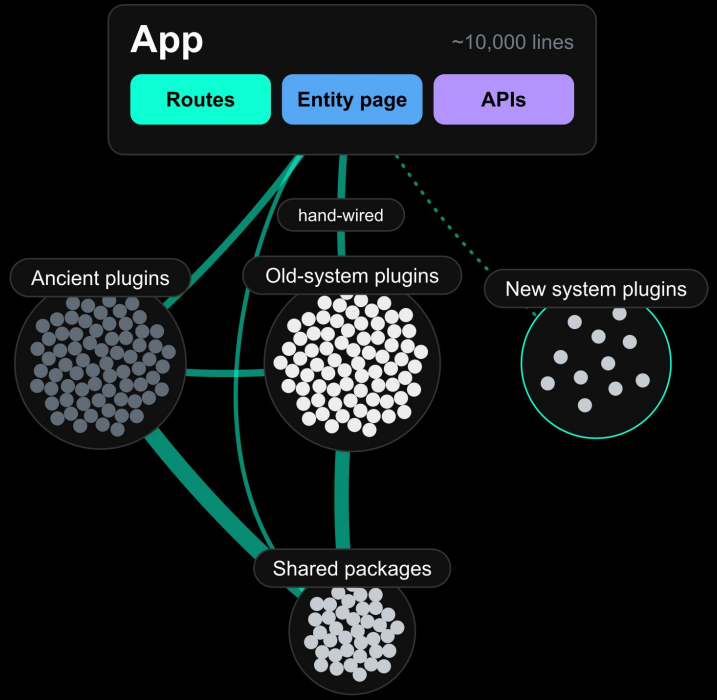
export default app.createRoot();
```

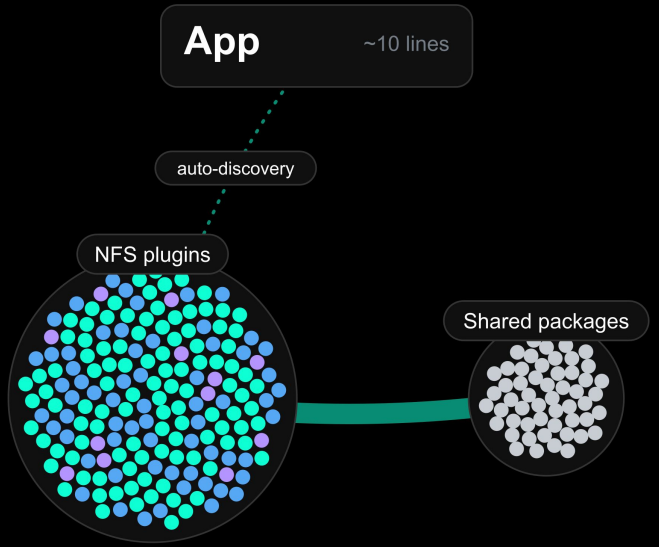
Backstage | c

Outcome

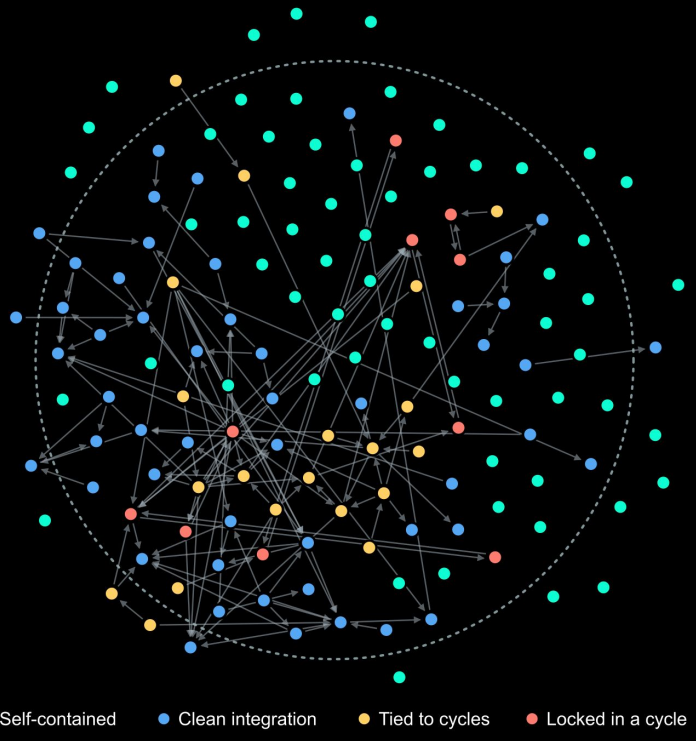


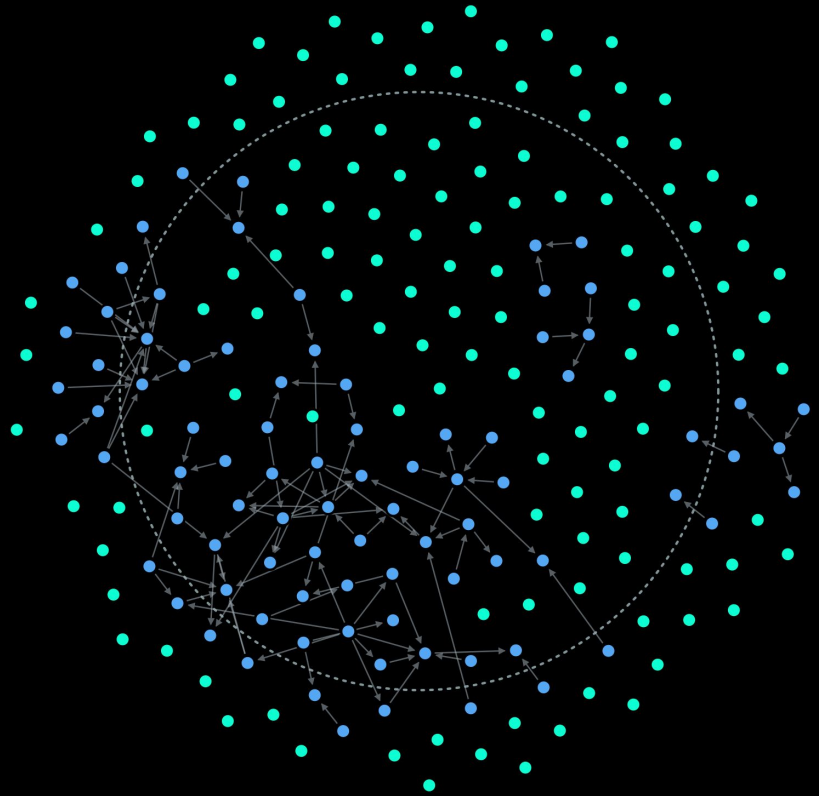
Outcome – Old vs new architecture



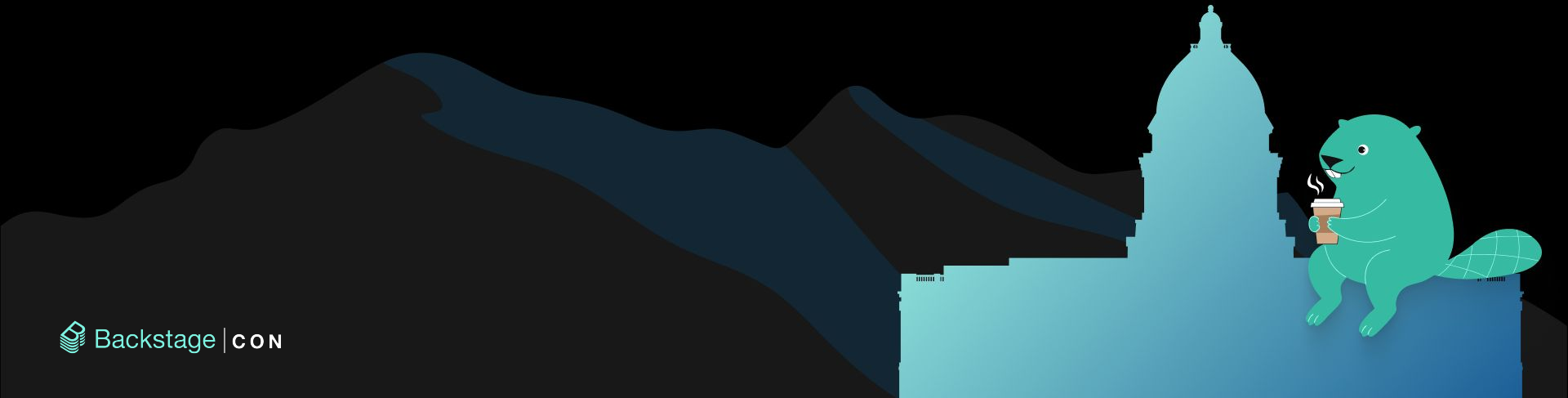


Outcome – Old vs new architecture





Demo





Learnings & Call-to-Action

Learnings

- The App stops being bottleneck
  - Plugins wire themselves; the central team reviews a plugin once, when it’s added
  - The migration clears out years of tech debt across the whole app
- Ready for what Backstage ships next
  - NFS is where the project is going and the only system plugins will support
  - Small, isolated, consistent changes are the best for coding agents
- One team can migrate everything
  - The changes are about wiring and don’t touch what a plugin does
  - Owners only review a small PR; they don’t have to do the work

Call-to-Action

- Spotify is fully moved over to the NFS
  - NFS is 1.0 in the upcoming releases and OFS will be deprecated
- Use **new-frontend-system-migration** skill to migrate
  - npx skills add https://backstage.io
  - github.com/backstage/backstage
- Learn more in “**Built for What’s Next: Backstage’s Role in AI-Assisted Development**” talk on Thu, Nov 12th; KubeCon Maintainer Track
- Come to the Backstage project pavilion, otherwise #frontend-system on Discord