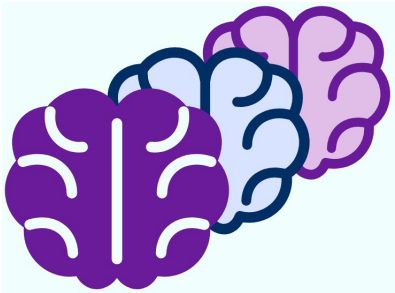
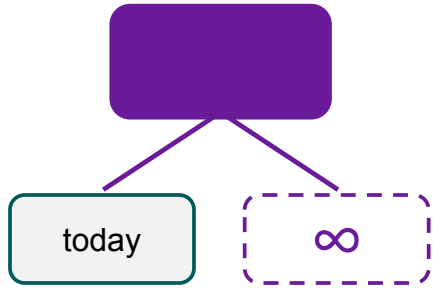
github.com/EPA-WG/cem/discussions/24

CEM DNA for structured data

A template for today's structure and tomorrow's evolution.







Thought

One idea gives rise to others.

DNA

A pattern carried through change.

CEM

A structured data that can evolve.

Sasha Firsov

12 November 2026 · Amsterdam

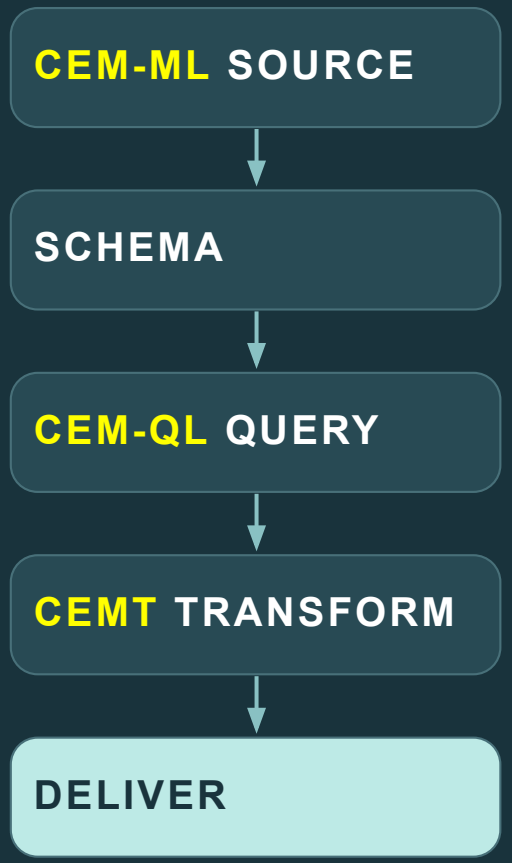
CEM-ML

Custom Element Material

Цьом (Tsyom)

Declarative document lifecycle engine

Evolutionary architecture for the declarative web stack



Sasha Firsov

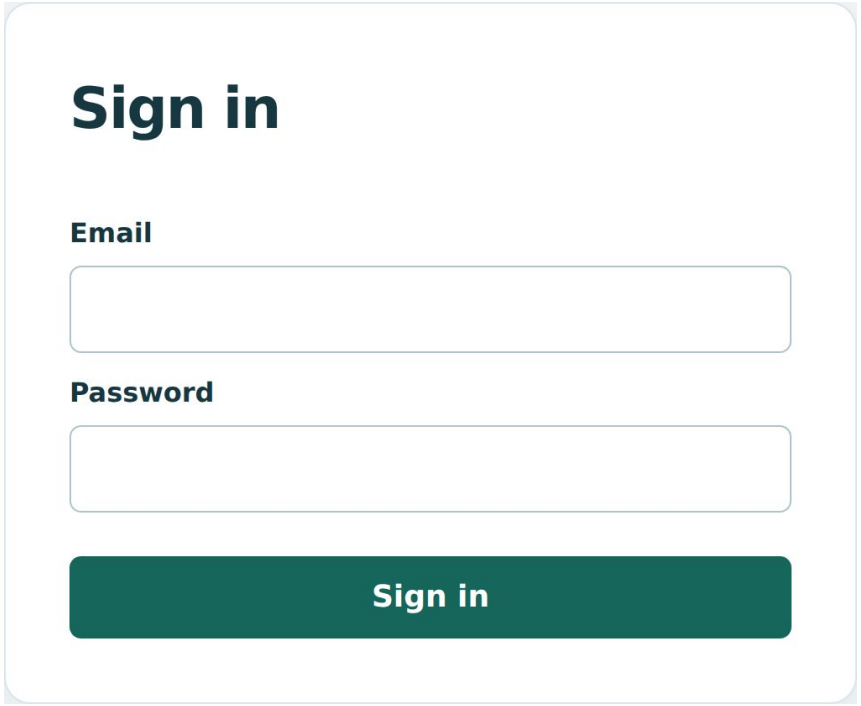
12 November 2026 · Amsterdam

Change the source. See the result.

One login document, followed through its lifecycle.

```
{h1 @id=login-title | Sign in}
```



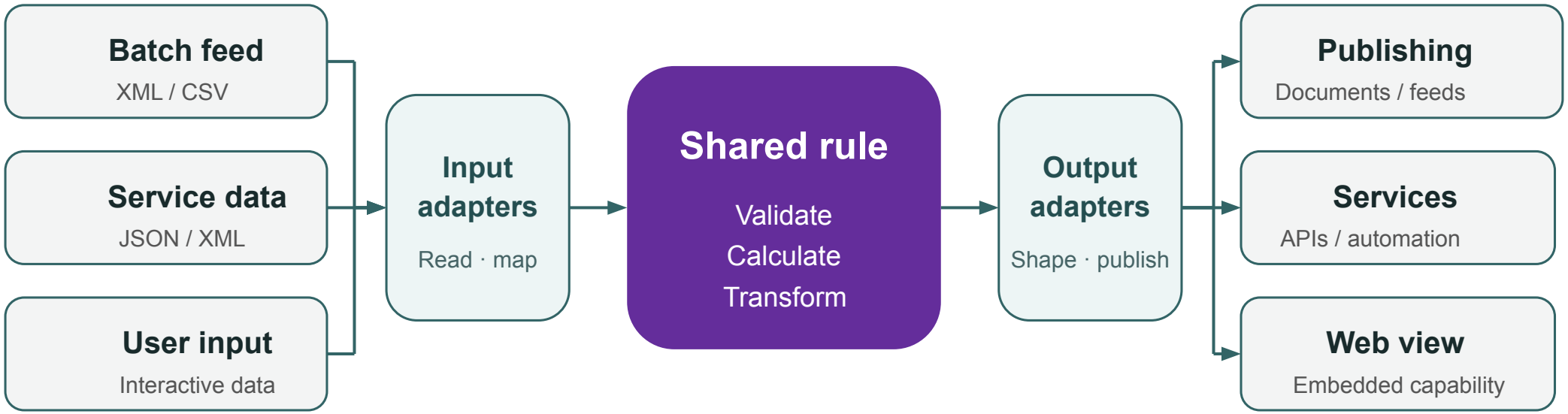


A sign-in view

Edit → validate → inspect → convert → query → transform → render

Keep the rule. Change the delivery path.

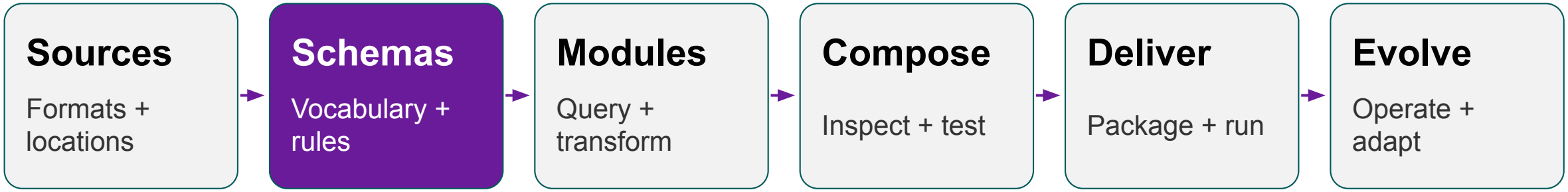
Reusable processing pattern · multiple inputs and outputs



Explicit adapters map vocabularies. Host permissions still apply.

Keep useful rules as systems change

Make the boundaries explicit. Protect the contracts with checks.



New feed. New vendor. New host.
Keep the agreed behaviour; adapt the affected boundary.

Keep your markup knowledge

The same heading and field, expressed in two notations.

XML / XHTML

```
<h1 id="login-title">Sign in</h1>
<label for="email">Email</label>
```

CEM-ML

```
{h1 @id=login-title | Sign in}
{label @for=email | Email}
```

Names · attributes · content

A small vocabulary for the whole journey

Structure remains visible while tools change around it.

```
{h1 @id=login-title | Sign in}
{label @for=email | Email}
```

Namespace

Which vocabulary owns a name

Node

An element and its children

@ attribute

A named value on the node

| content

Text or nested structure

Make one change

Change the heading from “Sign in” to “Welcome back”.

BEFORE

```
{h1 @id=login-title | Sign in}
```

AFTER

```
{h1 @id=login-title | Welcome back}
```

Predict the visible result. Then run it.

Pick, extend, or create your own schemas

Define vocabulary, types and validation rules.

01

Pick

Start with an existing schema and its agreed vocabulary.

02

Extend

Add constraints or specialize a contract for your use case.

03

Create

Describe the vocabulary and rules your own domain needs.

Deliver formatters and colorers with the schema.

Catch the mistake at its source

A teaching schema requires an explicit form method.

```
{form @action="#session"}
```

demo.login.method_required

Login form must declare its method.

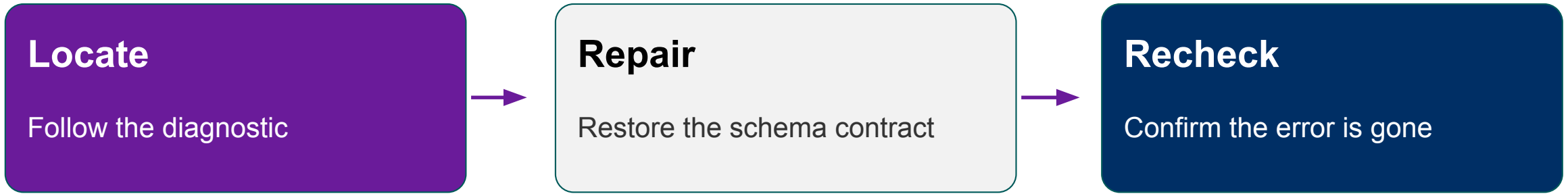
What failed?

Where is the source?

Which rule applies?

Repair the cause. Run the check again.

Use the source location to make one deliberate correction.




```
{form @action="#session" @method=post}
```

Repaired: exit 0 · zero errors · three baseline warnings

Inspect the meaning behind the syntax

Find the same node in the source and the typed structure.

```
{node @id=node-16 @kind=element @order=4 @source-
id=login.xml @parent-id=node-9 @name=input
@byte-offset=252 @byte-length=78
@namespace=http://www.w3.org/1999/xhtml @line=1 @column=253}
```

name = input

source = login.xml

bytes = 252…330

Make structure readable

Adopt pretty and tabular formatters, color output in HTML and terminals.

```
{h1 @id=login-title | Sign in}
{label @for=email | Email}
```

SCHEMA DELIVERY

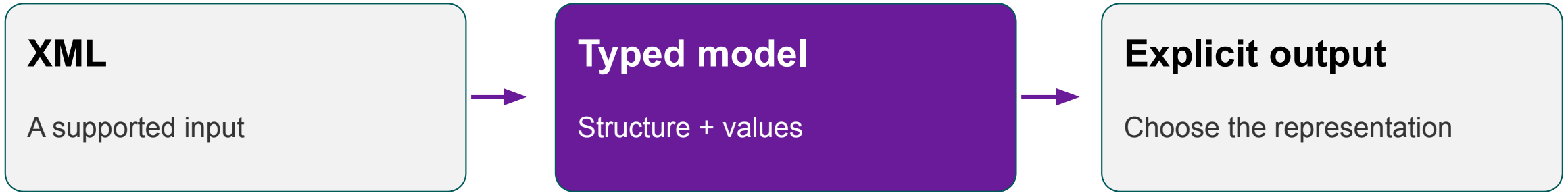
Readable by design

Keep formatting, token colors and diagnostics alongside the vocabulary.

Inspect the structure. Read the report. Follow the source.

Change representation. Preserve agreed meaning.

Import an XML export of the login through the same lifecycle.

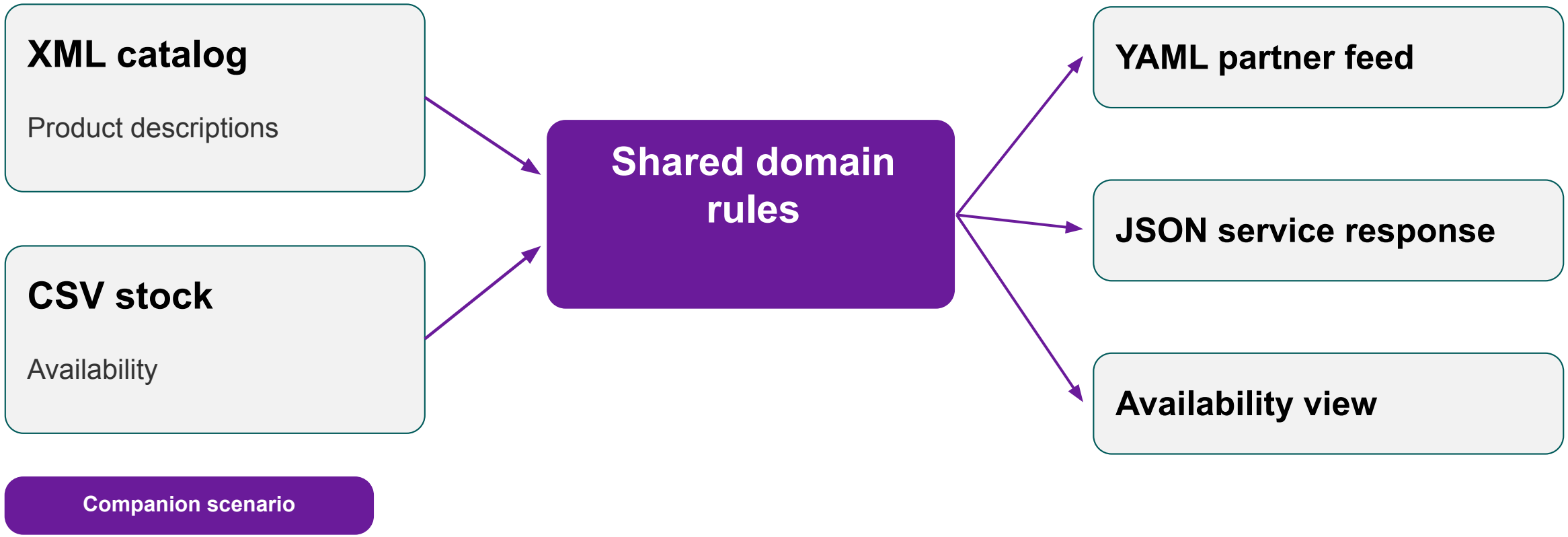


Compare the heading, field names and attributes.

Spelling and mappings can change. State what must stay equivalent.

Multiple inputs. Multiple useful outputs.

Each source and destination can use its own format.



Ask a familiar question

Select the input fields in the login document.

XPATH

```
//*[local-name() = "input"]
```

CEM-QL

```
module "urn:demo:login-inputs"

seq:where(input.descendants,
  fn(n) => n.kind == "element" &&
           n.name == "input")
```

Same two fields

email

password

Declare the transformation

Select data, evaluate expressions, and produce a view.

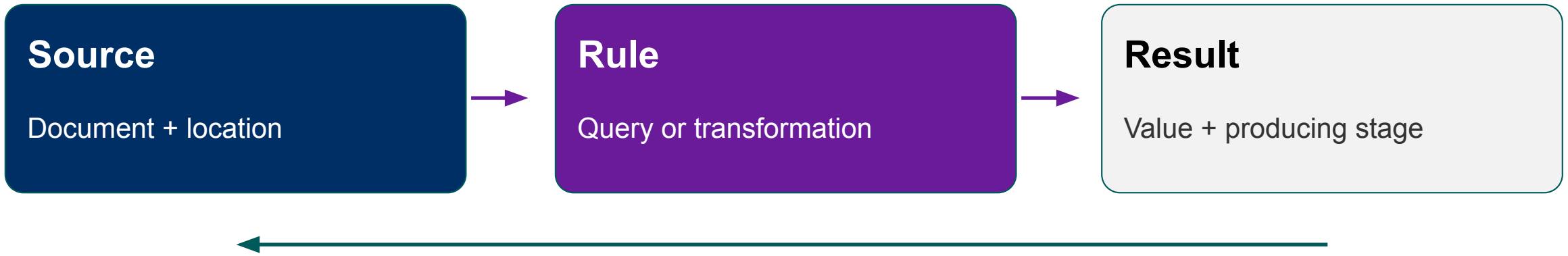


Keep modules reusable. Make dependencies explicit.

Reuse CEM-QL and a defined XSLT profile.

Follow the result back to its source

Source identity and transformation context belong in the lifecycle.

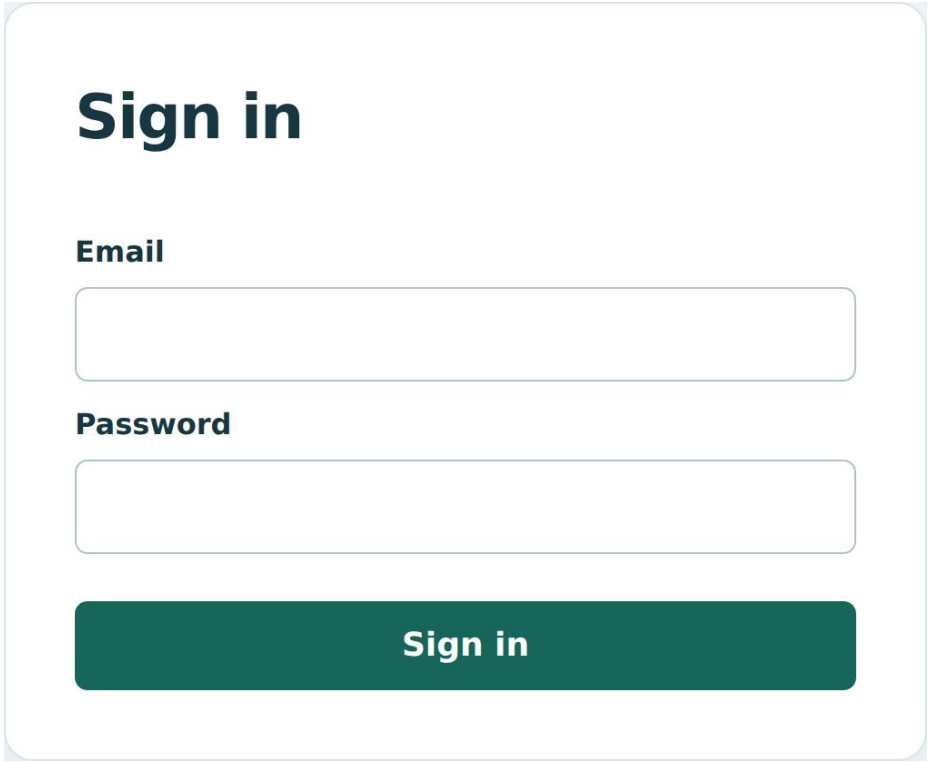


Where did this value come from?

Inspect source spans, diagnostics and the operation report.

Render into the document you can inspect

Declarative data, events, queries and templates meet in <cem-element>.



Static HTML preview

Data → query → template → light DOM

Use ordinary browser inspection.
Keep semantic markup and public styling hooks.

Embedding applications

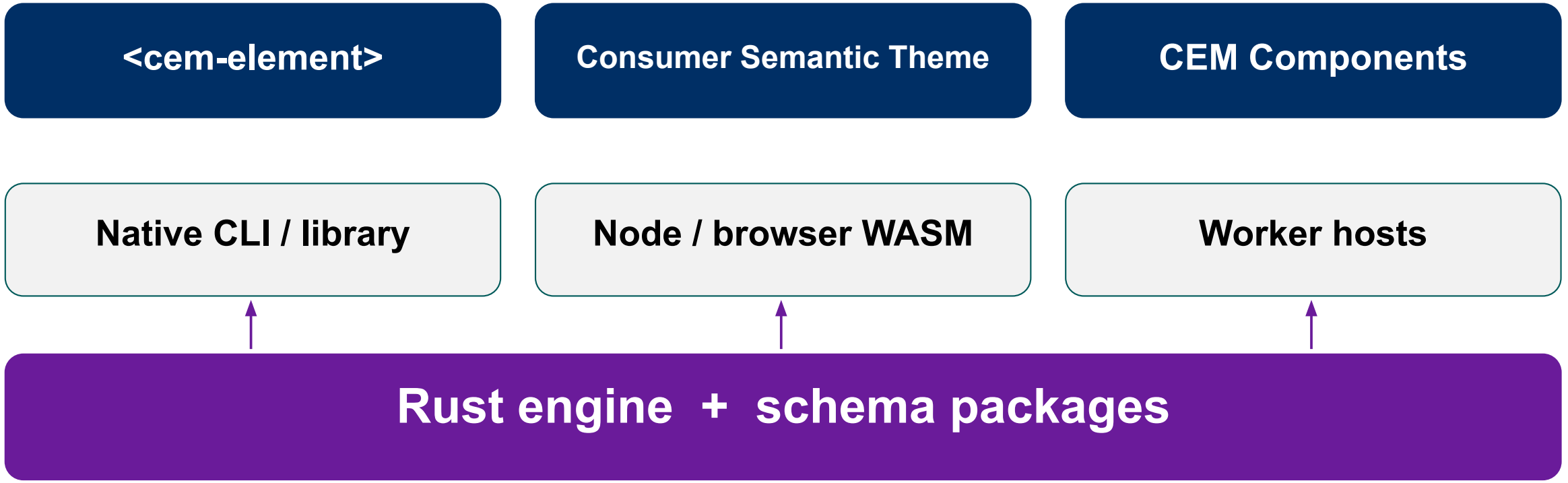
Put a capability inside the workflow that needs it.

“Put a supplier’s availability check inside a buyer’s ordering workflow.”
“Put a specialist transformation inside the pipeline that needs it.”

Embeddable Provider apps maintain the capability.
Host app supplies context, permitted actions and visual integration.

Find your place in the platform

One processing foundation, several ways to adopt it.



CEM Studio · workbench in development

CEM Site · docs and examples

Put processing where the work happens

Connectors read sources and publish to destinations.

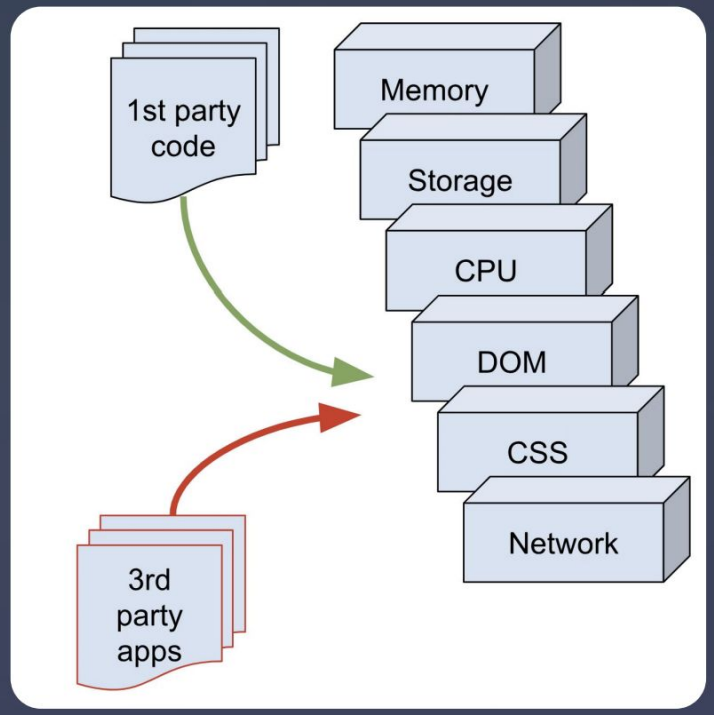
Linux native CLI	Batch, publishing, validation, CI
Library	Services, editors and developer tools
Node / browser WASM	Supported operations in each host
Declarative components	Interactive views in host workflows

Build, extend and customize connectors for your own systems.

Browser vs DCE security

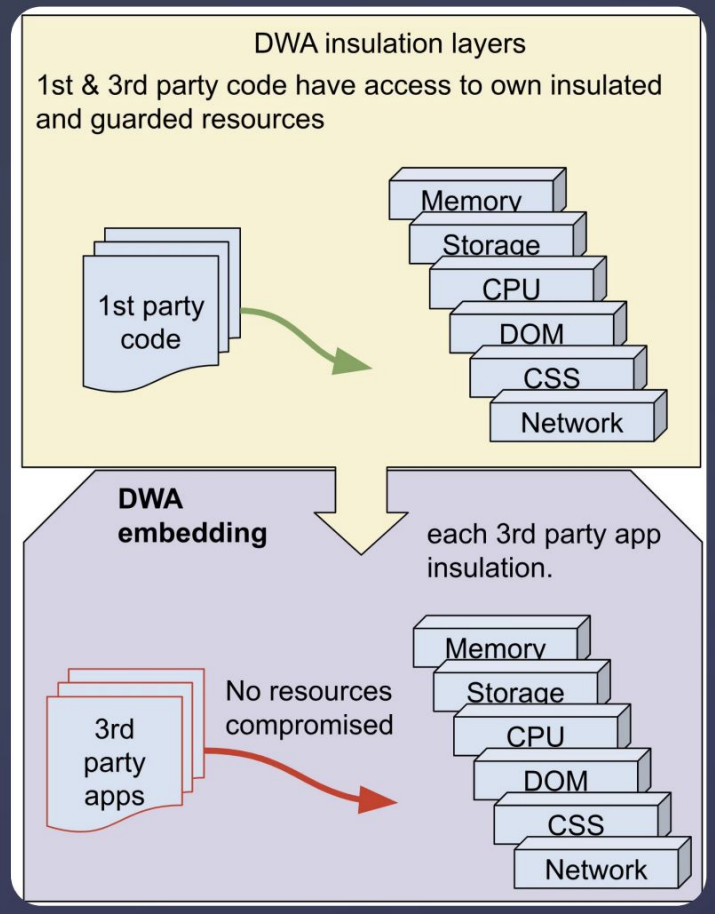
JS apps

1st and 3rd party has a full access



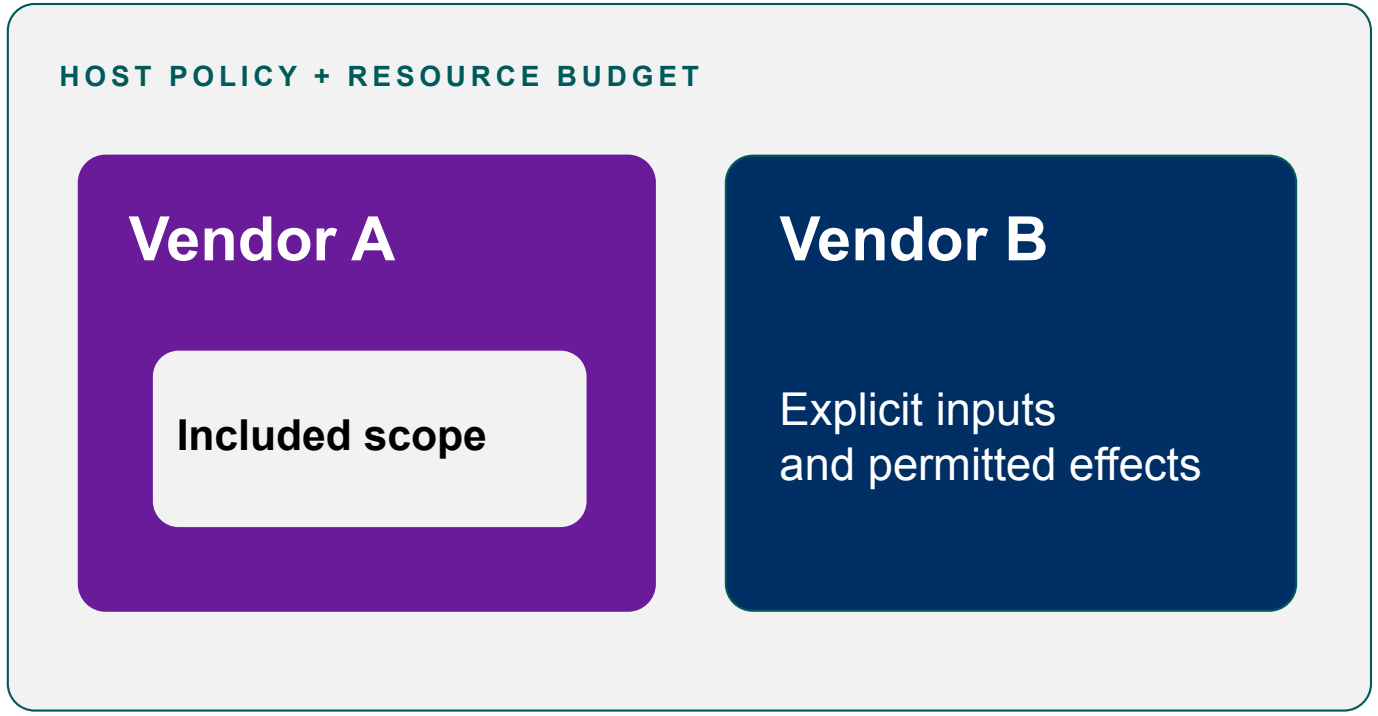
DCE

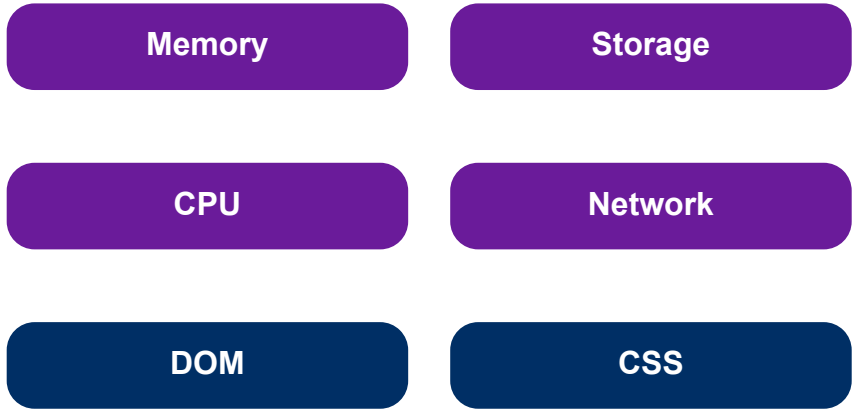
1st and 3rd party have access only to own insulated set of resources



Insulate each vendor’s contribution

Requests → host grants → runtime enforcement





Nested scopes stay within parent authority and budgets.

Architecture target · current controls are partial

Six layers: foundations and boundaries

A control is only as strong as the effect paths it covers.

LAYER	CURRENT FOUNDATIONS	BOUNDARY STILL TO ESTABLISH
Memory	Accounted allocations, inherited ceilings	Complete per-vendor private-memory isolation
Storage	Host ports, explicit export and redaction	Every-path authority and namespace quotas
CPU	Bounded workers, cancellation, deadlines	Preemption of arbitrary in-process code
Network	Resolver policy before configured access	All-path egress, redirects and credentials
DOM	Owned ranges, inert state, workers	Hostile same-page-script confinement
CSS	Native @scope and ownership contracts	Containment of every style/resource effect

Make performance a lifecycle property

Reduce repeated work. Bound resource use. Preserve useful state.

DATA

Retain typed structure

Avoid unnecessary parse / serialize cycles between compatible stages.

EXECUTION

Bound independent work

Use explicit dependencies, worker limits and suitable progressive stages.

DELIVERY

Resume component state

Adopt compatible server output and load declarations when needed.

Transformation graphs connect data pipelines and web asset assembly.

Host and workload determine the benefit. Full bundler parity is a goal.

Follow the source and the evidence

The CLI and code preview work independently of Studio.

SOURCE	examples/cem-ml/login.cem
CODE PRESENTATION	Native tabular formatter → CEM HTML colorer
ASSET EVIDENCE	slide-assets / evidence and command transcript
RESOURCES	github.com/EPA-WG/cem/discussions/24

Use the recorded checkpoints if a local setup fails.

32 schema packages

Data formats, languages and platform contracts; support varies by profile.

Data and markup · 9

csv · html · json · markdown · mathml
svg · xhtml · xml · yaml

Query, style, transformation · 5

css · css-selector · scss
xpath · xslt

External schema resources · 2

json-schema · relax-ng

CEM authoring and schemas · 7

cem-ml · cem-ql · cem-native-template
cem-element-template · cem-transform
schema · schema-package

Platform contracts · 6

cem-data-island · cli-command
module-map · module-map-v2
module-map-v3 · studio-project

Structural projections · 3

cem-ast-projection
cem-dom-projection
cem-events-projection

Includes internal contracts and three module-map revisions.

Reuse expertise within explicit profiles

Start with the operation and input you can verify.

XML / (X)HTML	Import, document structure and explicit delivery mappings
XPath / CEM-QL	Verified equivalent queries on the selected input
XSLT	A defined compilation and runtime profile
Formats + locations	Adapters extend representation; connectors extend access
Hosts	Native, Node and browser capabilities differ

Bring one useful integration

Bring your data. Keep your expertise. Compose the capability you need.

01 **A format boundary to simplify**

02 **A processing rule to reuse**

03 **A partner capability to embed**

Resources and updates

github.com/EPA-WG/cem/discussions/24