When Blicca and Aurora meet to have fun

sharing blocks across Plone's two UI's

Maik Derstappen - md@derico.de - derico.de

Two names, before we start

Blicca

the new name of Plone's **Classic UI**: Python-rendered templates plus JavaScript UI components

Aurora

the successor of **Volto**: the modern, modular full-stack JavaScript frontend

- Same products, new names: the rest of this talk only says **Blicca** and **Aurora**



Chapter 1 — Twenty years of putting things on a page

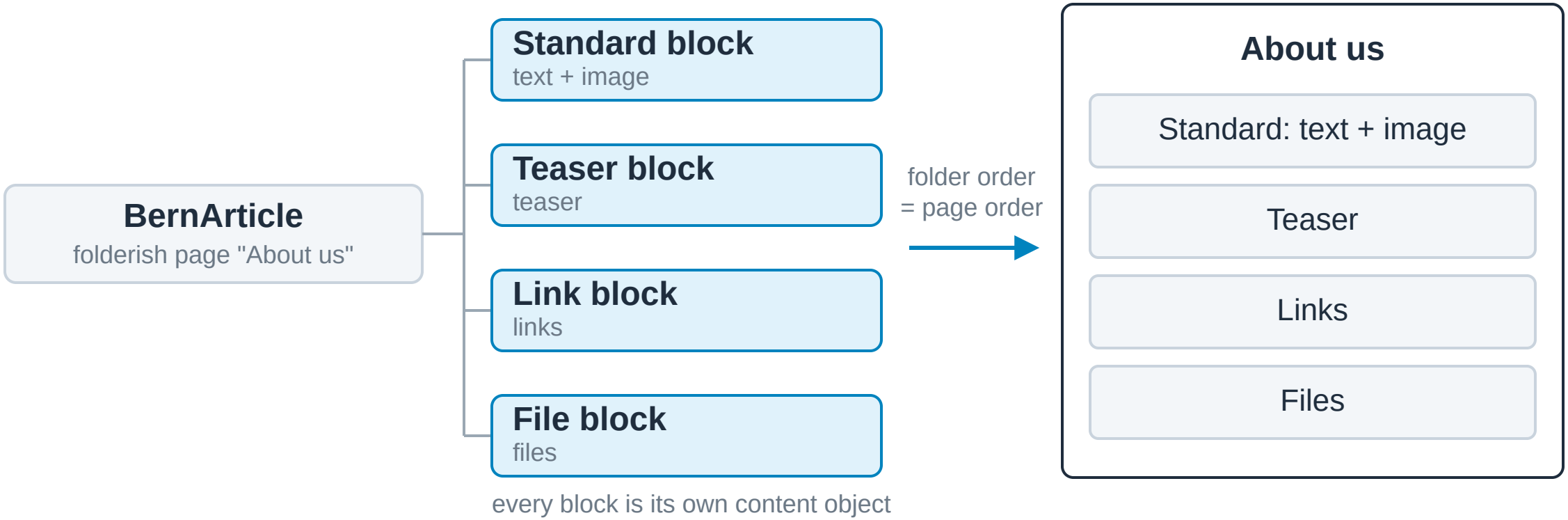
A short history of page composition in Plone

Every solution answered the same two questions:

- **Where does the layout live?**
- **Where does the content live?**

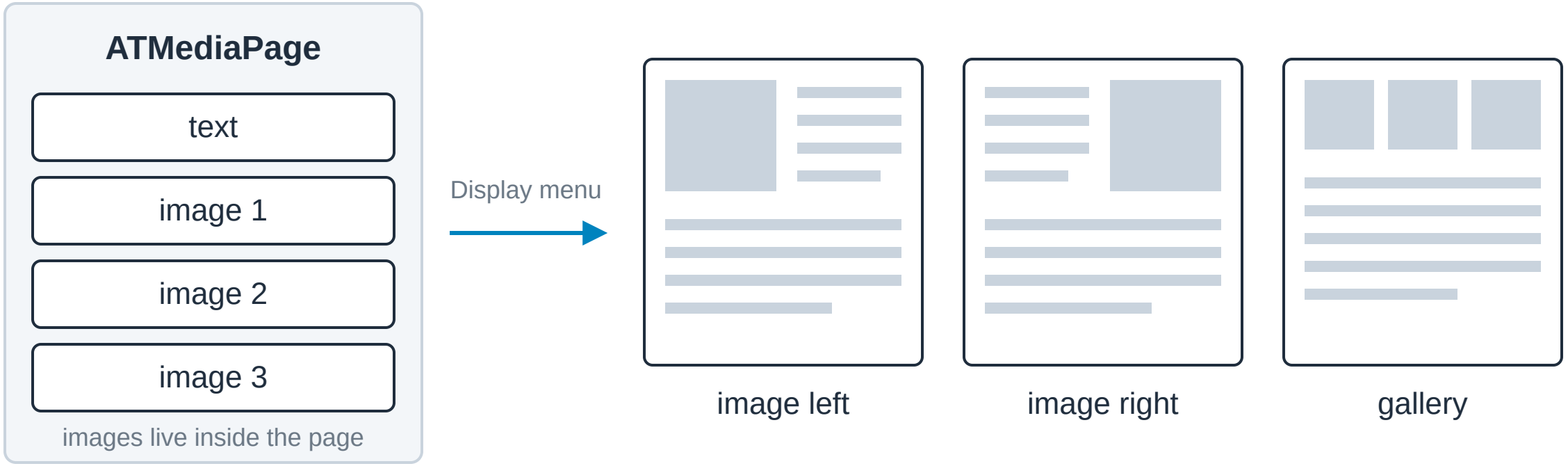


1.1 Act 1 — The content *is* the page



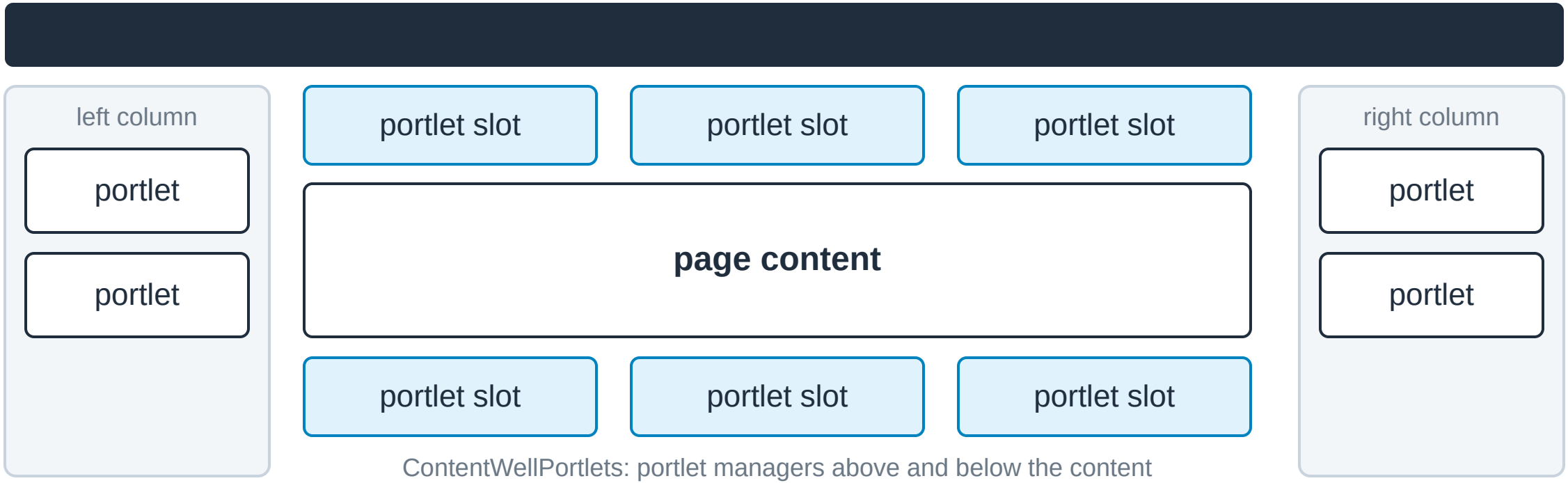
- **BernArticle** (Plone 2.x): the page is a folder, its **children are the blocks**
- Plone had "blocks" long before Volto

1.2 Act 1 — The layout is a view



- **ATMediaPage** (Plone 2.x): fixed content, images live *inside* the page
- The layout is a **selectable view template**, switched in the Display menu

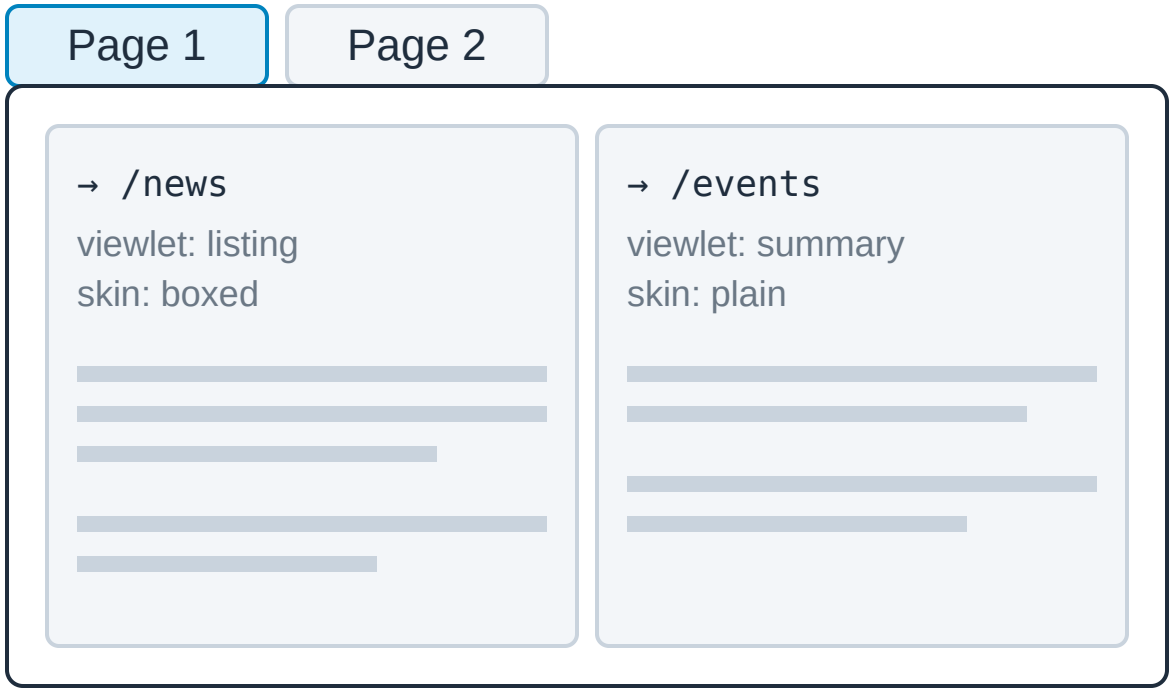
1.3 Act 2 — Borrow the machinery

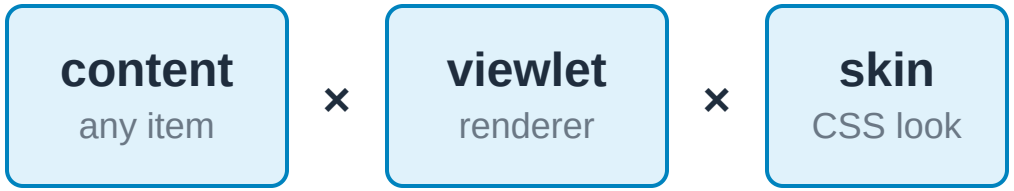


ContentWellPortlets: portlet managers above and below the content

- Plone 3 brought a powerful **portlet engine**: per context, inherited, blockable
- **PortletPage**: a page made of portlet columns · **ContentWellPortlets**: slots around the content

1.4 Act 3 — Compose from what already exists



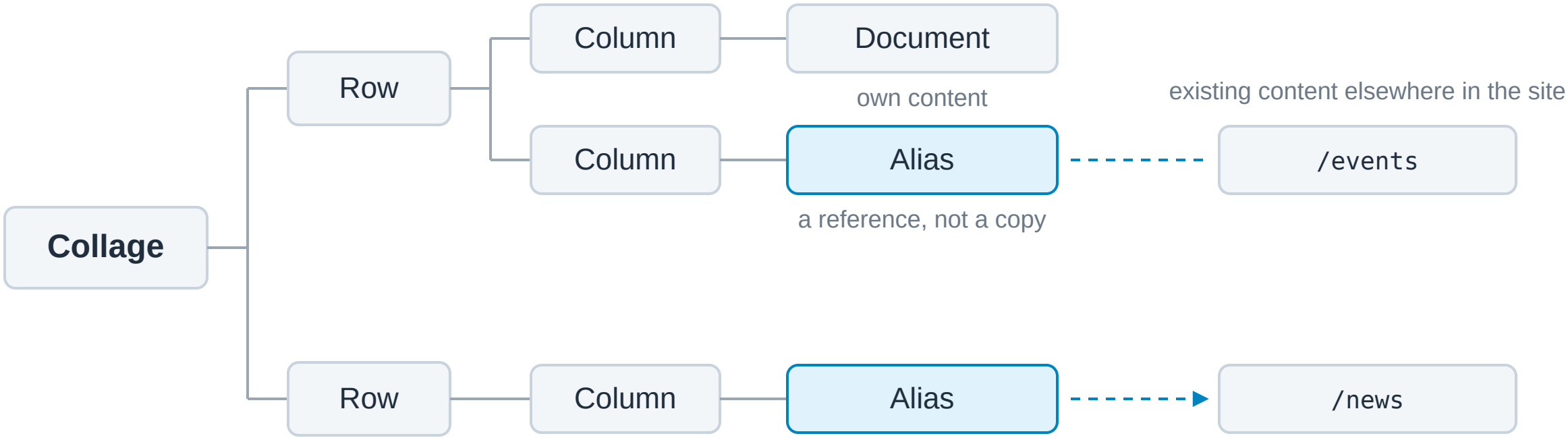


a panel = content × viewlet × skin

stored as configuration on the page

- **CMFContentPanels** (Plone 2 – 3): pages become **compositions** of content from anywhere in the site
- A panel is a stored configuration: which item, which viewlet, which skin

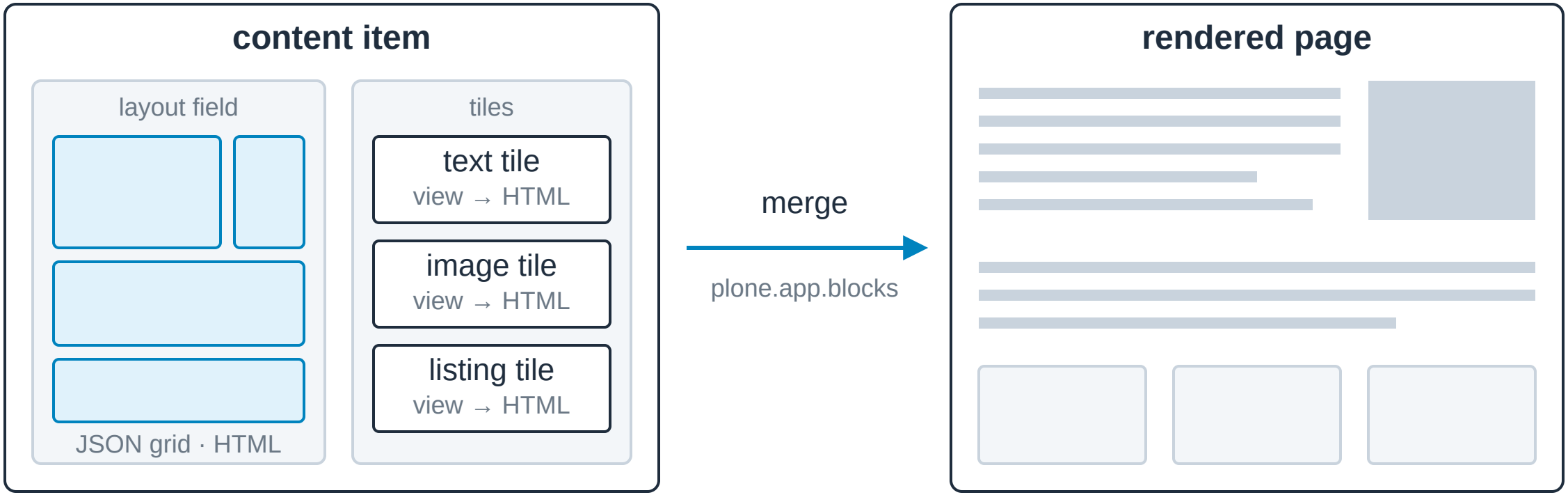
1.5 Act 3 — The structure itself is content



Row and Column are content types too: the structure itself is content

- **Collage** (Plone 3 – 4): Row and Column are content types, **aliases** reference existing items
- Each item picks its own renderer, a browser view

1.6 Act 4 — The layout becomes data: tiles



- **collective.cover · Mosaic** (Plone 4 – 6 Blicca): the layout is stored **in a field**
- Content lives in **tiles**, small views returning HTML; the server merges both

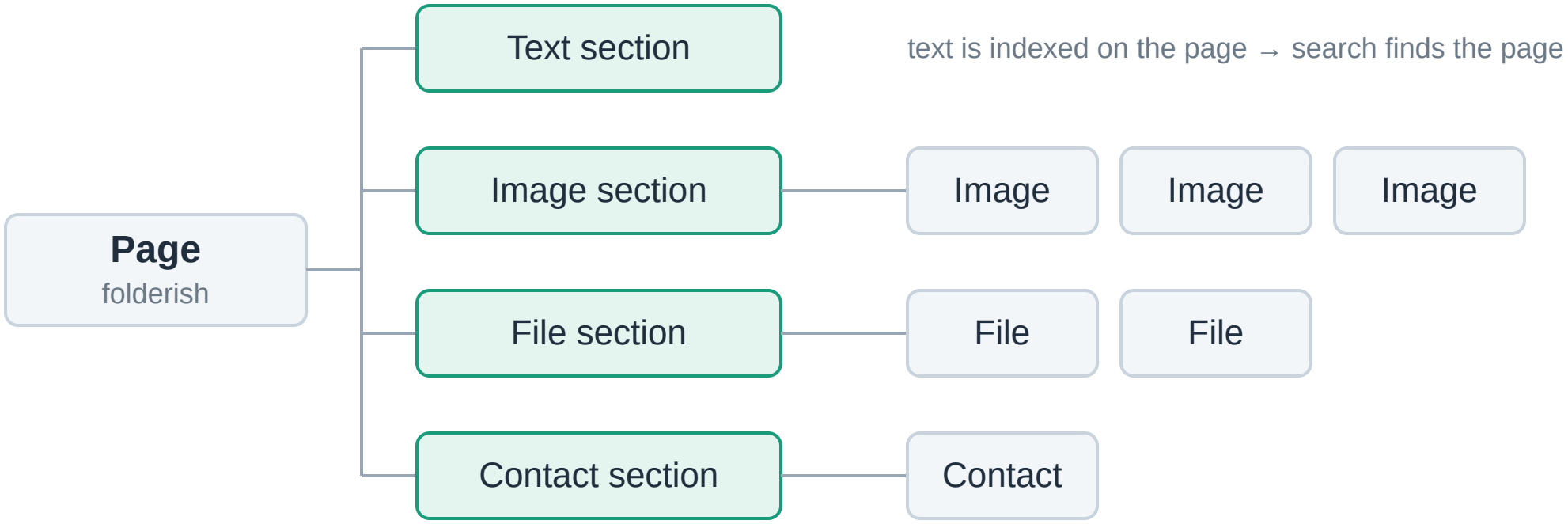
1.7 Act 5 — The page becomes a data document: blocks

```
{
  "blocks": {
    "5f1c…": {"@type": "title"},
    "a93e…": {"@type": "slate", "value": [...]},
    "c07b…": {"@type": "teaser", "href": [...]}
  },
  "blocks_layout": {"items": ["5f1c…", "a93e…", "c07b…"]}
}
```

- **Volto** (Plone 6): the whole page is **JSON data** on the content item
- Delivered by plone.restapi, **rendered in React**

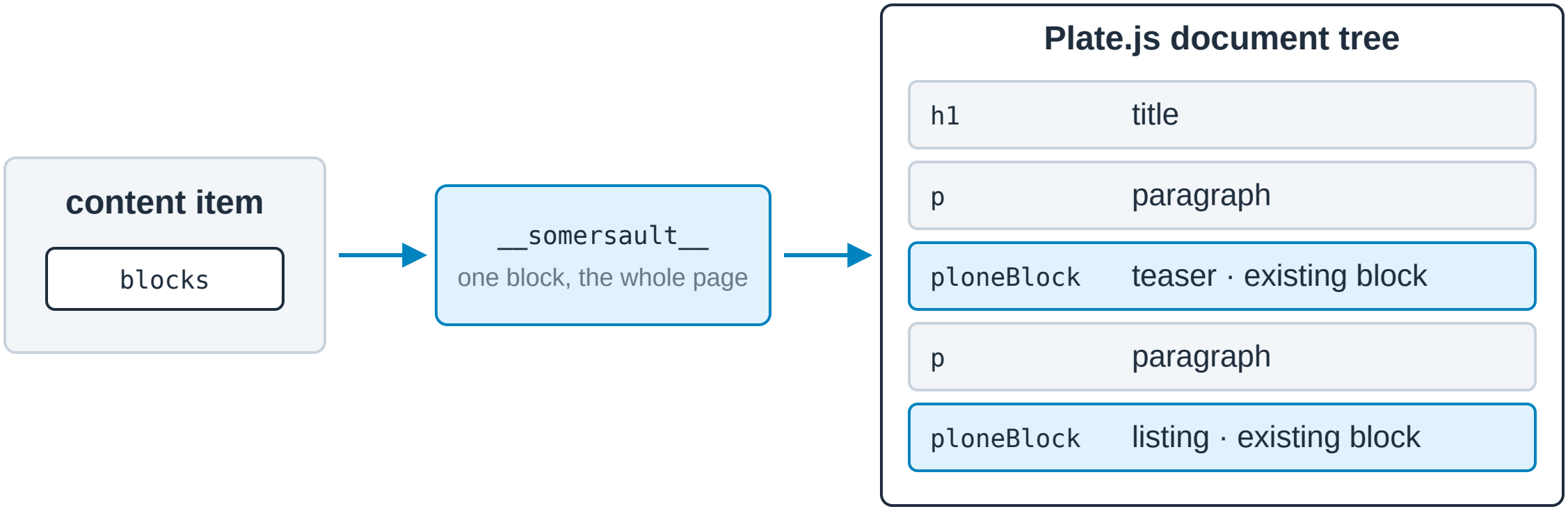


1.8 Plot twist — back to the tree



- **collective.contentsections** (Plone 6 Blicca): sections are plain **Dexterity items**
- BernArticle's idea, modernised: beautiful Blicca sites in less than a day

1.9 And forward — one editor, one tree



- **Aurora** (Plone 6, successor of Volto): the page is **one Plate.js tree**, not a list of isolated editors
- Existing blocks travel along as `ploneBlock` nodes, the block model stays

1.10 The pattern: a pendulum

OBJECTS

layout = content items

DOCUMENTS

layout = data on the item



BernArticle · ATMediaPage
Collage
Contentsections

PortletPage · ContentPanels

cover · Mosaic
Volto
Aurora

every generation rediscovered one side of the pendulum

1.11 Objects or documents

Objects give us	Documents give us
workflow, permissions, search, references per piece	fluid editing, copy & paste, portable JSON
Plone does the work	but opaque to the rest of Plone



1.12 What survived

Idea	Born in	Lives on today in
Building blocks as items	BernArticle, Collage	**Contentsections** · Blicca
Layout as data + server renderers	ContentPanels, cover	**Mosaic tiles** · Blicca
Page as a self-contained document	Volto	**Blocks** · Volto, Aurora

➡ *Three concepts still alive, split across two UIs*



Chapter 2 — Where we stand today

Two UIs, two ways to compose a page

- **Blicca** — Mosaic tiles, Contentsections · Python
- **Volto** → **Aurora** — blocks · JavaScript



2.1 Blicca — pages rendered on the server

Mosaic

Page.customContentLayout

HTML with tile placeholders

```
<div data-tile="…/standardtiles.html/t1">
```

plone.app.blocks transform

merges layout + tile HTML

Contentsections

Page (folderish)

Text section | Image section | Listing section

one view + template per section

schema · view · page template

Python renders on the server → HTML that search, workflow, RSS, mail and PDF can use

- **Python-rendered templates** plus JavaScript UI components
- Add-ons: one Python package, `pip install`, ZCML, GenericSetup

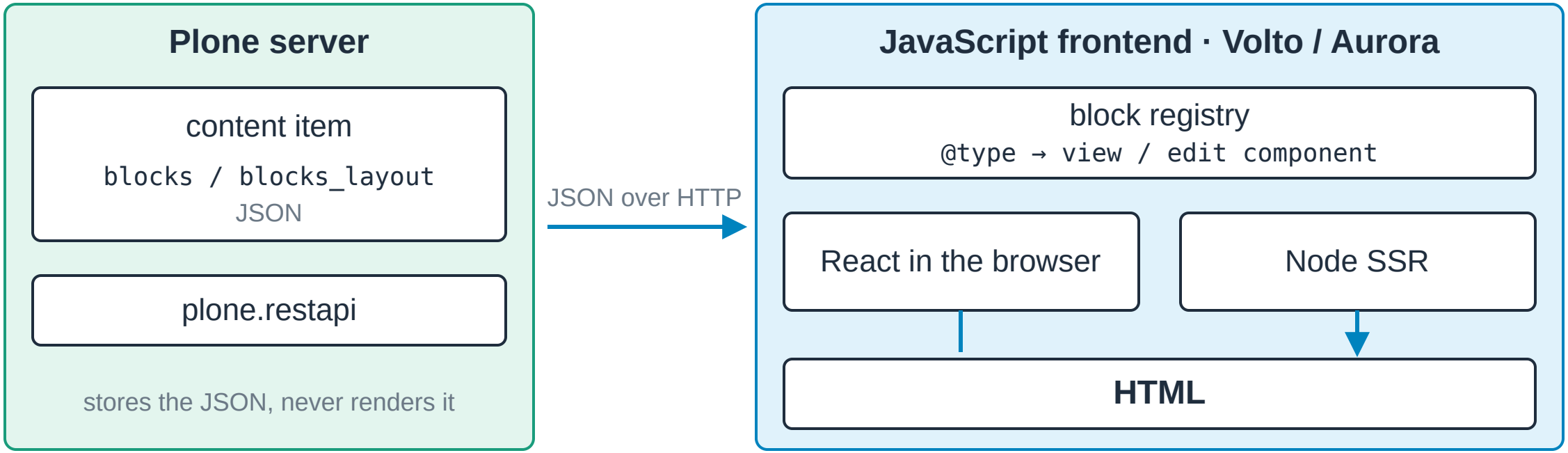
2.2 Volto → Aurora — blocks are JavaScript

```
// Volto and Aurora share the same block registry
config.blocks.blocksConfig.teaser = {
  id: "teaser",
  view: TeaserView,          // React
  edit: TeaserEdit,          // React
  blockSchema: TeaserSchema, // JS object
};
```

- The block registry, the schema and the view are **JavaScript**
- Aurora keeps the blocks and replaces the editor: one Plate.js document



2.3 Volto → Aurora — pages rendered in JavaScript



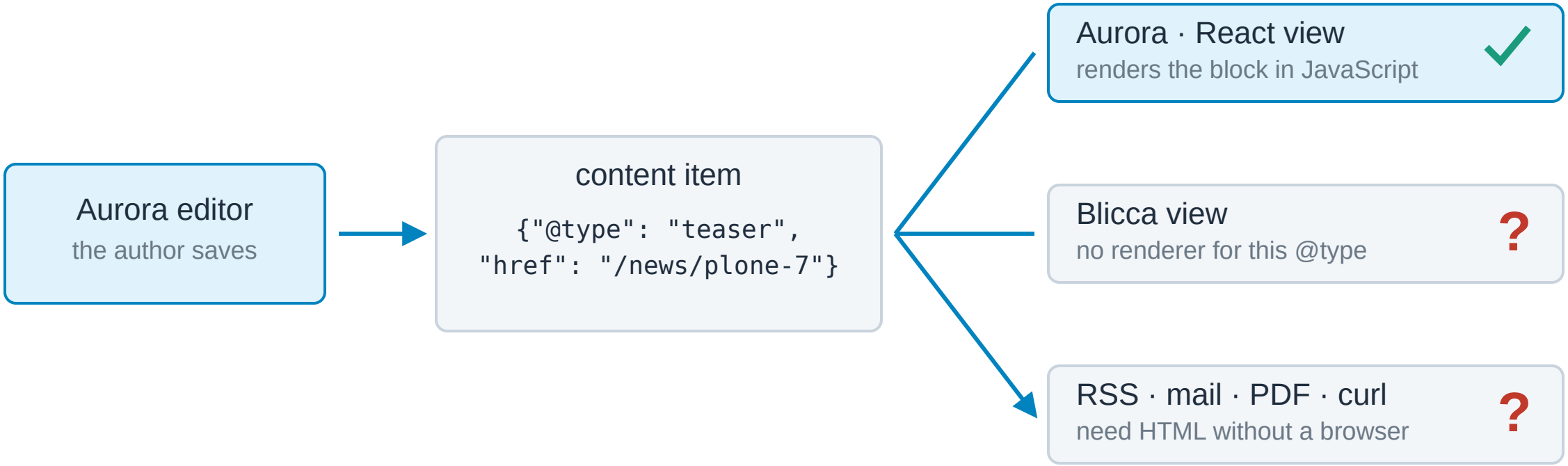
- Storage: `blocks` / `blocks_layout` JSON on the item, served by plone.restapi
- Rendering: **React**, in the browser and via Node SSR; the server **never renders it**

2.4 Same site, two worlds

	Blicca	Volto / Aurora
Page model	Mosaic tiles, Contentsections	JSON blocks
Rendered by	Server · Chameleon	React · browser + Node
Editor tech	patternslib / edit forms	React · Plate.js
Add-on =	Python package on PyPI	npm package
Who builds it	Python integrator	JS/TS developer

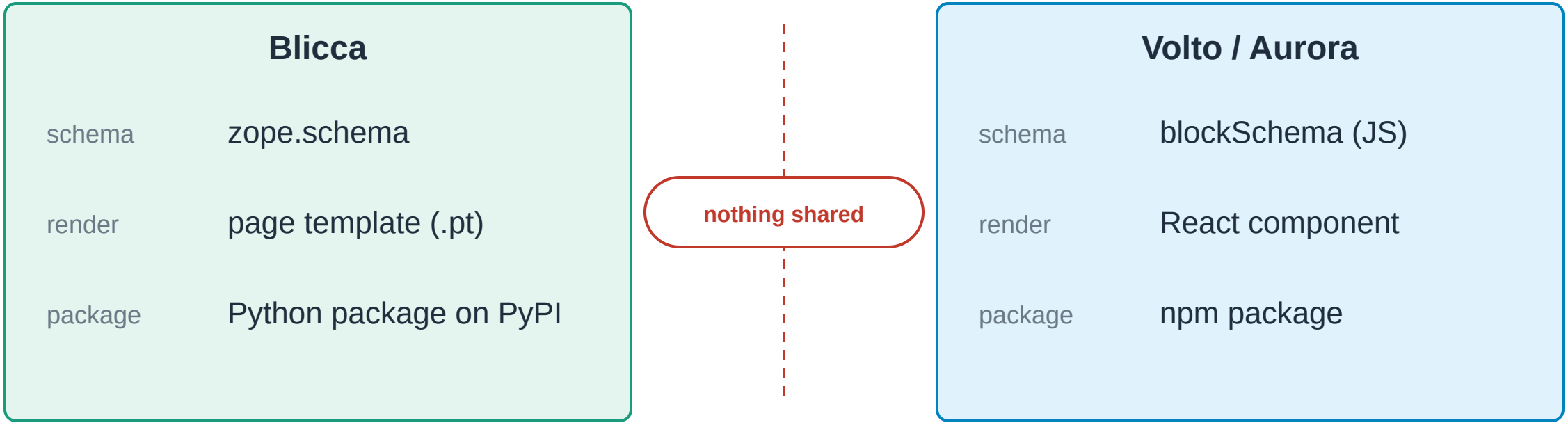


2.5 The gap: a block is a React component



- The block's `view` lives in a **JS bundle the server never loads**
- Open a Volto-edited page in Blicca: an empty body

2.6 Same feature, built twice



the same feature is built twice, and every team picks a side

- A Python shop cannot use a Volto block, a React shop cannot use a tile
- Two UIs, two block systems, **zero shared rendering**

➡ *What if one block could render in both?*

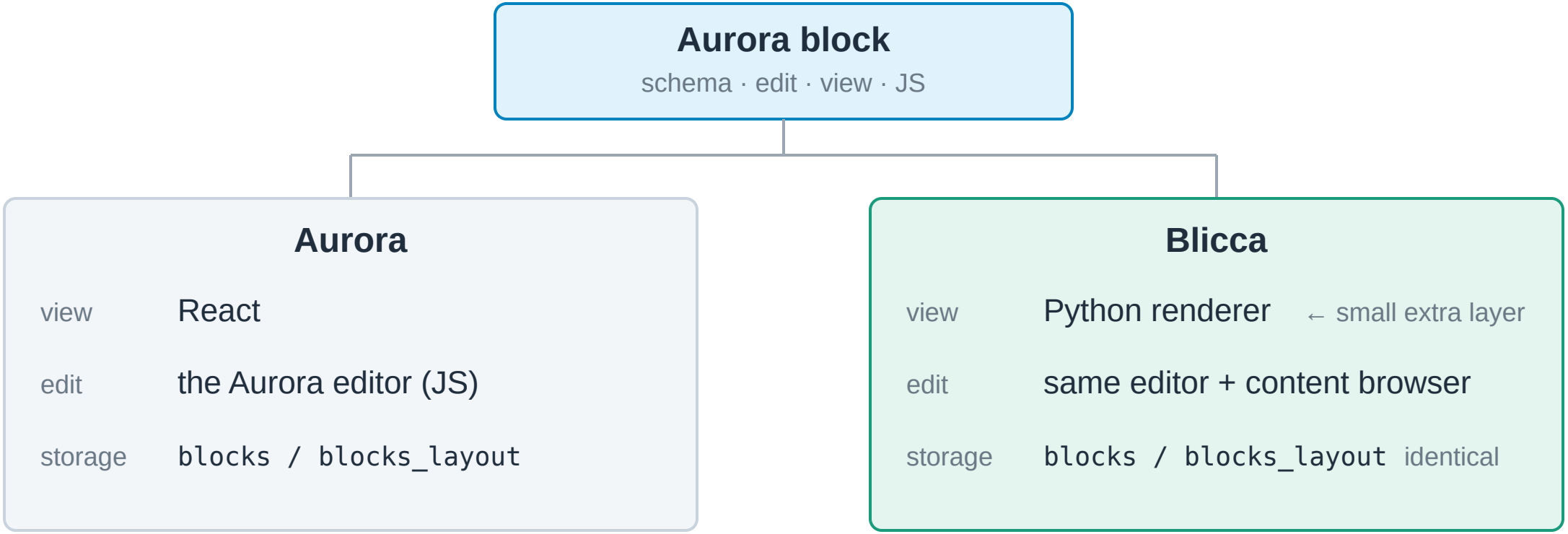
Chapter 3 — A better future

Aurora blocks, in both UIs

- **Aurora** — successor of Volto, the modern, modular full-stack JavaScript frontend
- **Blicca** — the well-known UI, Python-rendered templates and JavaScript UI components
- **Both** use Aurora blocks

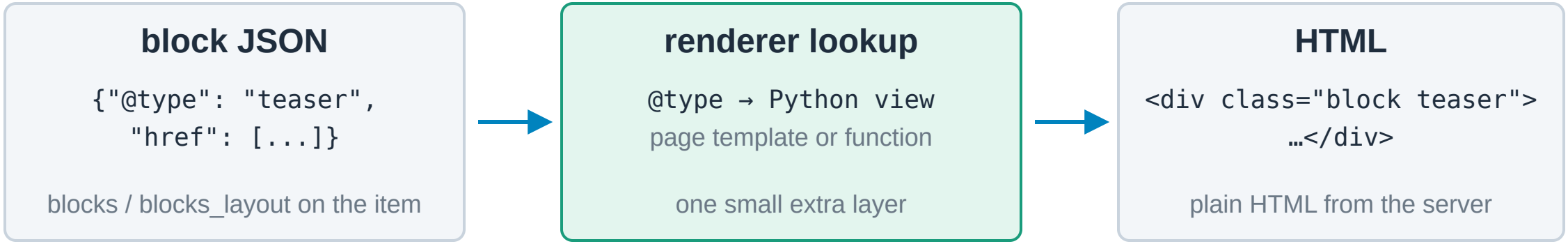


3.1 Two UIs, one block vocabulary



- Both UIs use **Aurora blocks** and the **same Aurora editor**
- Blicca adds a small **Python rendering layer**; same JSON, same block types

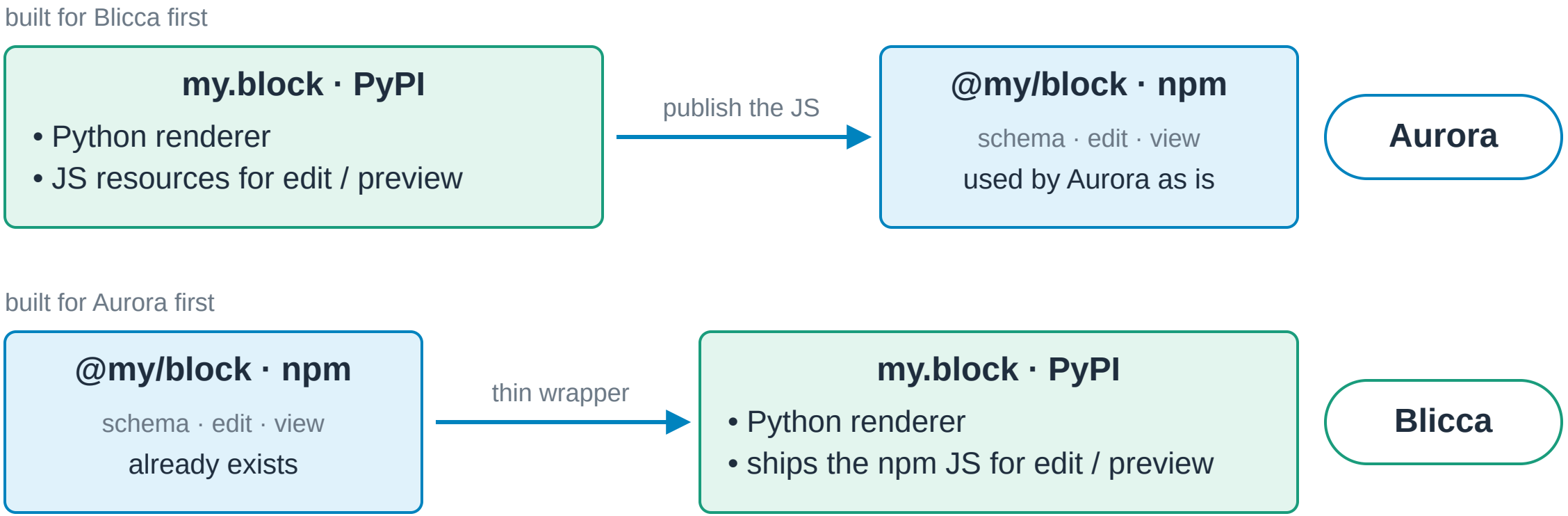
3.2 The Blicca rendering layer



search snippets · RSS · mail · PDF · curl: everything server side works again

- Look up a browser view named after the `@type`, render with Python, done
- Editing and preview: the shared Aurora editor loads the block's **JS resources** in Blicca

3.3 Sharing is caring



- Build a block once and **use it in both UIs**
- Blicca-first blocks publish their JS to npm; Aurora-first blocks get a thin Python wrapper

Demo

3.4 Under the hood

How Blicca renders Aurora blocks

- Stock Plone, Barceloneta theme, `volto.blocks` behavior
- For Python developers who know browser views

3.5 Two renderers, one page

Editor		Published page	
who	the author	who	the visitor
runs	React, in the browser	runs	Python, on the server
draws	the block's JS component	draws	a browser view for the block type

same data · same CSS classes · same stylesheet

- React lives **only** in the editor; the published page is plain server HTML
- What the author sees is what the visitor gets

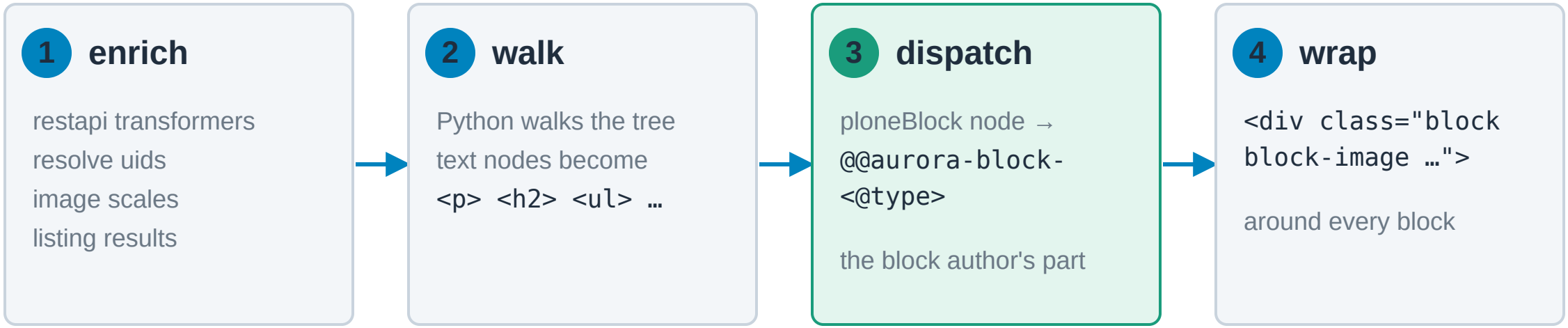
3.6 What gets stored

```
{
  "blocks": {
    "__somersault__": {
      "@type": "__somersault__",
      "value": [
        {"type": "h2",         "children": [{"text": "About us"}]},
        {"type": "p",          "children": [{"text": "Hello Plone"}]},
        {"type": "ploneBlock", "@type": "image", "url": "../resolveuid/8f3a…"}
      ]
    }
  }
}
```

- The standard `blocks` field holds **one** block: the whole page as a tree
- Saved exactly as the editor emits it, no transformation

3.7 From JSON to HTML

`@@view`



the same transformers run for the editor and for the page, so both see the same data

- Every block gets the same wrapper classes on the canvas and on the page
- One CSS file styles both

3.8 A block is a browser view

```
<browser:page
    name="aurora-block-promo"
    for="*"
    class=".promo_block_view.PromoBlockView"
    template="promo_block_view.pt"
    permission="zope2.View" />
```

- Register a view named `aurora-block-<@type>` and the block renders
- That is the whole contract

3.9 The view class

```
class PromoBlockView(BaseBlockView):
    @property
    def title(self):
        return self.data.get("title")
```

- `self.data` holds the block's data, already enriched
- Unknown type or broken renderer: an empty placeholder, **the page still renders**

3.10 The editor half

```
export default function install(config) {
  config.blocks.blocksConfig["promo"] = {
    id: "promo", title: "Promo",
    edit: PromoEdit, view: PromoView, blockSchema: PromoSchema,
  };
  return config;
}
```

- A regular Aurora block: `edit` and `view` components plus a schema
- Publishable to npm and usable in Aurora as is

3.11 Telling Blicca where the bundle lives

```
<records interface="…IAuroraBlockAddon" prefix="…blockaddons/my.promo">
  <value key="bundle">++plone++my.promo/promo-block.js</value>
  <value key="css">++plone++my.promo/promo-block.css</value>
</records>
```

- Loaded only in `@@aurora-edit`, never on the public page
- A broken bundle is skipped: the block leaves the menu, pages keep rendering

3.12 One add-on, both halves

```
my.promo/
├── browser/
│   ├── configure.zcml
│   ├── promo_block_view.py
│   └── promo_block_view.pt
├── profiles/default/registry.xml
└── static/
    ├── promo-block.js
    └── promo-block.css

pip install my.promo
```

Python half

renderer: one browser view

`aurora-block-promo`

registry record: where the bundle lives

JS half

built editor bundle + one stylesheet

loaded only in the edit view

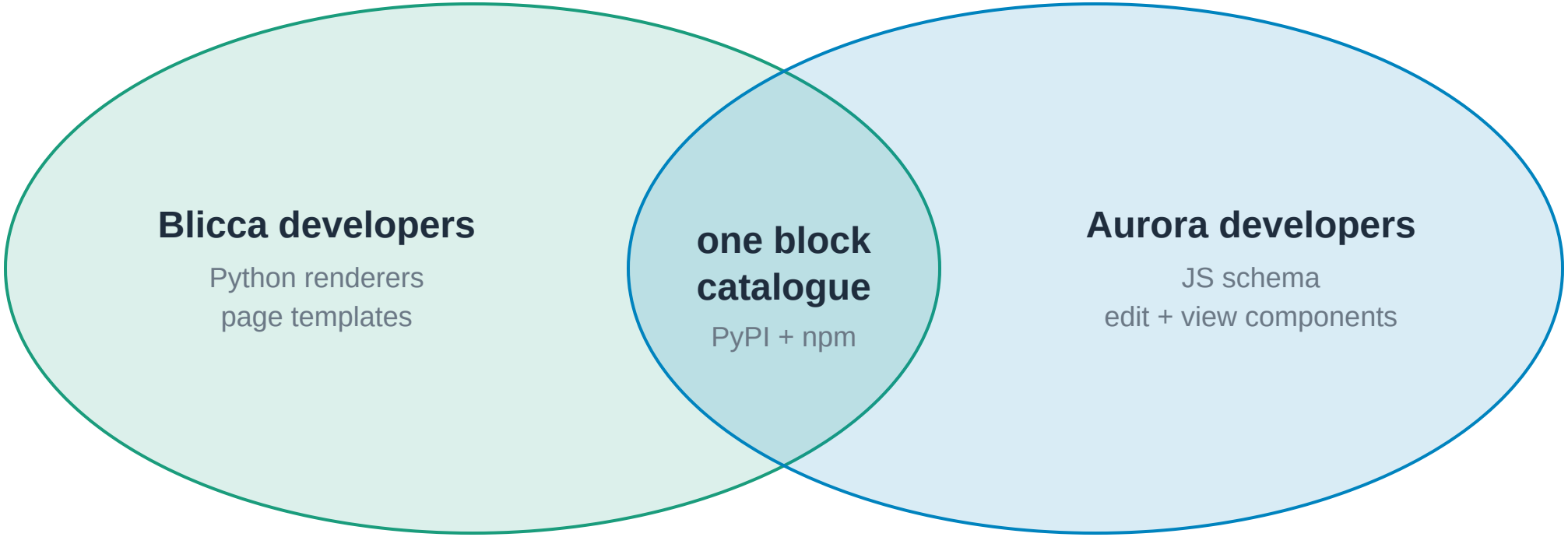
- Python renders, JS edits, one package ships both
- No Node on the server, no editor rebuild

3.13 Three rules

Rule	Meaning
Store as-is	one block, one tree, saved as the editor emits it
Render on the server	Python walks the tree, one browser view per block type
Ship both halves	editor bundle and renderer view in one Plone add-on

Same data, same classes, same CSS: canvas and page stay in lockstep.

3.14 Let's build blocks together



- Python developers bring renderers, JS developers bring editors
- Every block on PyPI **and** npm doubles its audience and its maintainers

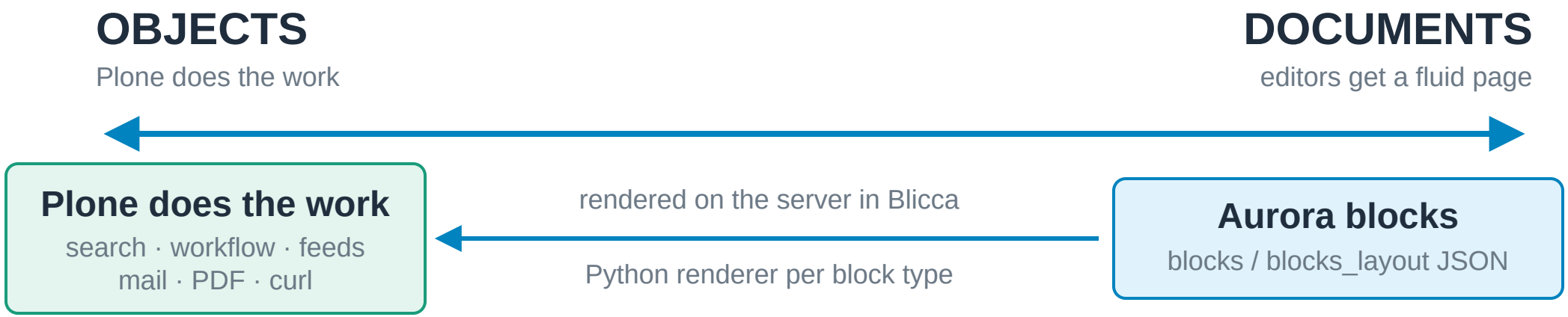
3.15 Building a website with Aurora blocks

Mockup element	Use
Grid, teaser, listing, text, image	**Existing block** + CSS in the theme
Hero with a fixed design	**Custom brand block** in the theme package
Logo wall, decorative SVG, HTML snippet	**Fragment**: static markup, no schema

- Start with **existing blocks + CSS**; only brand-specific pieces need their own block



3.16 One block, two UIs



the pendulum stops: both sides at once

- **Aurora blocks** and the **Aurora editor** are shared; Aurora renders in JavaScript, Blicca in Python
- Ship blocks on **PyPI and npm**: build once, use in both UIs

plone.blicca.auroraeditor

- currently private, needs funding.
- will be published in Github Plone org

Integration is intended to be used in any Blicca site, existing or new. It will not change anything in existing sites. You can activate it on any content types as you like.

Thank you

Maik Derstappen - md@derico.de - derico.de

Questions?

