dymaptic

Blazor State Management

Managing User Data Across Client and Server

Tim Purdum

Tech Bash

November, 2025

techbash

techbash

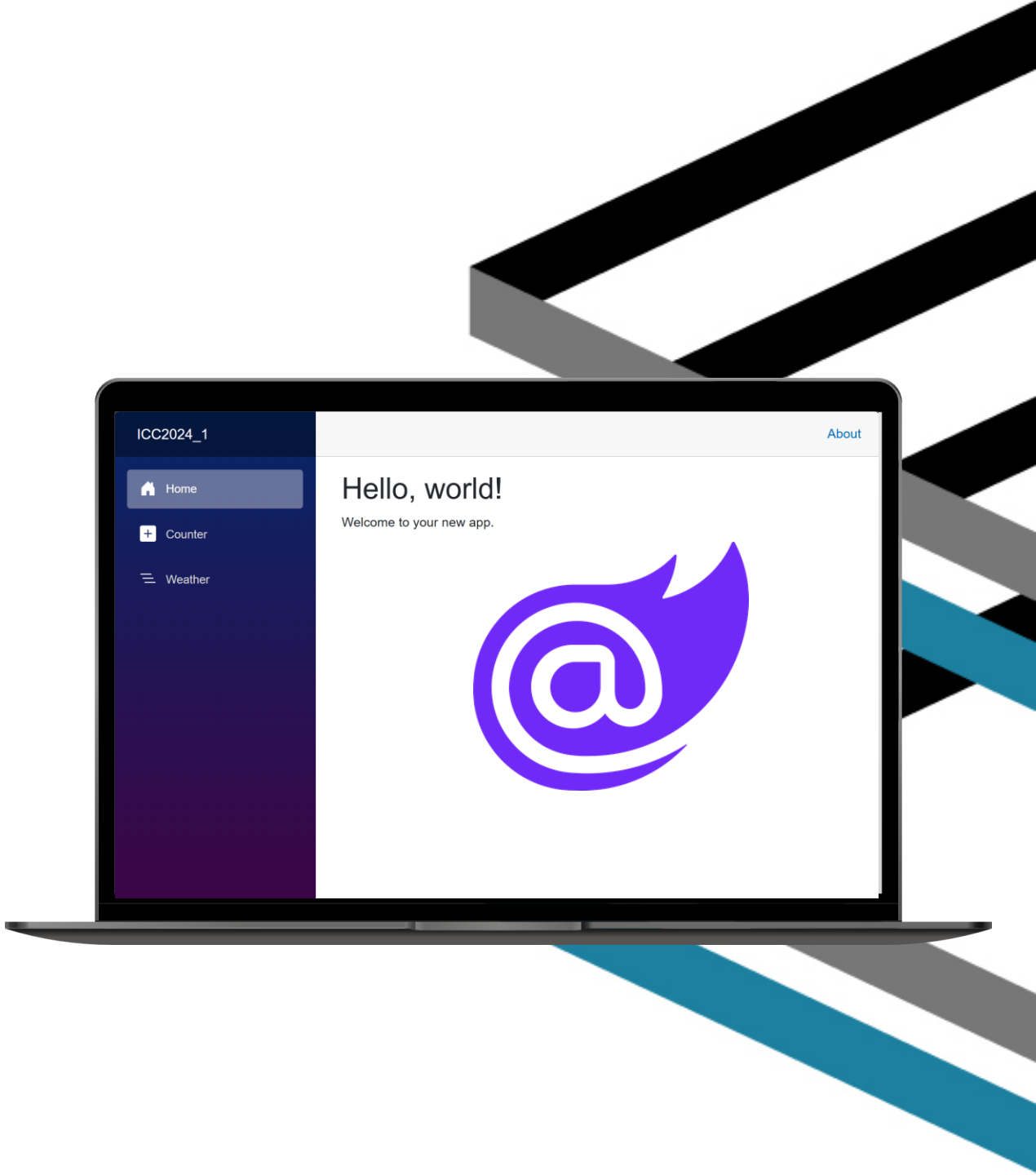
Goals of the Session

- **Identify types of state management in Blazor and the tools and patterns used**
- **Learn about how the Blazor rendering modes and render cycles impact state management**
- **Identify larger architectural patterns and practical examples for managing state in a Blazor application**

techbash

What is Blazor?

- Modern full-stack web framework
- Built on Asp.NET Core and Modern .NET
- Released with .NET Core 3.1 in 2018
- Component-based reactive framework
- Static and dynamic Server-Side rendering
- Client WebAssembly SPA applications or individual components
- High productivity with a single unifying language and framework



Blazing Shipments

techbash

As we look at this web app, consider the following questions:

- Where are the pages being rendered?
- How does it know what data to load?
- Are the pages comprised of a single component, or many?
- How does the site respond to user interaction?
- If we needed to store data, where would we store it?

techbash

What is State Management?

- **“State management refers to the management of the state of one or more user interface controls such as text fields, submit buttons, radio buttons, etc. in a graphical user interface.”**
  - ***from Wikipedia (based on redux.js.org)***

techbash

Types of State in Web Development

- **Component State**
- **Application State**
- **User/Session State**
- **Persistent State**

techbash

Component State

- Stored in component fields/properties or a model object
- Bound to HTML input and display elements
- Unsaved changes are lost on navigation/refresh

```
<p role="status">Current count: @currentCount</p>
<button class="btn btn-primary"
        @onclick="IncrementCount">Click me</button>
@code {
    private int currentCount = 0;
    private void IncrementCount() => currentCount++;
}
```

techbash

Component State

- Custom razor syntax for binding

```
<input type="text" @bind="fieldOrProp" />
```

- fires with the onchange event
- Change the event with `@bind:event="oninput"`
- Add a change handler method with `@bind:after="HandlerMethod"`
- For Razor Components, the syntax changes to bind to parameters: `<TestComponent @bind-ParameterName="fieldOrProp" />`

techbash

Application State

- State shared across components using
  - **Parameters**
  - **CascadingValues**
  - **EventCallbacks**
  - **Service Classes**

Application State: Parameters

- **C# public properties with [Parameter] attribute on a child component**

MapView.razor

```
[Parameter]
public double? Latitude { get; set; }

[Parameter]
public double? Longitude { get; set; }
```

- **In consuming (parent) class markup, parameters display like HTML attributes with capital letters**

```
<MapView Latitude="@shipment.Latitude" Longitude="@shipment.Longitude">
  <Map>
    <Basemap>
      <BasemapStyle Name="BasemapStyleName.ArcgisStreets"/>
    </Basemap>
  </Map>
</MapView>
```

Application State: Cascading Values

- Wrap child components with markup tags

```
<CascadingValue Value=" @User" Name="CurrentUser">
   <ProfileSelector />
</CascadingValue>
```

- All descendant components can receive the values as properties with the [CascadingParameter] attribute

```
[CascadingParameter(Name="CurrentUser")]
public ApplicationUser? CurrentUser { get; set; }
```

Application State: EventCallbacks

techbash

- A type of Parameter
- Async-supporting Event triggers

```
[Parameter]
public EventCallback<LayerViewCreateEvent> OnLayerViewCreate { get; set; }
```

- Bind to a parent component method instead of field or property

```
<MapView  OnLayerViewCreate="OnLayerViewCreate">
   <Map>
      <FeatureLayer OutFields="@(["*"])">
         <PortalItem PortalItemId="234d2e3f6f554e0e84757662469c26d3" />
      </FeatureLayer>
   </Map>
   </Extent>
</ MapView>
```

```
private async Task OnLayerViewCreate(LayerViewCreateEvent createEvent)
{
   if (createEvent.Layer is FeatureLayer)
   {
      // query the feature service
   }
}
```

Application State: Service Classes

techbash

- Any C# Class can be injected via Property Injection
  - In Razor Markup

```
@page "/order"
@inject StateManagementService StateManagementService
```

  - Or in C#

```
@code {
    [Inject]
    private StateManagementService? StateManager { get; set; }
}
```

- Allows offloading State Management logic from Pages and Components
- Share state between Components
- Use traditional .NET events/EventHandlers to notify different components about changes

techbash

User/Session State

- Authentication
- Authorization
- Profile
- Records
- Work Progress

techbash

User/Session State

- Browser Persistence
  - Query String https://blazingshipments.com?id=12345
  - Tokens
  - Cookies
  - localStorage
  - sessionStorage
  - indexedDb
- Server Persistence
  - Persistent Cache (e.g., Redis)
  - Database

Persistent State: Browser Storage

techbash

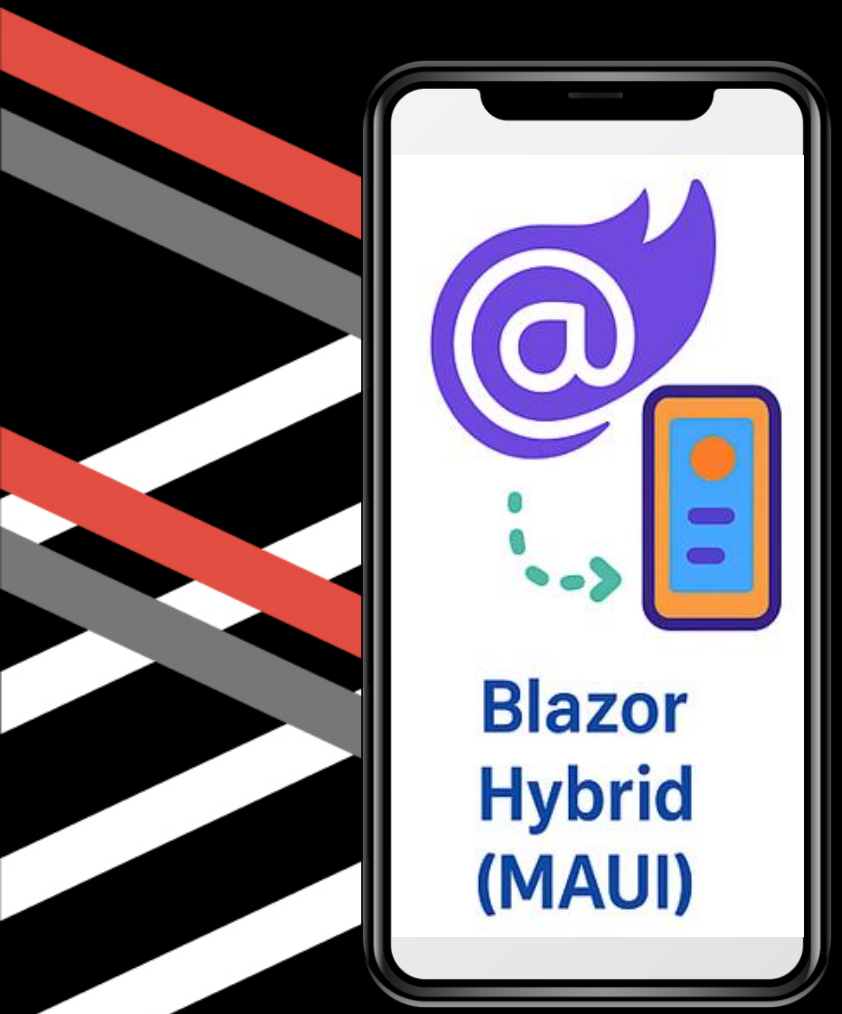
- **localStorage**
  - persists when tab/browser is closed, across multiple tabs
- **sessionStorage**
  - isolates data between tabs to prevent issues, data also is lost when tab is closed
- **IndexedDb**
  - Object-store structured database
  - Create an object store with a key path (aka ID) or a key generator
  - Also supports indexes
  - Transaction-scoped access: add, put (update), get, delete
- All require JavaScript or NuGet JS wrappers to interact.
- Available in “Interactive Render Modes”

Persistent State: Server Storage

- MemoryCache
- Redis cache
- HybridCache
- Database
- Only available from “Interactive Server” or via web API calls.

techbash

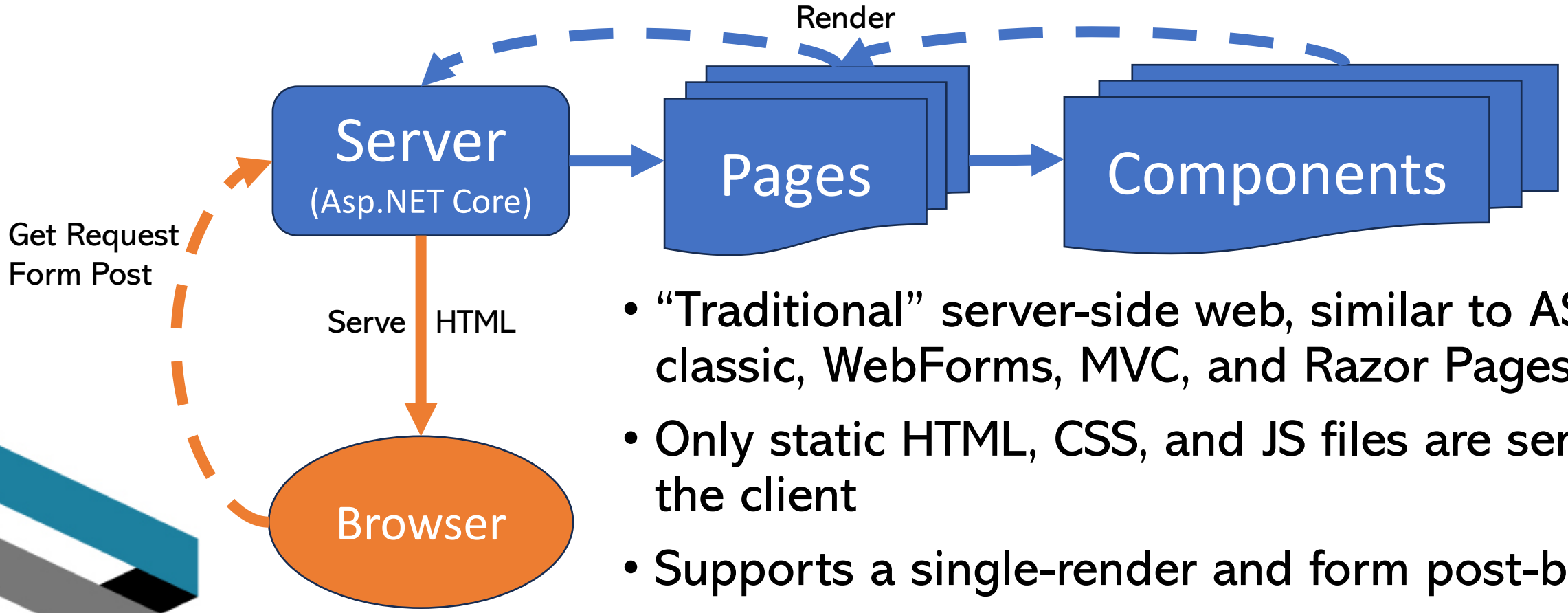
Blazor Component Render Modes



- Static Server Mode
- Interactive Server Mode
- Interactive WebAssembly Mode
- Interactive Auto Mode
- Blazor Hybrid *

** technically a "Blazor Hosting Model", not a render mode*

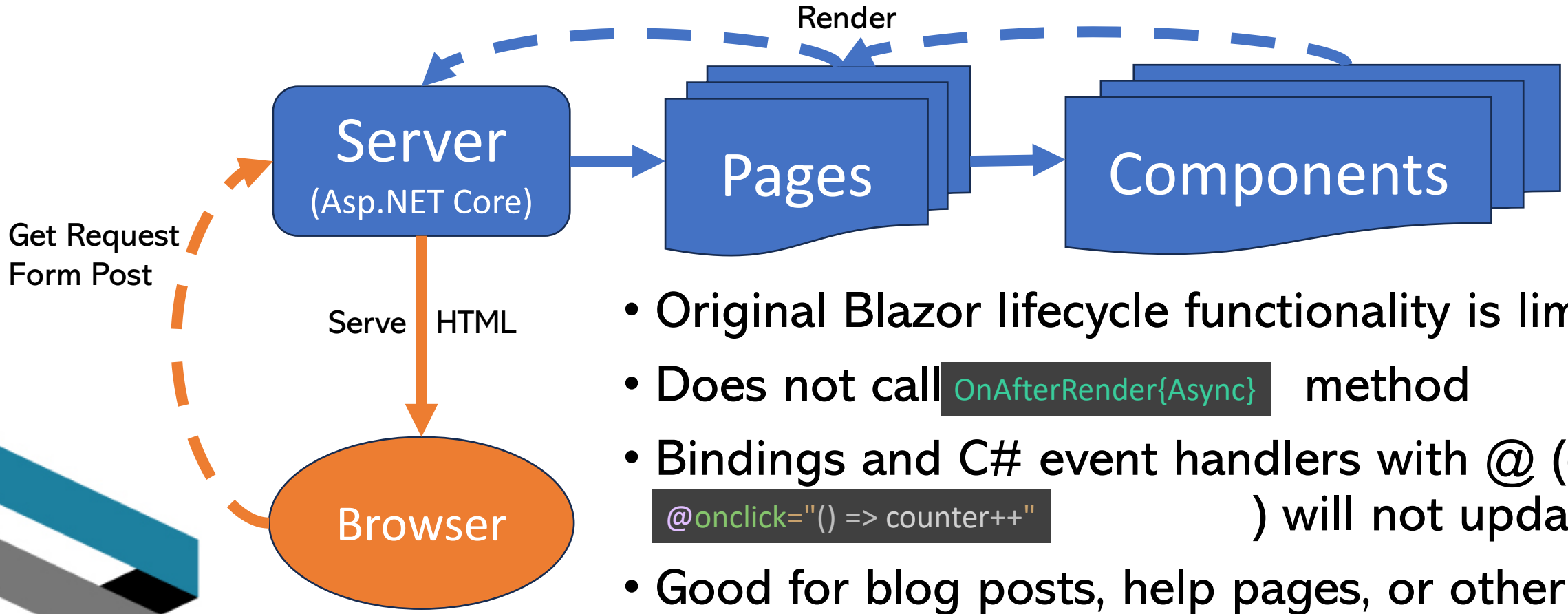
Blazor Render Modes: Static Server



- "Traditional" server-side web, similar to ASP classic, WebForms, MVC, and Razor Pages
- Only static HTML, CSS, and JS files are sent to the client
- Supports a single-render and form post-backs
- No interactive updates via C# (can still use JS)

techbash

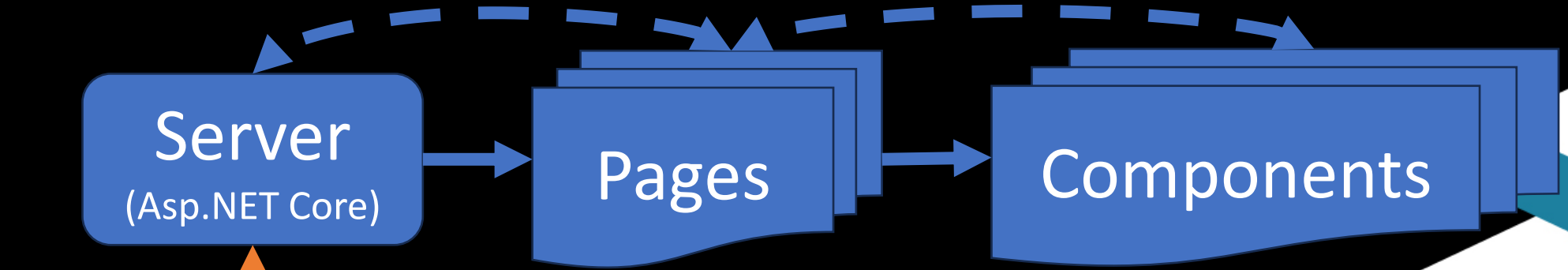
Blazor Render Modes: Static Server (cont.)



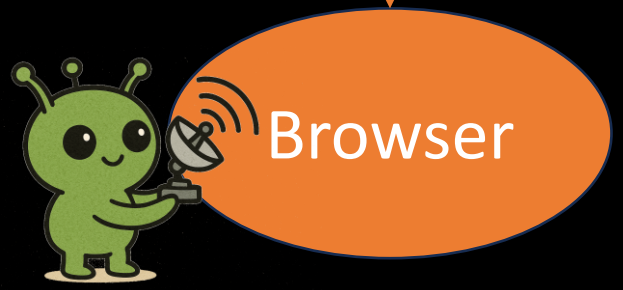
- Original Blazor lifecycle functionality is limited
- Does not call `OnAfterRender{Async}` method
- Bindings and C# event handlers with @ (e.g., `@onclick="() => counter++"`) will not update
- Good for blog posts, help pages, or other read-only content and simple forms

techbash

Blazor Render Modes: Interactive Server



SignalR 2-way Communication

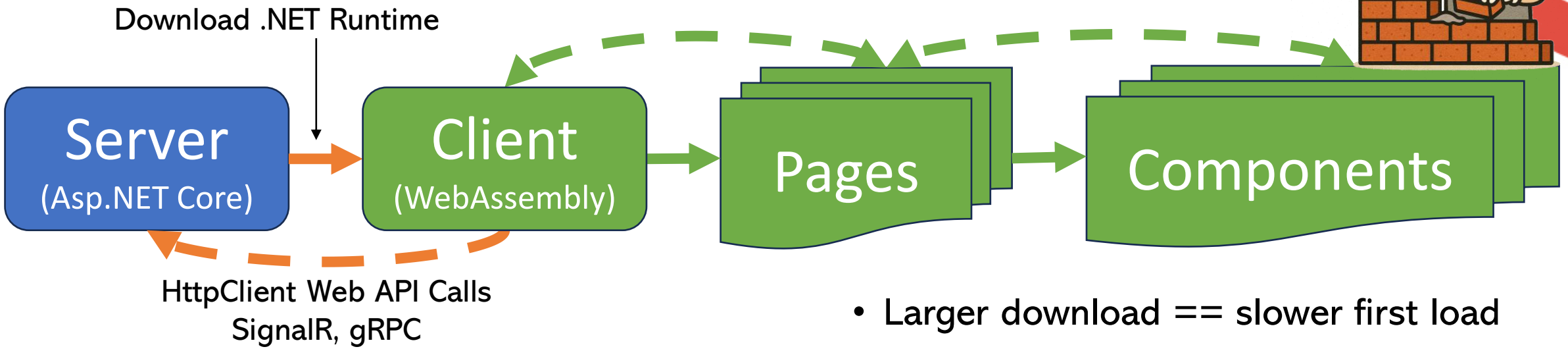


- Continuous websocket connection between client and server with SignalR
- Live data-binding, real-time updates, JavaScript interop
- Direct access to server data store
- Fast on first load
- Can introduce network lag

techbash

Blazor Render Modes: Interactive WebAssembly

techbash



- Runs in the client browser
- Live data-binding, real-time updates, JavaScript interop
- HttpClient calls to communicate with server web API
- Single-threaded

- Larger download == slower first load
- Faster interactions after first load (no network latency on events)
- Closest in approach to most JS SPA frameworks
- Available in the hosted Blazor Web App and standalone WebAssembly projects

Blazor Render Modes: Interactive Auto

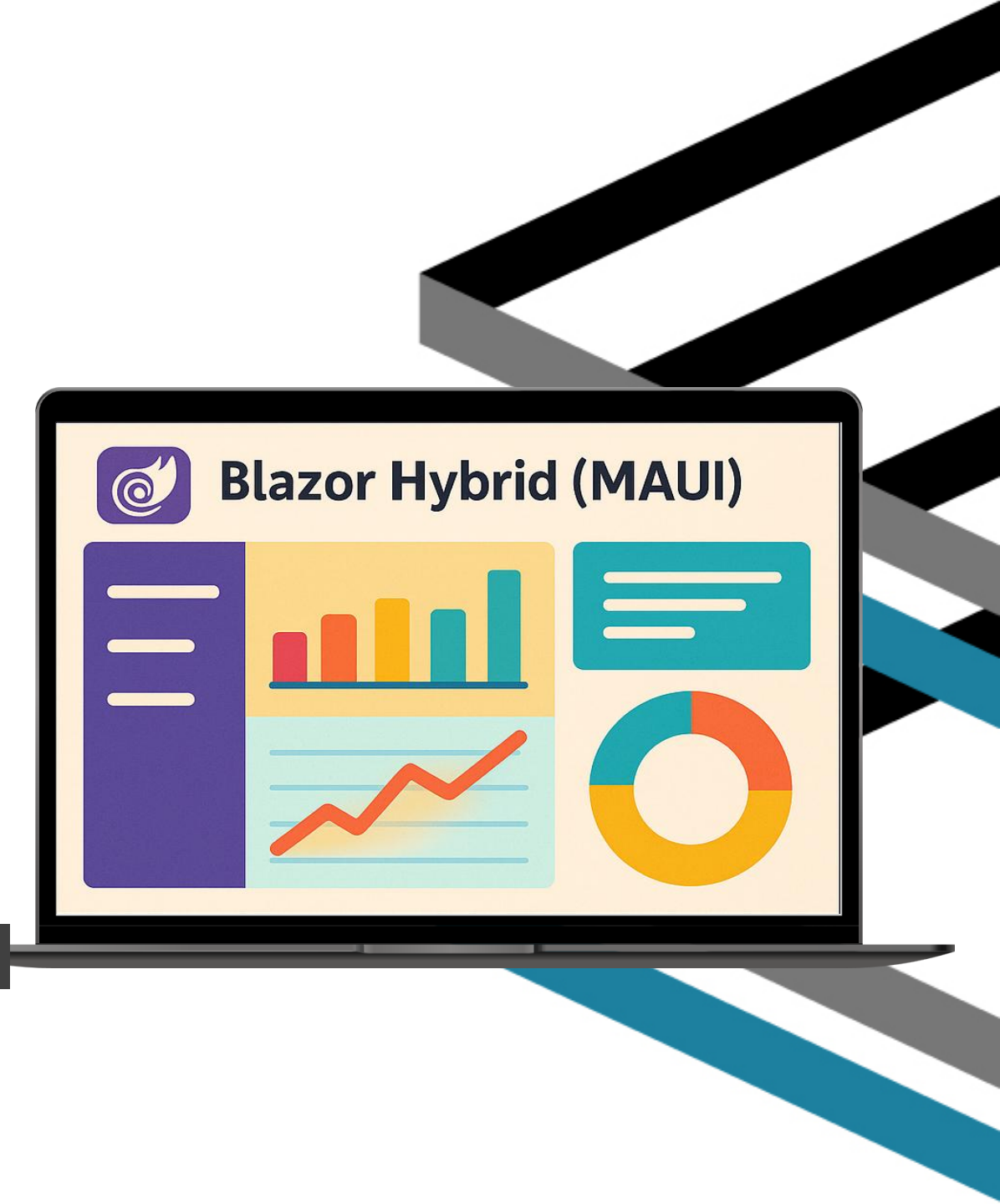
- On first load, runs from server, creating SignalR connection
- In the background, downloads .NET runtime and client code
- On next load, switches to running from WebAssembly
- “Best of both worlds”
  - Fast start on first load (server)
  - More responsive and robust interactions (client)
- Requires flexible data handling/abstraction to handle both client and server modes

techbash

techbash

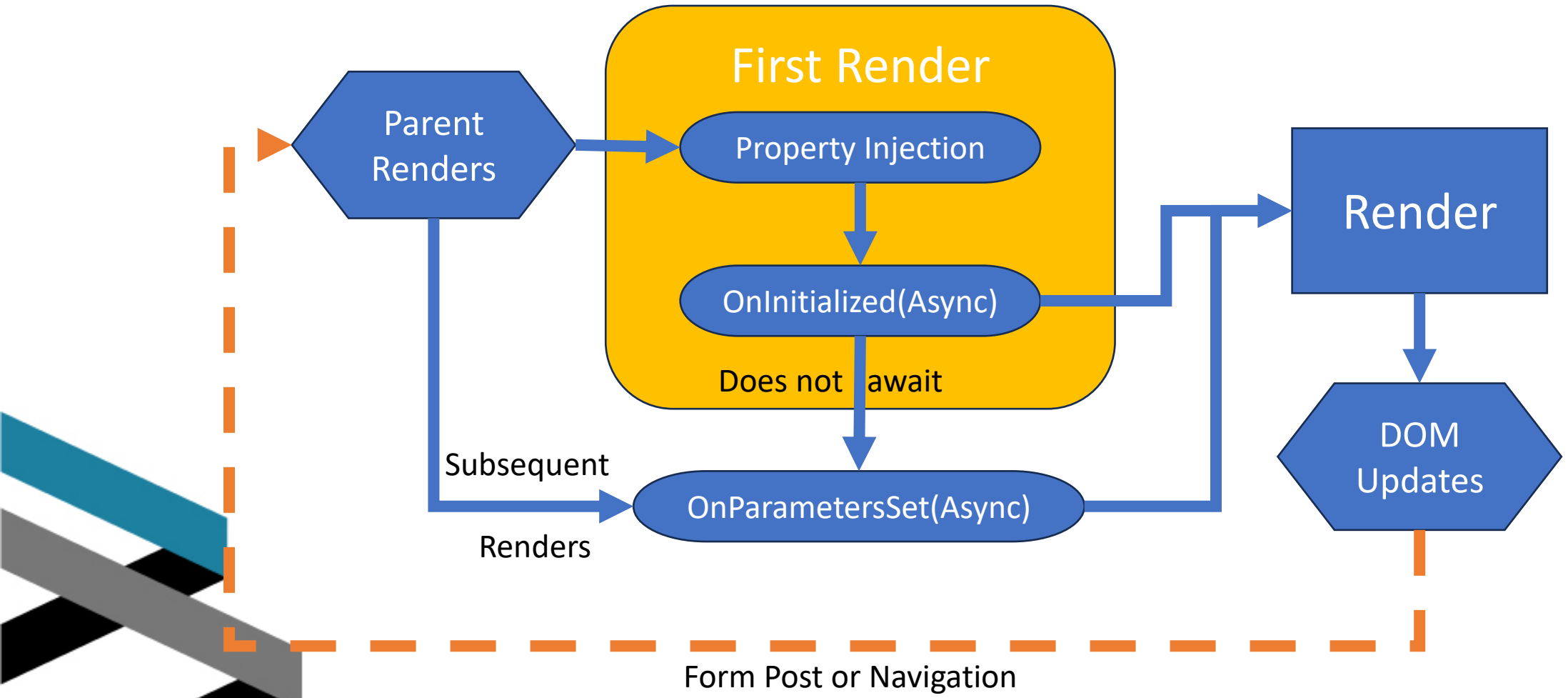
Blazor Hybrid

- Runs in a WebView in .NET MAUI (iOS, Android, Mac, Windows), WPF, or Windows Forms
- Native .NET multi-threaded code execution (not WebAssembly)
- Access to device APIs (GPS, Bluetooth, photos, etc.)
- Can reuse components or entire UI applications between web, desktop, and mobile
- Always interactive, fires `OnAfterRender{Async}`
- Does not require defining `@rendermode`



techbash

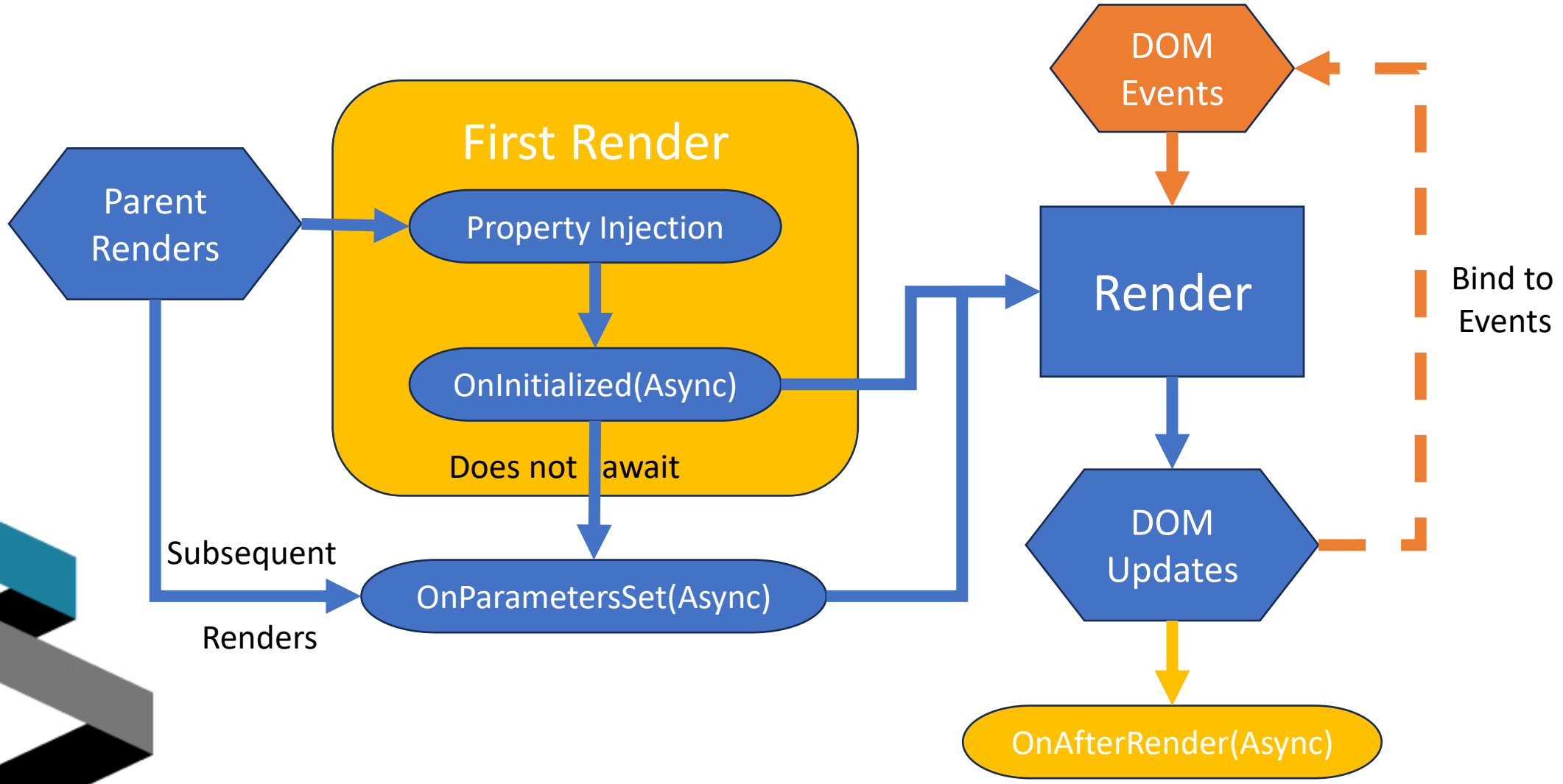
Razor Component Lifecycle: Static Server Mode



State set in OnInitialized and OnParametersSet should be Idempotent

techbash

Razor Component Lifecycle: Interactive Modes



Don't set state that will cause a render cycle in OnAfterRender!

Architectural Patterns for State Management

- Some frameworks encourage you to manage state in a specific pattern
  - React – Flux/Redux/MVU
  - XAML Frameworks – MVVM
  - Asp.NET Core MVC – …MVC
- Blazor does not have a "default" named architectural pattern, but the decisions we make still impact how we manage the user and application state

techbash

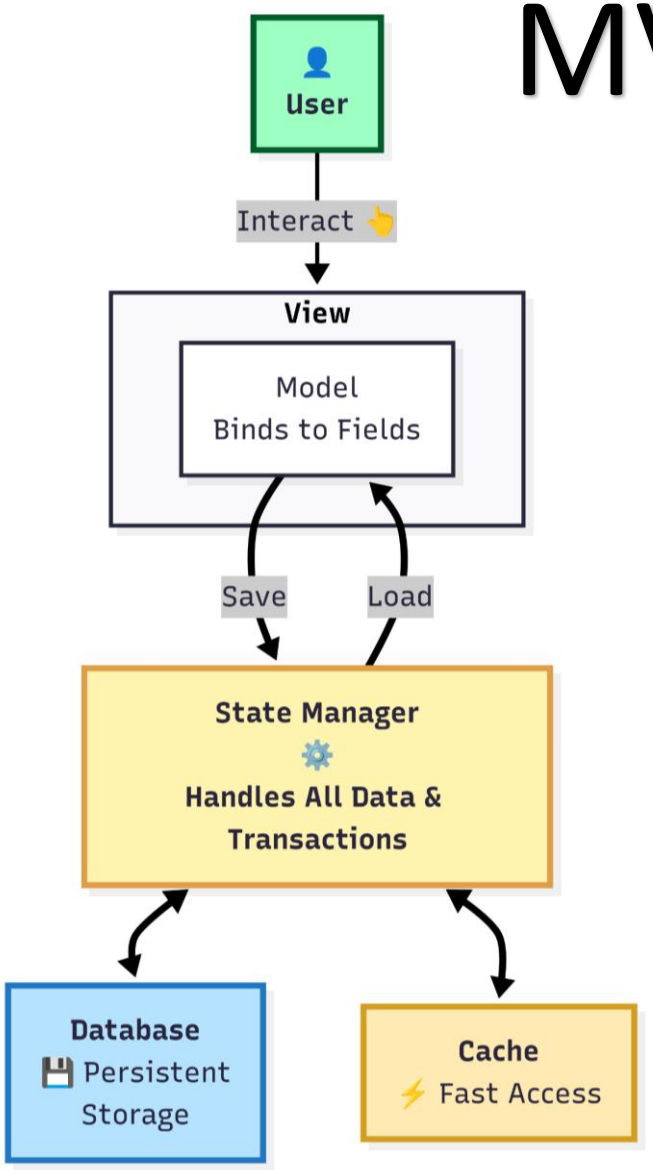
Architectural Patterns for State Management

- **Goals for Blazor State Management**
  - **Flexible components that will work in both Interactive Server and Interactive WebAssembly modes**
  - **Reduced boilerplate logic like pass-through methods**
    - ***(e.g., clientComponent => clientService => webApi => webService => dataRepository)***
  - **Consistent patterns for communication between components**
  - **Abstract away communication from WebAssembly client to Server**
  - **Keep pages and components lightweight and easy to read**
  - **Allow generic implementations for simple use cases**

techbash

Architectural Patterns for State Management

MVSM™

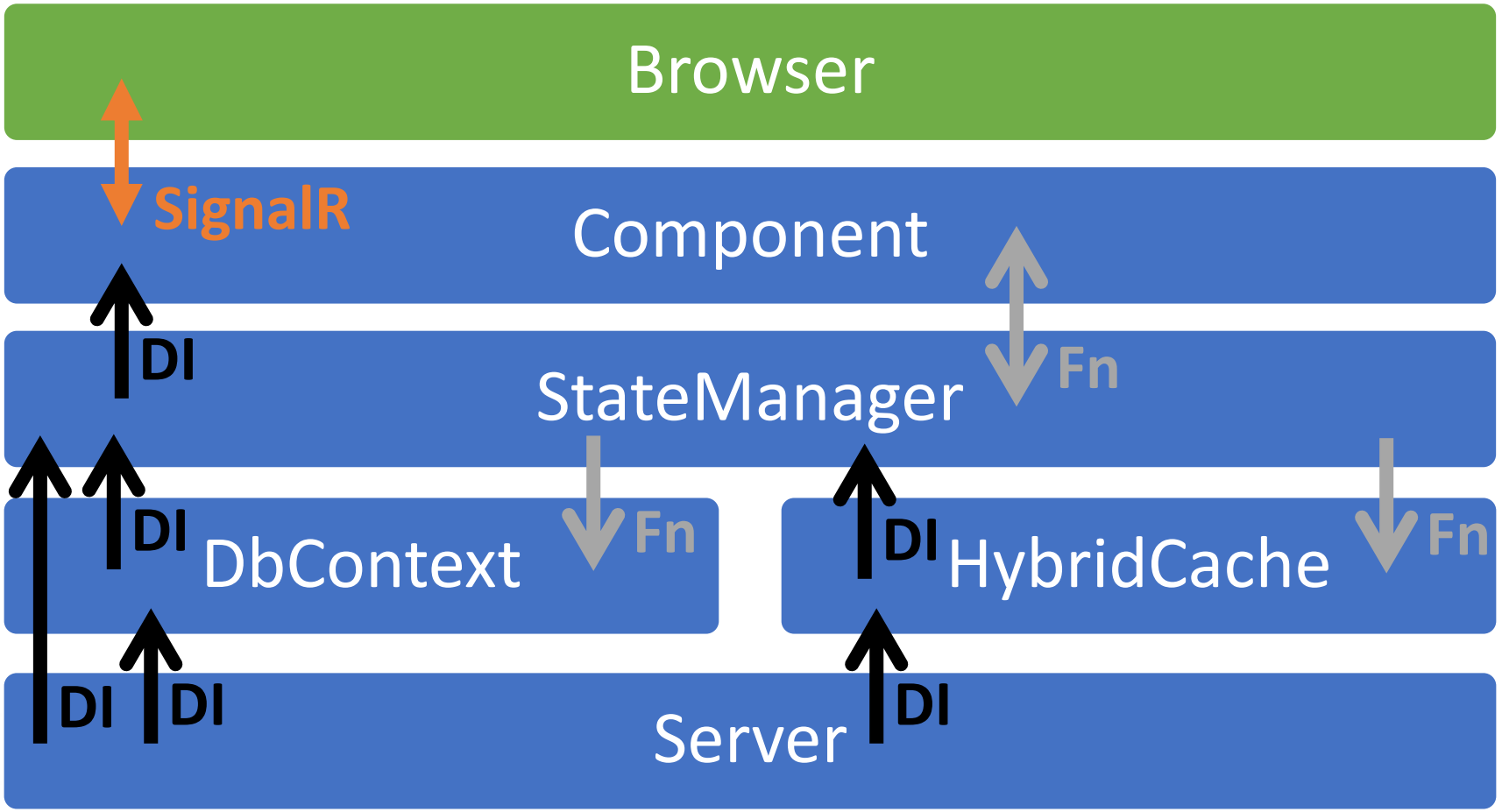


- **Model**
- **View**
- **State Manager**
  - Model and View designed to work together with two-way binding
  - Model can live in either the View or the State Manager class
  - State Manager is responsible for abstracting transport and any data transformation

techbash

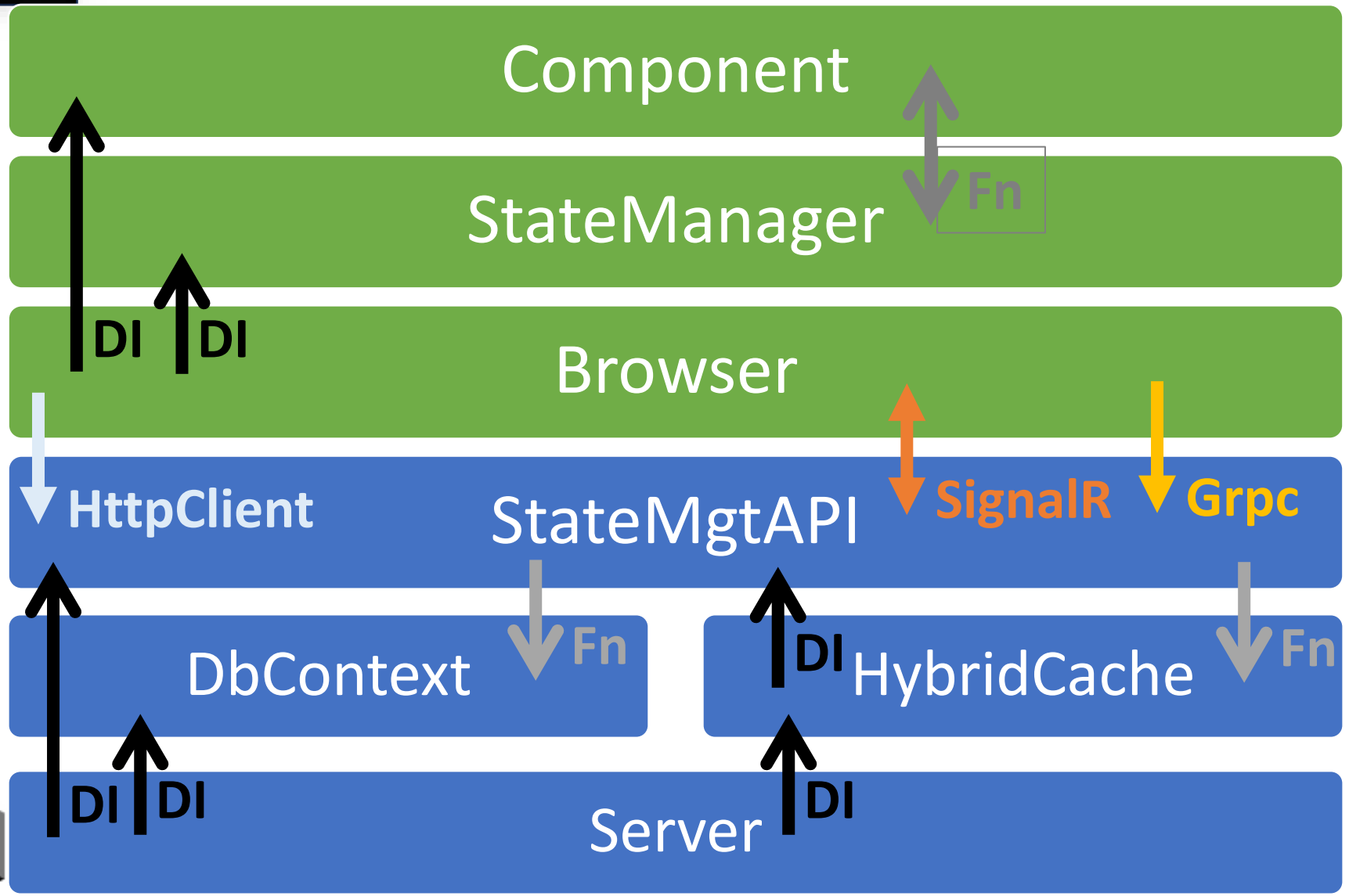
techbash

Server Component



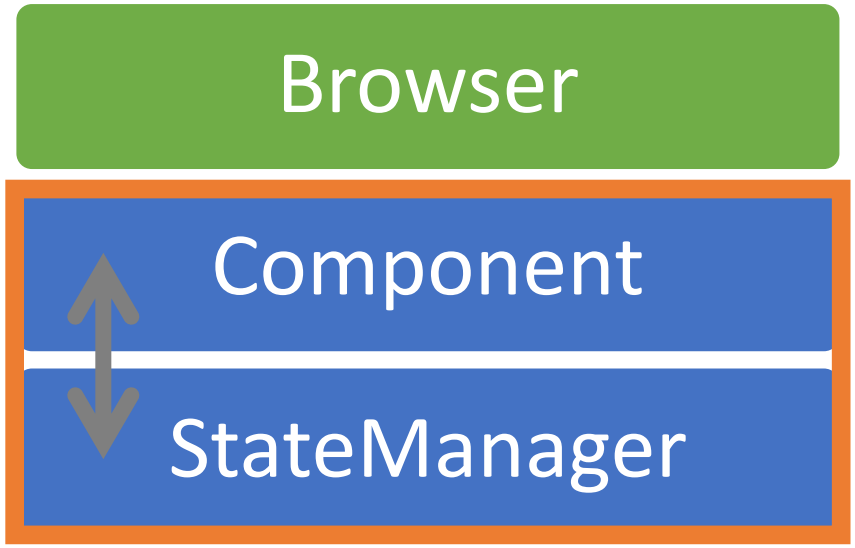
techbash

Client Component

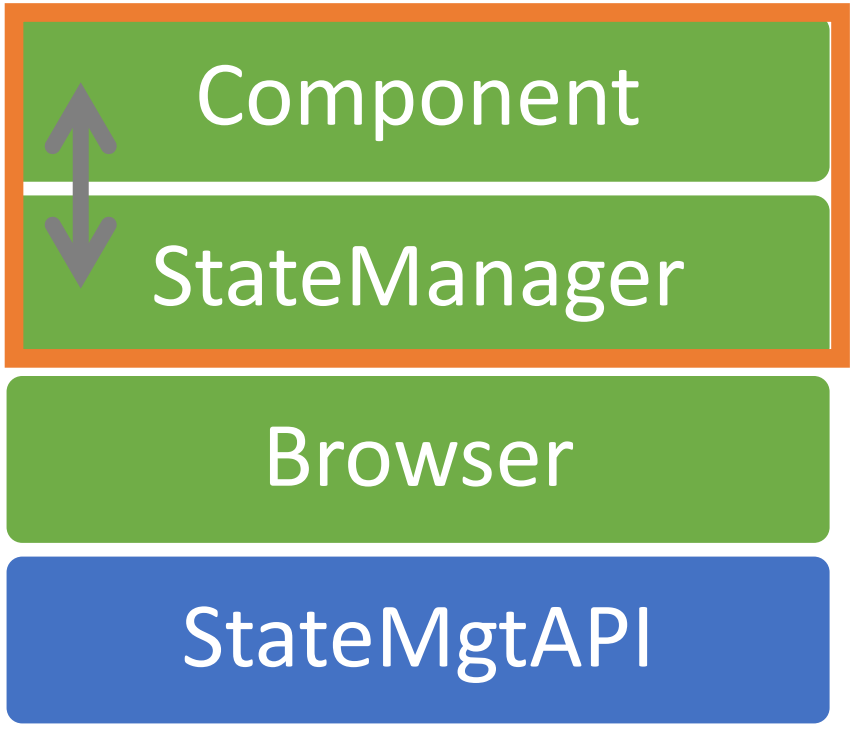


techbash

Server Component



Client Component



In both cases, the Component only ever has one consistent IStateManager interface to interact with

Review

techbash

🧱 Core Concepts of State Management

- **Definition**: State management involves tracking the dynamic data of a user interface—across components, sessions, and storage layers.
- **Types of State**:
  - *Component State*: Temporary, lost on refresh or navigation.
  - *Application State*: Shared across components using cascading values, DI services, etc.
  - *User/Session State*: Stored in browser memory (e.g., localStorage, sessionStorage, indexedDb), usually not synced with the server.
  - *Persistent State*: Long-term data stored in a database or API.

Review

techbash

Blazor Render Modes & Their State Implications

- **Static Server Mode:**
  - Simple form submission and HTML rendering.
  - Limited interactivity and no real-time state updates.
  - Persistence tools: cookies, tokens, query strings.
- **Interactive Server Mode:**
  - Real-time two-way binding using SignalR.
  - Enables in-memory server-side tracking and real-time updates.
  - Challenges: reconnection handling, distributed server sync.
- **Interactive WebAssembly Mode:**
  - Fully client-side execution.
  - Rich interactivity with flexible state control
  - .Risks of state desynchronization and ID conflicts for new data.
- **Interactive Auto Mode:**
  - Hybrid approach: server-rendered first load, client-side on reload.
  - Balances fast startup with responsive interactivity

Review

techbash

🧩 Patterns for Binding & Application State Sharing

- *Binding*:
  - @bind, @bind:event, @bind:get/set, and @bind:after allow seamless two-way data binding in Razor.
- *Component Communication*:
  - Parameters, CascadingValues, EventCallbacks, and DI Services are used to maintain shared state and coordination.

🗃️ Browser Storage Techniques

- *localStorage and sessionStorage*:
  - Simple key-value stores for persistence.
- *IndexedDb*:
  - Structured object store with indexing and transaction support. Can be wrapped with JS + C# logic or NuGet packages.

Review

techbash

🧠 Architectural Patterns for Blazor

- **MVU**: Immutable, Redux-style, but not ideal for Blazor's reactive capabilities.
- **MVVM**: Familiar in .NET but verbose; Blazor doesn't require INotifyPropertyChanged.
- **MVC**: Suited for non-interactive, server-rendered apps—less effective in Blazor.

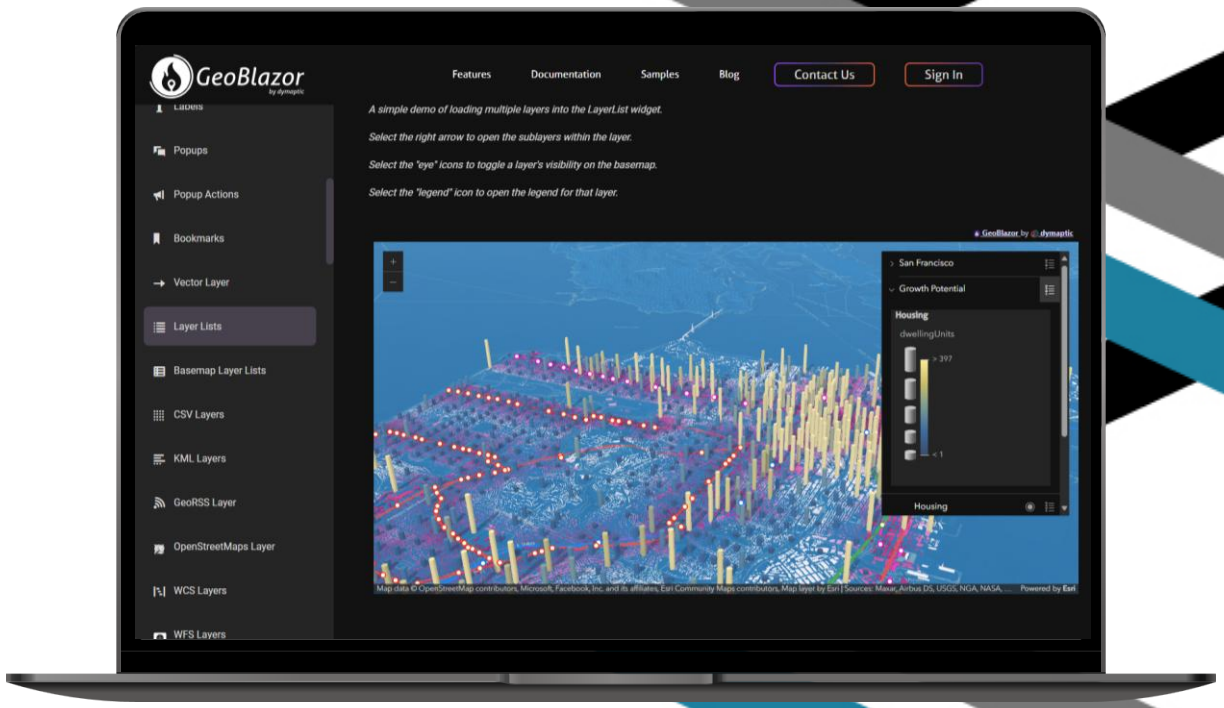
🧩 MVSM™ – A Blazor-Centric Pattern

- **Model-View-State Manager**:
  - Two-way binding between View and Model
  - .State Manager handles all data transport, persistence, and API abstraction.
  - Designed for extensibility using generics, reflection, and browser storage.

techbash

Check out https://samples.geoblazor.com

- Fully interactive application samples written in C# and Razor
- Each page is written to run in both Client and Server mode (live sample is Client mode)
- GeoBlazor library utilizes JSRuntime to interact with the ArcGIS Maps SDK for JavaScript, so GeoBlazor *users* don't have to switch to JavaScript



GeoBlazor

Thank You!

dymaptic

Notes & Links @
https://timpurdum.dev

techbash

GeoBlazor