Esri International Developer Summit

Palm Springs, CA

Integrating w. Enterprise Business Systems

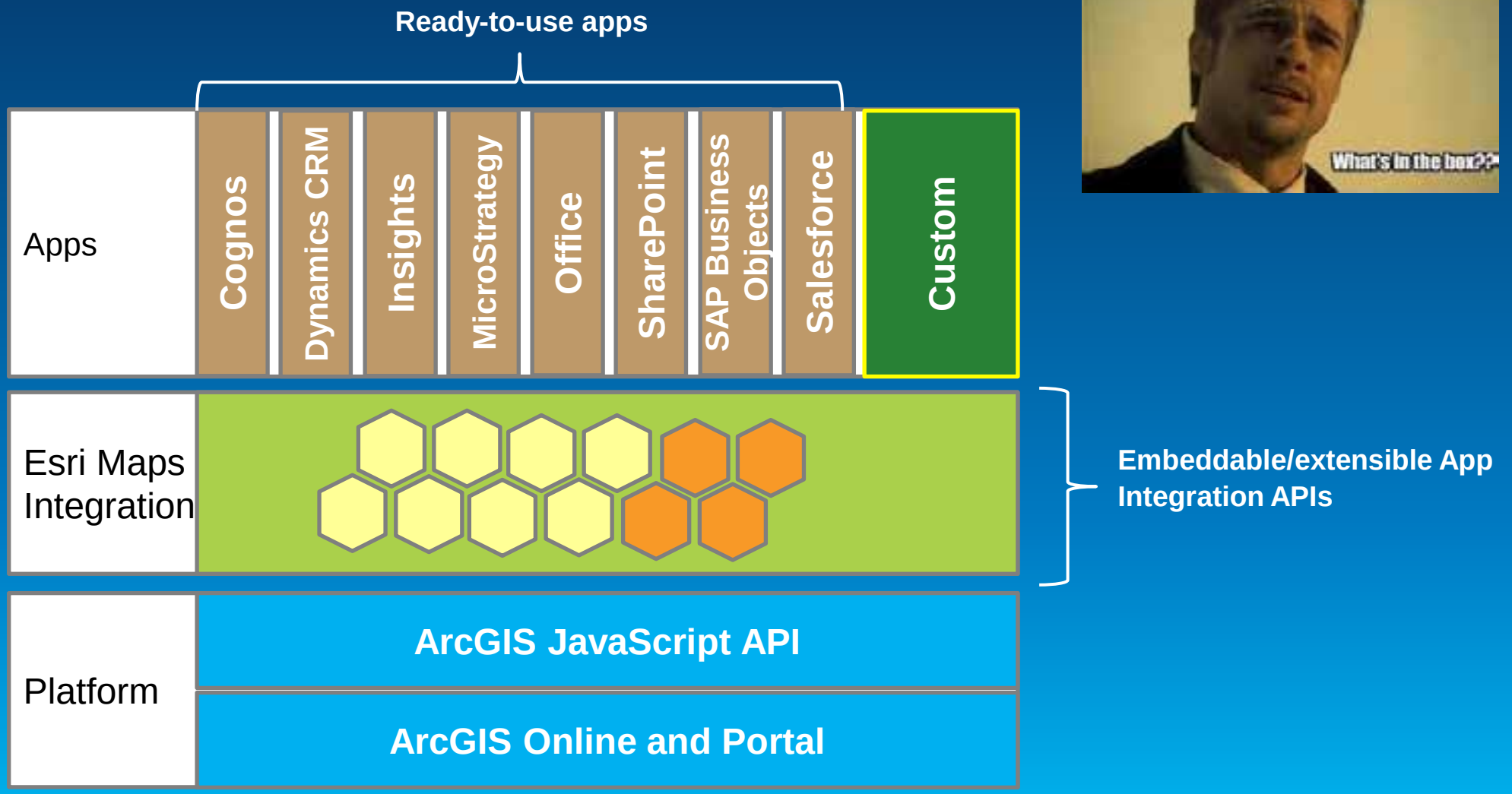
Patrick Brennan

Mark Mallany

What are we talking about?

- A new partner/distributor option to implement an ‘Esri Maps’ style app
- Expose how we have abstracted access to a diverse set of 3rd party business systems
  - We use it ourselves (Cognos, MicroStrategy, SharePoint, etc)
- Allow you to develop custom functionality (beyond what we make available ‘out of the box’)
- Built on the ArcGIS platform
- Part of Esri’s JavaScript story

Esri Maps Framework



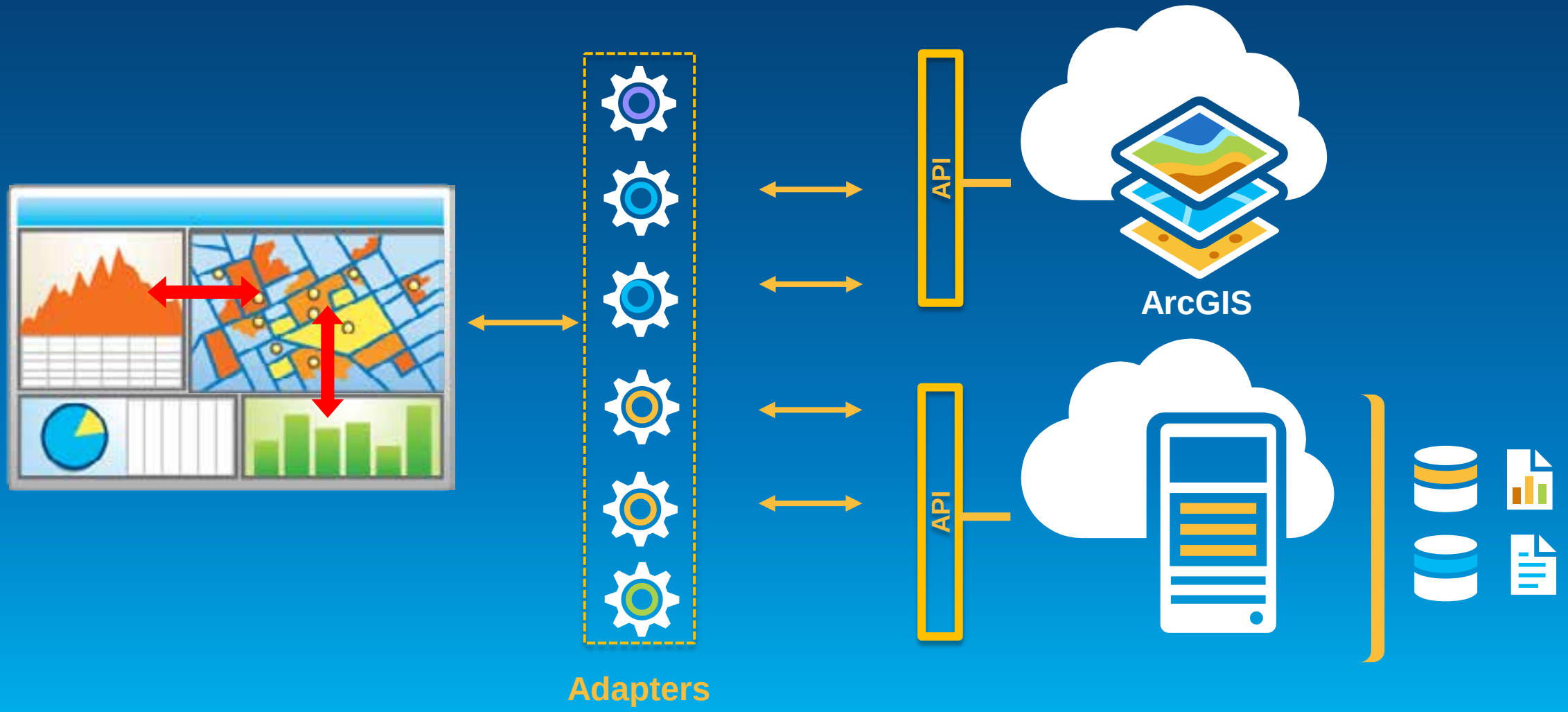
Embeddable App

Core capabilities

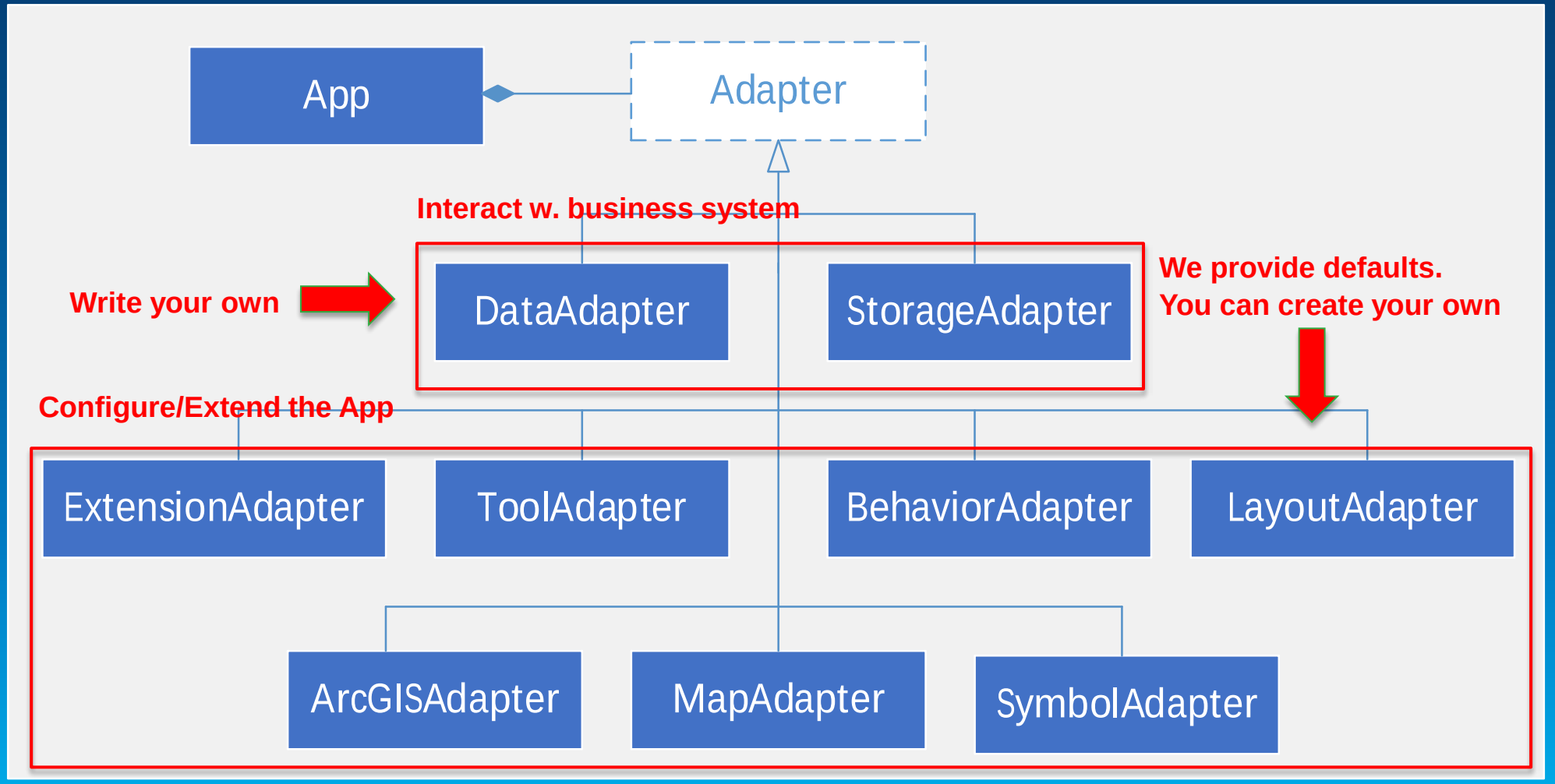
Integration

Esri Maps Integration

Concept



Adapters


App
Adapter
Interact w. business system
Write your own
DataAdapter
StorageAdapter
We provide defaults.
You can create your own
Configure/Extend the App
ExtensionAdapter
ToolAdapter
BehaviorAdapter
LayoutAdapter
ArcGISAdapter
MapAdapter
SymbolAdapter

Interact with the business system

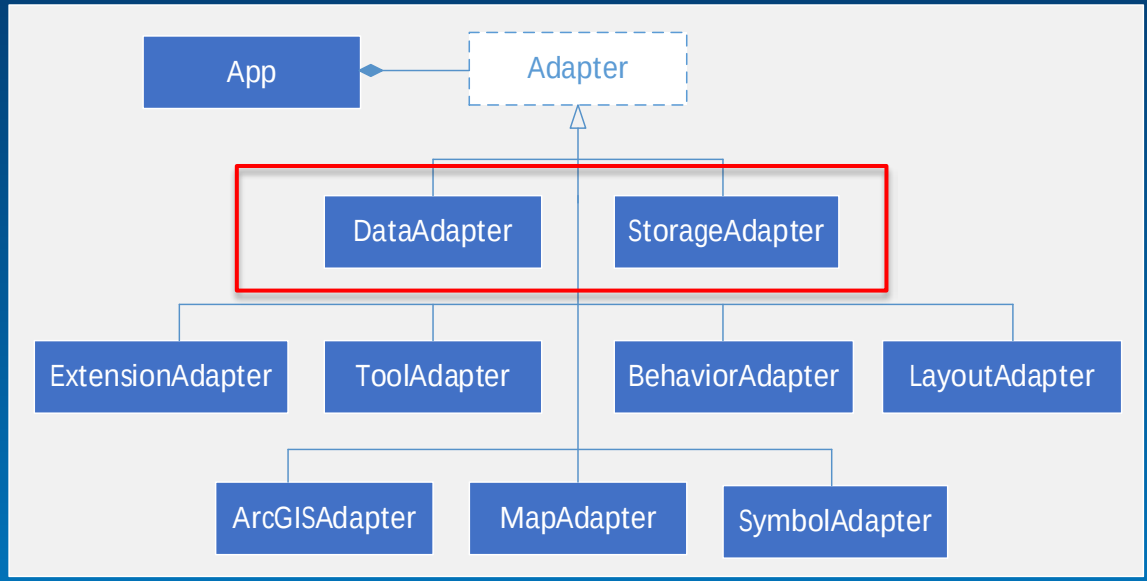
DataAdapter, StorageAdapter

Data Adapter:

- Convey information about what data is available
- Describe the business system's capabilities
- Query the system for information

Storage Adapter:

- Stores and retrieves various aspects of the application
- Examples:
  - App config
  - Map config
  - Location types
  - ArcGIS token



Data Adapter – Considerations

- Can be fairly complex
- Know your system (every system has different strengths weaknesses)
- Know its limitations (un-bounded queries? write-back?)
- Use the system's APIs access the data in a meaningful ways
  - Be 'context aware' (e.g. if you're in a dashboard, get the dashboard's current data)
- System native data types – need to me mapped to ArcGIS API for JavaScript field types (e.g. esriFieldTypeString)

Storage Adapter

Storage should be ‘context aware’

- If the App is embedded in a report, then the storage should occur on a report level
- If the App is embedded in a dashboard, then the storage should occur on a dashboard level
- Etc

Configure/Extend the App

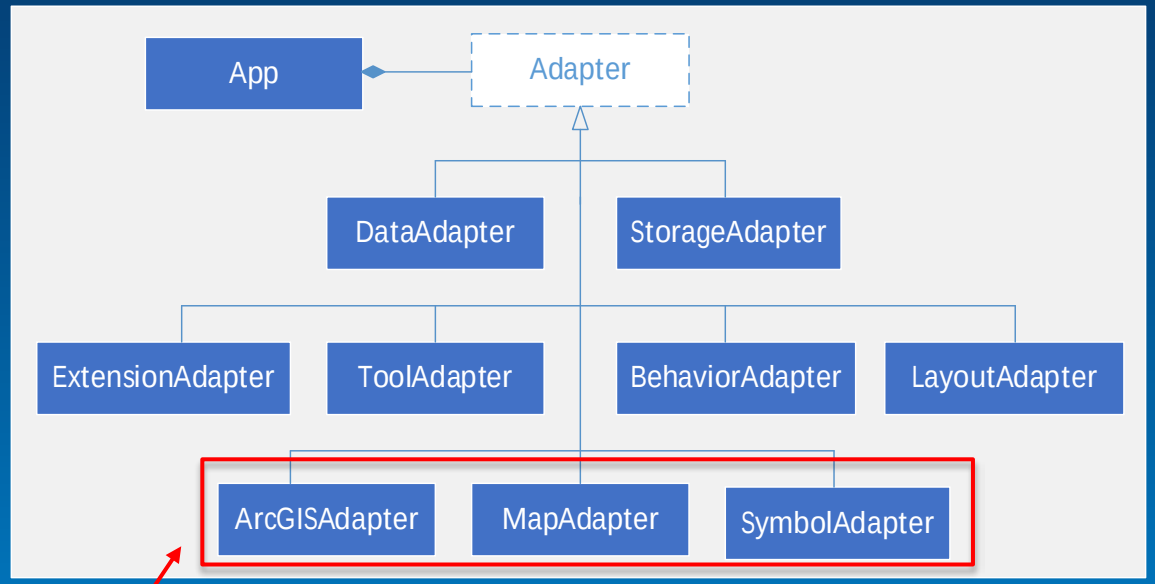
ArcGISAdapter, MapAdapter, SymbolAdapter

ArcGISAdapter - interacts with ArcGIS (Portal or online)

- Built-in users
- Enterprise logins (Online)
- PKI / IWA / LDAP (portal for ArcGIS)

- **MapAdapter - provides a default map for your App**
  - ArcGIS REST API web map

- **SymbolAdapter - provides the default symbols used by the App**



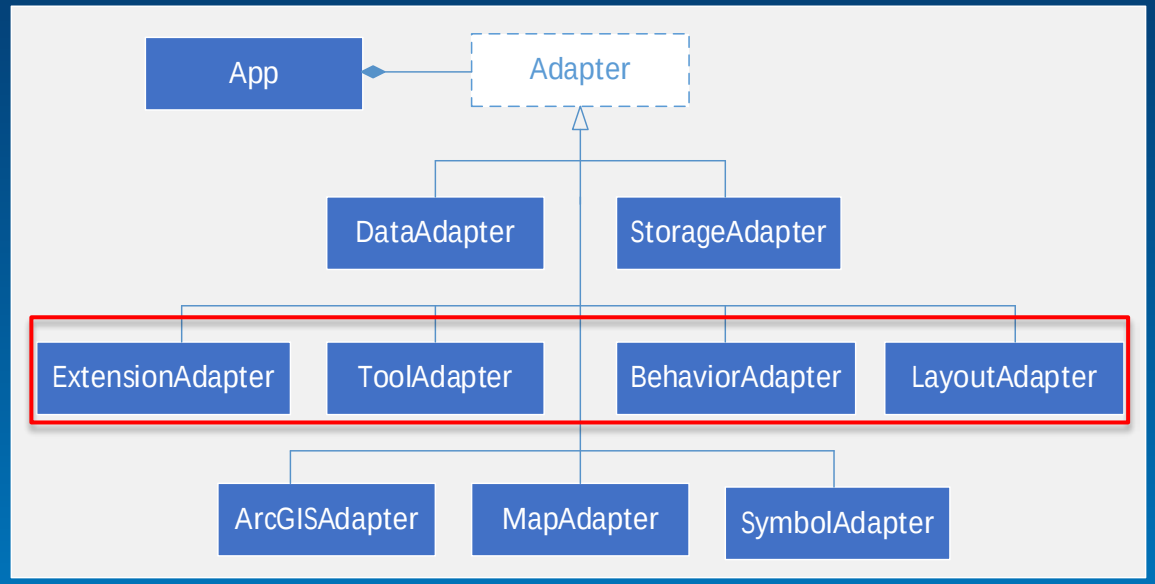
We provide good defaults

Configure/Extend the App

ExtensionAdapter, ToolAdapter, BehaviorAdapter, LayoutAdapter

We will talk about these during the 'Extensibility' section (after the first set of demos)

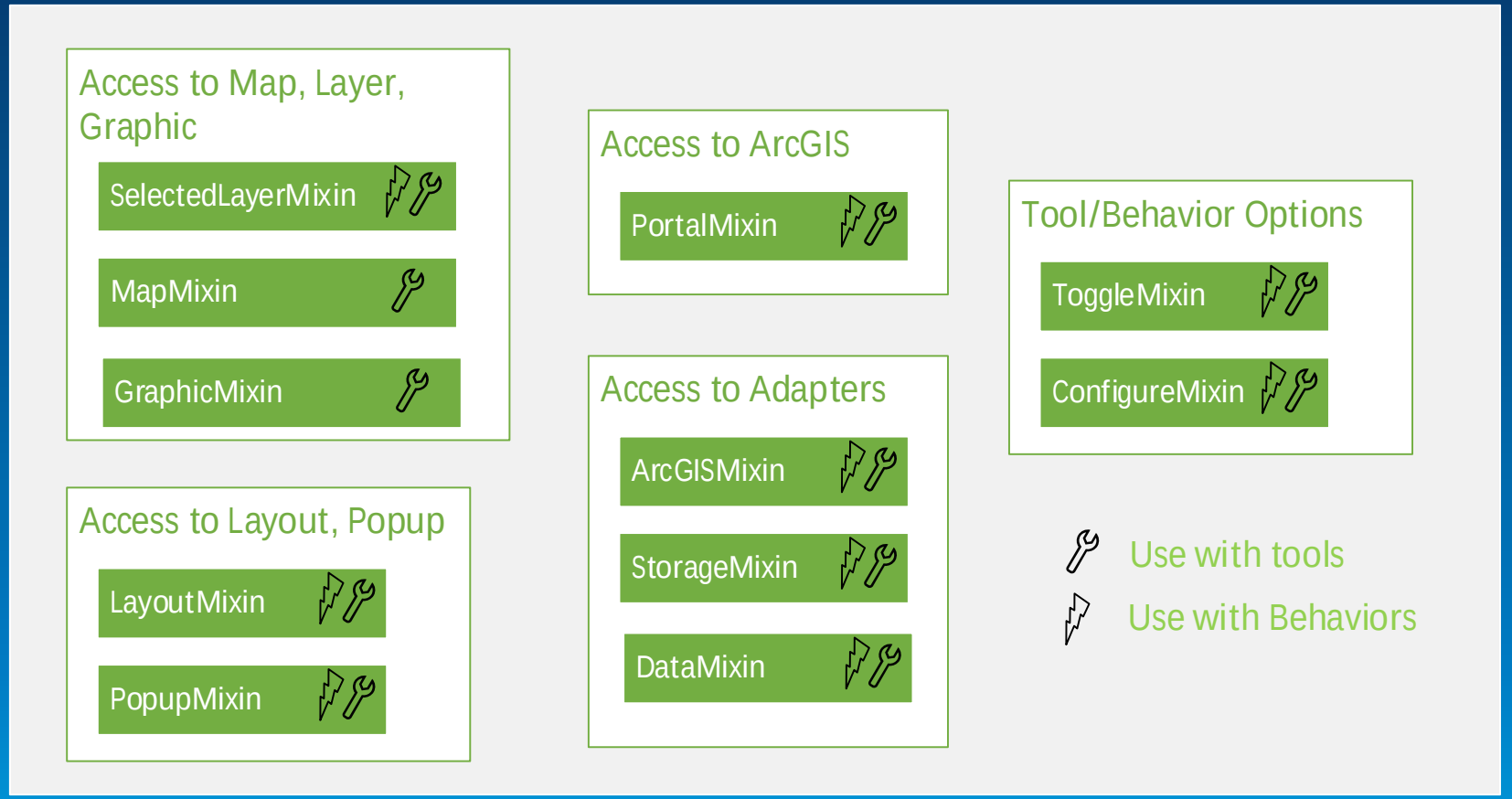
Only needed if you are writing customizations



DEMO

Extensibility

A note about mixins



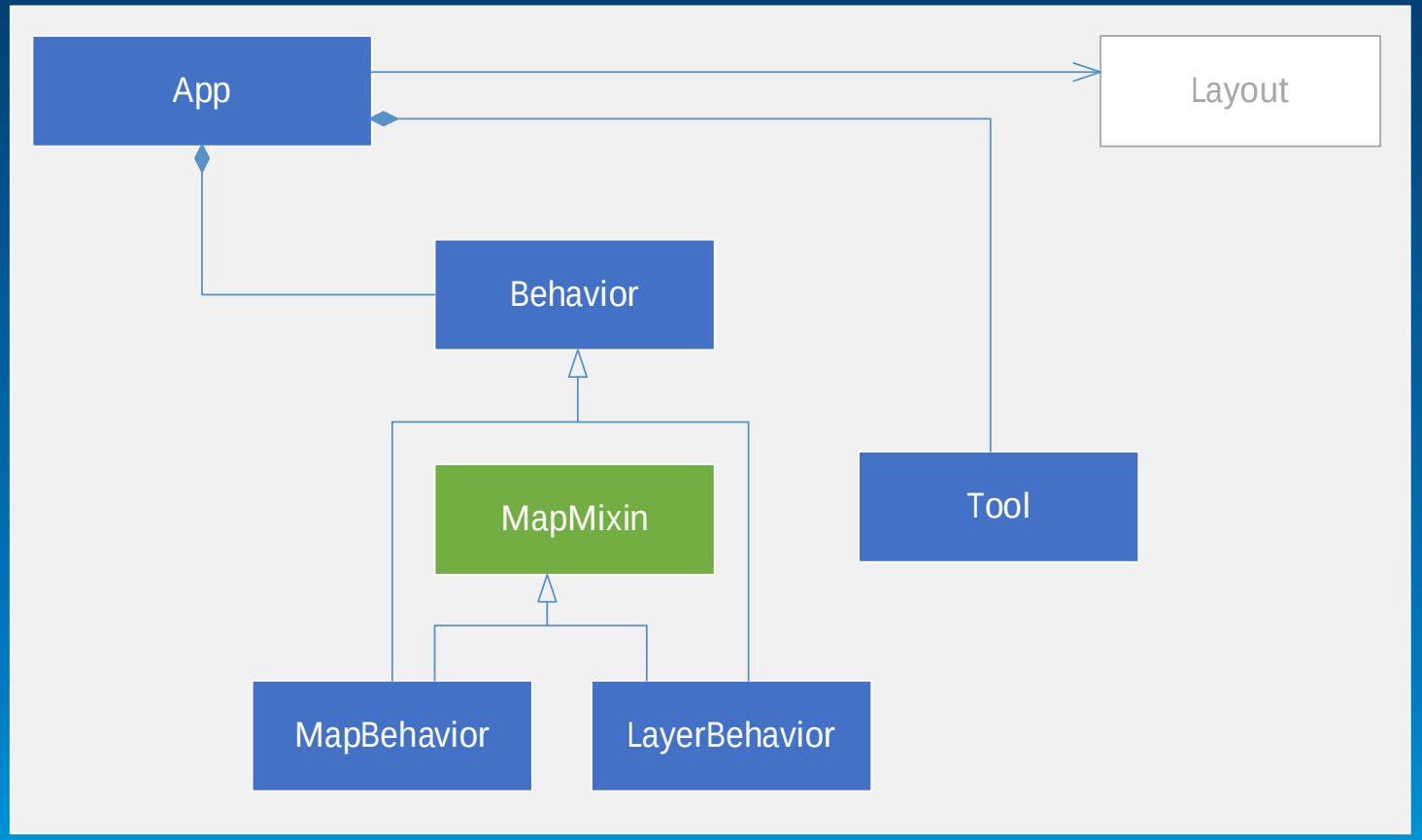
- **Use mixins to add functionality to tools/behaviors**
- **We use these (and other) mixins in our ‘core’ implementation**

Extending the App

Tools and Behaviors

Two ways to add new functionality:

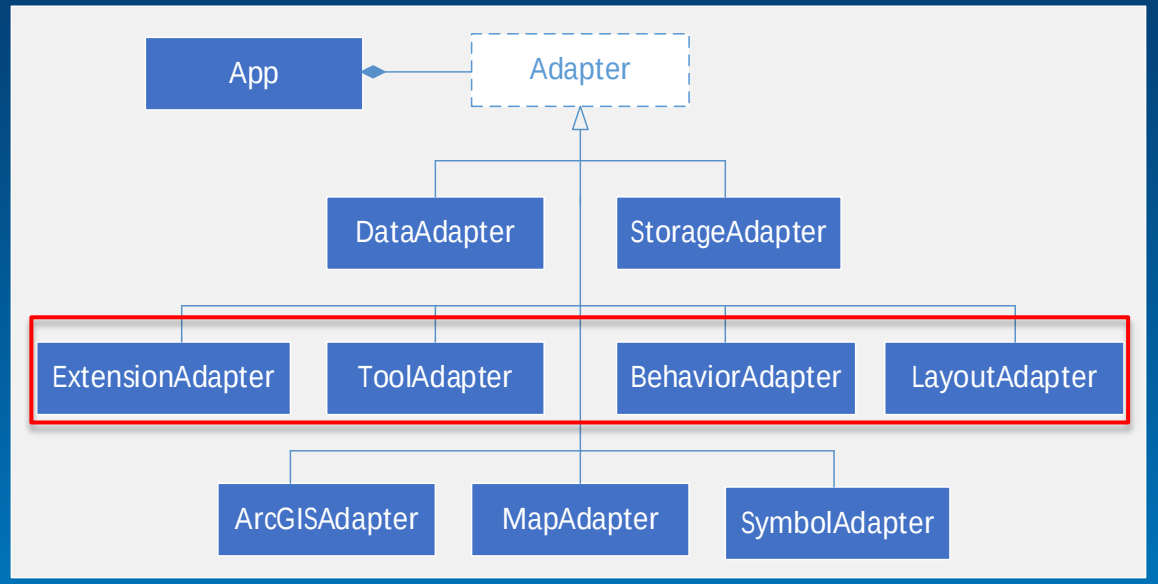
1. **Tool**
   - Triggered by user interaction
2. **Behavior**
   - Triggered by events



Extending the App

Adapters

- ExtensionAdapter – gathers tools and behaviors from one or more ‘extension packs’
- ToolAdapter – responsible for the list of tools
- BehaviorAdapter – responsible for the list of behaviors
- LayoutAdapter – responsible for where the tools are placed on the layout



Extending the App

ExtensionAdapter

- An ExtensionAdapter returns an array of ‘extension packs’ (i.e. you can reference more than one extension pack)
- Extension pack
  - File named manifest.json.txt (accessed via url)
  - Contains lists of tools and behaviors

MyExtensionAdapter.js

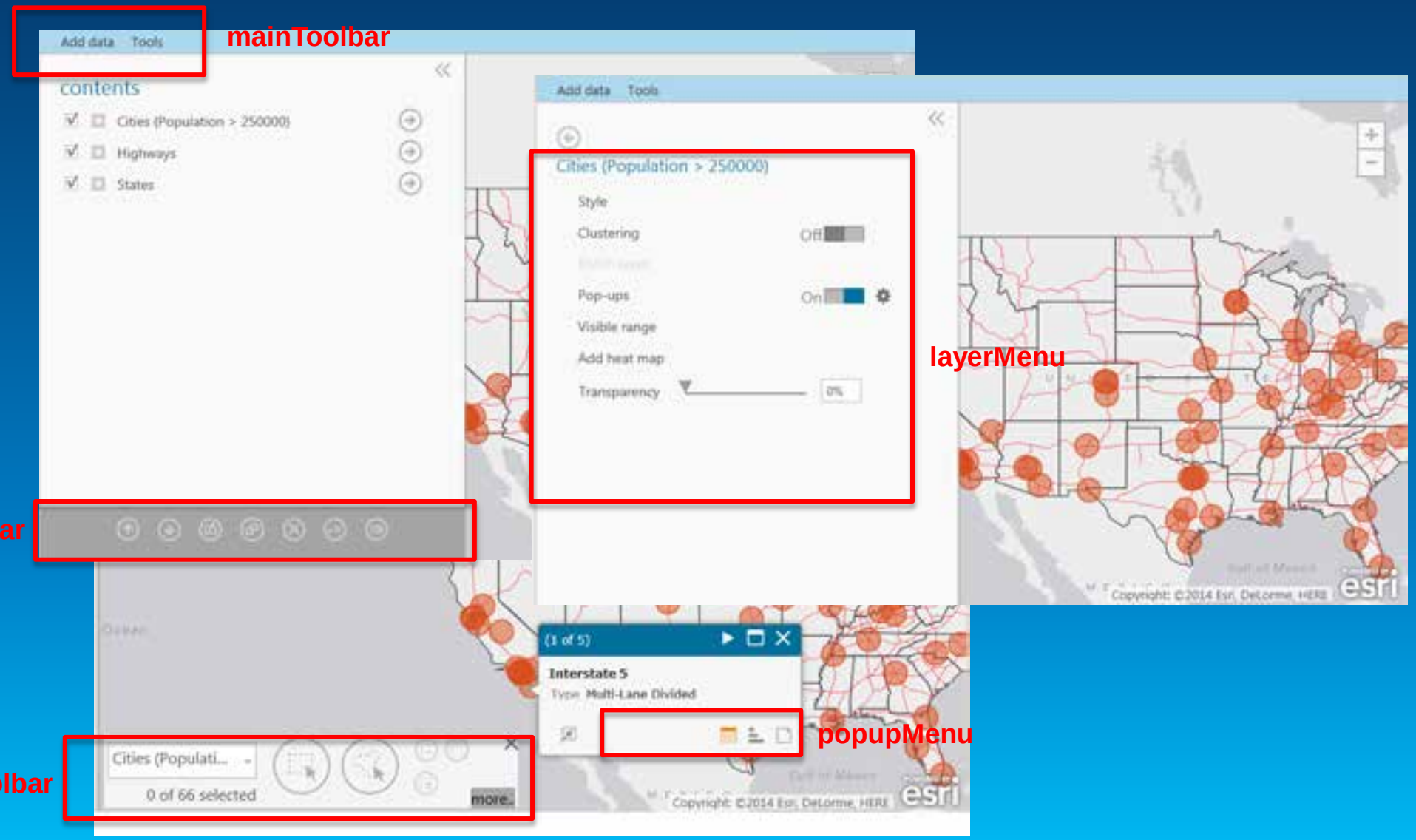
```
define([
    "dojo/_base/declare",
    "esriMaps/adapters/ExtensionAdapter"
], function(declare, ExtensionAdapter) {

    // Create a custom extension adapter by inheriting Extension Adapter class
    return declare(ExtensionAdapter, {

        getDefaultExtensions: function() {
            return [{
                name: "MyExtensions",
                location: location.pathname.replace(/\/[^/]+$/, '/') + "/resources"
            }];
        }
```

manifest.json.txt

```
{
    "tools": [{
        "name": "MyTool",
        "path": "./AlertTool"
    }],
    "behaviors": [{
        "name": "MyBehavior",
        "path": "./AlertBehavior"
    }]
}
```

Extending the App

‘Tool map’



Checklist: Deploy a custom tool/behavior

Step 1	• Create an ‘extension pack’ (manifest.json.txt) to list your new tools/behaviors
Step 2	• Write an ExtensionAdapter to point to the manifest
Step 3	• Write tools/behaviors (use mixins for your use-case)
Step 4	• Write a ToolAdapter / Behavior Adapter
Step 5	• Pass your adapters to the App constructor

DEMO

FAQ

FAQs

- How do I get EMF?
- When is it being released?
- Do you support enterprise logins?
- Do you support an on-premise implementation?
- Can I use ArcGIS for JavaScript digits? (Etc)
- How does this differ from ArcGIS WebApp Builder?


esri
Understanding our world.