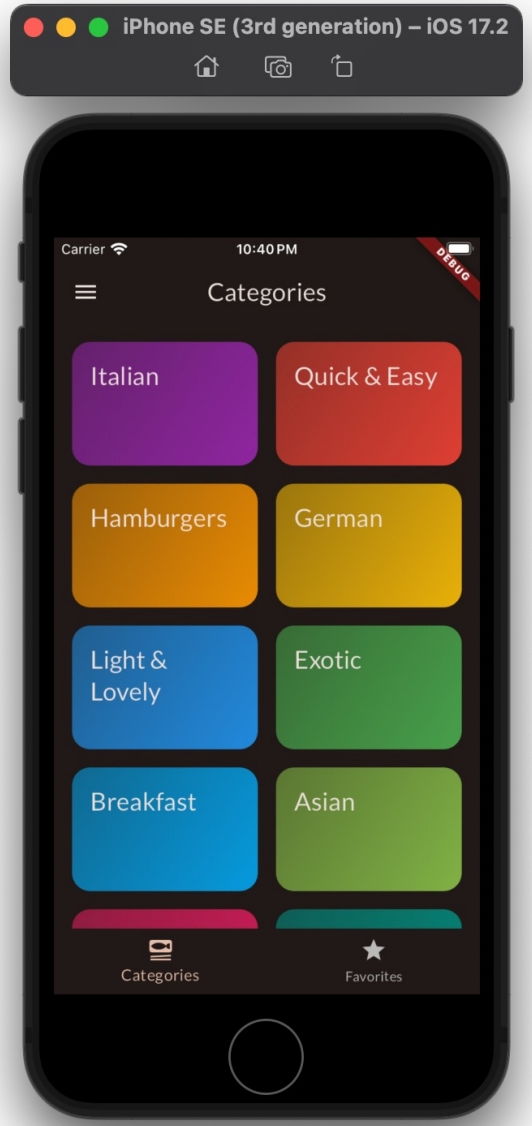
Global State Management & Navigation

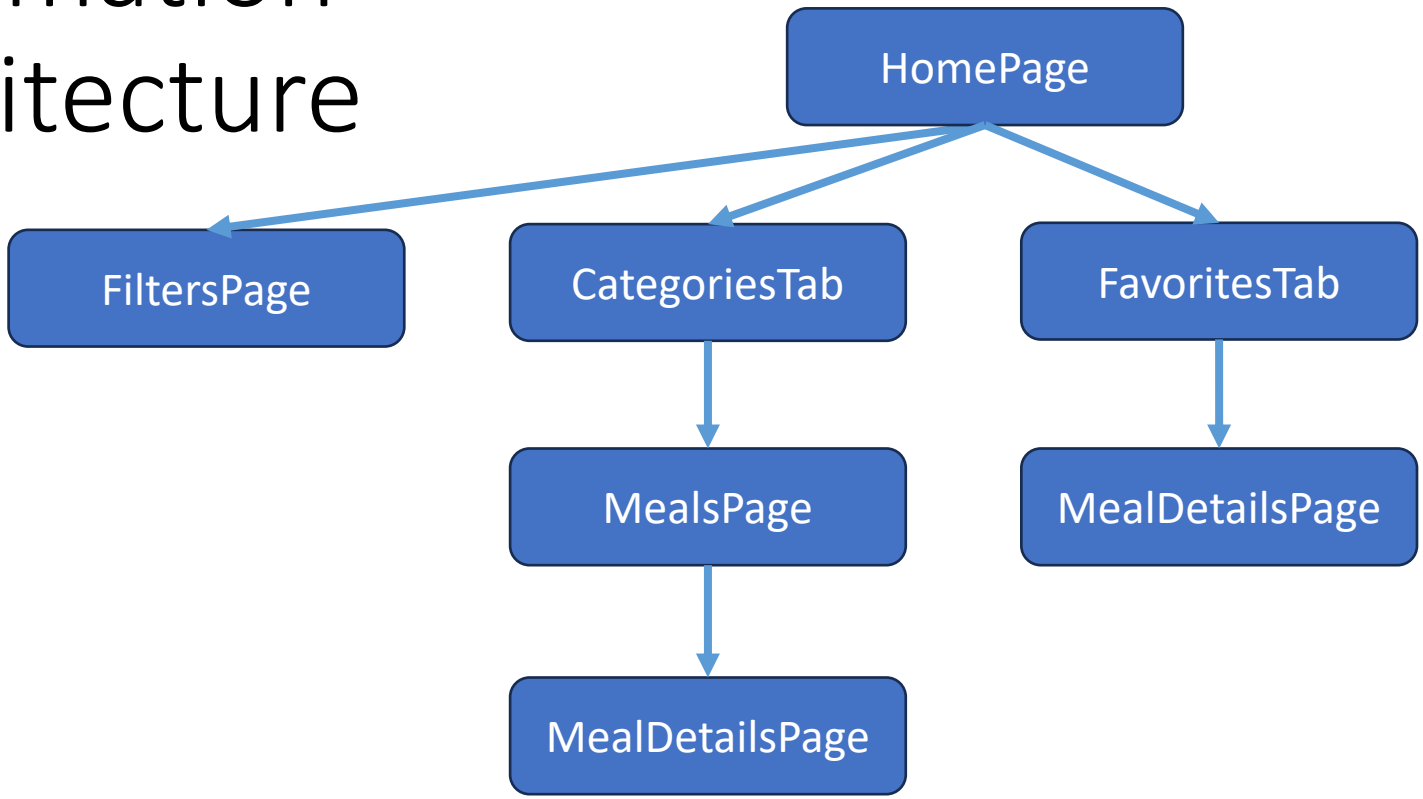
Shan-Hung Wu
CS, NTHU

Let’s Cook with Help of Meals App

- Meals displayed in categories
- Filters for meals
- Favorite meals
- Meal details & scroll tracking
  - via `NotificationListener`
  - for dynamic opacity of `AppBar`

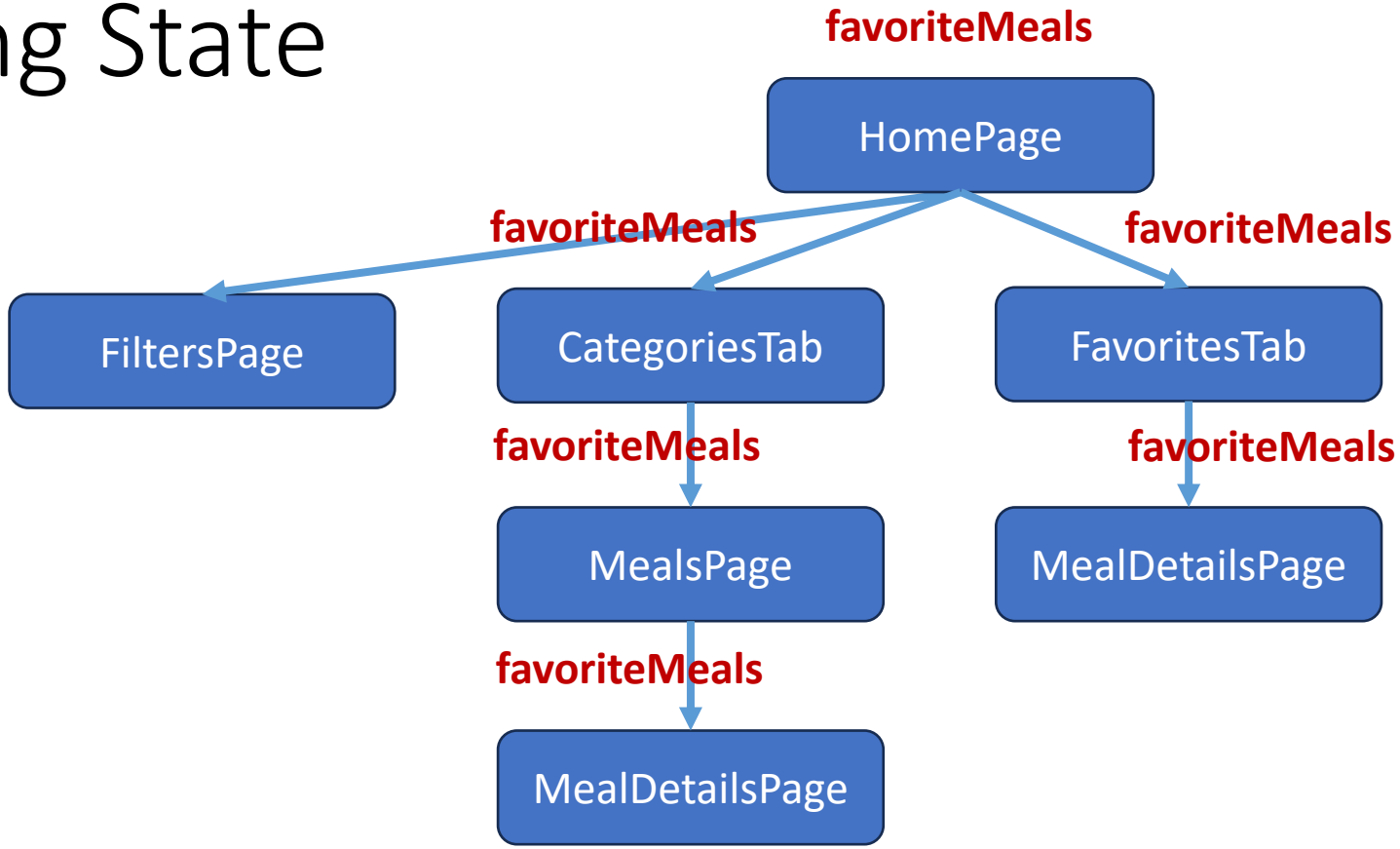


Information Architecture



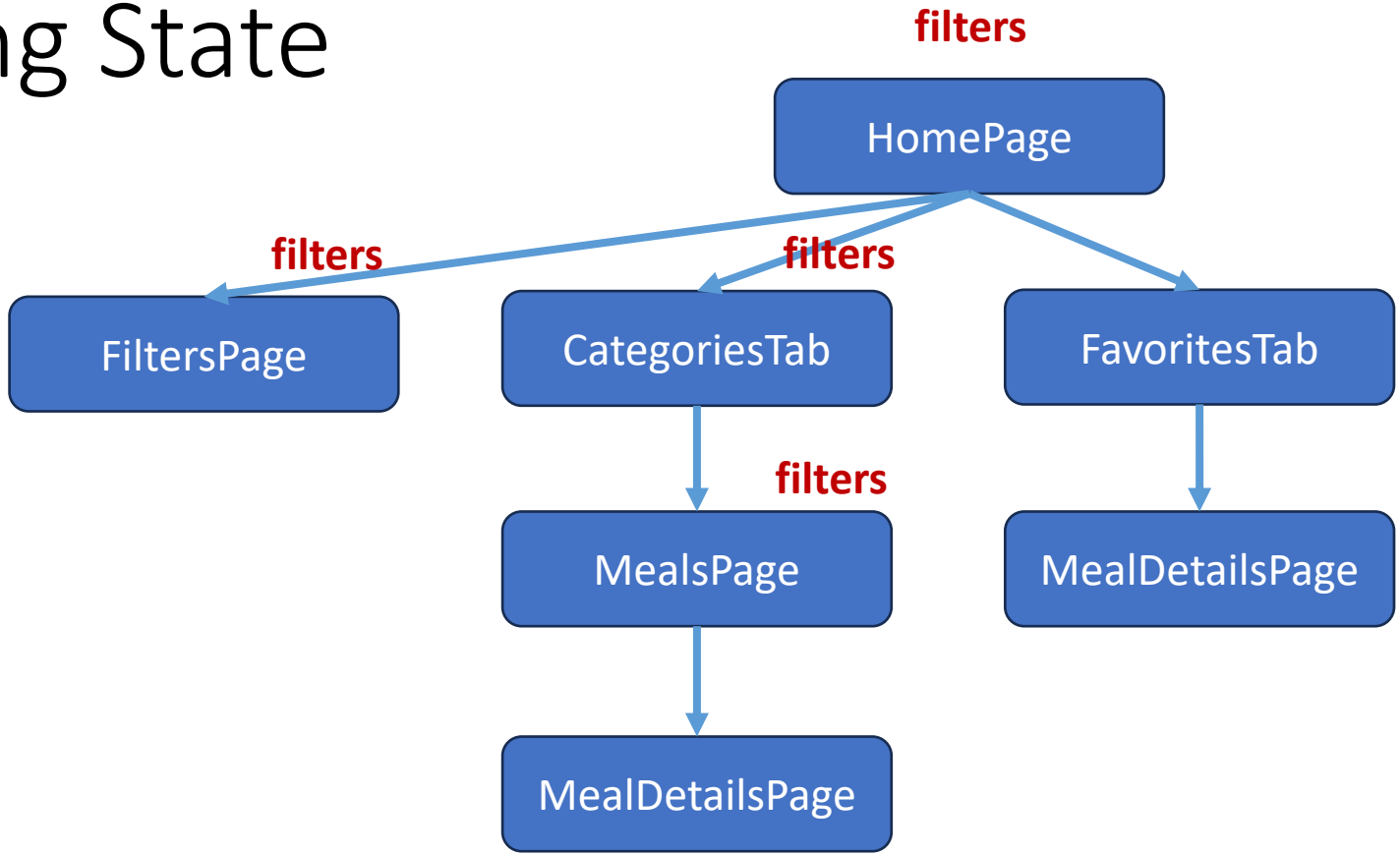
- Users can toggle favorite meals at bottoms of the two branches
- Where to keep the `favoriteMeals` state?

Lifting State Up



- The `favoriteMeal` is passed around as constructor arguments everywhere

Lifting State Up



- Similar problem exists for the `filters` state

State Management

- Flutter provides **`InheritedWidget`** to solve the problem
- Data are “inherited” by ***every child widget*** in widget tree
- Each child depending on the data can choose to ***rebuild*** whenever the data change

- In practice, we use `Provider`, a wrapper of `InheritedWidget`, to reduce boilerplate code

Providing Changing Data

```
// 1. define data
class Counter
    extends ChangeNotifier {
  int _count = 0;
  int get count => _count;

  void increment() {
    _count++;
    notifyListeners();
  }
}
```

```
// 2. top widget
ChangeNotifierProvider<Counter>(
  create: (context) =>
      Counter (),
  child: ...,
)
```

- `Provider` creates an `InheritedWidget` internally
- It also manages `ChangeNotifier`'s lifecycle
  - E.g., Calling `ChangeNotifier.dispose()` when removed from widget tree

Reading Data

```
// in child's build()
final counter = Provider.of<Counter>(
    context, listen: true);
final count = counter.count;
// or
final counter = context.watch<Cunter>();
final count = counter.count;
// or
return Consumer<Counter>(
  builder: (context, counter, child) {
    final count = counter.count;
    ...
  }
);
```

1. Start from element associated with context
2. Moving up along element tree to find ***the nearest data of matching type***
3. If `listen` is true, ***rebuild widget*** when data change

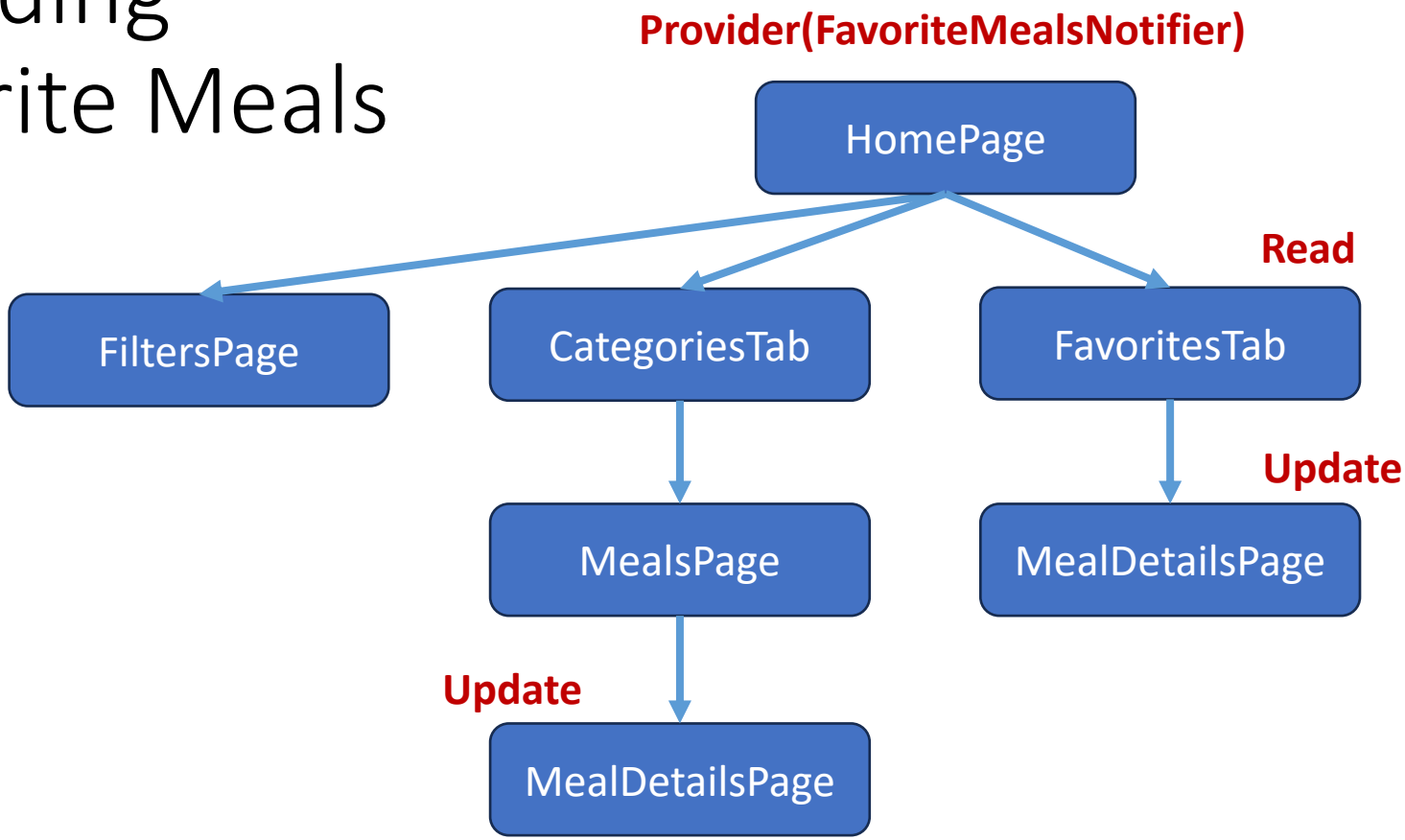
Updating Data in Child Widget

```
// data
class Counter
  ...
  void increment() {
    _count++;
    notifyListeners();
  }
}
```

```
// in child's build()
return Widget(
  button: FilledButton(
    onPressed: () {
      Provider.of<Counter>(
        context,
        listen: false,
      ).increment();
      // or context.read<Counter>()
      ...
    },
  ),
);
```

- `increment()` calls `notifyListeners()` in turn, resulting in rebuild of all listening widgets

Providing Favorite Meals



- No constructor arguments passed around

Other Providers

- Provider
  - For constant value that won't change over time
- StreamProvider
  - For `Stream` and exposing the latest value emitted
- MultiProvider
  - Provides multiple types of data at once
- ProxyProvider
  - Provide data that depends on other providers
  - Common variant: ChangeNotifierProxyProviderK

ChangeNotifierProxyProvider2

```
// provided data
class A extends ChangeNotifier {
  ...
}
class B extends ChangeNotifier {
  ...
}
```

- Data `C` depend on `A` and `B`

```
// new provided data
class C extends ChangeNotifier {
  int _c;

  void updateData(int _a, int _b) {
    _c = _a + _b;
    notifyListeners();
  }
  ...
}
```

ChangeNotifierProxyProvider2

```
// in widget's build()
return MultiProvider(
  providers: [
    ChangeNotifierProvider<A>(create: (context) => A()),
    ChangeNotifierProvider<B>(create: (context) => B()),
    ChangeNotifierProxyProvider2<A, B, C>(
      create: (context) => C(),
      update: (context, a, b, prevC) =>
        prevC!..updateData(),
      ),
  ],
  child: ...,
);
```

- `update` is called when `a`/`b` is initialized or changes
- Dart's cascade operator "`..`"

Navigation

```
// in widget
Navigator.of(context).push(
  MaterialPageRoute(
    builder: (context) => ...,
  ),
);
Navigator.of(context).pop();
```

- Problems?
- ***Imperative*** screen transitions and nav state
- ***Scattered*** transition logic
- ***No web*** support

Navigation 2.0 and Routing

- ***Declarative***: define “routes” declaratively
- Advanced routing logic such as ***nested navigators***
- Supports web and ***deep linking***
- ***Nav state restoration*** after app is killed

```
// in App
Widget build(BuildContext context) {
  return MaterialApp.router(
    theme: ...,
    routerDelegate: ...,
    routeInformationParser: ...,
    restorationScopeId: 'myapp',
  );
}
```

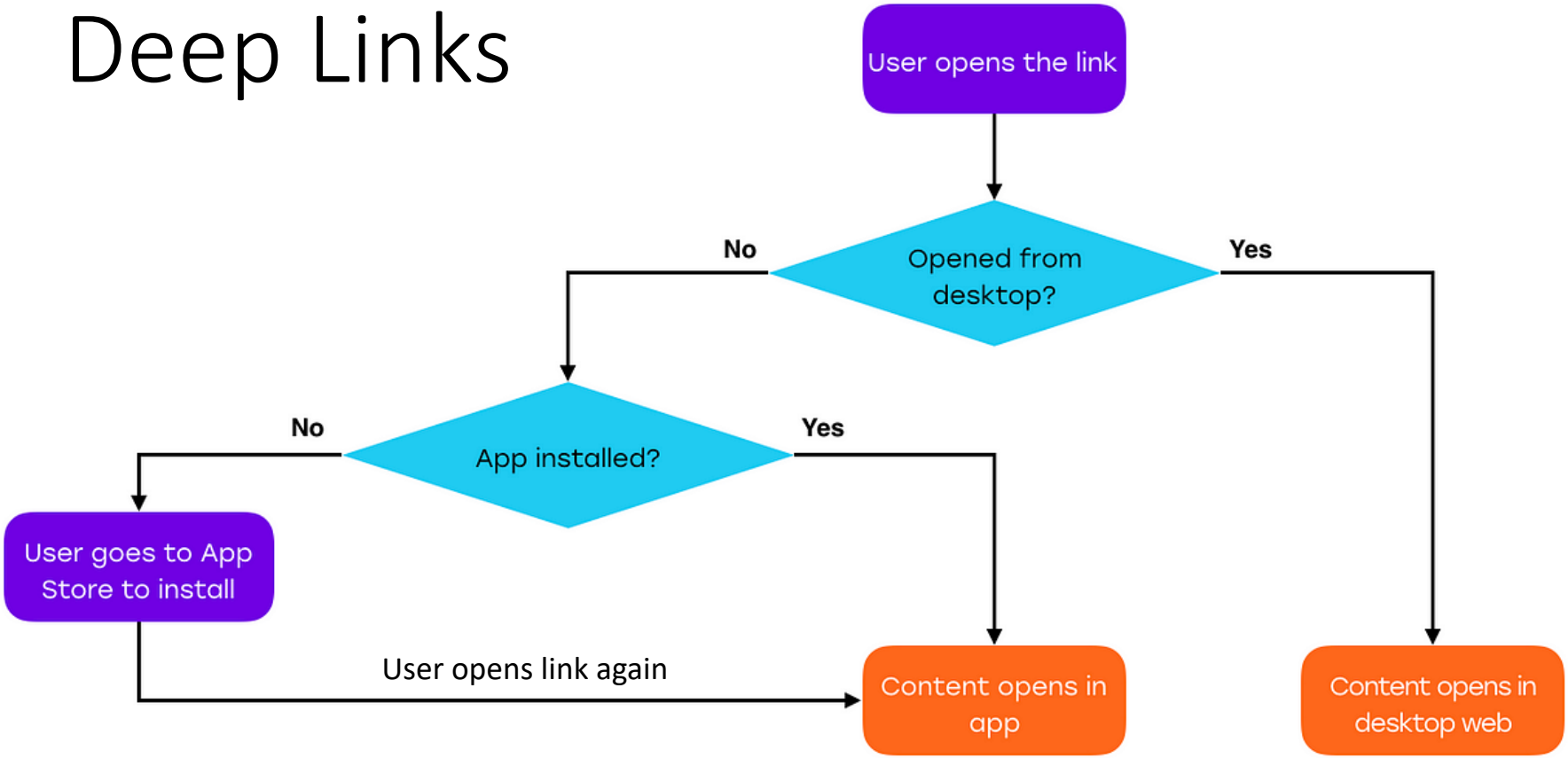
Package go_router

- Abstracts the complexity of Navigator 2.0 away with built-in `RouterDelegate` and `RouteInformationParser`
- Built-in support for deep links

```
// in App
final _routerConfig = GoRouter(...); // define routes

Widget build(BuildContext context) {
  return MaterialApp.router(
    theme: ...,
    routerConfig: _routerConfig,
    restorationScopeId: 'myapp',
  );
}
```

Deep Links

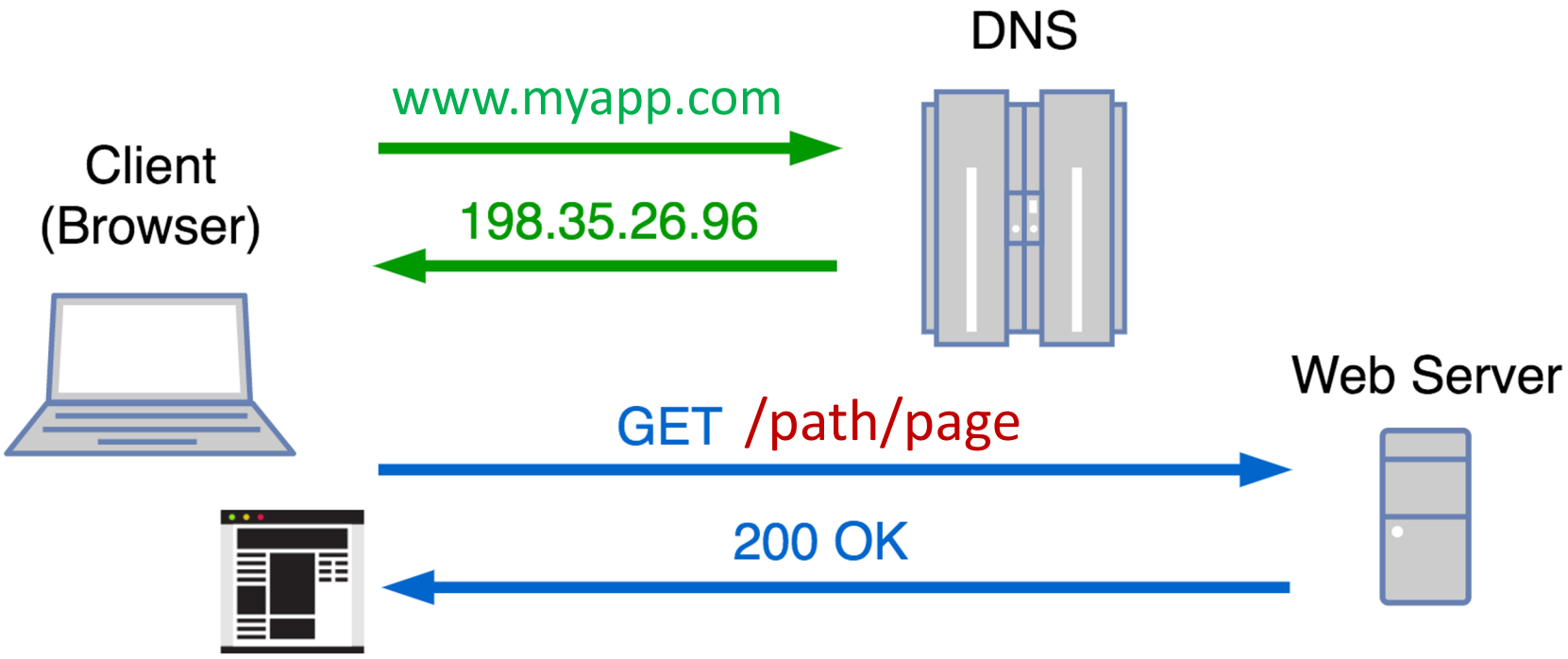


- Links are URLs defined in HTTP protocol

To define routes, we need some knowledge about web development…

From URL to Web Page

https://www.myapp.com/path/page



HTTP

- A ***protocol*** is language spoken by machines
  - Defines structure of messages to be exchanged
- HyperText Transfer Protocol (HTTP) defines:
  - Messages: HTTP ***request*** and HTTP ***response***
  - Requests: accessing ***resources*** (web pages) via **GET**, **POST**, **PUT**, and **DELETE** methods
  - Responses: ***2***00 OK, ***3***01 Moved, ***4***04 Not Found, ***5***00 Server Error, etc.

HTTP Messages

- Each HTTP message have
  - Initial line, header lines, and optionally body

Request:

```
GET /path/page HTTP/1.1
Host: www.myapp.com
Accept: application/json
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 3324

{
  "field1": "value1",
  "field2": "value2",
  ...
}
```

- A resource can have different ***presentations***
  - HTML, XML, etc.

Defining URLs for Web + App

- E.g., listing docs, create a doc, edit title, etc.
- But HTTP defines only 4 methods
  - GET, POST, PUT, DELETE
- Option 1: define new “verbs”
  - Always POST to the same URL
  - Define body by following SOAP protocol
- Option 2: define new “nouns”
  - E.g., edit title → POST /title
  - Different URLs for different nouns/resources
  - Called RESTful URLs

Restful URLs

URLs\Methods	GET	POST	PUT	DELETE
https://${host}/${resource}**s**	List all resources (with query "?...")	Create a new resource (unknown ID)	Replace the entire collection	Delete the entire collection
https://${host}/${resource}s/**${id}**	Read a specific resource	Treat this resource as a collection and create a new member	Update this resource or create one (known ID)	Delete this resource

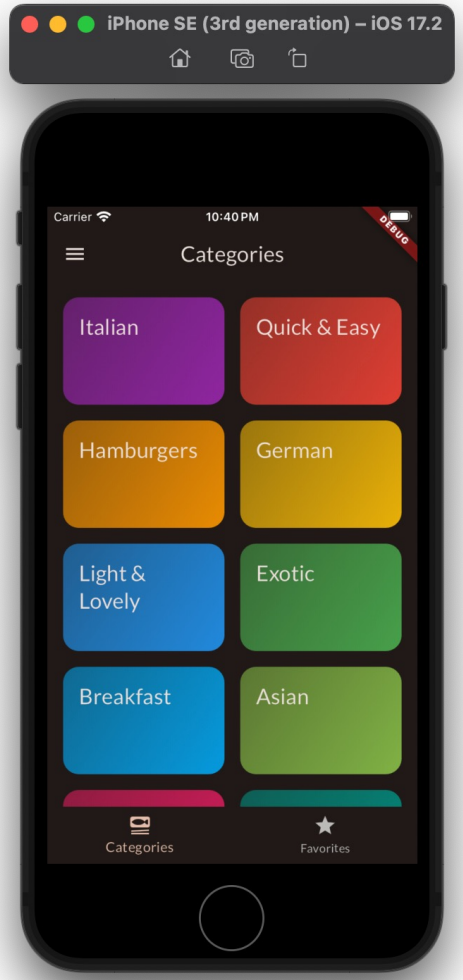
- Each resource type maps to 2 URL types
  - Collection URLs vs resource URLs

- Search docs: `GET /docs?query=...&sort=...`
- Create doc: `POST /docs`
- Edit title: `POST /docs/${id}/title`

`NavigationService` in Meals App

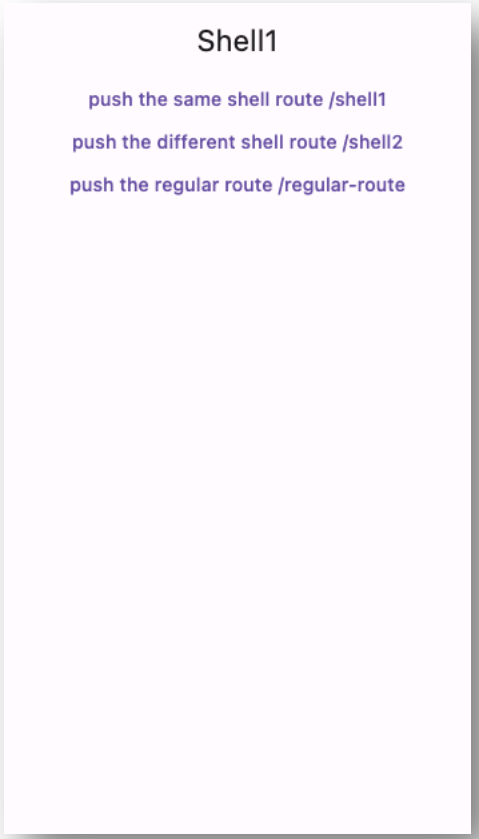
- Centralizes declaration of routes/pages
  - with nested routes
- Centralizes route transition logic
  - via `push()` or `go()`

- Needs to be provided at top of widget tree

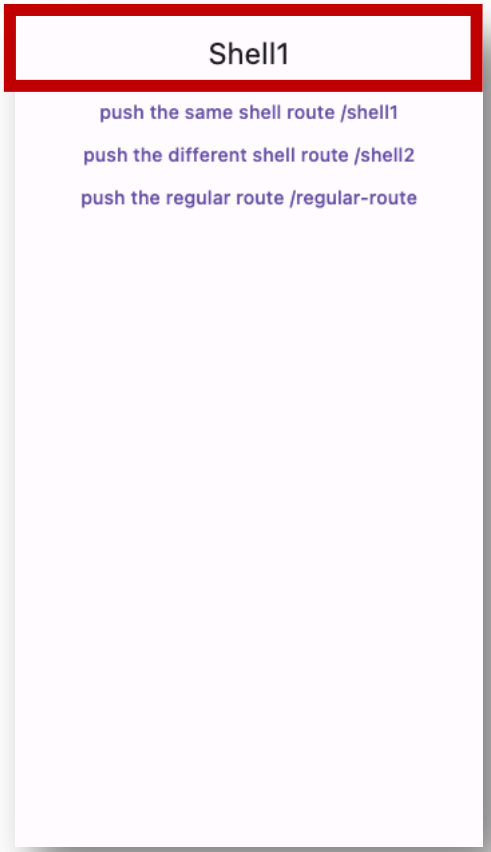


Shell Routes

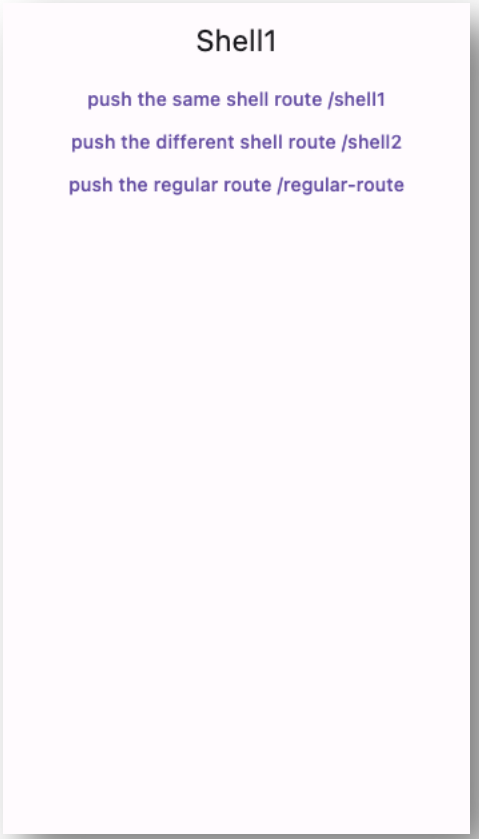
- Routes having “shells” outside of their nav stack



No shell



Same shell



Different shells

Assigned Reading

- Providers
  - StreamProvider
  - MultiProvider
  - ProxyProvider, in particular ChangeNotifierProxyProvider2
- Routes
  - ShellRoute with code example