# WooCommerce CRUD and Data Store

Akeda Bagus

# Akeda Bagus

- Code wrangler at Automattic.
- Currently working on WooCommerce extensions and marketplace.

# CRUD and Data Store in WooCommerce?

Why we need that?

# **C**reate, **R**ead, **U**pdate, **D**elete

- WooCommerce has some resource types: orders, products, customers, coupons, etc.

- With CRUD in resource, the developer doesn't need to know the internal complexities when interacting with WooCommerce resource types.

- Increase DRY. It can be used in WooCommerce REST API and CLIs.

- Once abstracted, the underlying data storage of resource can be anything (CPT, custom table, external service, or dummy storage).

- Increase test coverage.

- For more detail see https://woocommerce.wordpress.com/2016/10/27/the-new-crud-classes-in-woocommerce-2-7/.

# Resource Interaction without Data CRUD

```
$order_id = 123;


// Retrieving.
$billing_name = get_post_meta( $order_id,
‘_billing_first_name’, true ) . ‘ ‘ . get_post_meta( $order_id,
‘_billing_last_name’, true );

// Updating.
update_post_meta( $order_id, ‘_billing_first_name’, ‘Akeda’ );
update_post_meta( $order_id, ‘_billing_last_name’, ‘Bagus’ );
```

# Resource Interaction without Data CRUD

- Developer needs to know WooCommerce meta keys and expected value's type to store for each resource type.
- Data storage is tied to WP tables.
- Unit test needs to be set up with the test DB unless all WP dependencies are mocked.

# Resource Interaction with Data CRUD

```
$order_id = 123;
$order = wc_get_order( $order_id );


// Retrieving.
$billing_name = $order->get_billing_first_name() . ‘ ‘ .
$order->get_billing_last_name();

// Updating.
$order->set_billing_first_name( ‘Akeda’ );
$order->set_billing_last_name( ‘Bagus’ );
$order->save();
```

# Resource Interaction with Data CRUD

- Developer only needs to know the Data CRUD methods.

- Data CRUD objects handle flow of data and any validations. Less work for developers!

- Easier to test the resource.

# Resource Interaction with Data CRUD

```
// Setters and getters on the resource.
// Basically set_{property} and get_{property}.

$product = wc_get_product( 123 );

$product->set_description( ‘My cool product’ );
$product->set_sku( ‘WOO123’ );
$product->set_price( 15000 );

$product->get_description(); // My cool product
$product->get_sku();         // WOO123
$product->get_price();       // 15000
```

# Resource Interaction with Data CRUD

```
// Saving and deleting with save() and delete().

$product = wc_get_product( 123 );
$product->set_price( 25000 );

// Internally, this will pass the action to the data store to
// persist the change. If no ID, it implies creation otherwise
// it will update existing product.
$product->save();

// This will delete the product from the DB.
$product->delete();
```

# Resource Interaction with Data CRUD

- All properties on resource types can be found at WooCommerce GitHub's wiki.

- https://github.com/woocommerce/woocommerce/wiki/CRUD-Objects-in-3.0

- Once you know available properties, the pattern is easy to remember:
  - `get_{property}` to retrieve the property.
  - `set_{property}` to set the property.
  - Use `save()` and `delete()` to create/update and delete the resource.

# Resource Interaction with Data CRUD

- All Data CRUD objects must extend abstract `WC_Data`.

- `WC_Data` handles the communication with the data store, so the implementation only needs to extend the abstract and define some properties that are available via setters and getters.

- Examples of WooCommerce Data CRUD classes:

  - `WC_Coupon`
  - `WC_Order`
  - `WC_Product`
  - `WC_Customer`

# Data Store

- Bridge Data CRUDs and storage layer.

- If Data CRUDs need data, they ask Data Store. Same thing when saving and deleting.

- Data Store can be powered by CPT, custom table, remote API, or anything. It's easy to swap the data store of a Data CRUD class.

- WooCommerce built-in Data CRUDs are shipped with CPT and custom table Data Store.

# Data Store

- Data Store should be started by defining interface. The implementation can be CPT, custom table, external API, or anything.
- Methods defined in the interface can be `create()`, `read()`, `read_meta()`, `delete_meta()`, etc.
- Every Data Store for Data CRUD object should implement that interface. For CPT, the `create()` could use `wp_insert_post()`, while for external API the `create()` could use `wp_remote_post()`.
- A method may accepts reference to `$data`, which has a type of `WC_Data.`
- For more detail see https://github.com/woocommerce/woocommerce/wiki/Data-Stores.

# Data Store

```
// Example of Data Store interface.

interface WC_Object_Data_Store_Interface {
  public function create( &$data );
  public function read( &$data );
  public function update( &$data );
  public function delete( &$data, $args = array() );
  public function read_meta( &$data );
  public function delete_meta( &$data, $meta );
  public function add_meta( &$data, $meta );
  public function update_meta( &$data, $meta );
}
```

# Data Store

```
// Example create() method from Coupon CPT Data Store. With CPT
// as the underlying storage, create() uses wp_insert_post().

public function create( WC_Coupon &$coupon ) {
  // ...

  $coupon_id = wp_insert_post( [
    ‘post_type’ => ‘shop_coupon’,
    ‘post_title’ => $coupon->get_code(),
    // ...
  ] );

  if ( $coupon_id ) {
    $coupon->set_id( $coupon_id );
    // ...
  }
}
```

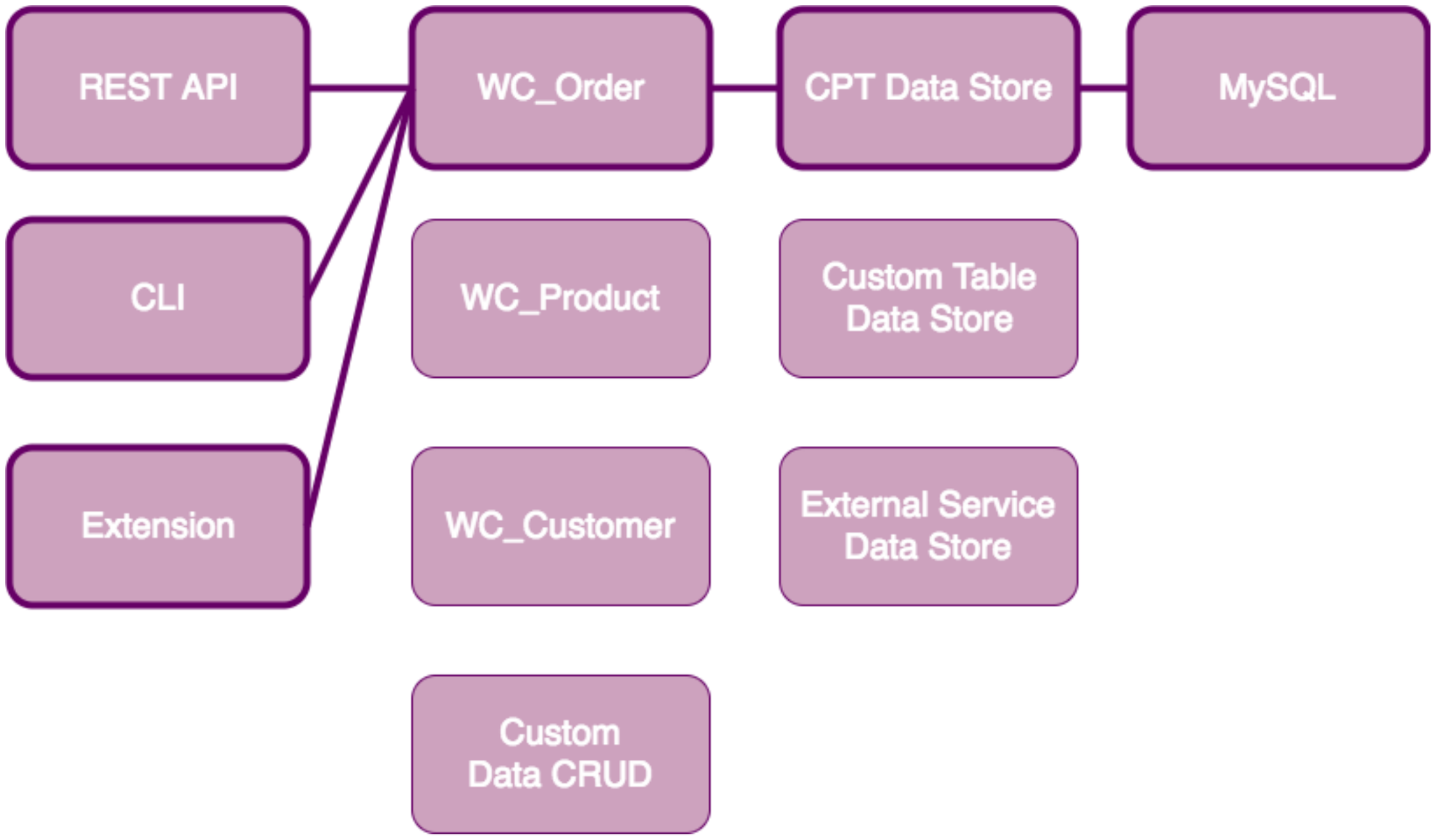
# Data Store

```
// Example delete_order_item() method from order item
// Data Store which uses custom table.

public function delete_order_item( $item_id ) {
  global $wpdb;
  $wpdb->query( /* ... */ );
  $wpdb->query( /* ... */ );
}
```

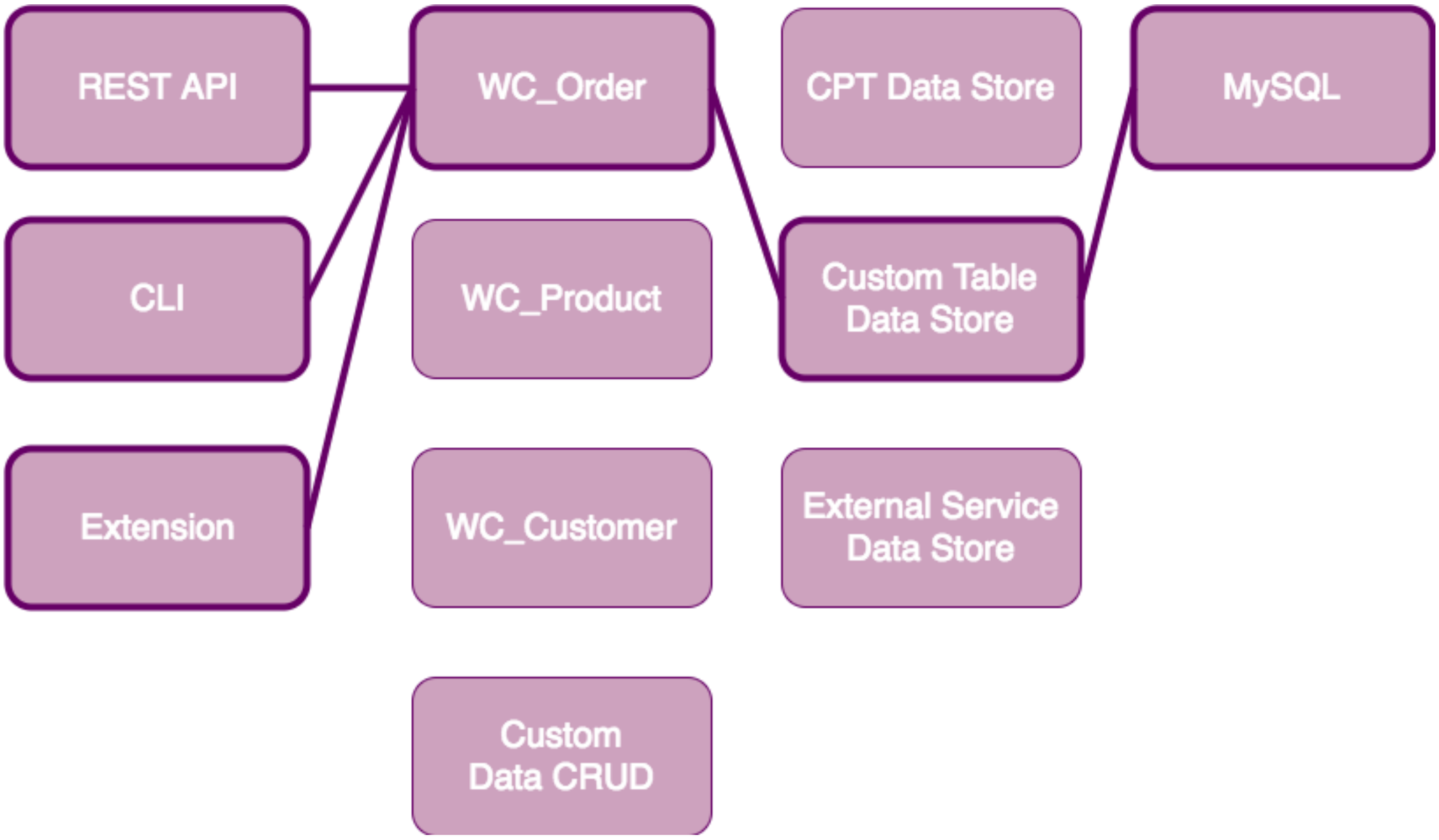
# Data Store

```
// Example create() with external API request.

public function create( WC_Data &$data ) {
  // ...

  $result = wp_remote_post( /* ... */ );
  if ( ! empty( $result['id'] ) ) {
    $data->set_id( $result['id'] );
  }

  // ...
}
```

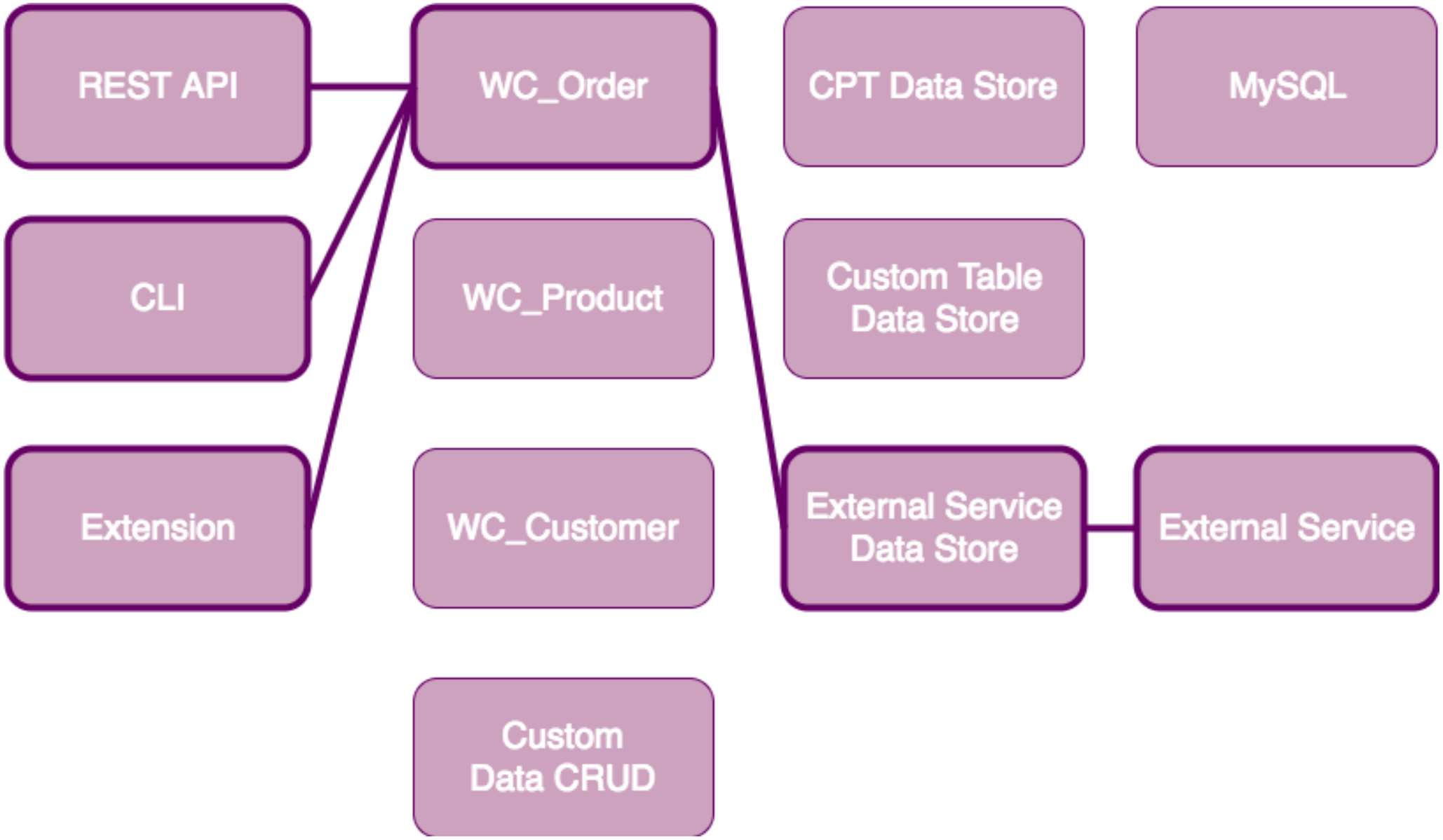
# Data CRUD and Data Store Interaction



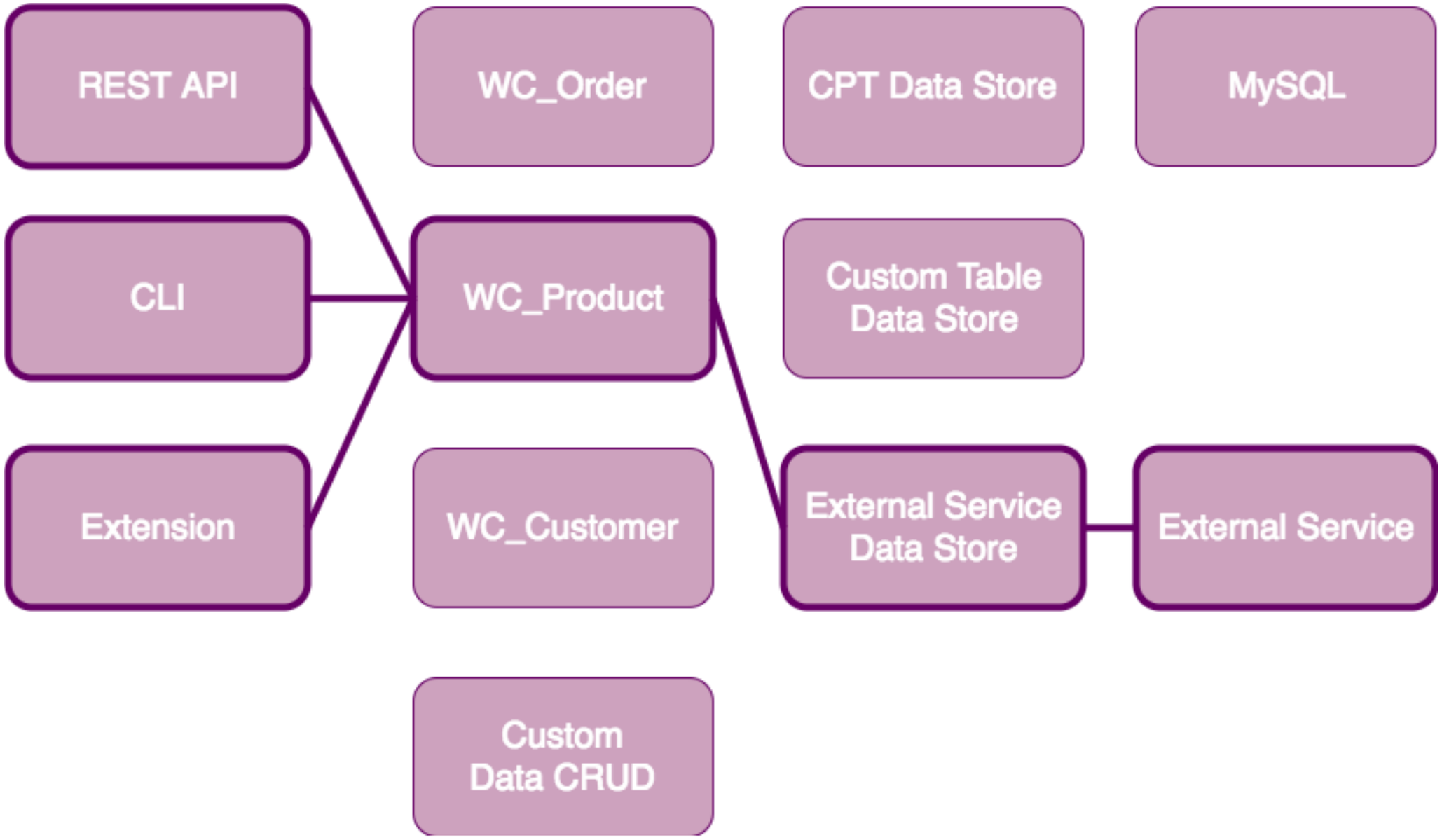
# Data CRUD and Data Store Interaction



# Data CRUD and Data Store Interaction



# Data CRUD and Data Store Interaction



# Creating your own CRUD in your extension (CPT Data Store)

- Let's use **Restaurant** as an example of a custom resource.
- First, define some properties that a **Restaurant** needs. For example:
    - Name
    - Address
    - Cuisine
    - Opening hours
- Map the properties in the CRUD class.
- Start **Restaurant** data store by using CPT as the underlying storage.

# Creating your own CRUD in your extension (CPT Data Store)

```
namespace Gedex\WCJKT\Restaurant\Data_Objects;

class Restaurant extends \WC_Data {

  // Name value pairs (name => default value).
  protected $data = [
    ‘name’ => ‘’,
    ‘address’ => ‘’,
    ‘cuisine’ => ‘’,
    ‘opening_hours’ => ‘9am - 10pm’,
  ];

  public function __construct( $restaurant = 0 ) { ... }
  public function get_name( $context = ‘view’ ) { ... }
  public function set_name( $name ) { ... }
}
```

# Creating your own CRUD in your extension (CPT Data Store)

```
namespace Gedex\WCJKT\Restaurant\Data_Objects;

class Restaurant extends \WC_Data {
  public function __construct( $restaurant = 0 ) {
    parent::__construct( $restaurant );

    if ( is_numeric( $restaurant ) && $restaurant > 0 ) {
      $this->set_id( $restaurant );
    } elseif ( $restaurant instanceof self ) {
      $this->set_id( $restaurant->get_id() );
    } else if ( ! empty( $restaurant->ID ) ) {
      $this->set_id( $restaurant->ID );
    } else {
      $this->set_object_read( true );
    }

    $this->data_store = \WC_Data_Store::load( ‘restaurant’ );

    if ( $this->get_id() > 0 ) {
      $this->data_store->read( $this );
    }
  }
}
```

# Creating your own CRUD in your extension (CPT Data Store)

```
namespace Gedex\WCJKT\Restaurant\Data_Objects;

class Restaurant extends \WC_Data {
  public function get_name( $context = ‘view’ ) {
    return $this->get_prop( ‘name’, $context );
  }

  public function set_name( $name ) {
    $this->set_prop( ‘name’, $name )
  }

  // Do the same for other properties.
  // ...
}
```

# Creating your own CRUD in your extension (CPT Data Store)

```
namespace Gedex\WCJKT\Restaurant\Data_Stores;

use Gedex\WCJKT\Restaurant\Data_Objects\Restaurant;

interface Restaurant_Interface {
  public function create( Restaurant &$restaurant );
  public function read( Restaurant &$restaurant );
  public function update( Restaurant &$restaurant );
  public function delete( Restaurant &$restaurant );
}
```

# Creating your own CRUD in your extension (CPT Data Store)

```
namespace Gedex\WCJKT\Restaurant\Data_Stores;

use Gedex\WCJKT\Restaurant\Data_Objects\Restaurant;

class Restaurant_CPT extends \WC_Data_Store_WP
 implements Restaurant_Interface
{
 public function create( Restaurant &$restaurant ) { }
 public function read( Restaurant &$restaurant ) { }
 public function update( Restaurant &$restaurant ) { }
 public function delete( Restaurant &$restaurant ) { }
}
```

# Creating your own CRUD in your extension (CPT Data Store)

- Let’s see in action.

# More Information for Data CRUD and Data Store

- https://github.com/woocommerce/woocommerce/wiki/CRUD-Objects-in-3.0

- https://github.com/woocommerce/woocommerce/wiki/Data-Stores

- Order with custom table is scheduled in Q1 2018. See the roadmap: https://trello.com/c/2QGrEbBW/113-an-order-database-that-can-scale-feature-plugin.

- If you develop WooCommerce extension make sure it's ready for this.

Thank you!