Service Integration Testing

17-423/723 Designing Large-Scale Software Systems

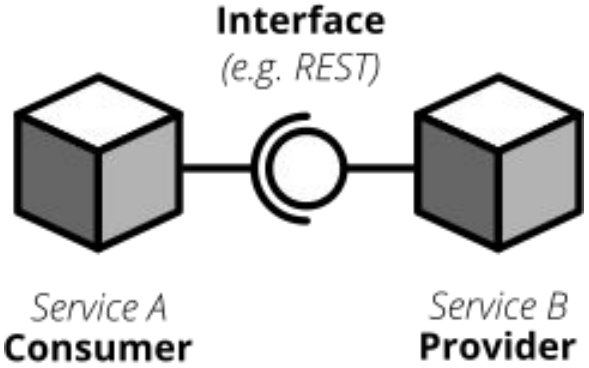
Recitation 6
Mar 14, 2025

Integration Testing

- Testing that multiple components/services work together as intended
- **Big-bang test**: Deploy & execute all services together under different test inputs & check the output
- **Q. Why is challenging to do in practice?**
  - Dependencies on external services
  - Debugging/localizing a buggy behavior to a single component
  - Generally, slow and expensive to set up
- Eventually, you will need some big-bang tests, to make sure that your system works under the production-like environment
- But can we also do something that’s less expensive & difficult?

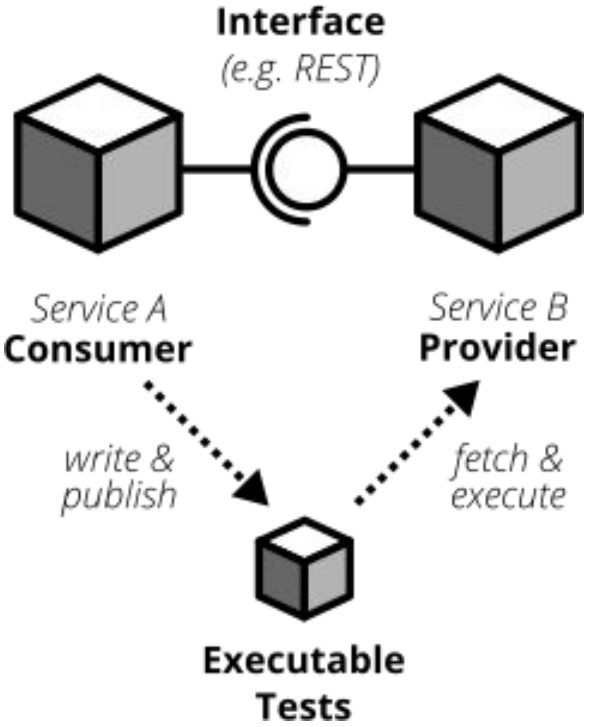
Contract Testing

- An incremental, service-by-service approach to integration testing
- **Provider:** Provides data to consumers
- **Consumer:** Processes data obtained from a provider
- **Consumer-driven Contract (CDC)**: Describes what the consumer expects from the provider as an output



Contract Testing: Steps

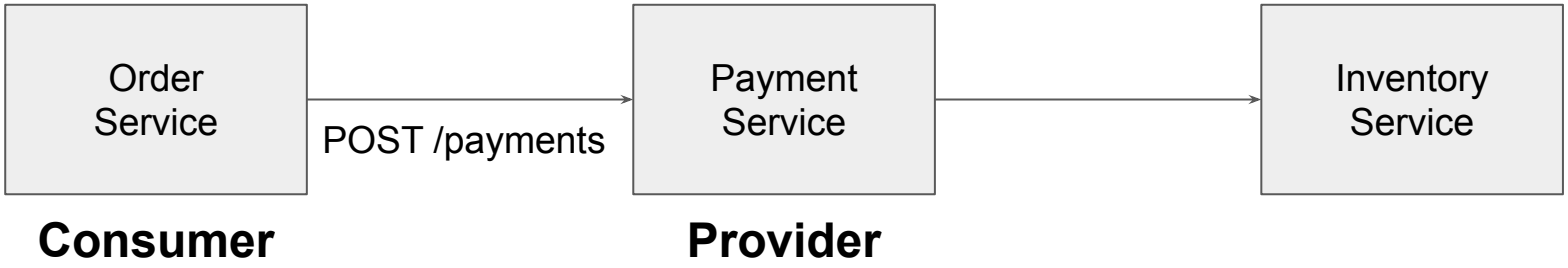
1. **Write a consumer unit test**: To do so, create a mock for the provider.
2. **Create a contract** (CDC) that describes (1) an input from the consumer to the provider and (2) the expected output.
3. **Run the consumer unit test** (with the mock).
4. **Publish the contracts:** Share all CDCs in a machine-readable form.
5. **Test the provider**: Run all CDCs against the provider.



Example: Online Store System

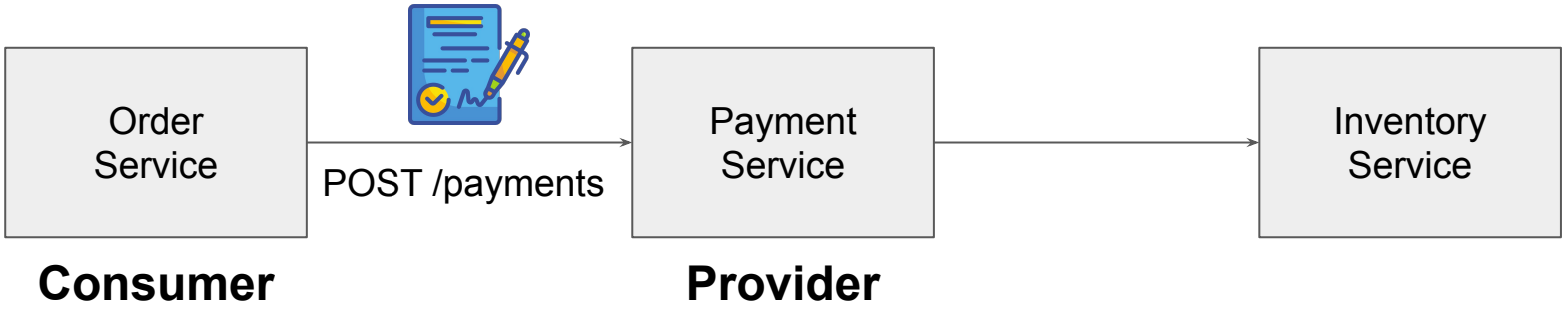
- **Order Service**: Handles orders from customers.
- **Payment Service**: Processes payments for orders.
- **Inventory Service**: Manage the stock levels for different items. Reserve items for a processed order.

Contract between Order & Payment Services



- **Consumer unit test**: Test a scenario where a customer order is successfully handled and finalized.
- The unit test includes (1) a mock of Payment Service and (2) a contract from Order to Payment Service

Contract between Order & Payment Services



Request

```
{
  "orderId": "ORD123",
  "amount": 100.50,
  "currency": "USD"
}
```

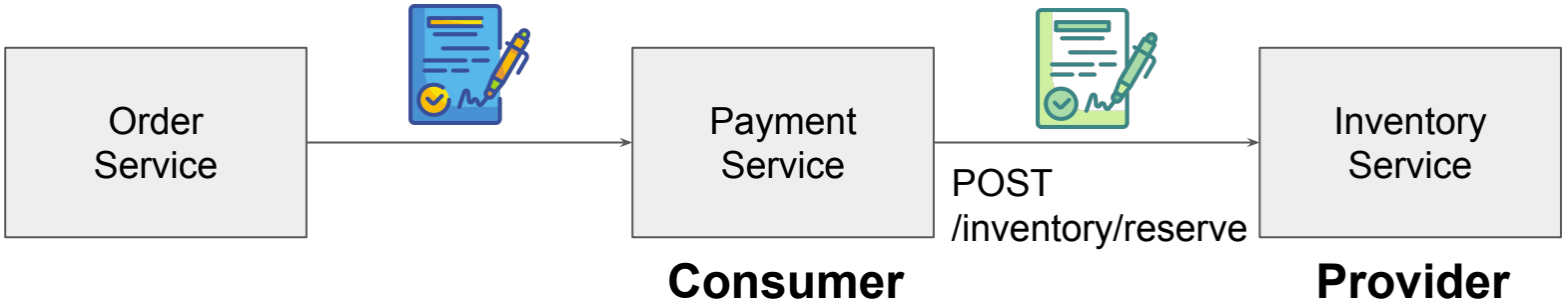
Expected response

```
{
  "paymentId": "PAY987",
  "status": "SUCCESS",
  "orderId": "ORD123"
}
```

Contract between Payment & Order Services

- This contract is stored and published, to be used as a test for the Payment Service
- Running this test requires another contract, this time from Payment to Inventory Service

Contract between Payment & Order Services



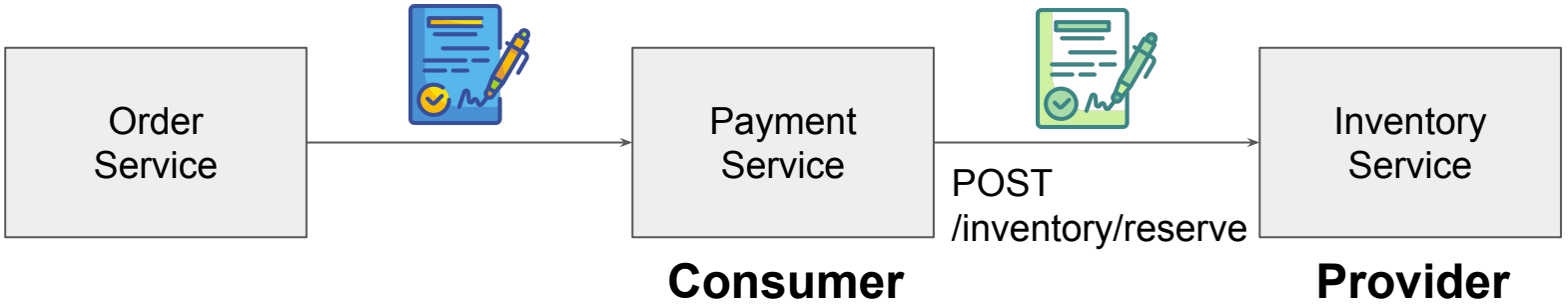
Request

```
{
  "orderId": "ORD123",
  "items": [
    { "productId": "P001", "quantity": 2 },
    { "productId": "P002", "quantity": 1 }
  ]
}
```

Expected response

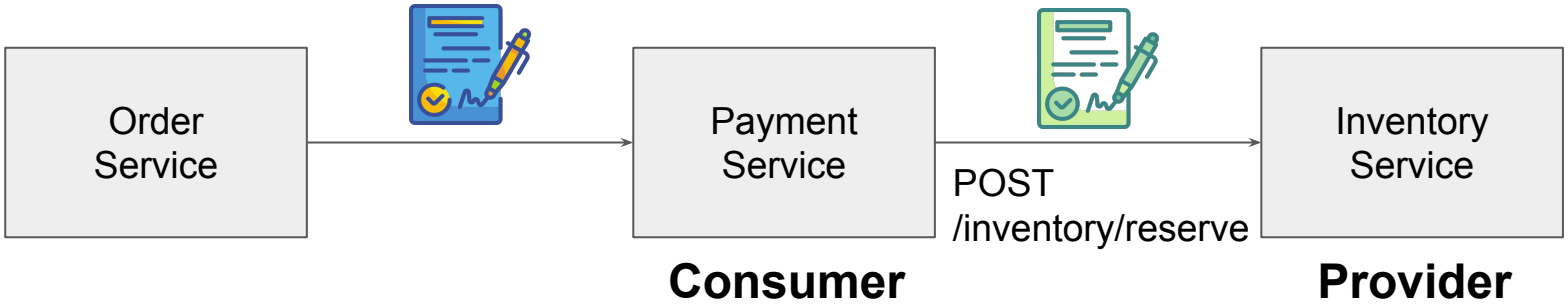
```
{
  "orderId": "ORD123",
  "status": "RESERVED"
}
```

Contract between Payment & Order Services



- This second contract is also stored and published, to be used as a test for the Inventory Service
- This whole process can be repeated for other tests for Order Service, resulting in additional contracts

Contract between Payment & Order Services



Request

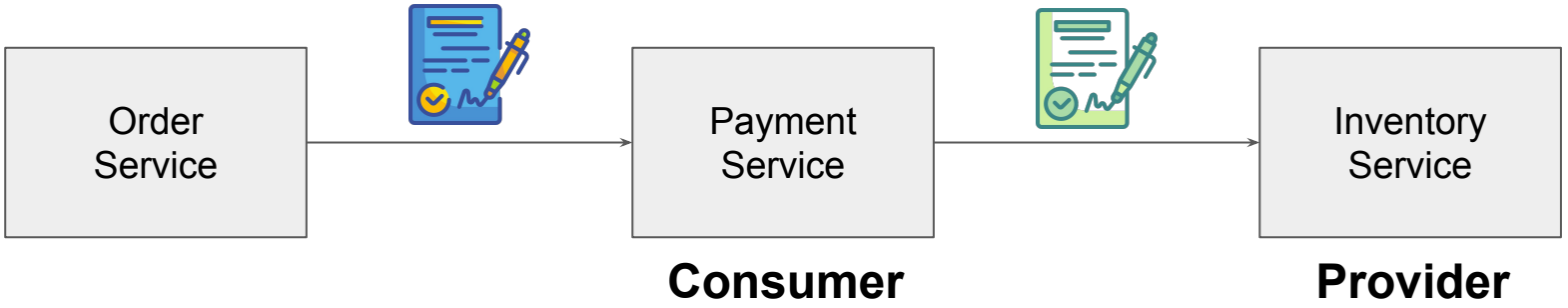
```
{
  "orderId": "ORD123",
  "items": [
    { "productId": "P001", "quantity": 2 },
    { "productId": "P002", "quantity": 1 }
  ]
}
```

Expected response

```
{
  "orderId": "ORD123",
  "status": "RESERVED"
}
```

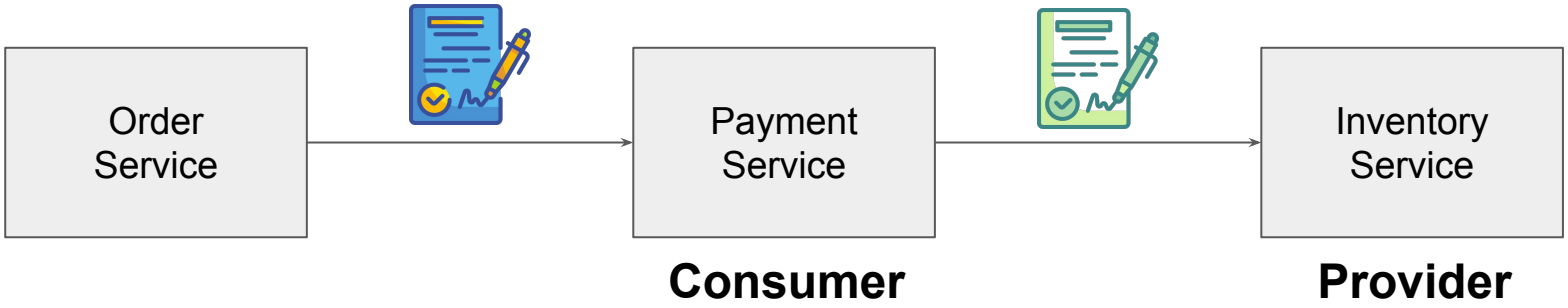
Note: Running this test requires Inventory Service to have sufficient stock for the items!

Contract with Precondition over Provider State



- In general, a contract may depend on the provider being under a particular state
- Such a contract must also specify **a precondition over the provider state**
  - e.g., “Inventory Service has at least 2 items for P001”

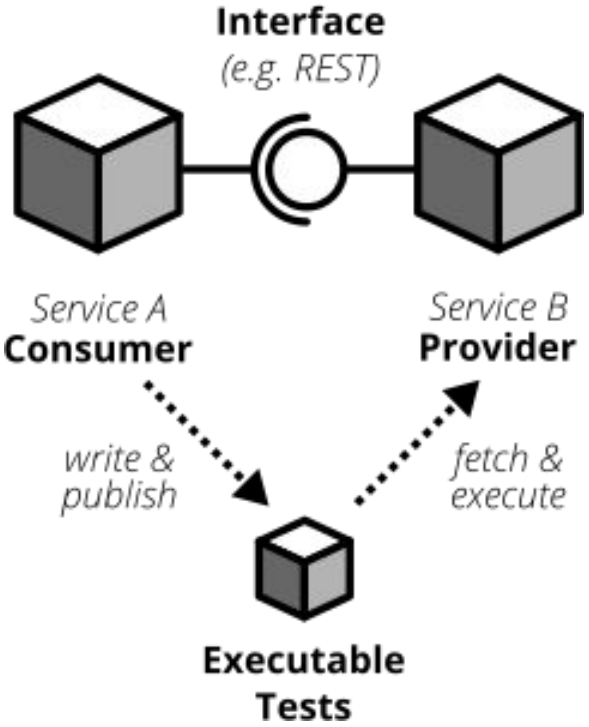
Contract with Precondition over Provider State



- In general, a contract may depend on the provider being under a particular state
- Such a contract must also specify **a precondition over the provider state**
- When testing the provider, write a setup code to bring the provider into a state that satisfies this precondition

Contract Testing: Benefits

- Allows services to be tested without having to run all of them
- When a provider changes, contracts can be used as regression tests, to detect whether the change affects its consumers
- **Q. How are contracts here different from interface specifications?**



Activity: Write Contracts for Project Services

- Link to the shared Google Doc
- Work with your team members
- For your scheduling application or the service that you are developing for M3/M4:
  - Develop a unit test to test behavior that involves a dependency on an external service
  - Write a contract for the external service, including (1) consumer input, (2) expected provider output, and (3) precondition on the provider state (if necessary)
  - Share the contract with the team for the provider service

Contract Testing Tools

- Pact: An open-source framework for contract testing
- Automates the process of creating, storing, and publishing contracts (Pact Broker)
- But contract testing can be done without a tool!
  - Document & share contracts with other teams!