Cornell University
Computing and Information Science

CS 5150 Software Engineering

13. Three Popular Architectural Styles

William Y. Arms

Example 1: Batch Processing with Master File Update

Examples

- Electricity utility customer billing (e.g., NYSEG)
- Telephone call recording and billing (e.g., Verizon)
- Car rental reservations (e.g., Hertz)
- Bank (e.g., Tompkins Trust)
- University grade registration (e.g., Cornell)

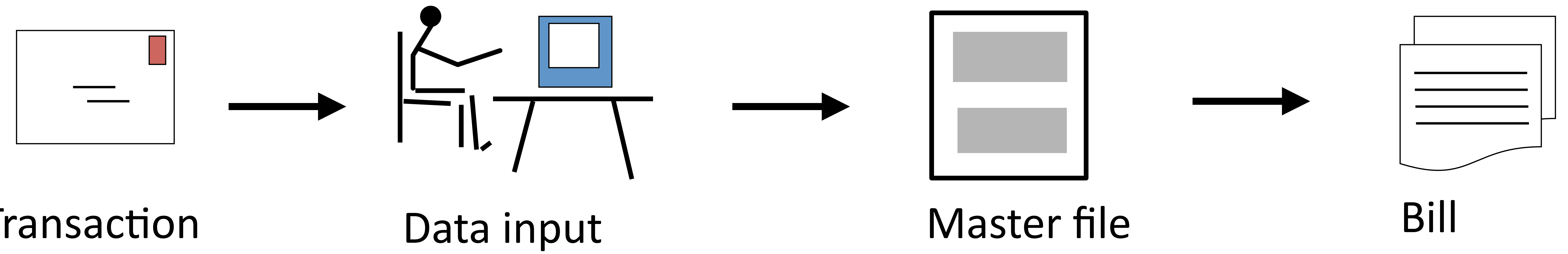
Master File Update

Example: Electricity Utility Billing

Requirements analysis identifies several transaction types:

- Create account / close account
- Meter reading
- Payment received
- Other credits / debits
- Check cleared / check bounced
- Account query
- Correction of error
- etc., etc., etc.,

First Attempt



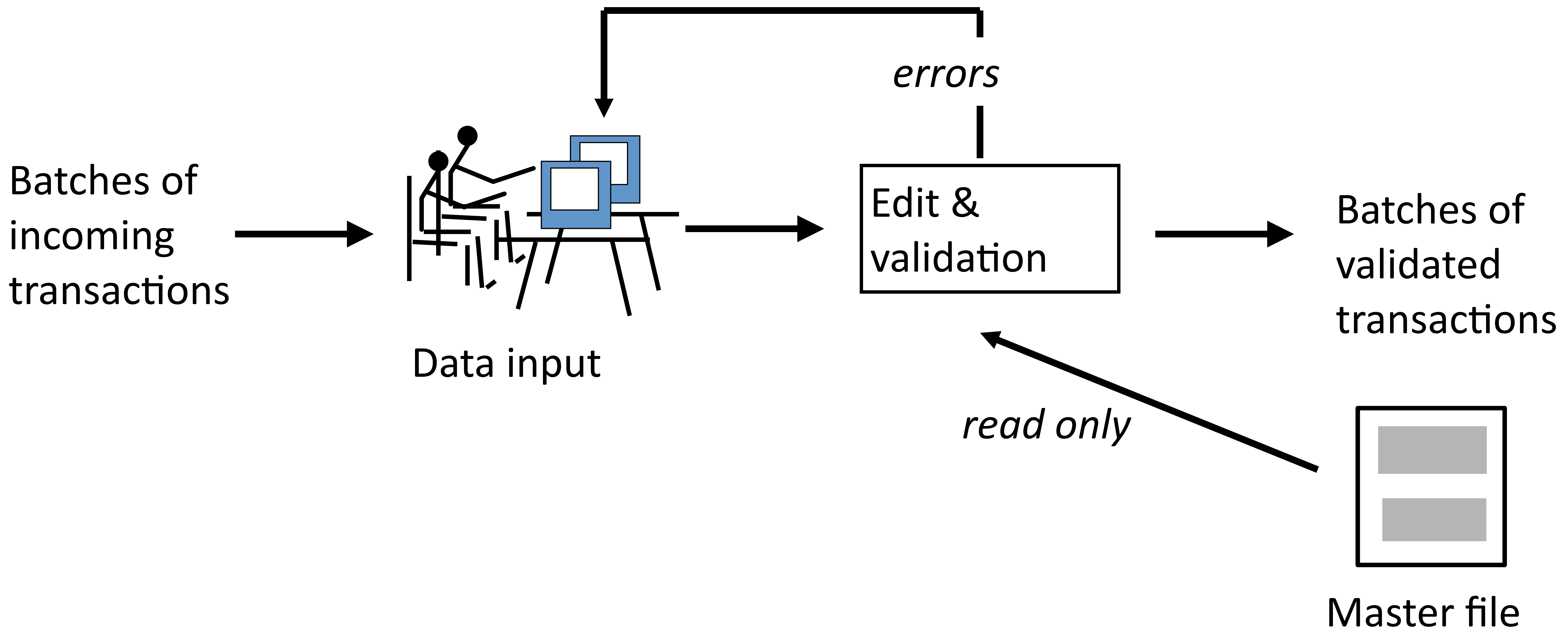
Each transaction is handled as it arrives.

Criticisms of First Attempt

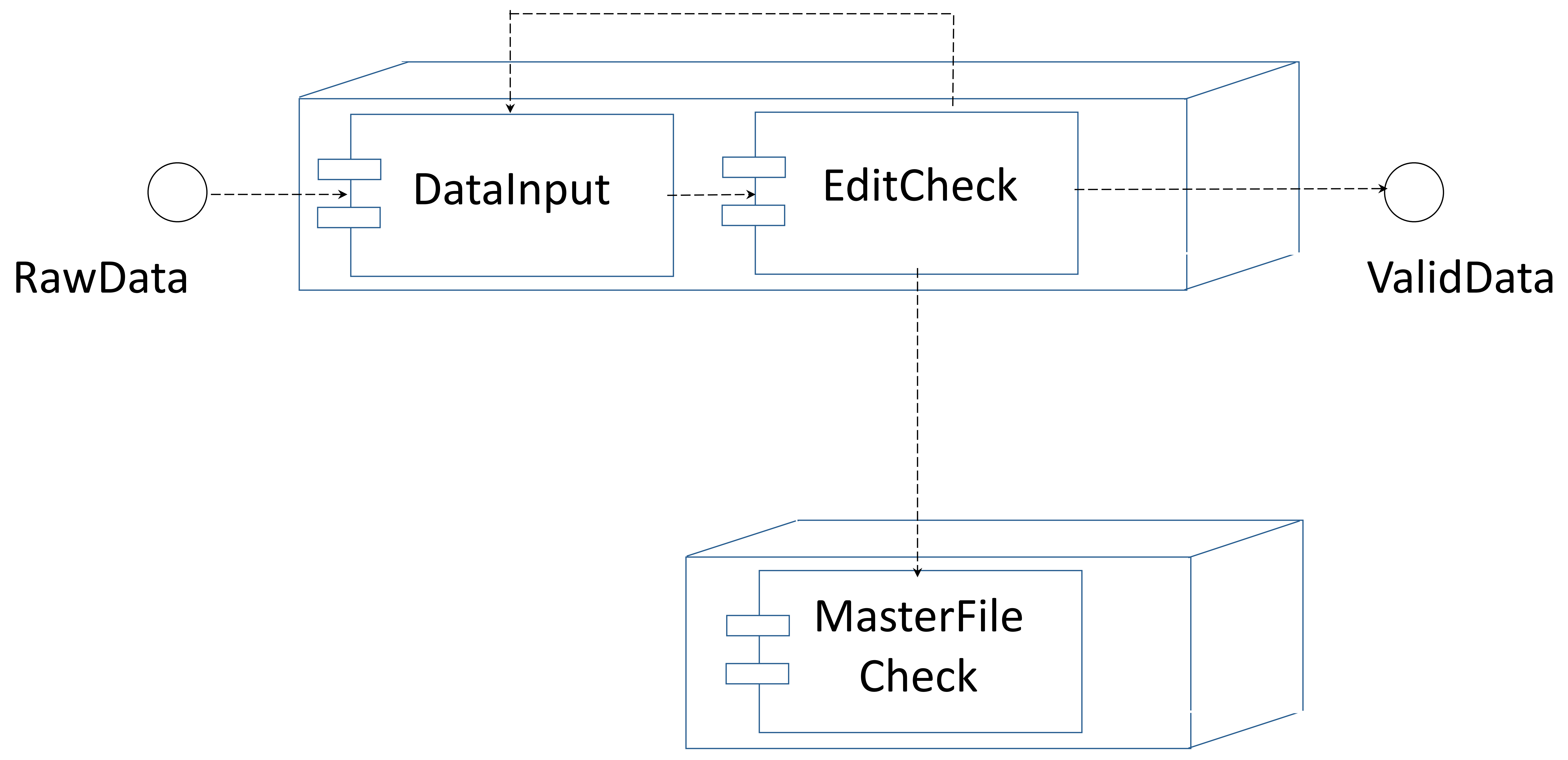
Where is this first attempt weak?

- All activities are triggered by a transaction.
- A bill is sent out for each transaction, even if there are several per day.
- Bills are not sent out on a monthly cycle.
- Awkward to answer **customer queries.**
- No process for **error checking** and **correction.**
- Inefficient in staff time.

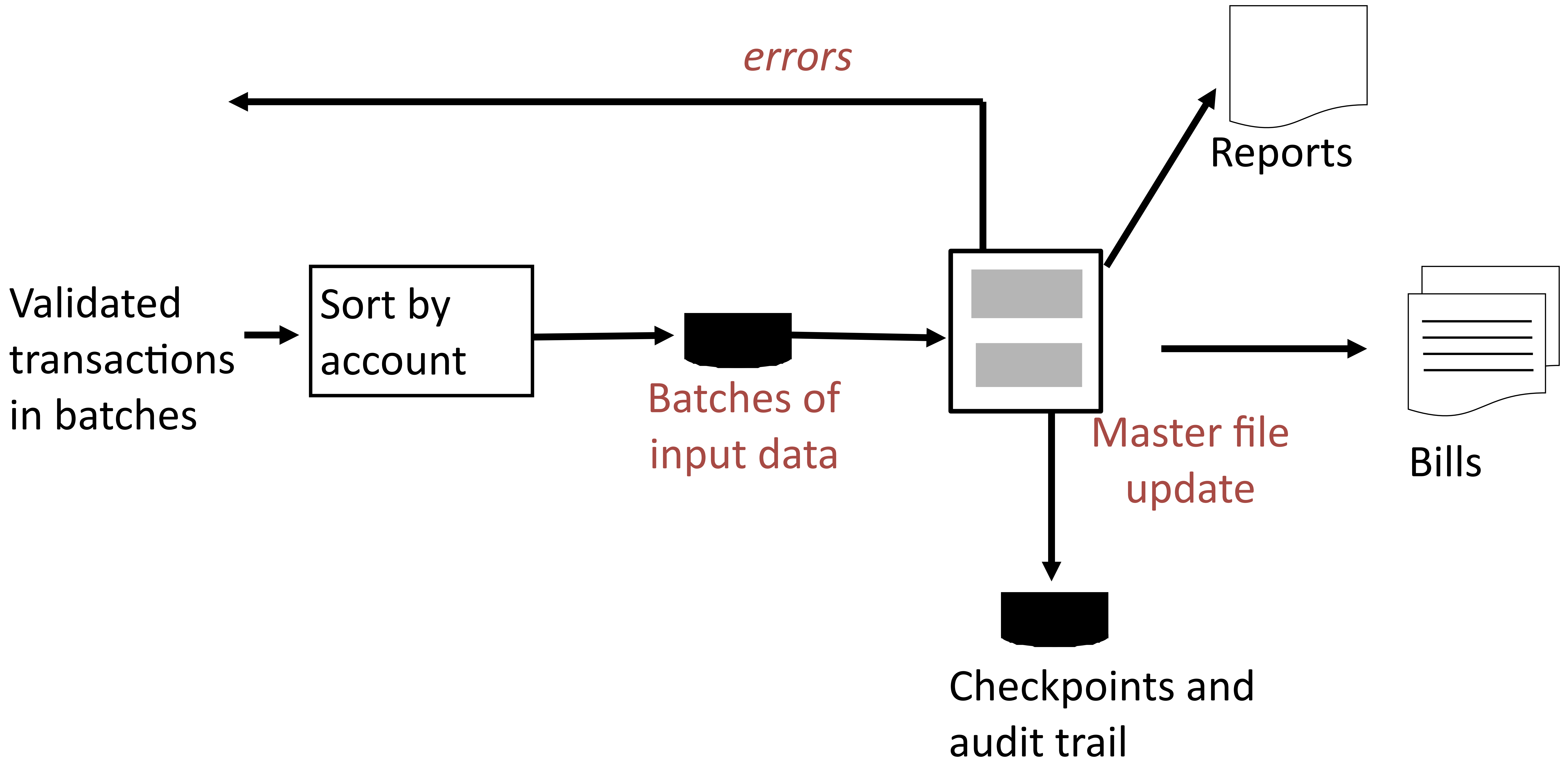
Batch Processing: Edit and Validation


Batches of incoming transactions
errors
Edit & validation
Batches of validated transactions
Data input
read only
Master file

Deployment Diagram: Validation



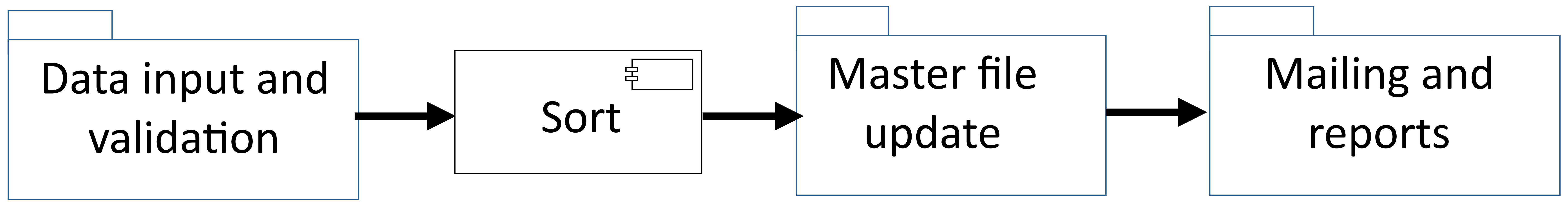
Batch Processing: Master File Update



Benefits of Batch Processing with Master File Update

- All transactions for an account are processed together at appropriate intervals, e.g., monthly.
- Backup and recovery have fixed checkpoints.
- Better management control of operations.
- Efficient use of staff and hardware.
- Error detection and correction is simplified.

Architectural Style: Master File Update (Basic Version)



Advantages:

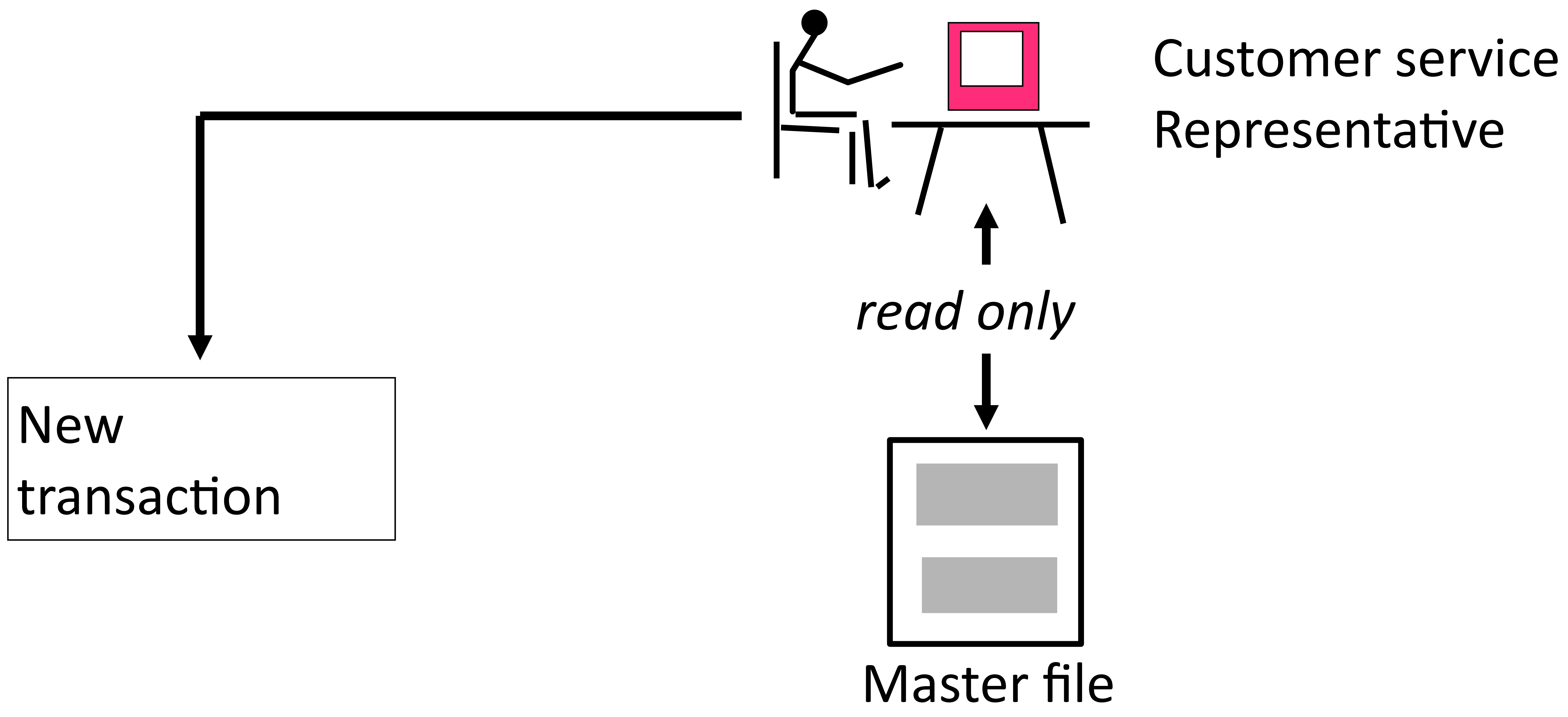
Efficient way to process batches of transactions.

Disadvantages:

Information in master file is not updated immediately. No good way to answer customer inquiries.

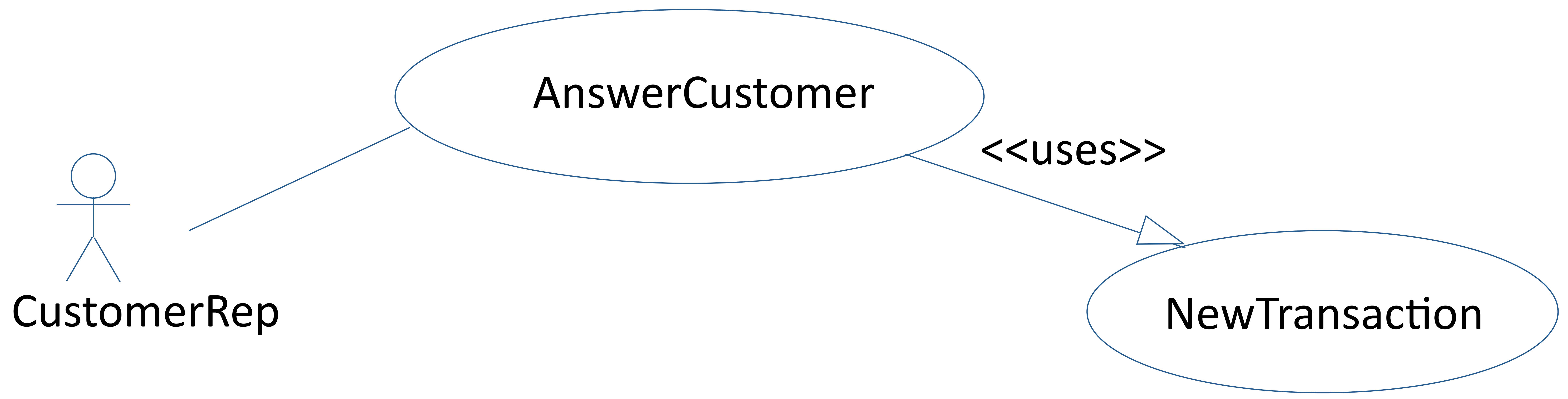
Online Inquiry

A customer calls the utility and speaks to a customer service representative.



Customer service department can read the master file, make annotations, and create transactions, but cannot change the master file.

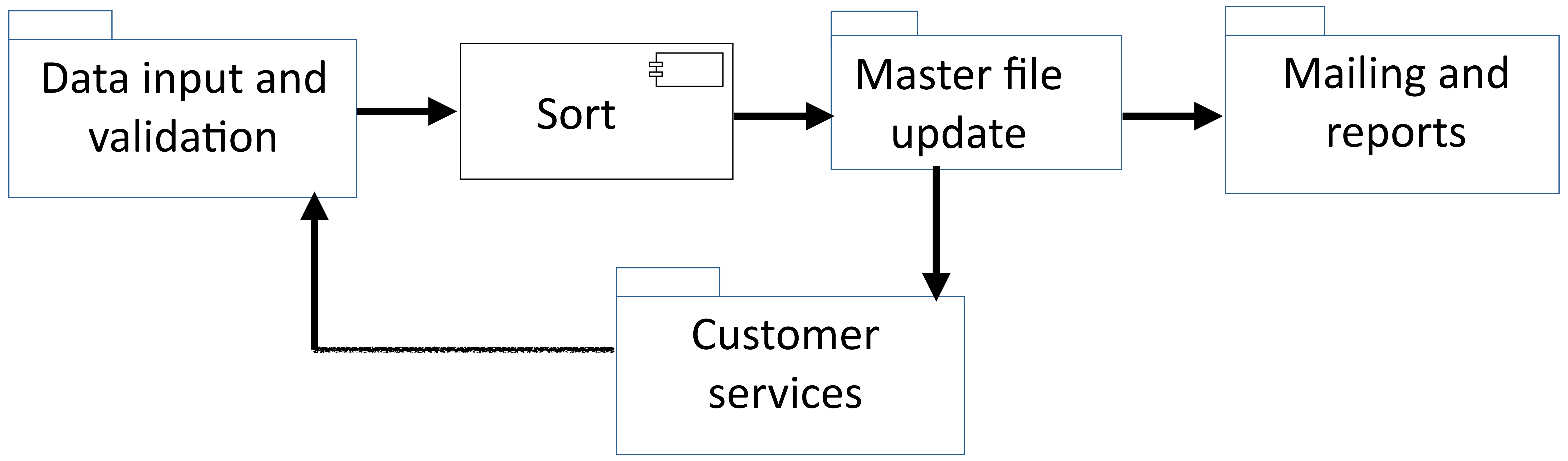
Online Inquiry: Use Case



The representative can read the master file, but not make changes to it.

If the representative wishes to change information in the master file, a new transaction is created as input to the master file update system.

Architectural Style: Master File Update (Full)



Advantage:

Efficient way to answer customer inquiries.

Disadvantage:

Information in master file is not updated immediately.

Example 2: Three Tier Architecture

The basic client/server architecture of the web has:

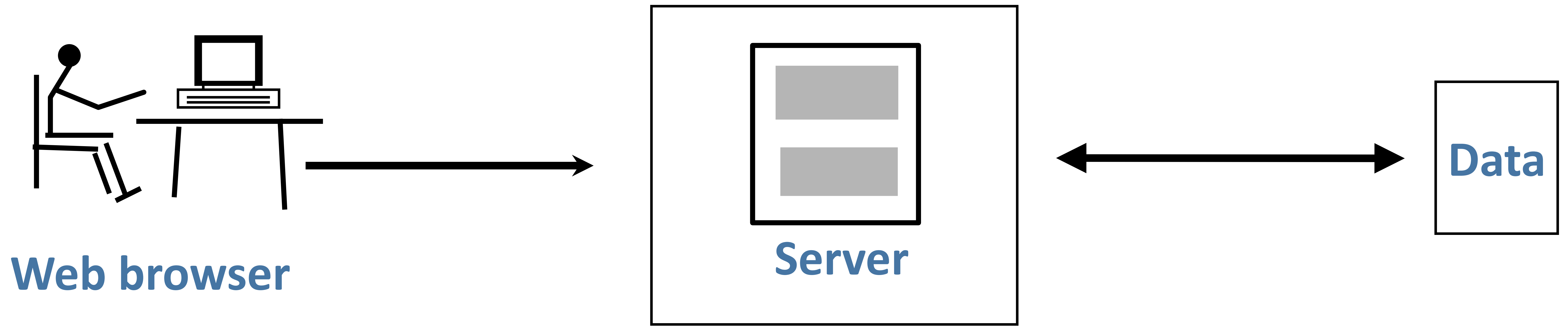
- a server that delivers static pages in HTML format
- a client (known as a browser) that renders HTML pages

Both client and server implement the HTTP interface.

Problem

Extend the architecture of the server so that it can configure HTML pages dynamically.

Web Server with Data Store



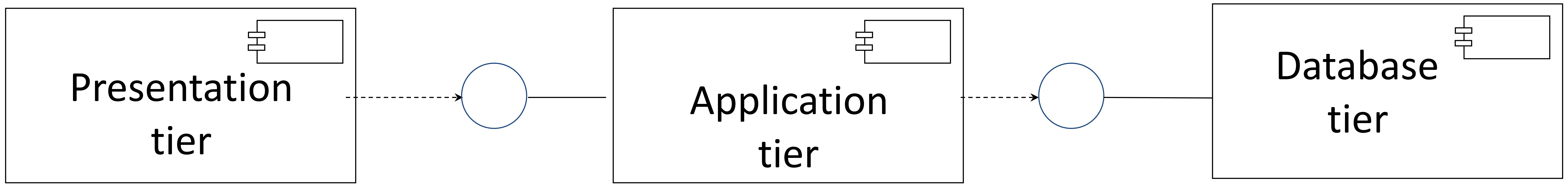
Advantage:

Server-side code can configure pages, access data, validate information, etc.

Disadvantage:

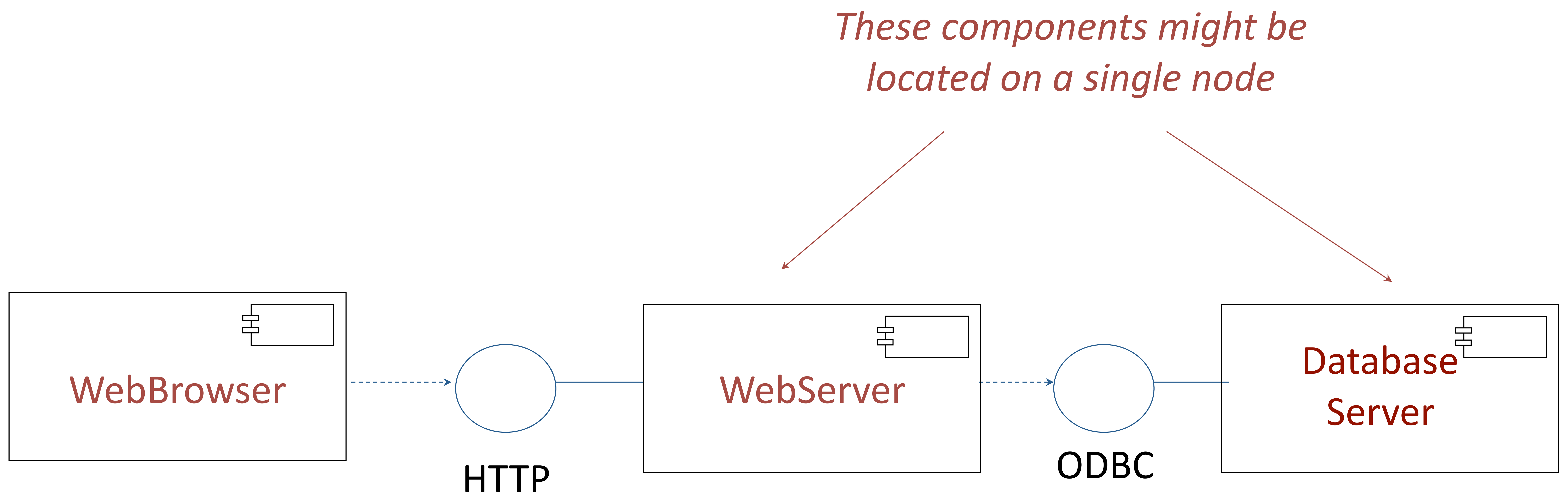
All interaction requires communication with server

Architectural Style: Three Tier Architecture

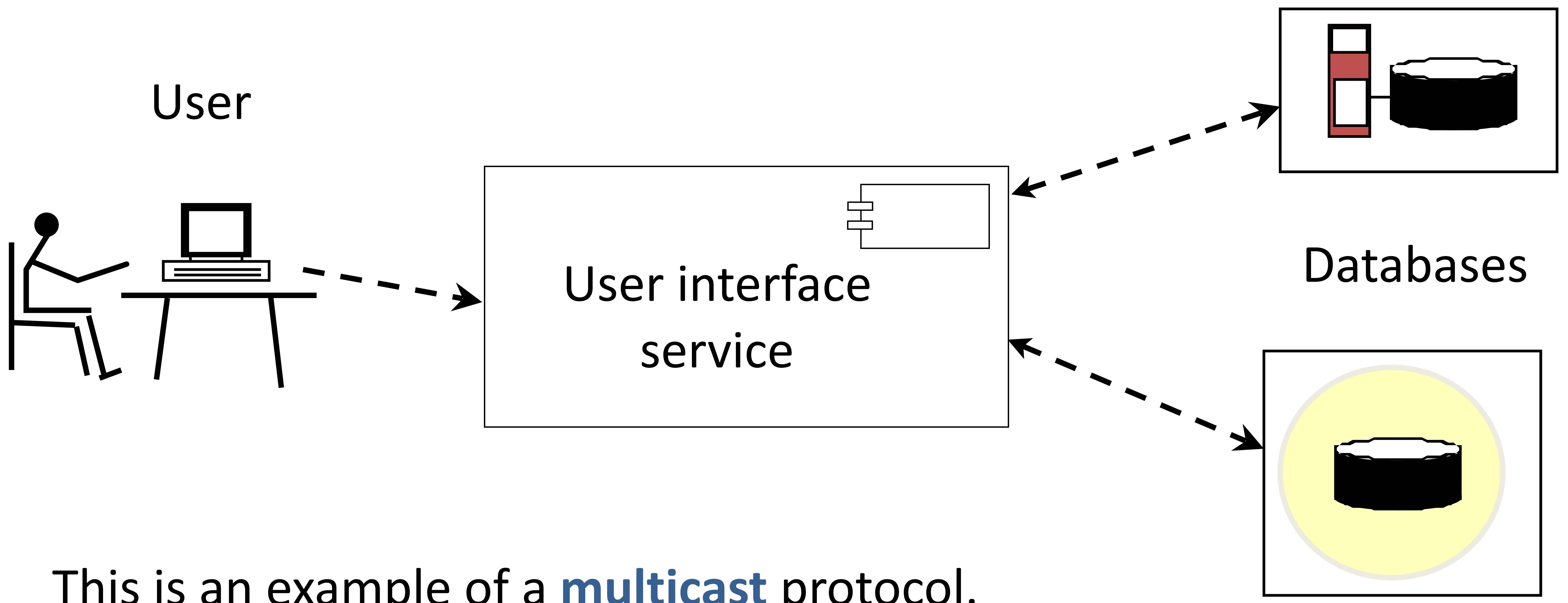


Each of the tiers can be replaced by other components that implement the same interfaces

Component Diagram



Three Tier Architecture: Broadcast Searching



This is an example of a **multicast** protocol.

The primary difficulty is to avoid troubles at one site degrading the entire system (e.g., every transaction cannot wait for a system to time out).

Extending the Architecture of the Web

Using a three tier architecture, the web has:

- a server that delivers dynamic pages in HTML format
- a client (known as a browser) that renders HTML pages

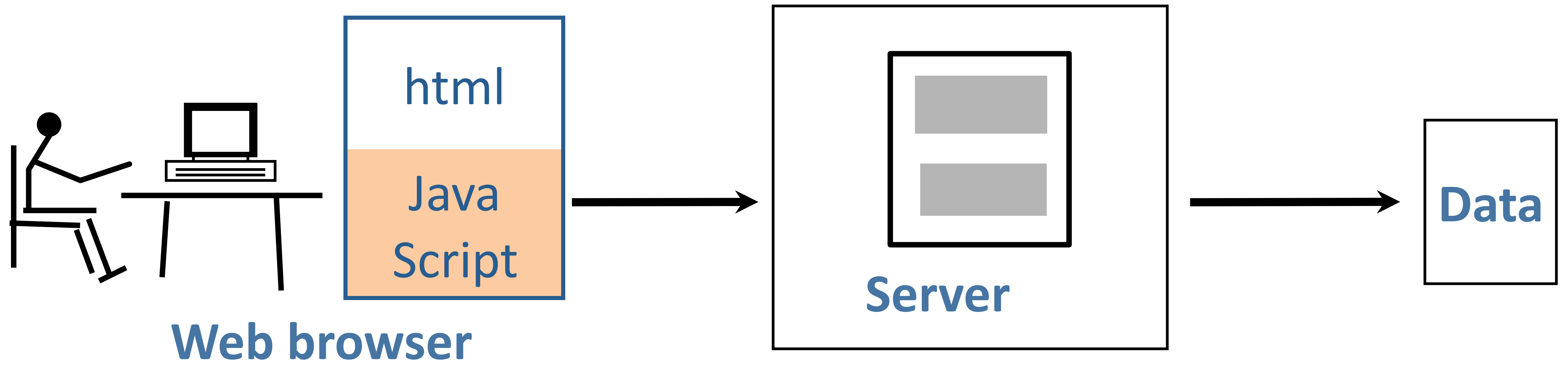
Both server and client implement the HTTP interface.

Every interaction with the user requires communication between the client and the server.

Problem 2

Extend the architecture so that simple user interactions do not need messages to be passed between the client and the server.

Extending the Web with Executable Code that can be Downloaded



Executable code in a scripting language such as JavaScript can be downloaded from the server

Advantage:

Scripts can interact with user and process information locally

Disadvantage:

All interactions are constrained by web protocols

Extending the Three Tier Architecture

In the three tier architecture, a web site has:

- a client that renders HTML pages and executes scripts
- a server that delivers dynamic pages in HTML format
- a data store

Further extensions

The three tier architecture with downloadable scripts is one the ways in which the basic architecture has been extended. There are some more:

- Protocols: e.g., HTTPS, FTP, proxies
- Data types: e.g., helper applications, plug-ins
- Executable code: e.g., applets, servlets
- Style sheets: e.g., CSS

Example 3: Model/View/Controller (MVC)

The definition of Model/View/Controller (MVC) is in a state of flux. The term is used to describe a range of architectures and designs.

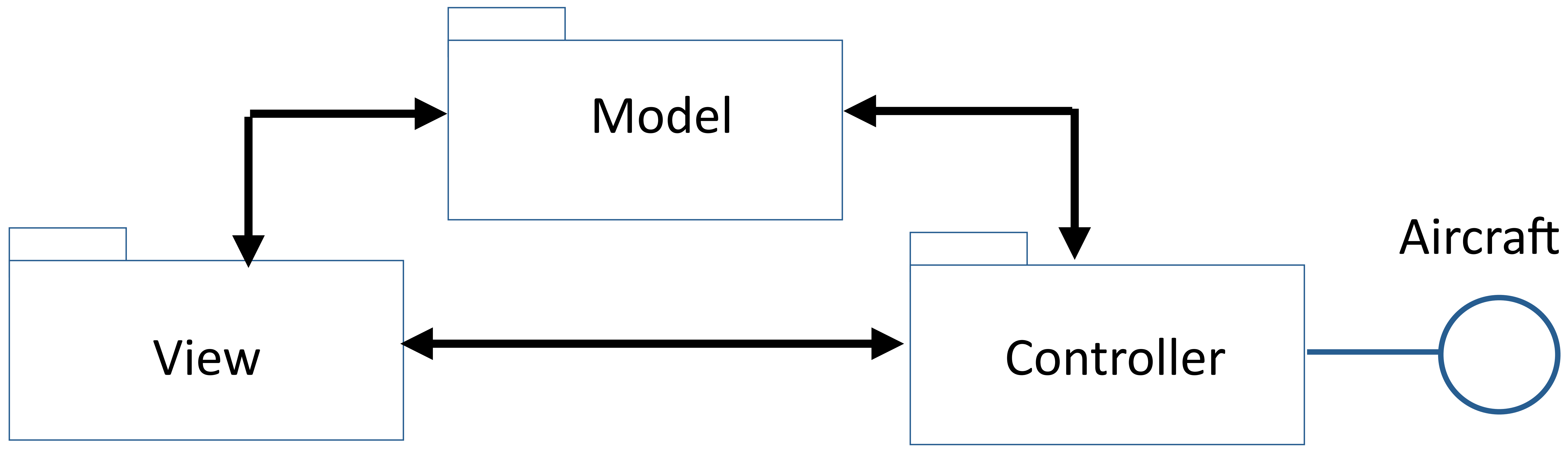
- Some are system architectures, where the model, view, and controller are separate components.
- Some are program designs, with classes called model, view, and controller.

We will look at three variants:

- An MVC system architecture used in robotics.
- A general purpose MVC system architecture used for interactive systems.
- Apple's version of MVC as a program design for mobile apps.

Model/View/Controller in Robotics

Example: Control of a unmanned model aircraft

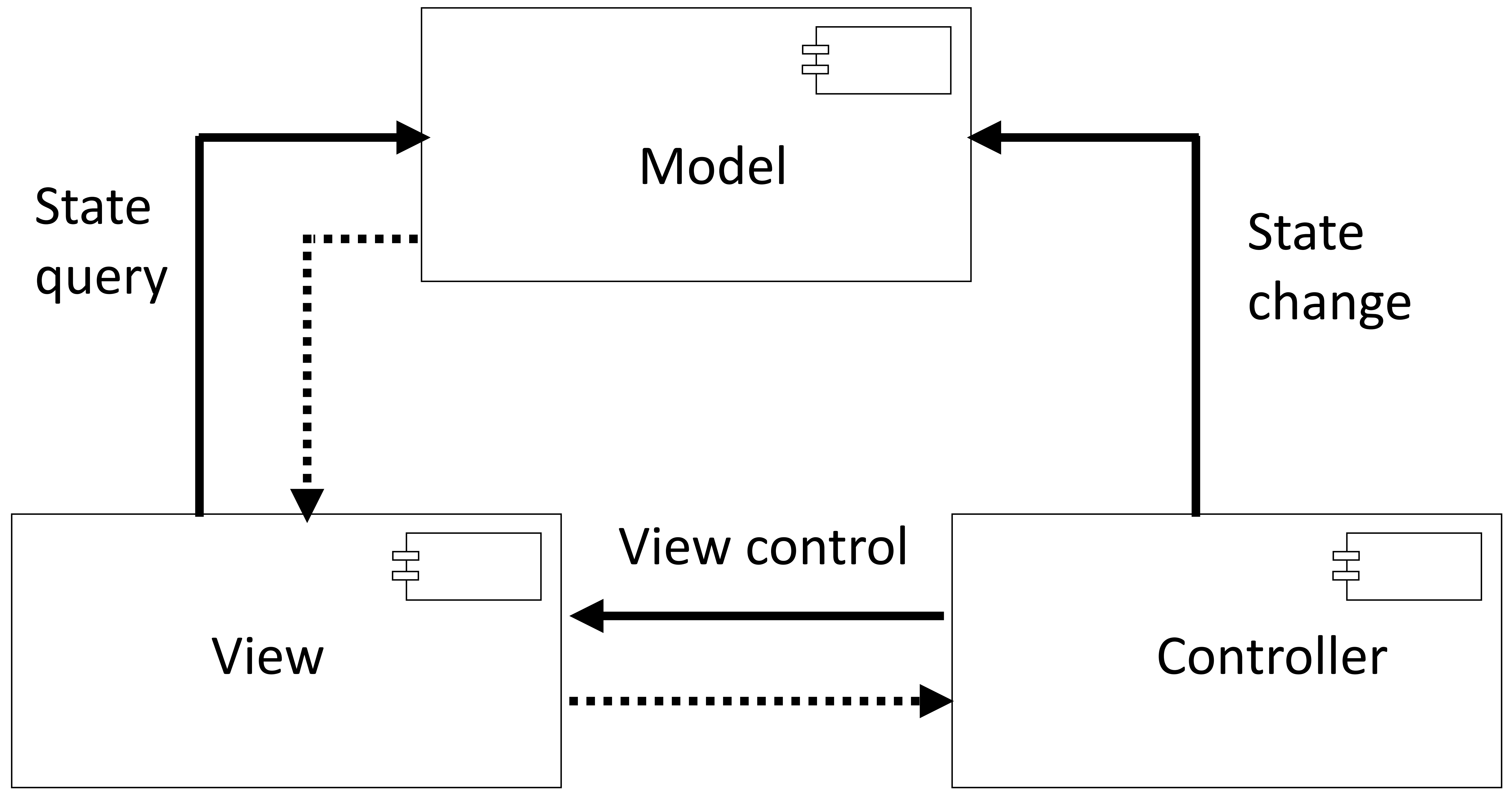


Controller: Receives instrument readings from the aircraft, updates the view, and sends controls signals to the aircraft.

Model: Translates data received from and sent to the aircraft, and instructions from the user into a model of flight performance. Uses domain knowledge about the aircraft and flight.

View: Displays information about the aircraft to the user on the ground and transmits instructions to the model via the controller.

Example 3. Model/View/Controller for Mobile Apps



Model

The model records the state of the application and notifies subscribers. It does not depend on the controller or the view.

- stores the state of the application in suitable data structures or databases
- responds to instructions to change the state information
- notifies subscribers of events that change the state
- may be responsible for validation of information

View

The view is the part of the user interface that presents the state of the interface to the user. It subscribes to the model, which notifies it of events that change the state.

- renders data from the model for the user interface
- provides editors for properties, such as text fields, etc.
- receives updates from the model
- sends user input to the controller

A given model may support a choice of alternative views.

Controller

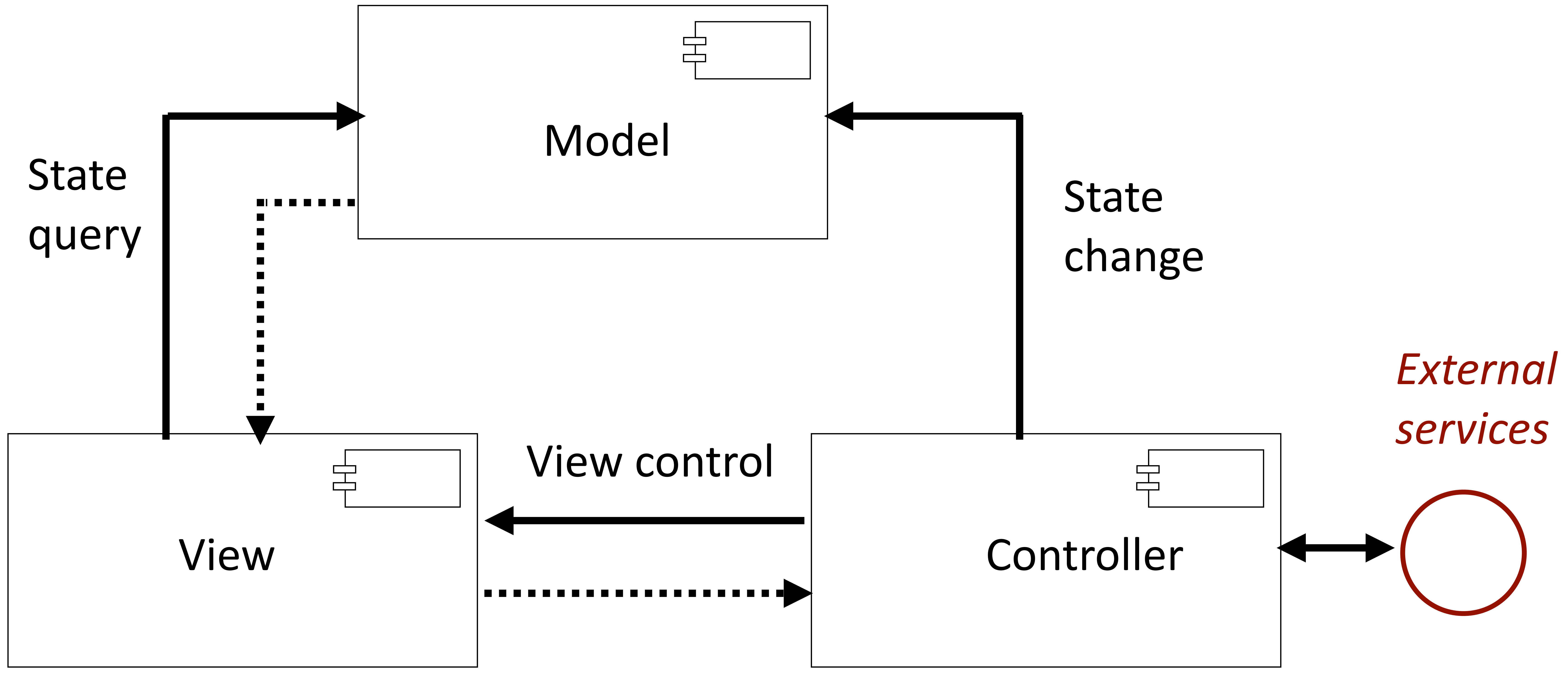
The controller is the part of the user interface that manages user input and navigation within the application.

- defines the application behavior
- maps user actions to changes in the state of the model
- interacts with external services via APIs
- may be responsible for validation of information

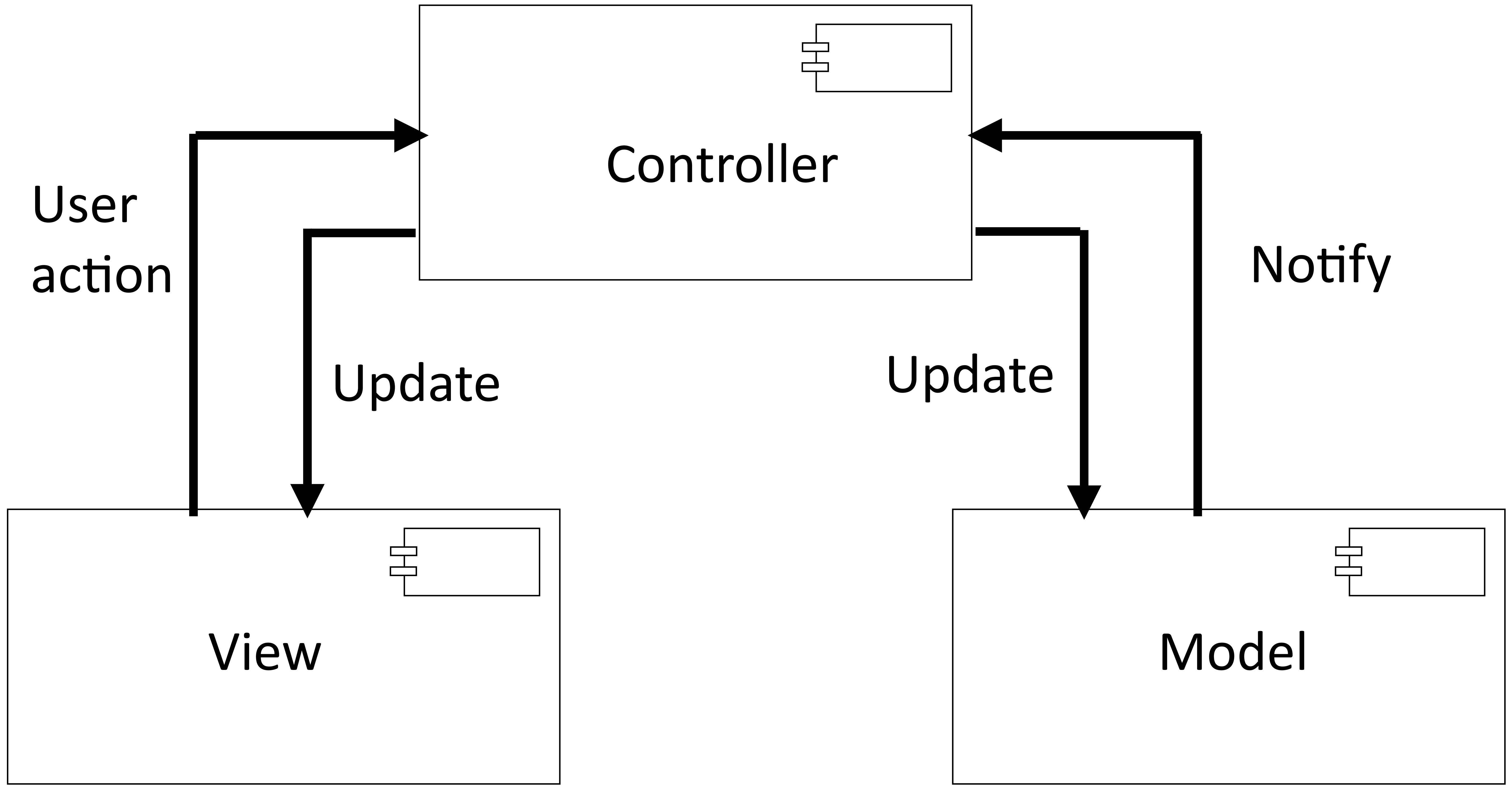
Different frameworks handle controllers in different ways. In particular there are several ways to divide responsibilities between the model and the controller, e.g., data validation, external APIs.

External Services for Mobile Apps

Mobile apps often make extensive use of cloud-based external services, each with an API (e.g., location, validation). These are usually managed by the controller.



Apple’s Version of Model/View/Controller



The diagram shows the model, view, and controller as components. In practice the MVC is a program design with three major classes.

Apple’s Version of MVC

Two challenges:

- A multi-screen app will have several views and controllers sharing the same model.
- It is easy to put too much code into the controller.

Architectural Styles and Design Patterns

There are many variants of the common architectural styles. Do not be surprised if you encounter a variant that is different from the one described in this course.

This is particularly true with the Model-View-Controller style. Several programming frameworks call classes that implement a variant of the Model-View-Controller architectural style a design pattern.

In this course we distinguish carefully between architectural styles and design patterns.

> Architectural styles are part of system design. They are defined in terms of subsystems, components, and deployment.
>
> Design patterns are part of program design. They are defined in terms of classes.