BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Module 7

Patterns – Part 3

Harvinder S Jabbal
SSZG653 Software Architectures

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Client-Server Pattern

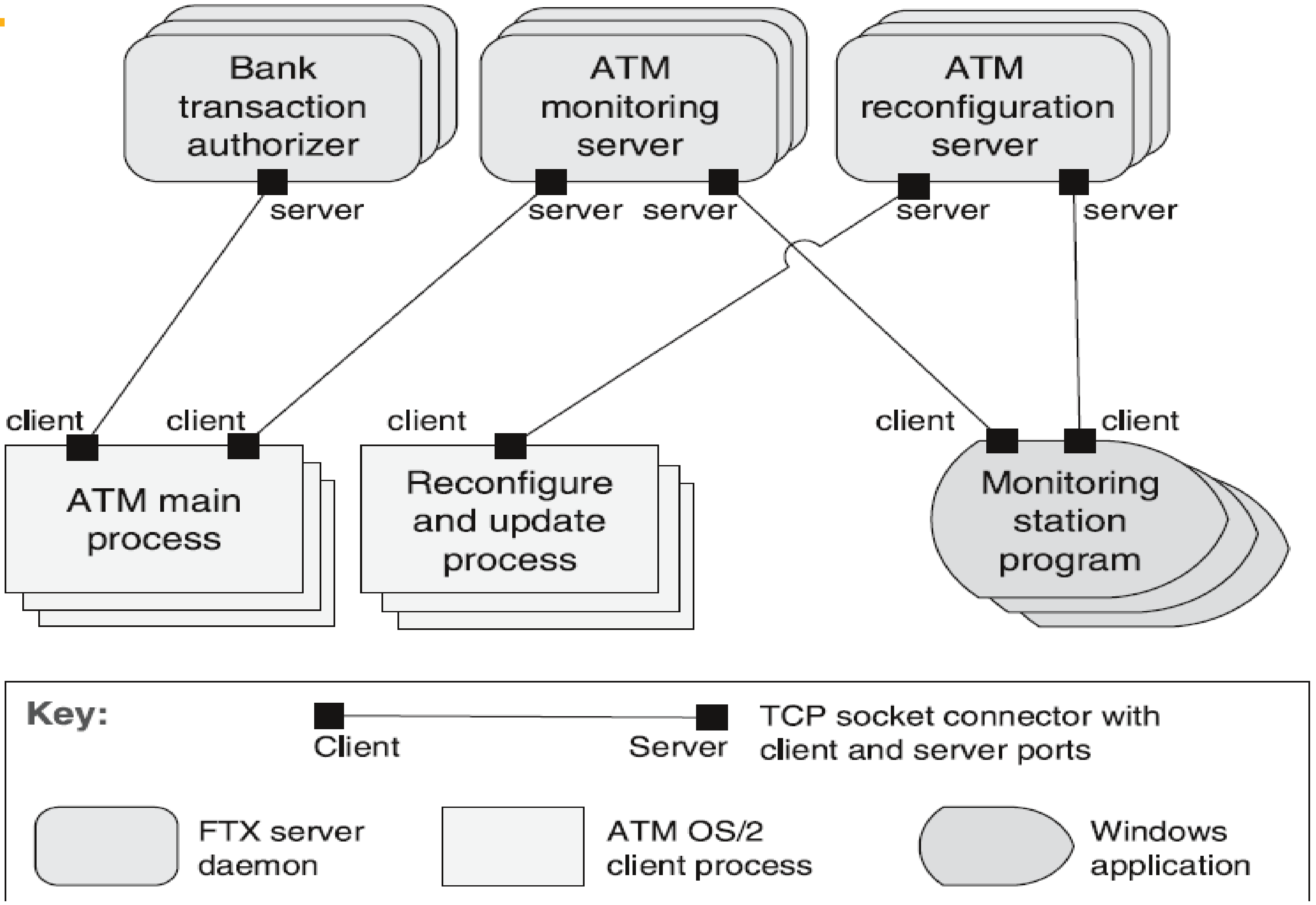
Client-Server Pattern

Context: There are shared resources and services that large numbers of distributed clients wish to access, and for which we wish to control access or quality of service.

Problem: By managing a set of shared resources and services, we can promote modifiability and reuse, by factoring out common services and having to modify these in a single location, or a small number of locations. We want to improve scalability and availability by centralizing the control of these resources and services, while distributing the resources themselves across multiple physical servers.

Solution: Clients interact by requesting services of servers, which provide a set of services. Some components may act as both clients and servers. There may be one central server or multiple distributed ones.

Client-Server Example



Client-Server Solution - 1

Overview: Clients initiate interactions with servers, invoking services as needed from those servers and waiting for the results of those requests.

Elements:

- *Client,* a component that invokes services of a server component. Clients have ports that describe the services they require.
- *Server:* a component that provides services to clients. Servers have ports that describe the services they provide.

Request/reply connector: a data connector employing a request/reply protocol, used by a client to invoke services on a server. Important characteristics include whether the calls are local or remote, and whether data is encrypted.

Client-Server Solution- 2

Relations: The *attachment* relation associates clients with servers.

Constraints:

- Clients are connected to servers through request/reply connectors.
- Server components can be clients to other servers.

Weaknesses:

- Server can be a performance bottleneck.
- Server can be a single point of failure.
- Decisions about where to locate functionality (in the client or in the server) are often complex and costly to change after a system has been built.

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Peer-to-Peer Pattern

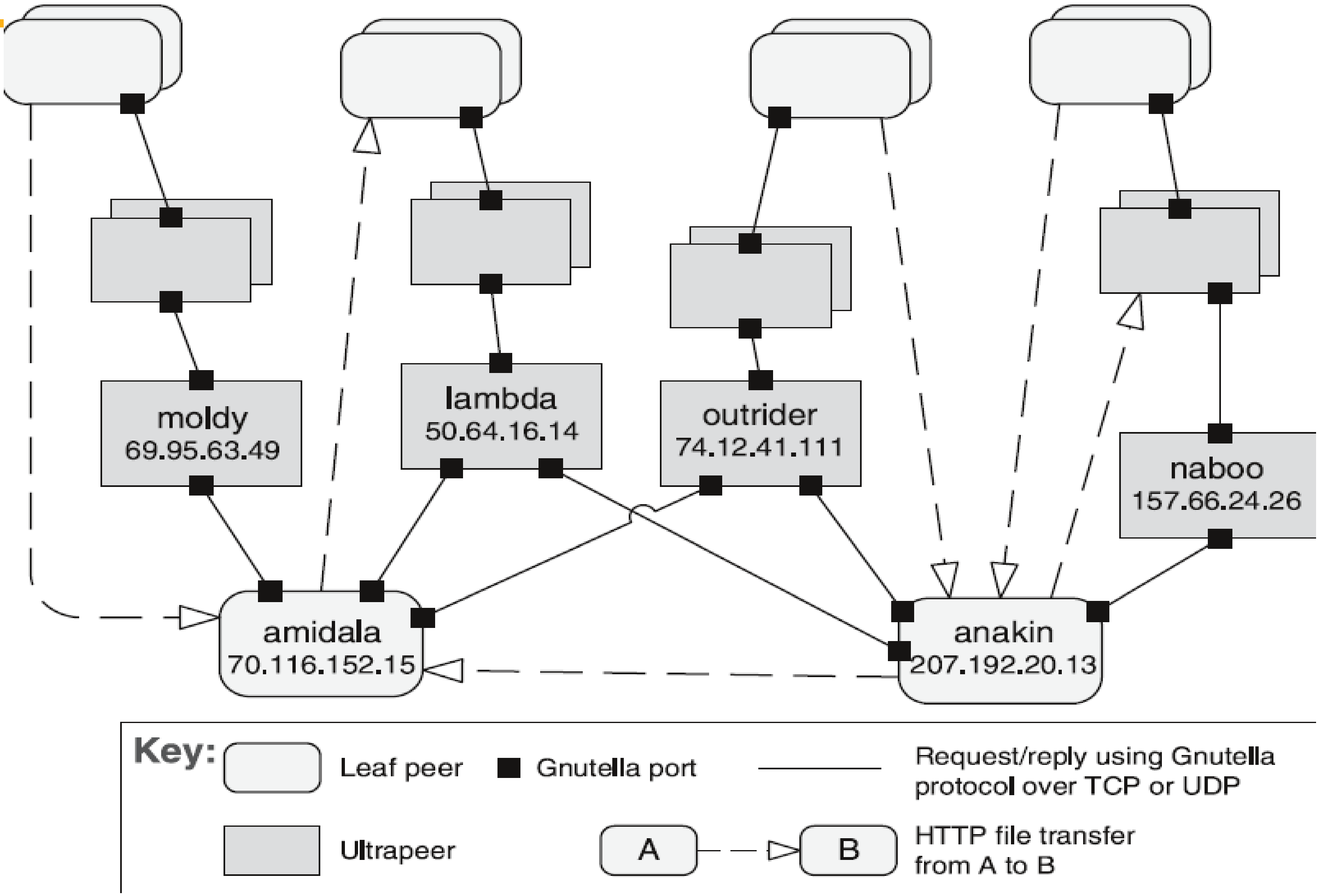
Peer-to-Peer Pattern

Context: Distributed computational entities—each of which is considered equally important in terms of initiating an interaction and each of which provides its own resources—need to cooperate and collaborate to provide a service to a distributed community of users.

Problem: How can a set of “equal” distributed computational entities be connected to each other via a common protocol so that they can organize and share their services with high availability and scalability?

Solution: In the peer-to-peer (P2P) pattern, components directly interact as peers. All peers are “equal” and no peer or group of peers can be critical for the health of the system. Peer-to-peer communication is typically a request/reply interaction without the asymmetry found in the client-server pattern.

Peer-to-Peer Example



Peer-to-Peer Solution - 1

Overview: Computation is achieved by cooperating peers that request service from and provide services to one another across a network.

Elements:

- *Peer,* which is an independent component running on a network node. Special peer components can provide routing, indexing, and peer search capability.
- *Request/reply connector,* which is used to connect to the peer network, search for other peers, and invoke services from other peers. In some cases, the need for a reply is done away with.

Relations: The relation associates peers with their connectors. Attachments may change at runtime.

Peer-to-Peer Solution - 2

Constraints: Restrictions may be placed on the following:

- The number of allowable attachments to any given peer
- The number of hops used for searching for a peer
- Which peers know about which other peers
- Some P2P networks are organized with star topologies, in which peers only connect to supernodes.

Weaknesses:

- Managing security, data consistency, data/service availability, backup, and recovery are all more complex.
- Small peer-to-peer systems may not be able to consistently achieve quality goals such as performance and availability.

Publish-Subscribe Pattern

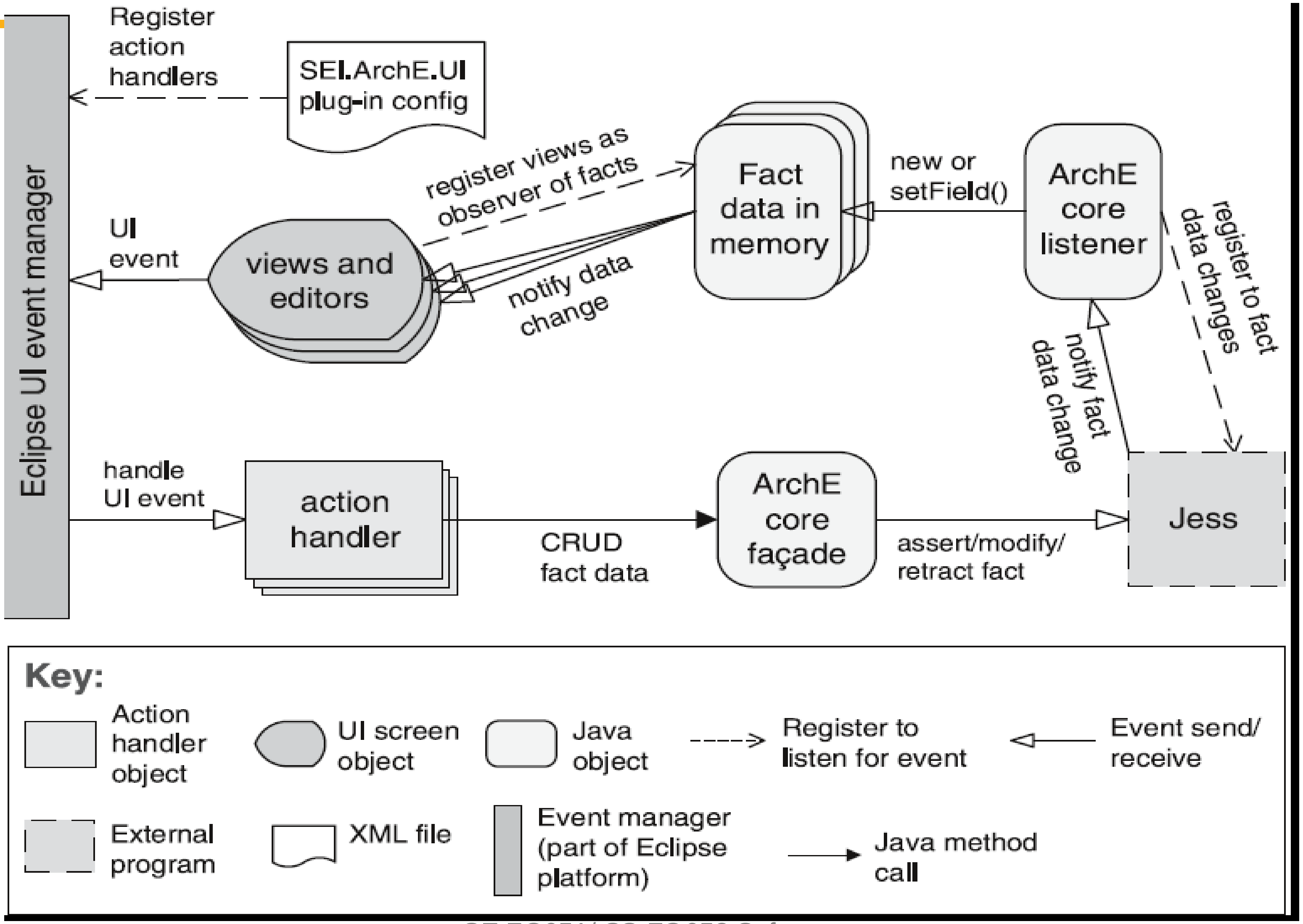
Publish-Subscribe Pattern

Context: There are a number of independent producers and consumers of data that must interact. The precise number and nature of the data producers and consumers are not predetermined or fixed, nor is the data that they share.

Problem: How can we create integration mechanisms that support the ability to transmit messages among the producers and consumers so they are unaware of each other's identity, or potentially even their existence?

Solution: In the publish-subscribe pattern, components interact via announced messages, or events. Components may subscribe to a set of events. Publisher components place events on the bus by announcing them; the connector then delivers those

Publish-Subscribe Example



Publish-Subscribe Solution – 1

Overview: Components publish and subscribe to events. When an event is announced by a component, the connector infrastructure dispatches the event to all registered subscribers.

Elements:

– *Any C&C component* with at least one publish or subscribe port.
– *The publish-subscribe connector,* which will have *announce* and *listen* roles for components that wish to publish and subscribe to events.

Relations: The *attachment* relation associates components with the publish-subscribe connector by prescribing which components announce events and which components are registered to receive events.

Publish-Subscribe Solution - 2

Constraints: All components are connected to an event distributor that may be viewed as either a bus—connector—or a component. Publish ports are attached to announce roles and subscribe ports are attached to listen roles.

Weaknesses:

- Typically increases latency and has a negative effect on scalability and predictability of message delivery time.
- Less control over ordering of messages, and delivery of messages is not guaranteed.

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Shared-Data Pattern

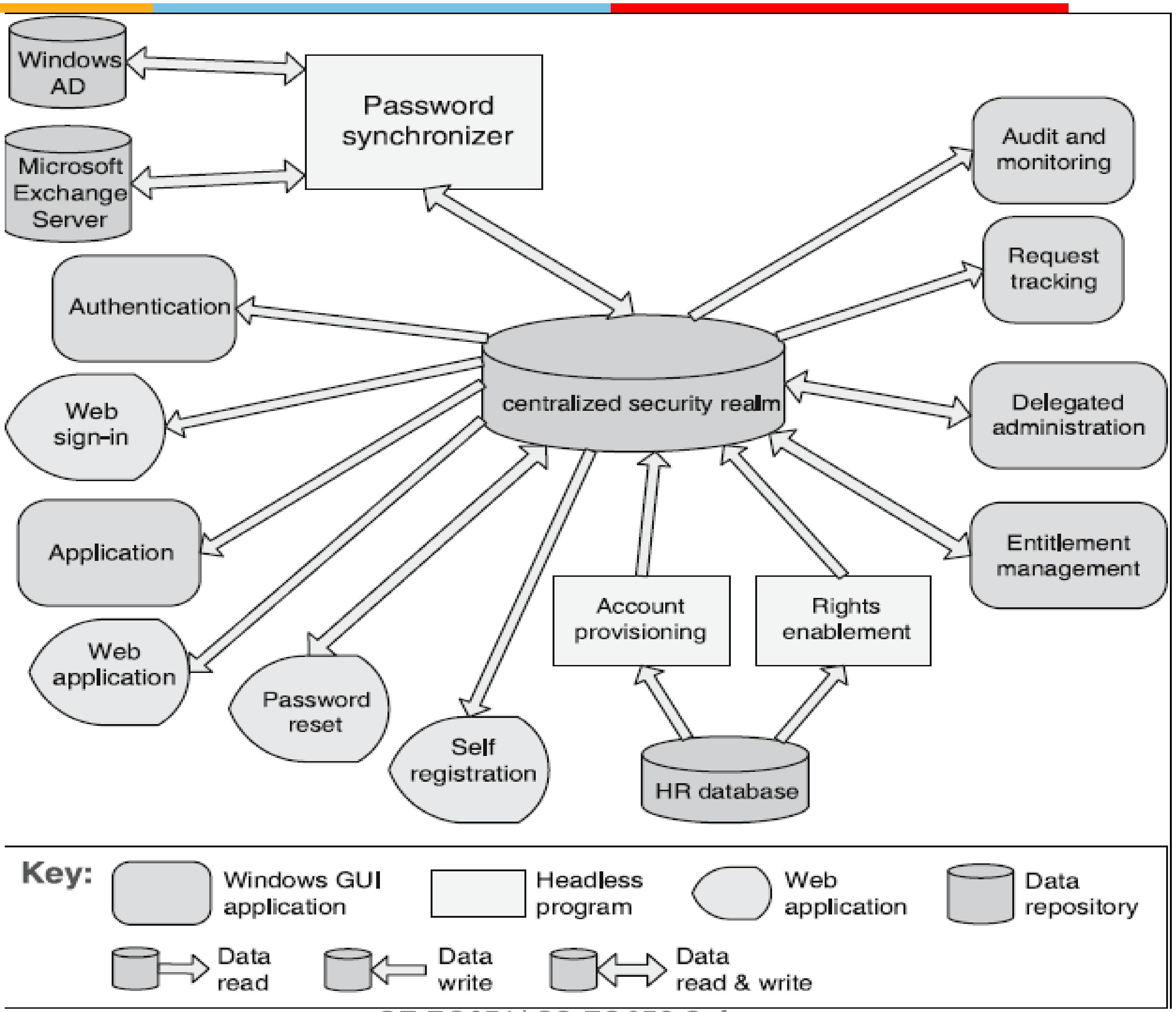
Shared-Data Pattern

Context: Various computational components need to share and manipulate large amounts of data. This data does not belong solely to any one of those components.

Problem: How can systems store and manipulate persistent data that is accessed by multiple independent components?

Solution: In the shared-data pattern, interaction is dominated by the exchange of persistent data between multiple *data accessors* and at least one *shared-data store*. Exchange may be initiated by the accessors or the data store. The connector type is *data reading and writing*.

Shared Data Example



Shared Data Solution - 1

Overview: Communication between data accessors is mediated by a shared data store. Control may be initiated by the data accessors or the data store. Data is made persistent by the data store.

Elements:

- *Shared-data store.* Concerns include types of data stored, data performance-oriented properties, data distribution, and number of accessors permitted.
- *Data accessor component*.
- *Data reading and writing connector*.

Shared Data Solution - 2

Relations: *Attachment* relation determines which data accessors are connected to which data stores.

Constraints: Data accessors interact only with the data store(s).

Weaknesses:

- The shared-data store may be a performance bottleneck.
- The shared-data store may be a single point of failure.
- Producers and consumers of data may be tightly coupled.

Thank you