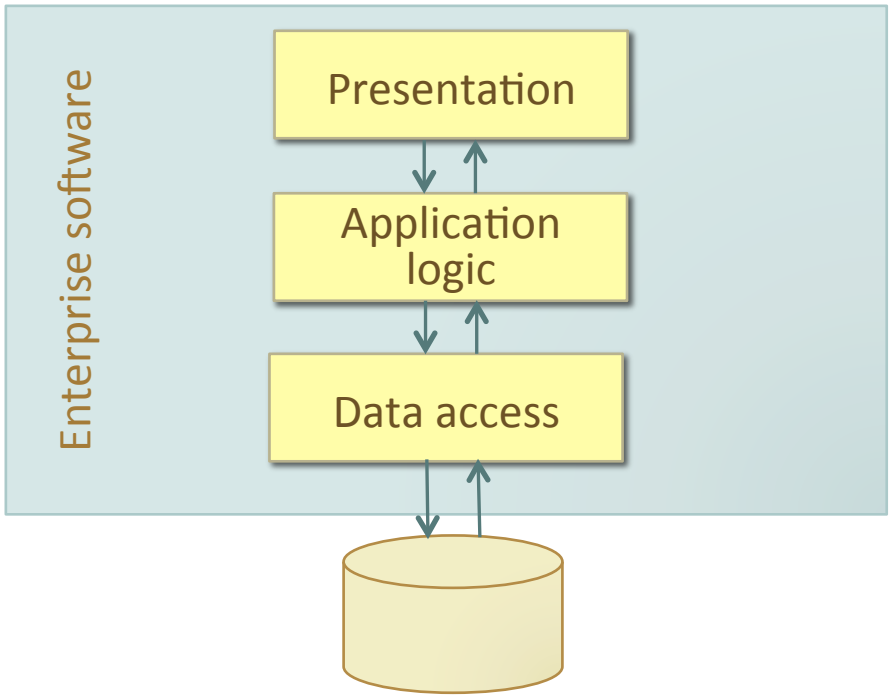
MTAT.03.229 Enterprise System Integration

Lecture 2: Multi-tier applications and middleware

Luciano García-Bañuelos

University of Tartu

The Anatomy of an Enterprise System



Application layers

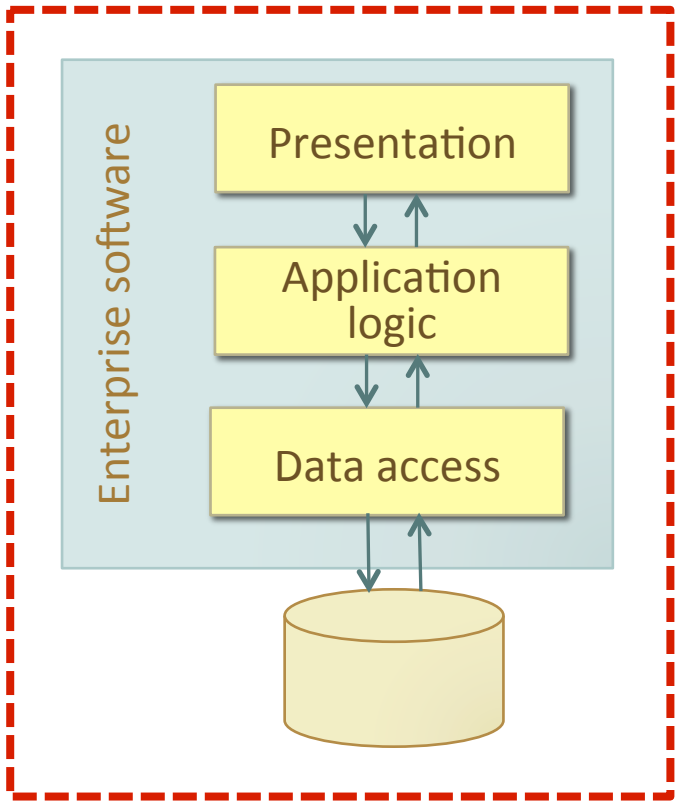
- Users interact with the system through a **presentation layer** (aka **user interface** or UI)
- The **application logic** (aka business logic) determines what the system actually does:
  - Enforces business rules
  - Coordinates business processes
- The **data access layer** facilitates the access to persistent data manipulated by the application logic.
  - Includes access to databases, search engines, document managers and/or a file system.

Tiers or Layers?

- N-tier architectures aim at splitting the application in components into different tiers/layers
- Tiers refer to **physical distribution** of components
  - Components can be executed over a collection of computers
- Layers refer to **logical separation** of components
  - Layered architecture: Communication should only happen with the contiguous layers
  - Promotes reuse and logic independence (components should be seamlessly replaceable)

☛ The number of tiers in the system somehow reflects the evolution of software architectures w.r.t. distribution

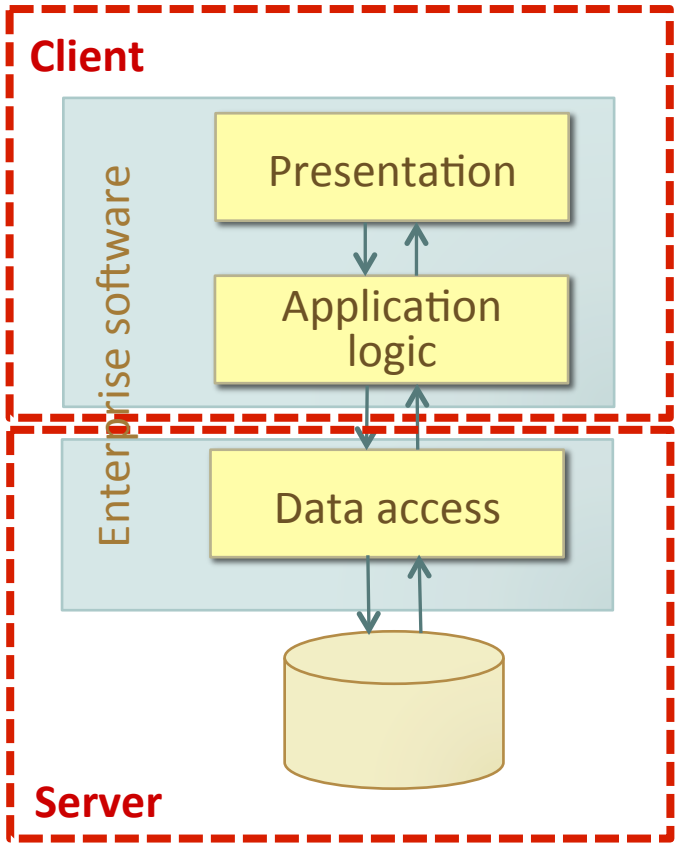
1-tier architecture



- All layers are bundled in a monolithic entity
- Typical "mainframe" architecture
  - Users access the system through dumb terminals
  - All computation happens in a single computer

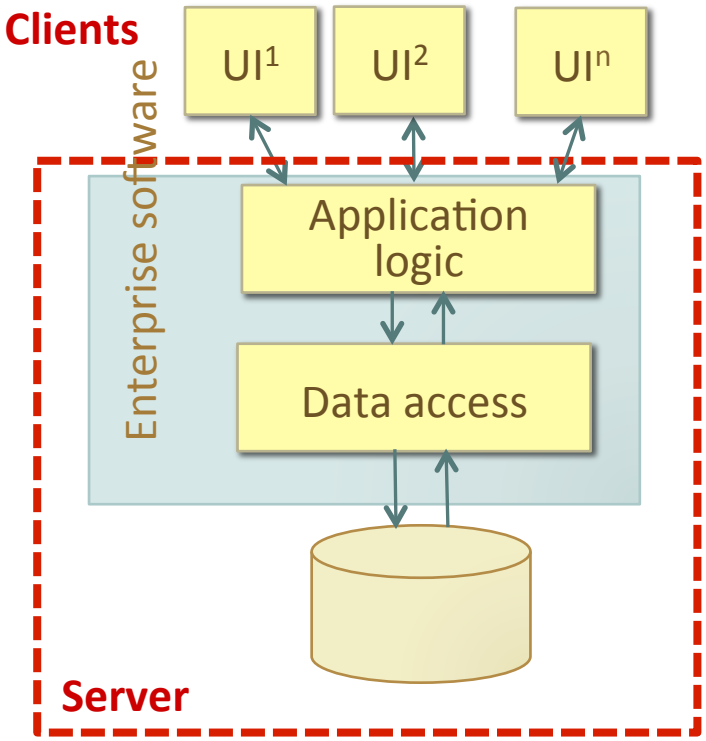
Pros	Cons
No context switching in the control flow	Limited scalability due to restrictions in the number of processors
Centralization eases resource management/ sharing	Oftentimes the code is platform specific, limiting portability
Code highly optimisable	Intertwined code hindering maintenance

2-tier architecture (1/2)



- With the introduction of computer networks, computation started to be physically distributed
- Application layers are distributed depending on the computing power of clients:
  - Thin clients execute only presentation layer
  - Fat clients execute both presentation and application logic layers
- The concept of API makes its appearance
- Notable example: Database management systems
  - The separation of data access layer promotes logical independence, reducing the impact of replacing a database technology on the presentation application logic layers

2-tier architecture (2/2)



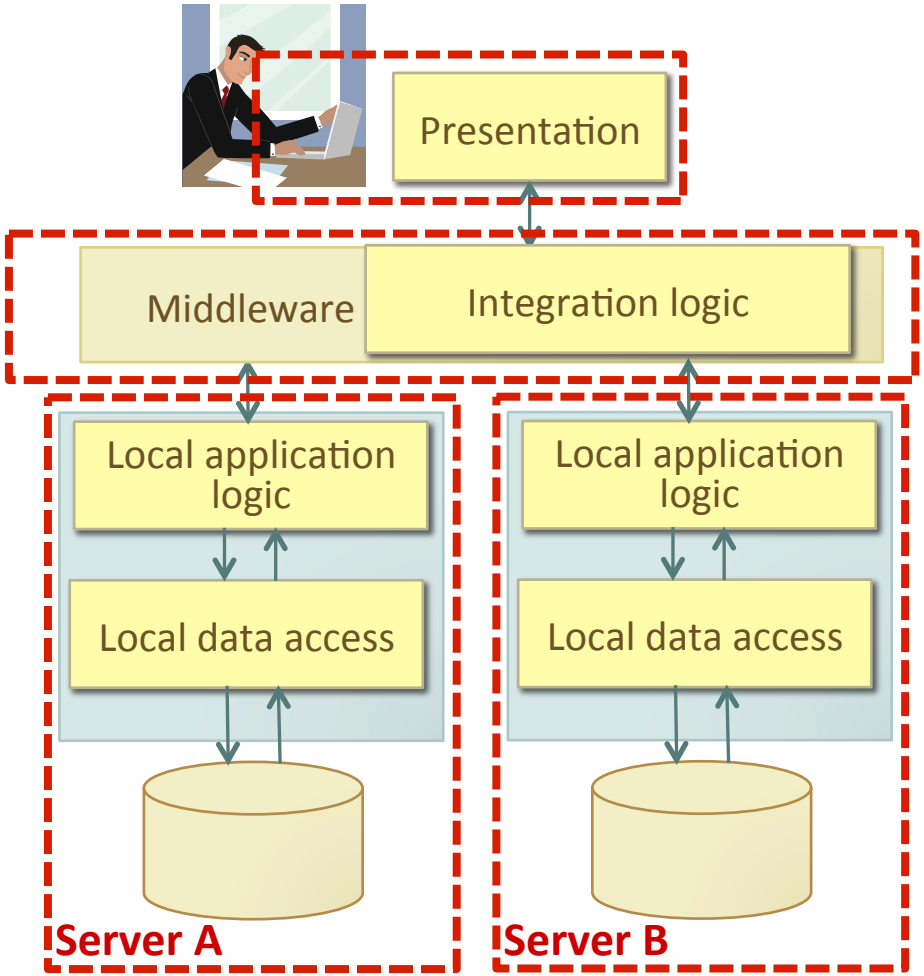
- With the arrival of PCs, the presentation layer moved to the client
  - Smartphones?
- With this approach, it is possible to have multiple presentation layers
  - Text (console) application
  - Graphical user interfaces (e.g., Java swing)
  - Web applications (e.g., HTML5, Javascript, etc.)
- Web as the universal platform for computing?
  - Google's Chromium OS

Issues with 2-tier architecture

- Problematic for a client to access multiple server applications
  - Server applications do not know each other
  - No common business logic
- The client if responsible for knowing where things are, how to get to them, and how to ensure consistency
- Little can be done to solve the problem with the 2-tier architecture
- One additional level of indirection is required:

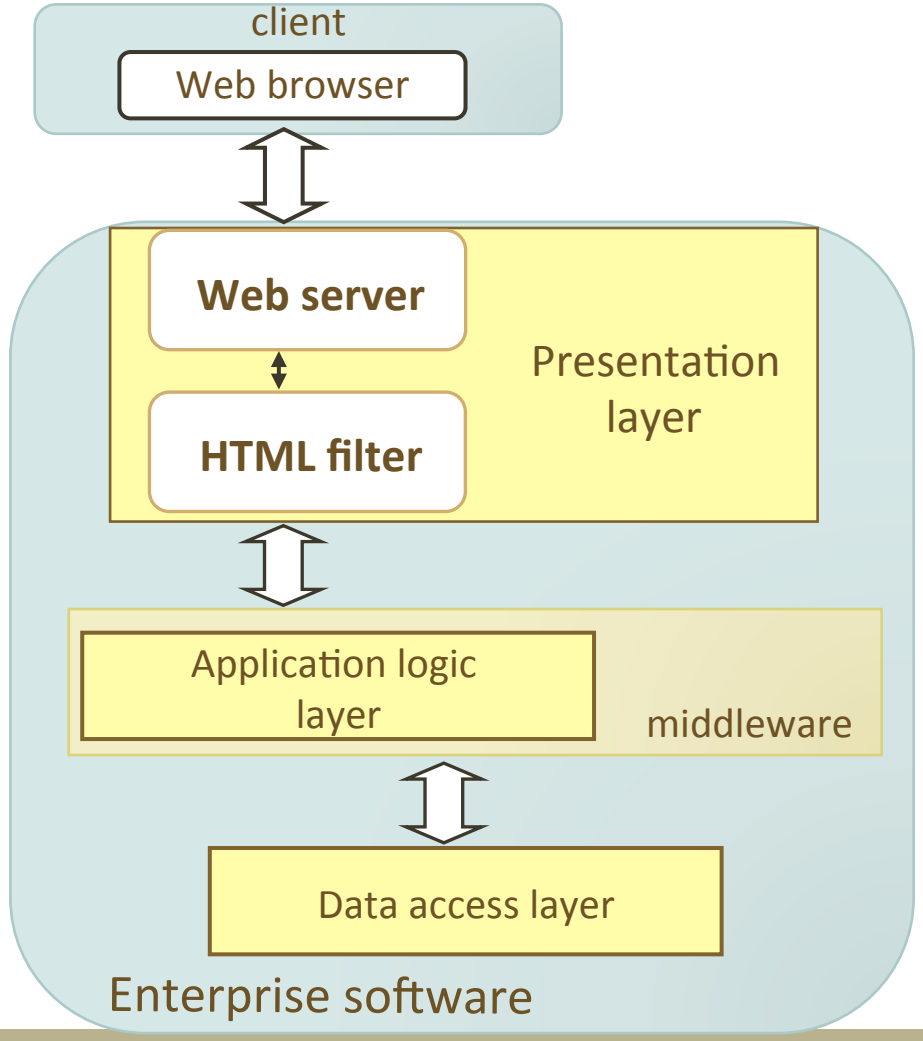
THE MIDDLEWARE

3-tier architecture: The middleware arrives



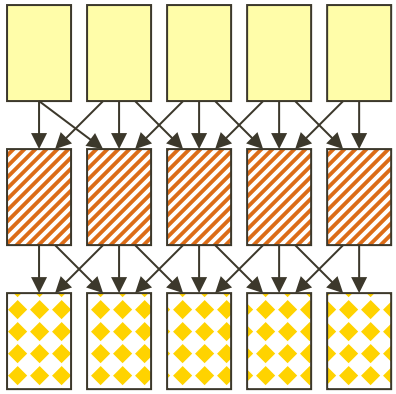
- Middleware is a level of indirection between clients and other layers
  - Simplifies the design of client applications by reducing the number of interfaces
  - Encapsulates integration logic and global application logic
  - Locates resources, accesses them, and integrates results (mediates between application logic/data access layers)

N-tier architecture



- N-tier architectures result from connecting several 3- tier systems and/or adding a layer to allow clients to access the system through a Web server ("Web layer")
- The Web layer is hosted in a Web application server: a middleware accessible through the Web.
- Web application servers are taking also parts of the functionality of traditional middleware – the boundary between the Web layer and the middleware is blurred.

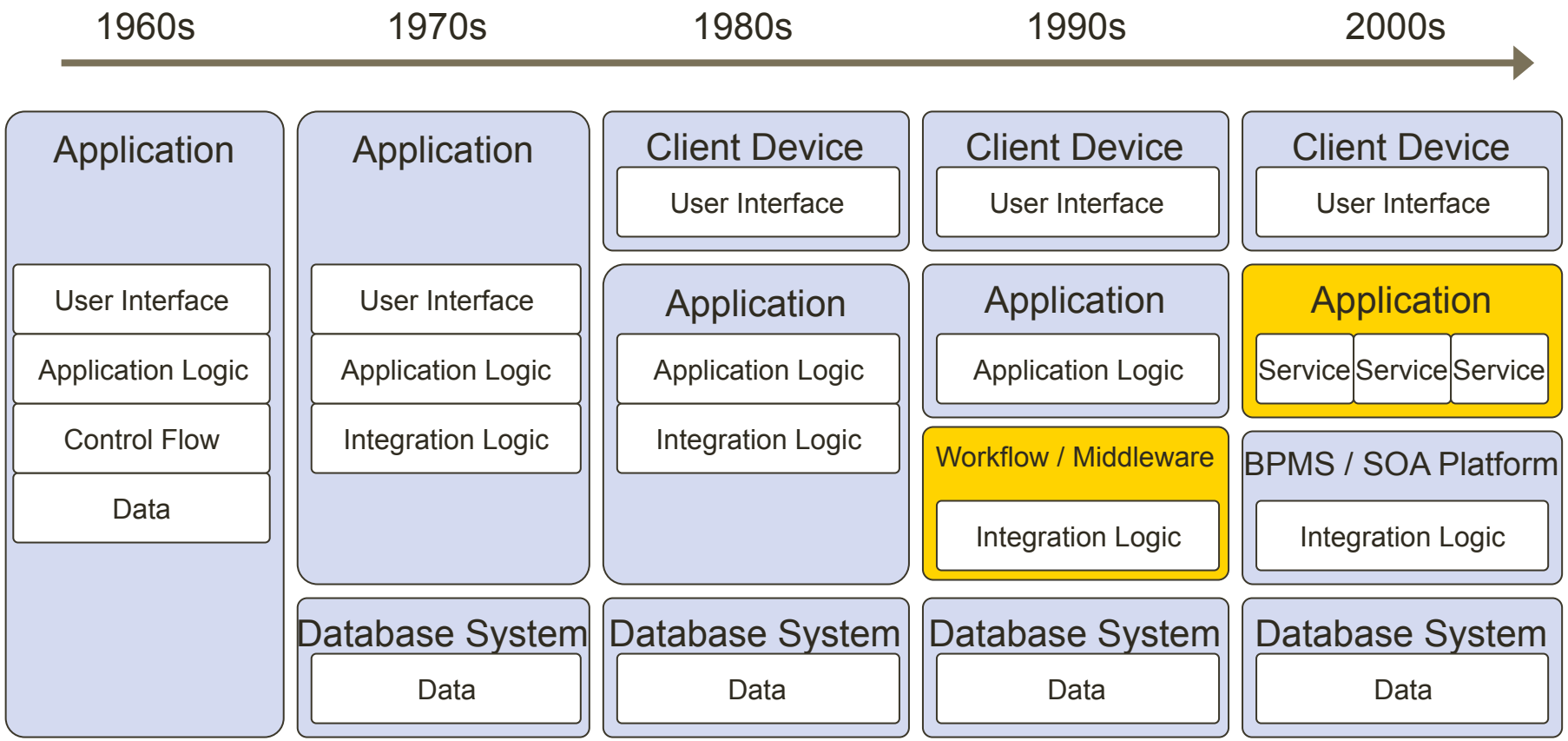
Flexibility versus Performance



There is no problem in system design that cannot be solved by adding a level of indirection. There is no performance problem that cannot be solved by removing a level of indirection.

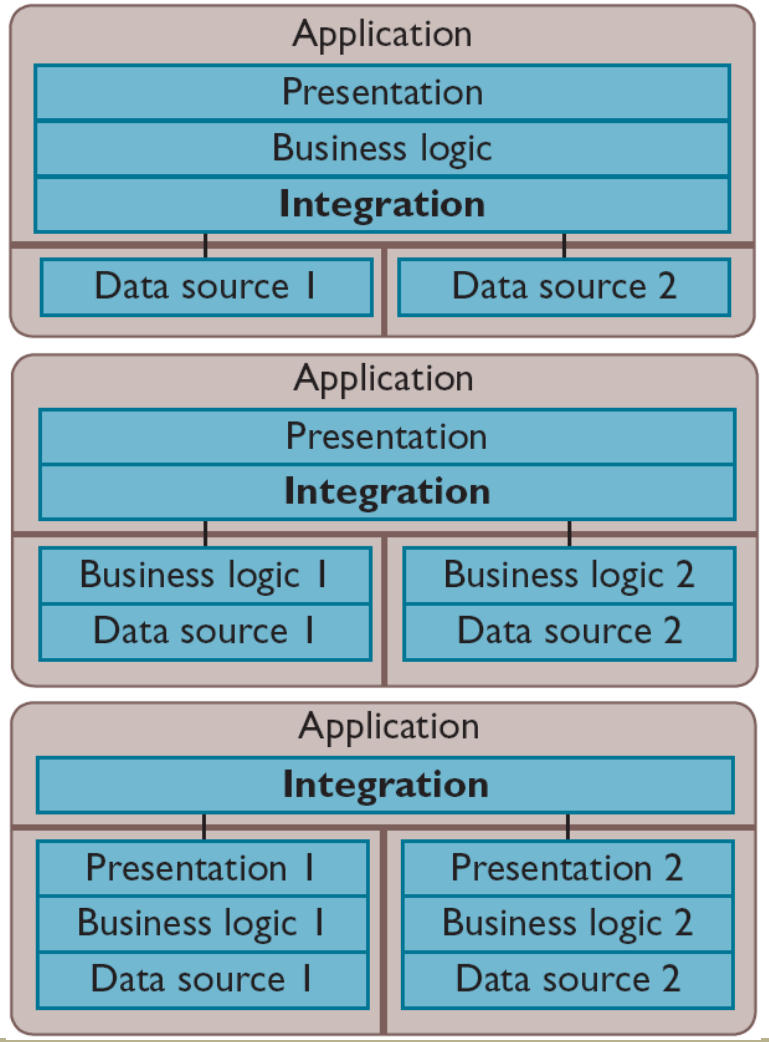
- The more boxes, the more modular the system: more opportunities for distribution and parallelism; more extensibility points.
- The more boxes, the more arrows: more connections need to be maintained, more coordination is needed. Complexity increases.
- The more boxes, the greater the number of context switches and intermediate steps to get to the data. Performance degrades.
- System designers try to balance the flexibility of modular design with performance demands.

Evolution of Information System Architectures



Integration Styles (classified by layer)

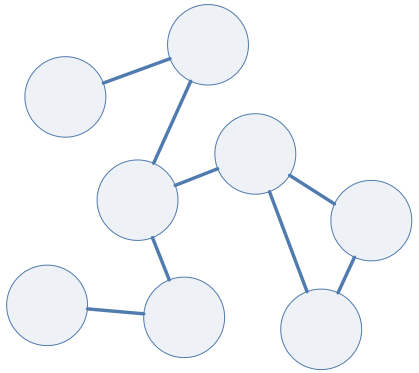
- Data access level
  - Shared/synchronized databases
  - Data warehousing
- Application logic level
  - Reuse application APIs
  - File transfer
  - RPC, message queues, workflow
- Presentation level
  - Screen scraping, revamping
  - Portlets, Web mashups



Integration Styles (classified by topology)

Point-to-point integration

- DB synchronization
- File transfer
- RPC / RMI (without registry)
- Simple message queues



Hub-and-spoke integration

- Data warehousing
- Federated databases
- Message brokers
- Object brokers
- Enterprise Service Bus
- Workflow / BPM systems