Client Server Architectures

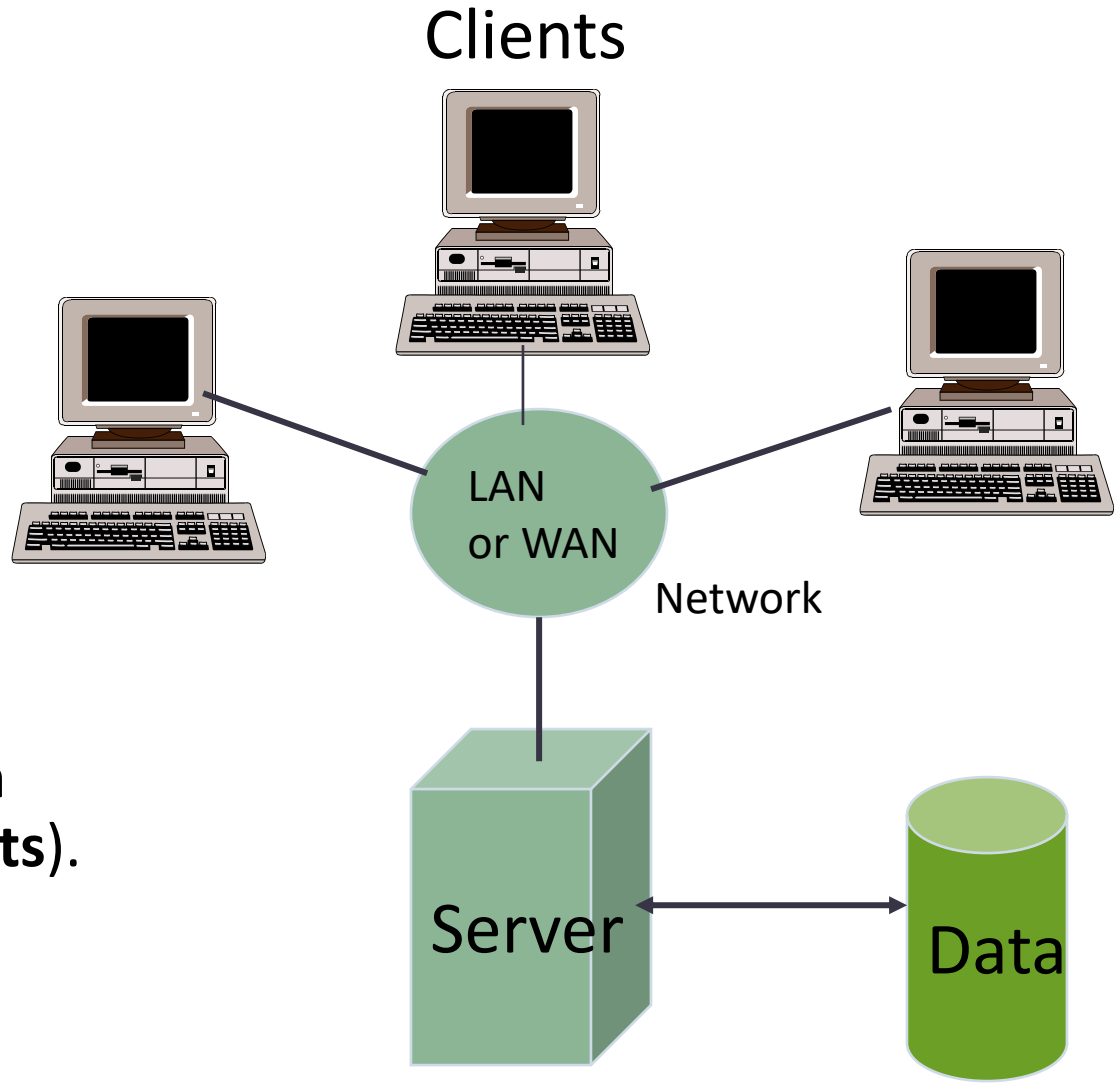
Platform Technologies

Based on Client/Server Architecture by Alex Berson &
Client/Server Distributed Systems by Dr. Andrew Davison

Client/Server Basics

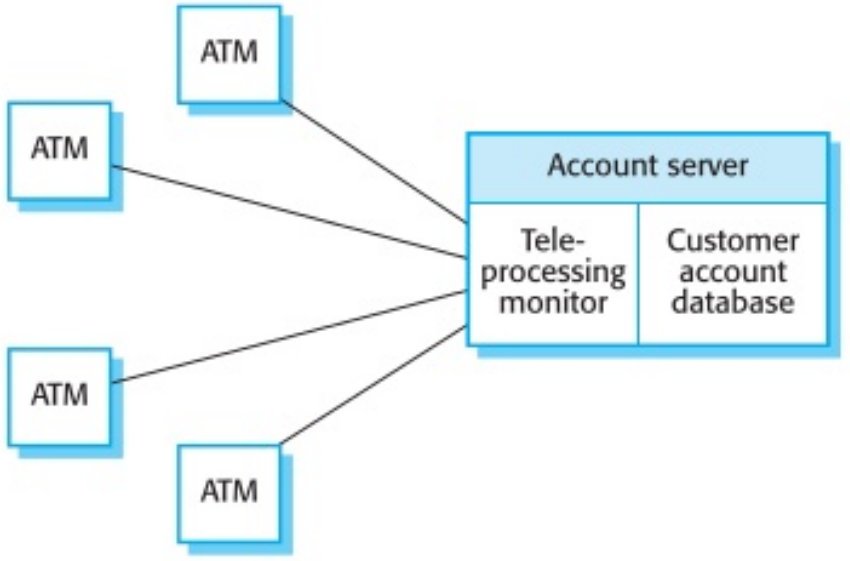
- A first examination of client/server functionality.

- A brief definition:
  - A **server** is a program (or collection of cooperating programs) that provides services and/or manages resources on the behalf of other programs (its **clients**).



Example: An ATM network

- The clients are the ATM machines
  - user interfaces
  - some simple application processing
- The server is at the bank
  - most application processing
  - very large database of customer accounts



Architectural Requirements

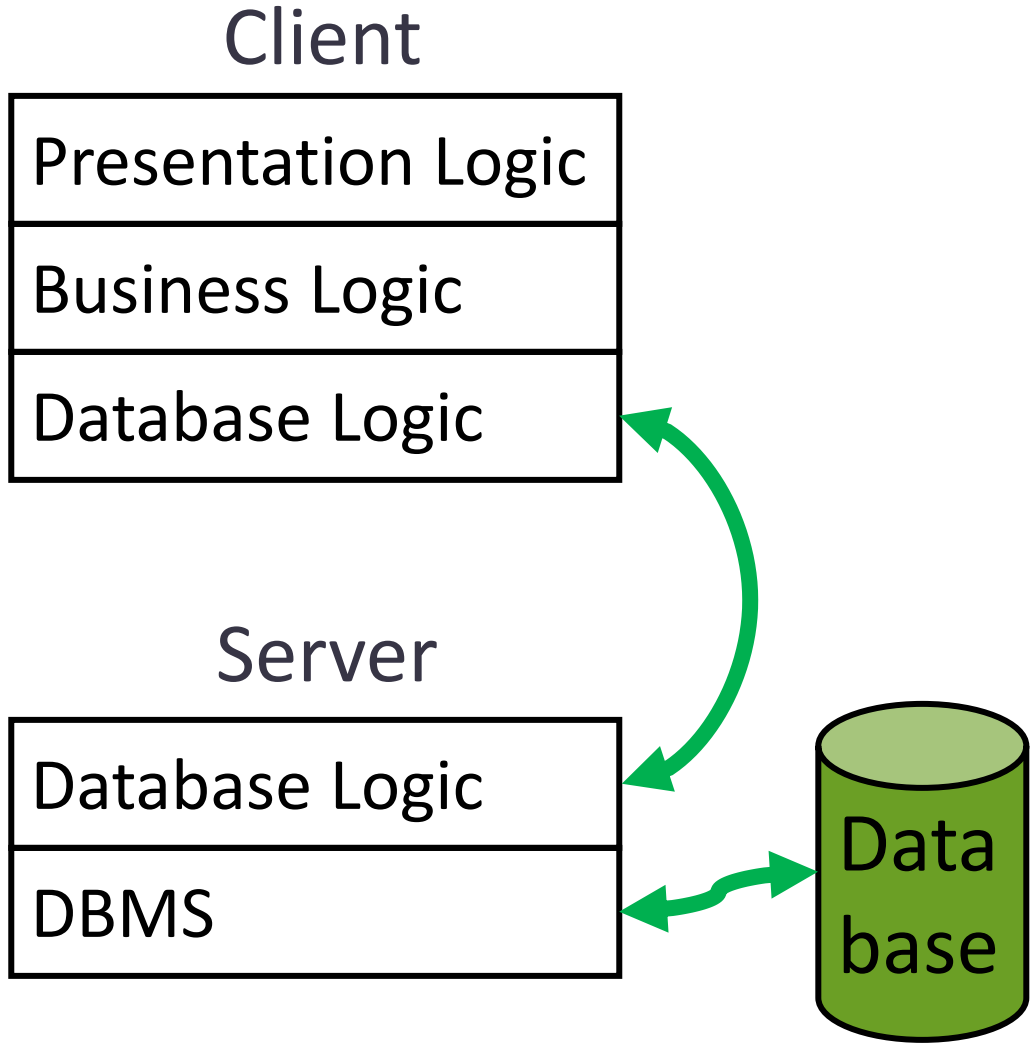
- Reliable, robust communication between the clients and server
- Client/server cooperation
- Server controls services/data that the client accesses
- Server handles conflicting requests
- Application processing is usually distributed between a client and the server

Client/Server Architectures

1. The 2-tier architecture
2. The 3-tier architecture
3. Locating the business logic
4. Locating the data
5. N-tier architecture
6. Microservices architecture
7. Web-Queue-Worker architecture
8. Big data, Big compute architectures

The 2-tier architecture

- The database is on the server
  - could some of it be moved to the client?
- Distributed database logic
  - most of it is on the client
- The client does the presentation.
- ‘Fat’ versus ‘thin’ clients.
- Much simpler if all the database servers are the same (homogenous).

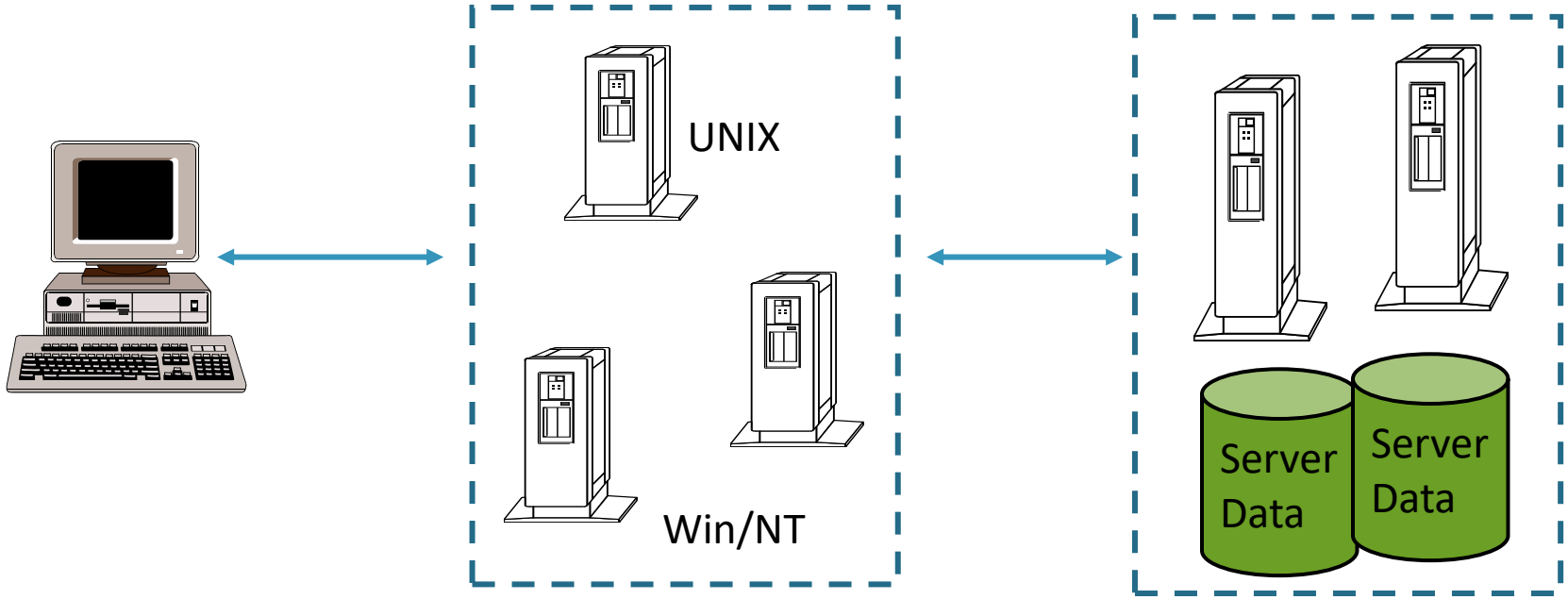


The 2-tier architecture: Drawbacks

- It is difficult to build **heterogeneous** database environments.
- Transaction processing is limited by the DBMS.
- Asynchronous processing is difficult
  - i.e. the client doesn’t wait for the server’s answer
- Scalability?

The 3-tier architecture

- The *back-end server* is usually a very large database (or databases)
- The *middle-tier server* usually holds shared applications (application/business logic)



The 3-tier architecture: Benefits over 2-tier

- The application logic in the middle-tier is more independent of the client and the back-end server
  - it should be more robust
- The application logic in the middle-tier can work more easily with data from multiple sources.
- Encourages multiple back-end servers
  - encourages data distribution

The 3-tier architecture: Drawbacks

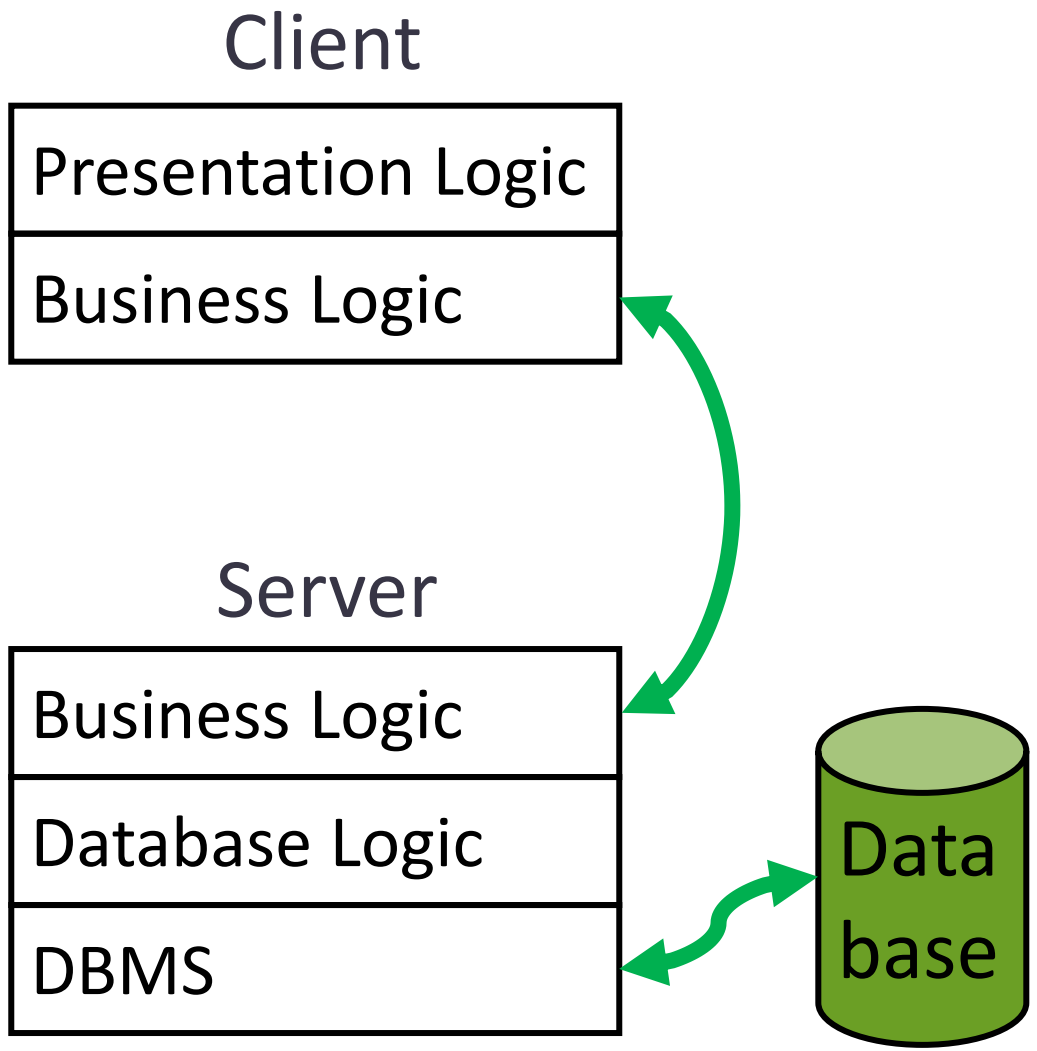
- Much more complex:
  - network management, data integrity, maintenance, development
- Still (partially) dependent on platforms
  - e.g. the client may still be restricted to a certain application server, but not (maybe) to any data server

The 3-tier architecture: Examples

- A ‘real’ ATM network
  - the ATM machines are the clients (as before)
  - the middle-tier servers provide certain processing
    - checking balances, money transfer requests
    - directing queries to the relevant back-end server
  - back-end server(s)
    - specialized by account type
    - very robust concurrency control, transaction processing
- Many Web applications are 3-tier:
  - the Web browser is the client software
  - the embedded components in Web pages come from the middle-tier
  - the back-end server contains the database/groupware

Locating the Business Logic

- Three ways of distributing the 'business logic' *(i.e. program code)*:
  - locate it entirely on the client
    - *fat* client
  - locate it entirely on the server
    - *fat* server
  - split it between the client and server



Fat Server Advantages

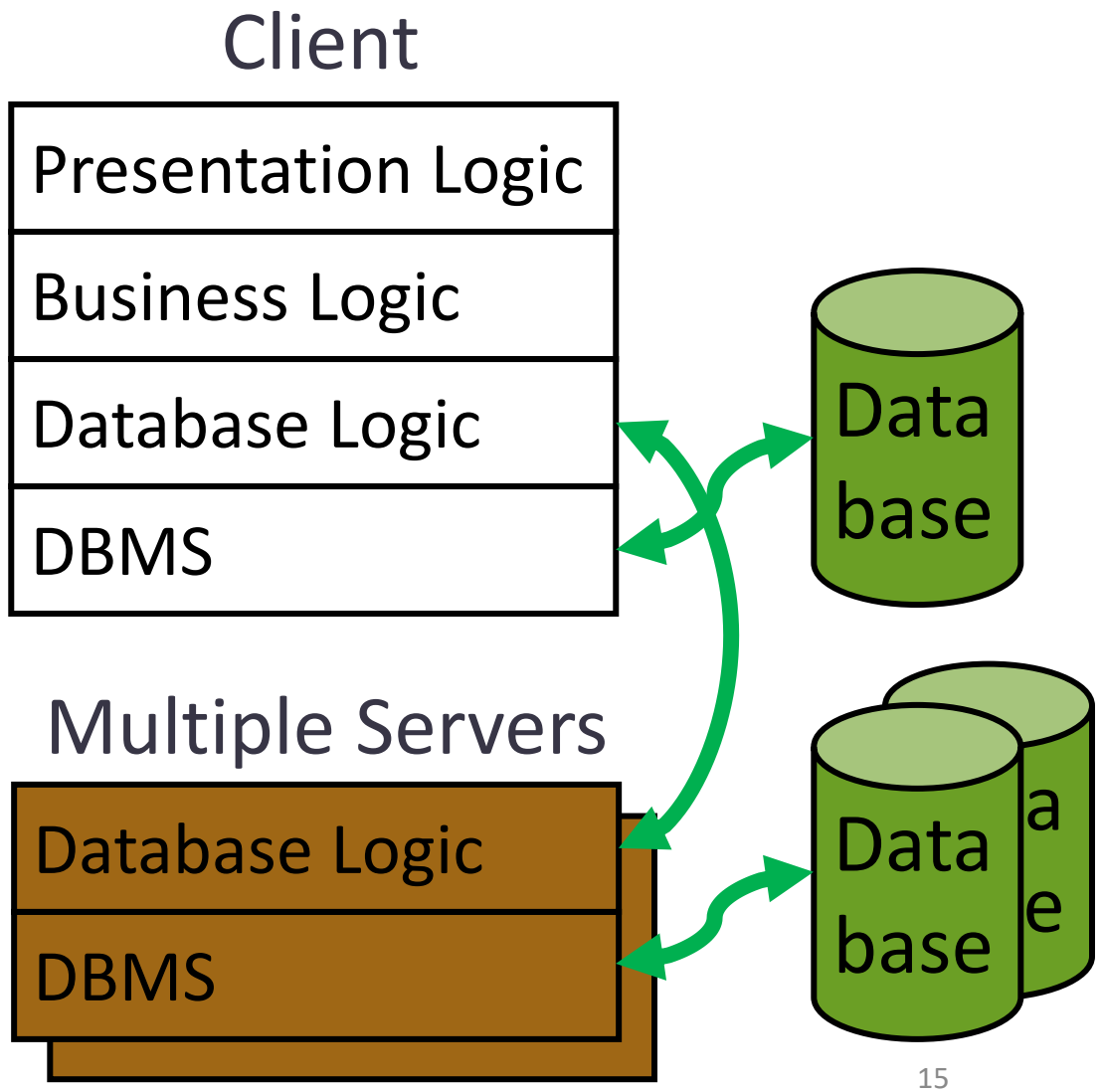
- Easier to update the application logic since clients not involved.
- Data is better hidden from clients.
- Easier to manage and debug since data and code is centrally located.
- Reduces bandwidth problems since data processing stays on the server.
- Better for mission-critical applications when fault-tolerance and stability are important.
- Encourages client simplicity and compatibility since the server must be able to work with many types of client.
  - e.g. support standard APIs

Fat Client Advantages

- The server is unaffected when updates are done to the client's application logic
  - the server will be more stable
- Easier to program
  - less networking
  - more direct access to client platform features, such as GUI

Locating the Data

- Issues:
  - Dividing up the data
  - Transparency of the distribution
  - Data integrity / synchronisation / consistency
  - Data administration / management

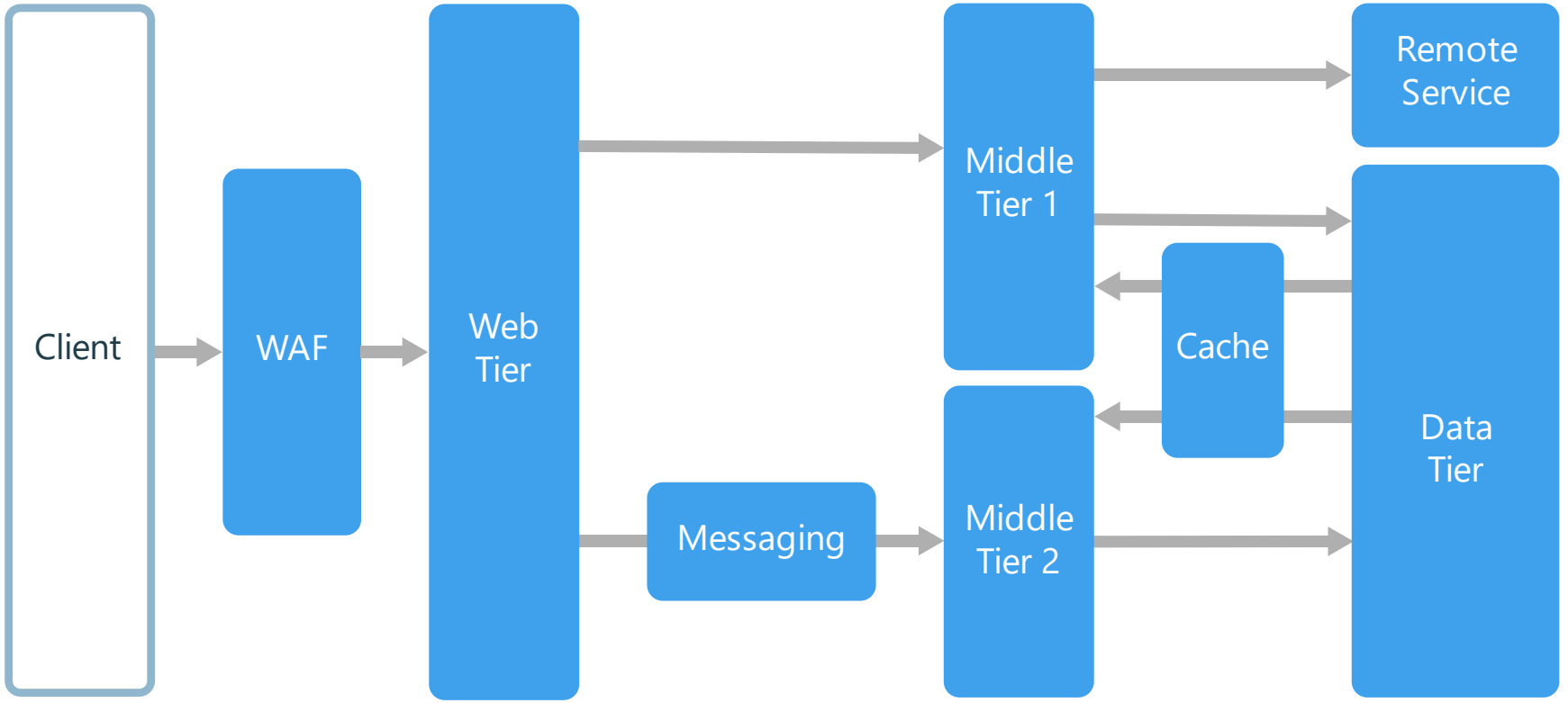


Transaction Processing

- A transaction is a sequence of actions which takes a system (usually a database) from one consistent state to another.
  - e.g. change a customer's record
- A transaction should possess the "*ACID*" properties:
  - **A**tomicity, **C**onsistency, **I**solation, **D**urability
- Recovery and concurrency mechanisms are necessary, typically implemented in a Transaction Processing Management (TPM) system.
- TPMs become very complex when data is distributed.
  - ACID must be distributed as well

N-tier architecture

- An N-tier architecture divides an application into **logical layers** and **physical tiers**.

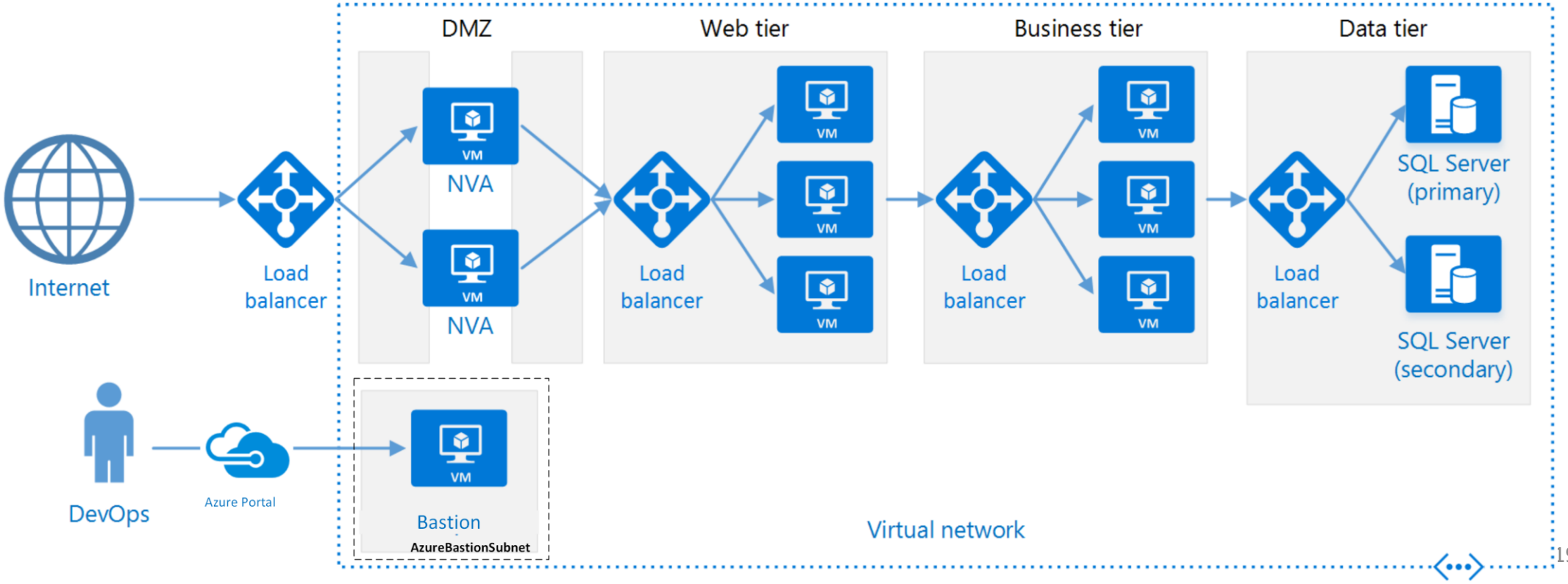


Common Features

- Asynchronous connectivity
- Data distribution using replication
- Name/directory services for resource location independence
- More complex data types
- More complex analysis
- Authentication services
- Distributed file system(s)
- Time services

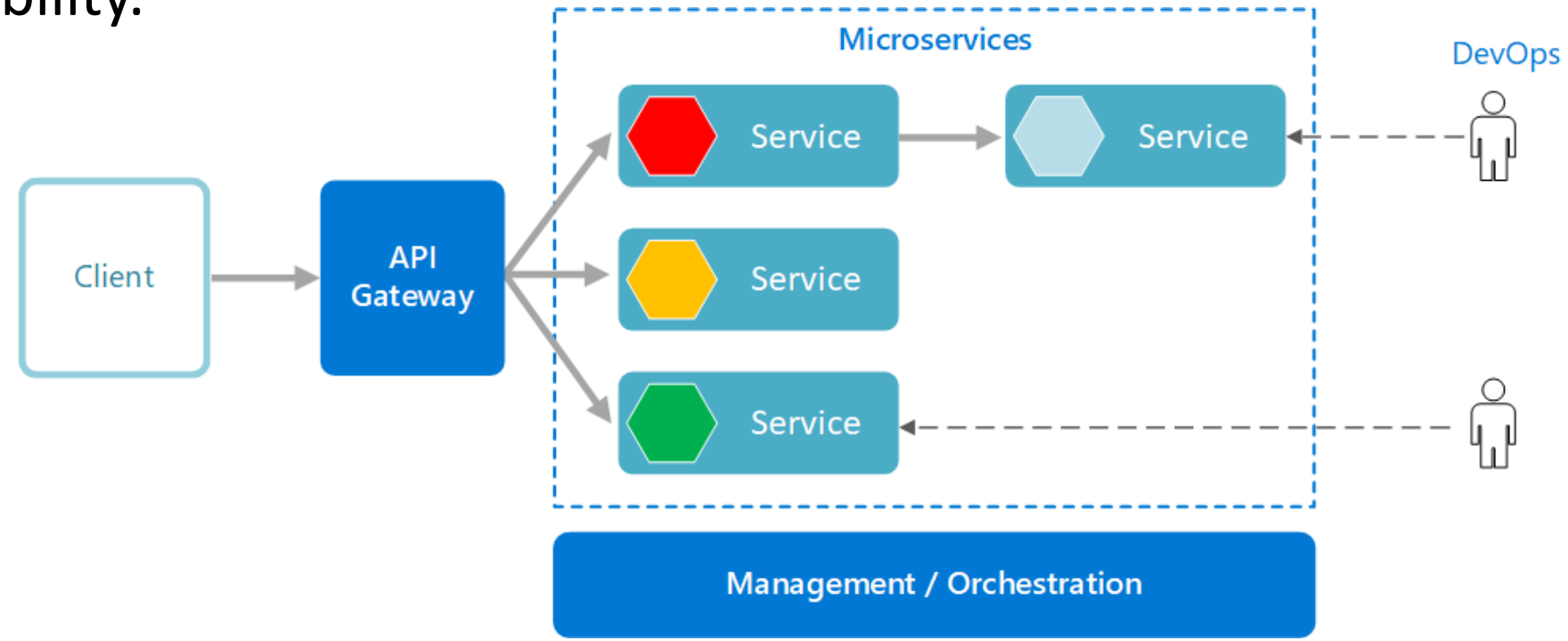
N-tier architecture on virtual machines

- Each tier consists of two or more VMs, placed in an availability set or virtual machine scale set. Multiple VMs can provide resiliency in case one VM fails.



Microservices architecture

- A microservices architecture consists of a collection of small, autonomous services.
- Each service is self-contained and should implement a single business capability.



Microservices architecture

Benefits

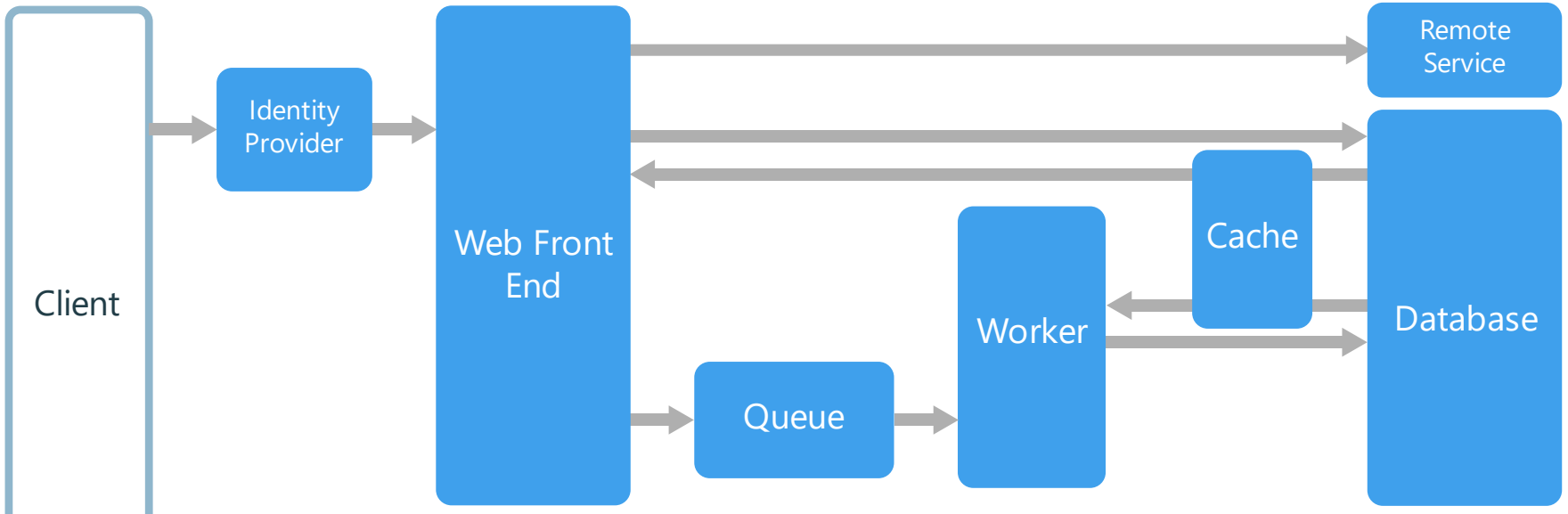
- Agility
- Small, focused teams
- Mix of technologies
- Fault isolation
- Scalability
- Data isolation

Challenges

- Complexity
- Development and testing
- Lack of governance
- Network congestion and latency
- Data integrity
- Management

Web-Queue-Worker architecture

- A **web front end** that serves client requests
- A **worker** that performs *resource-intensive tasks, long-running workflows, or batch jobs*.
- The web front end communicates with the worker through a **message queue**.



Web-Queue-Worker architecture

Benefits

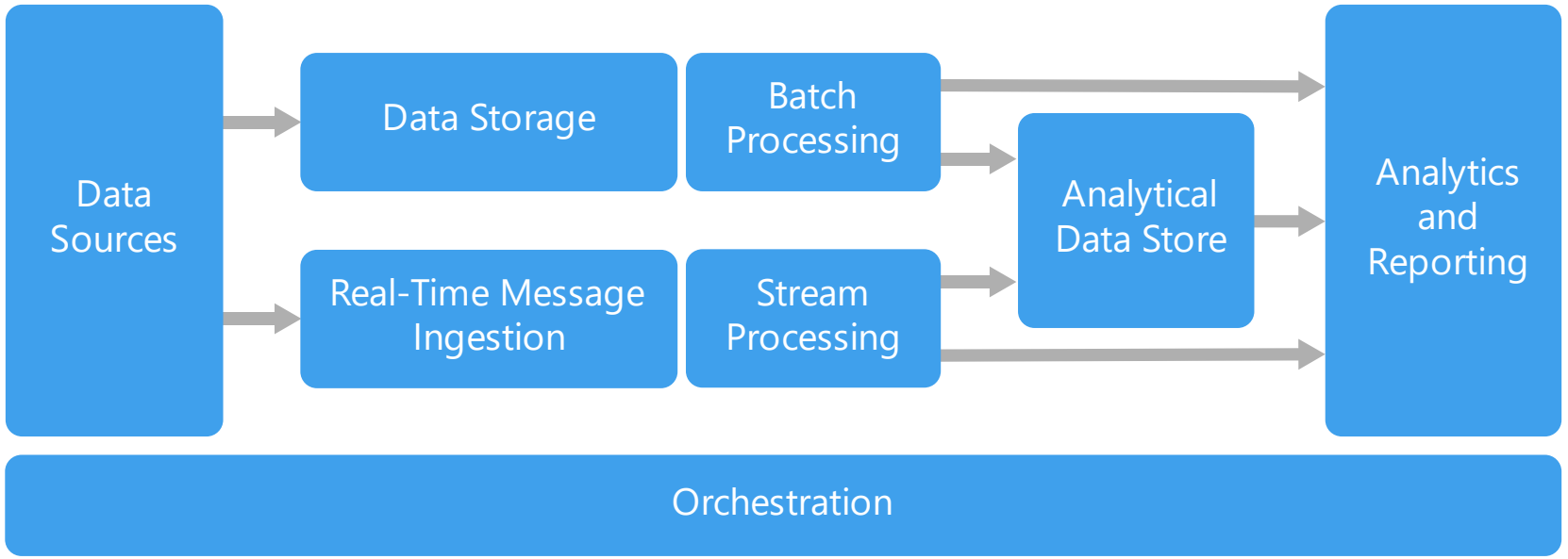
- Relatively simple architecture that is easy to understand.
- Easy to deploy and manage.
- Clear separation of concerns.
- The front end is decoupled from the worker using asynchronous messaging.
- The front end and the worker can be scaled independently.

Challenges

- Without careful design, the front end and the worker can become large, monolithic components that are difficult to maintain and update.
- There may be hidden dependencies, if the front end and worker share data schemas or code modules.

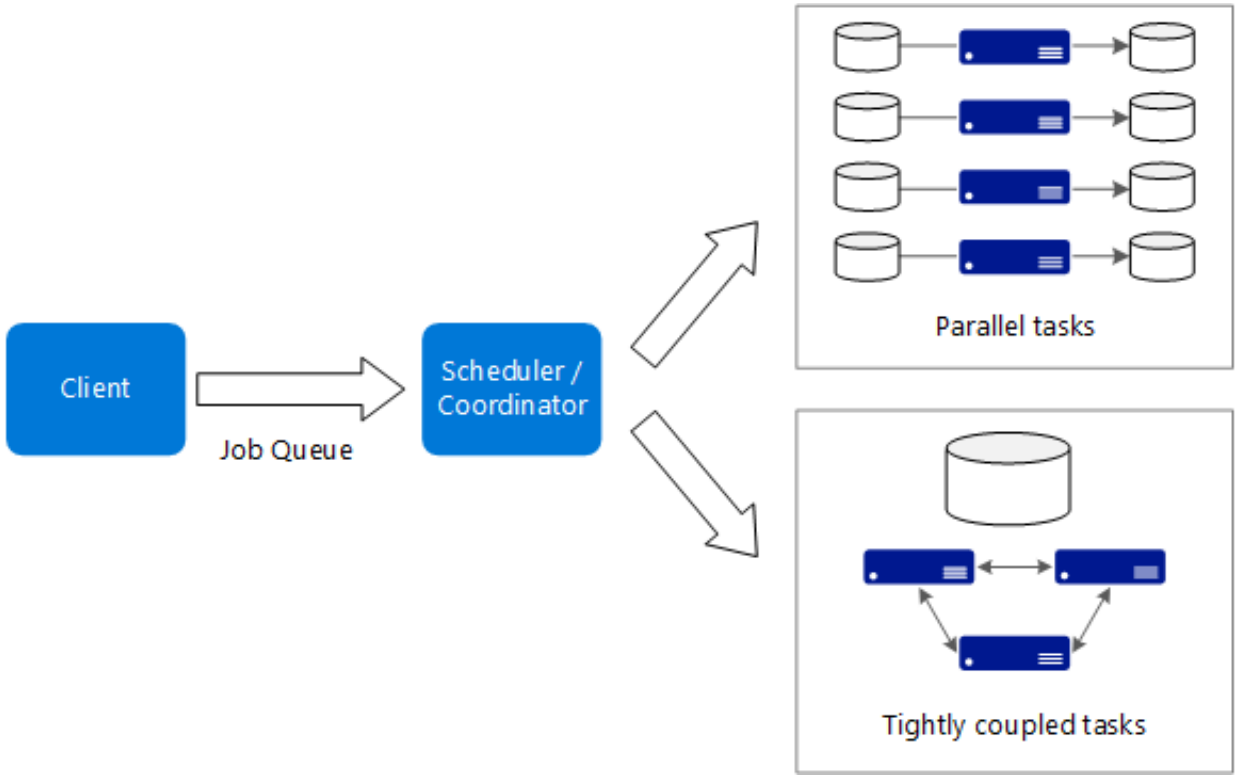
Big data architecture

- For handling the ingestion, processing, and analysis of data that is too large or complex for traditional database systems.



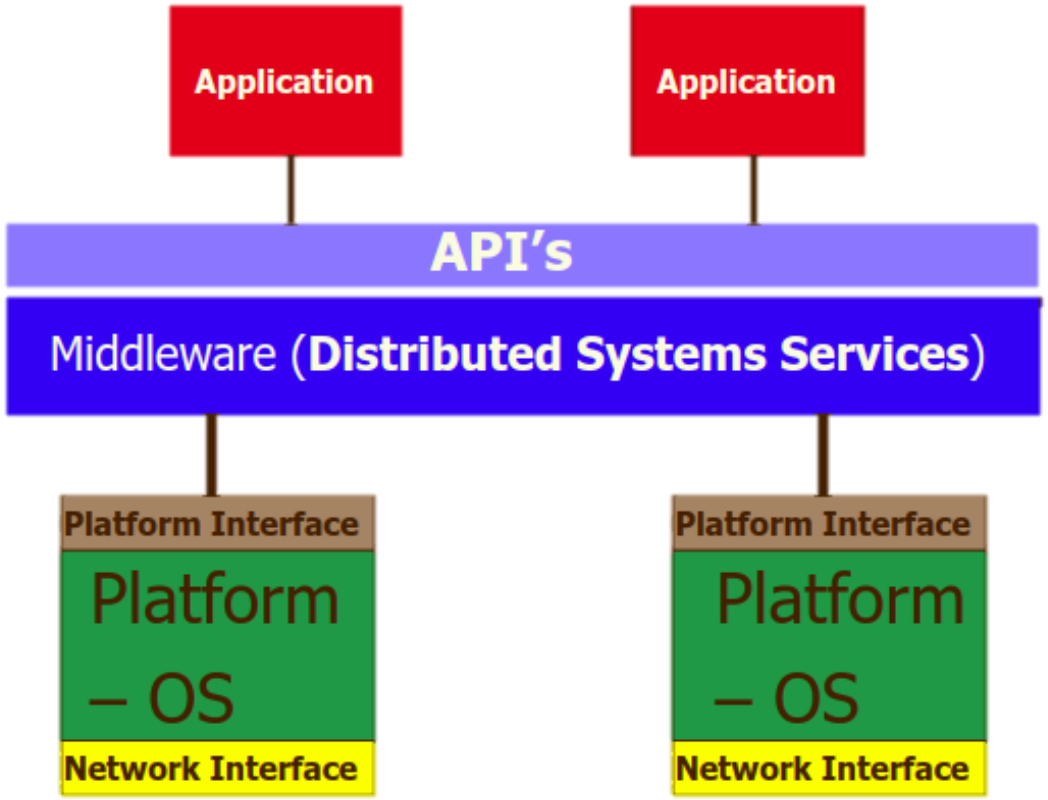
Big compute architecture

- For large-scale workloads that require a large number of cores, such as image rendering, fluid dynamics, financial risk modelling, oil exploration, drug design, and engineering stress analysis



Middleware

- A set of tools that provide a uniform means and style of access to system resources across all platforms
- Enable programmers to use the same method to access data regardless of the location of that data



Recurring Issues with Client/Server

- LAN, WAN, Internet scaling
- Data distribution/replication
- Distributed processing
- System management/maintenance
- Choice of middleware
- Standards / open systems

Advantages of Client/Server

- Mainframe functionality can be made widely available
  - cost benefits
- Processing and data are localised on the server
  - reduces network traffic, response time, bandwidth requirements
- Business logic can be distributed (in 3-tier model)
  - reuse, portability
- Present-day systems are too large and involve too many users to be located on one machine.

Disadvantages of Client/Server

- The server becomes a bottleneck
- Distributed applications are much more complex than non-distributed ones
  - i.e. in development, run time, maintenance, upgrades
- Requires a shift in business practises
  - local, simple data --> distributed, open, complex data

Client/Server Terminology

- Client
  - A networked information requester, usually a PC or workstation, that can query database and/or other information from a server
- Server
  - A computer, usually a high-powered workstation, a minicomputer, or a mainframe, that houses information for manipulation by networked clients
- Applications Programming Interface (API)
  - A set of function and call programs that allow clients and servers to intercommunicate
- Middleware
  - A set of drivers, APIs, or other software that improves connectivity between a client application and a server