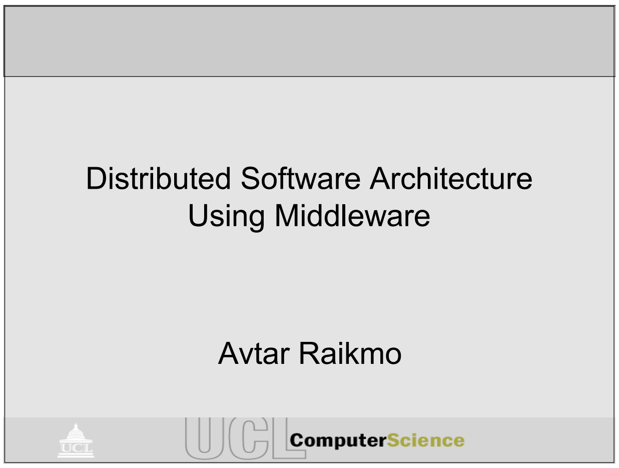
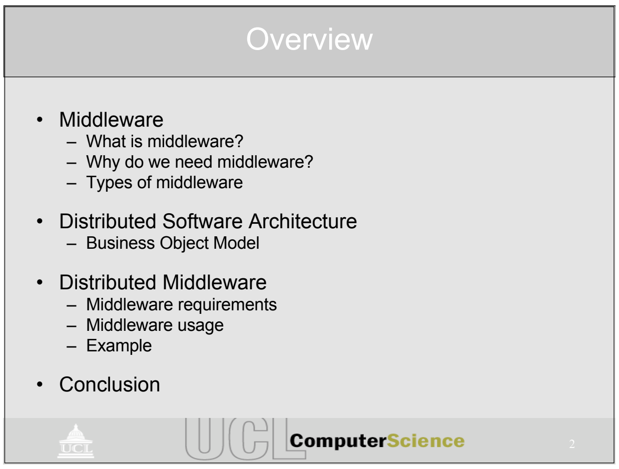


Distributed Software Architecture Using Middleware

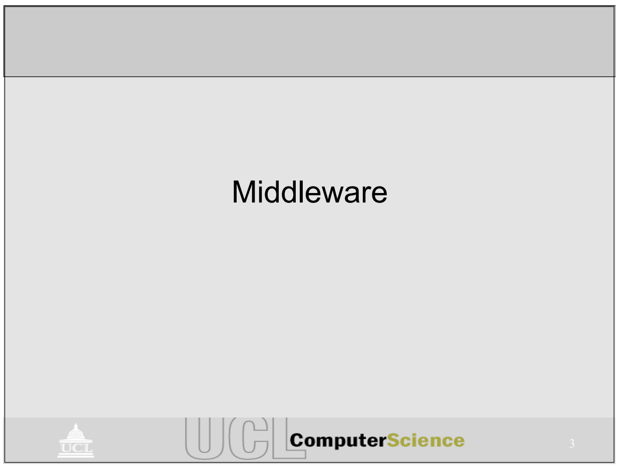
Avtar Raikmo



Overview

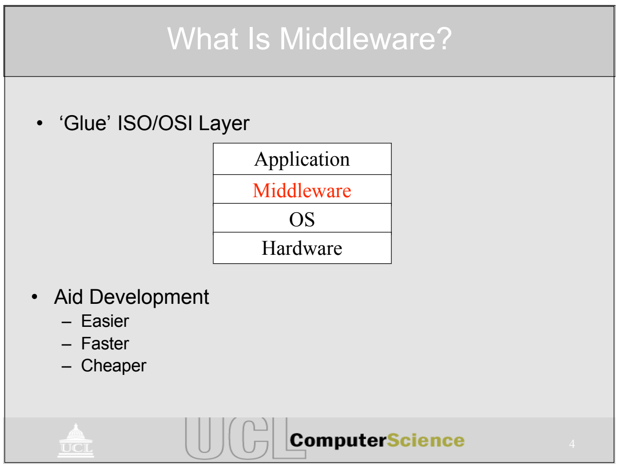
- Middleware
  - What is middleware?
  - Why do we need middleware?
  - Types of middleware
- Distributed Software Architecture
  - Business Object Model
- Distributed Middleware
  - Middleware requirements
  - Middleware usage
  - Example
- Conclusion

2



Middleware

3



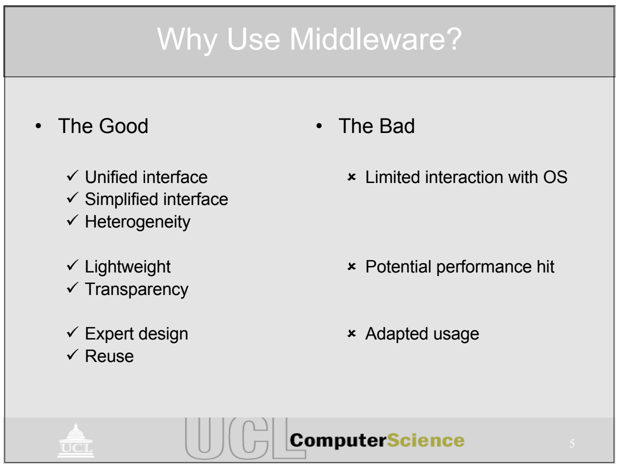
What Is Middleware?

- 'Glue' ISO/OSI Layer

Application
Middleware
OS
Hardware

- Aid Development
  - Easier
  - Faster
  - Cheaper

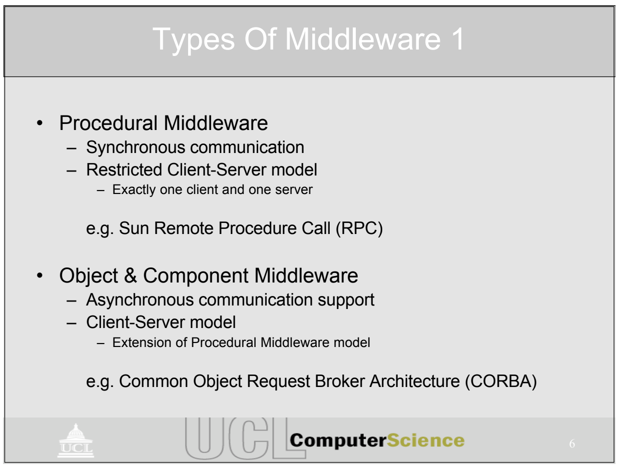
4



Why Use Middleware?

• The Good	• The Bad
✓ Unified interface ✓ Simplified interface ✓ Heterogeneity	× Limited interaction with OS
✓ Lightweight ✓ Transparency	× Potential performance hit
✓ Expert design ✓ Reuse	× Adapted usage

5



Types Of Middleware 1

- Procedural Middleware
  - Synchronous communication
  - Restricted Client-Server model
    - Exactly one client and one server

  e.g. Sun Remote Procedure Call (RPC)

- Object & Component Middleware
  - Asynchronous communication support
  - Client-Server model
    - Extension of Procedural Middleware model

  e.g. Common Object Request Broker Architecture (CORBA)

6

Types Of Middleware 2

- Transactional Middleware
  - Asynchronous communication support
  - Client-Server model
    - Distributed transactions, using 2PC

  e.g. IBM Customer Information Control System (CICS)

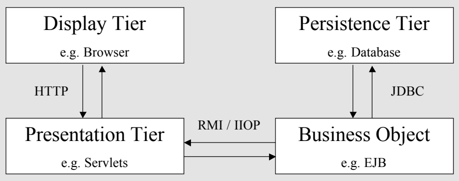
- Message-Orientated Middleware
  - Asynchronous message exchange
  - Point-to-Multipoint support
    - Use of topics and subscription

  e.g. Sun Java Message Service (JMS)

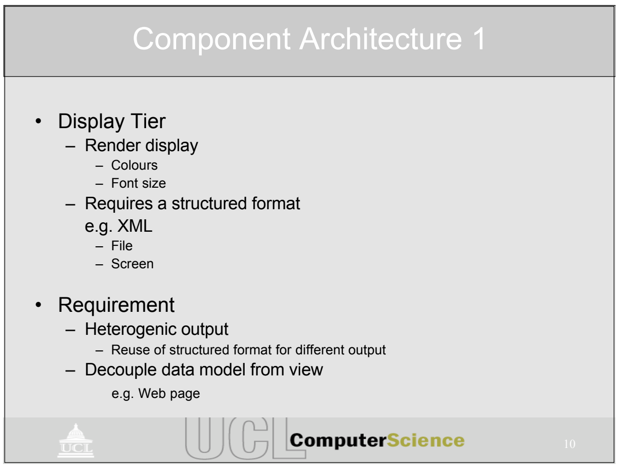
Distributed Software Architecture

Distributed Software Architecture

- Component Based Architecture



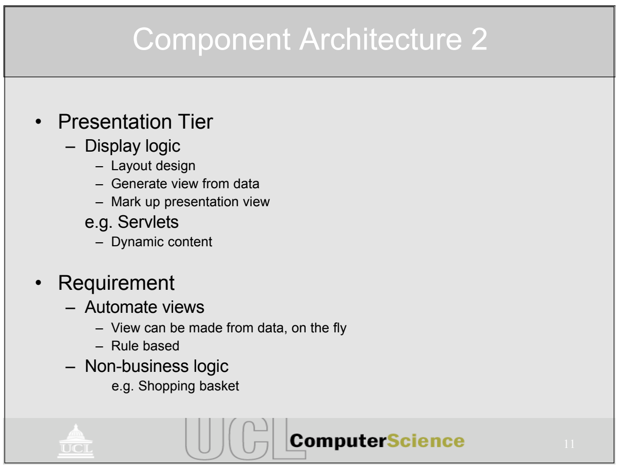
Business Object Model



Component Architecture 1

- Display Tier
  - Render display
    - Colours
    - Font size
  - Requires a structured format
    e.g. XML
    - File
    - Screen
- Requirement
  - Heterogenic output
    - Reuse of structured format for different output
  - Decouple data model from view
    e.g. Web page

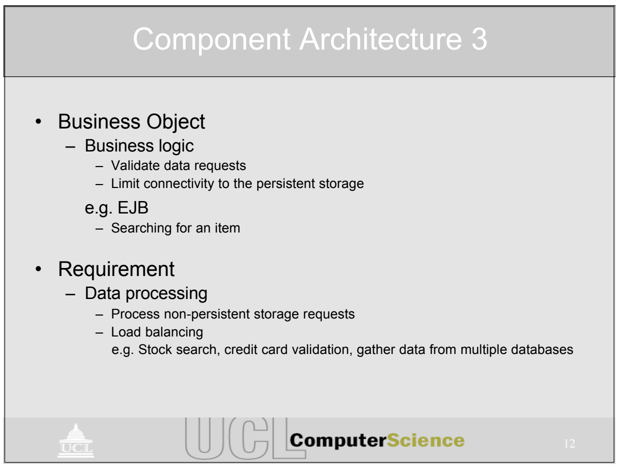
10



Component Architecture 2

- Presentation Tier
  - Display logic
    - Layout design
    - Generate view from data
    - Mark up presentation view
    e.g. Servlets
    - Dynamic content
- Requirement
  - Automate views
    - View can be made from data, on the fly
    - Rule based
  - Non-business logic
    e.g. Shopping basket

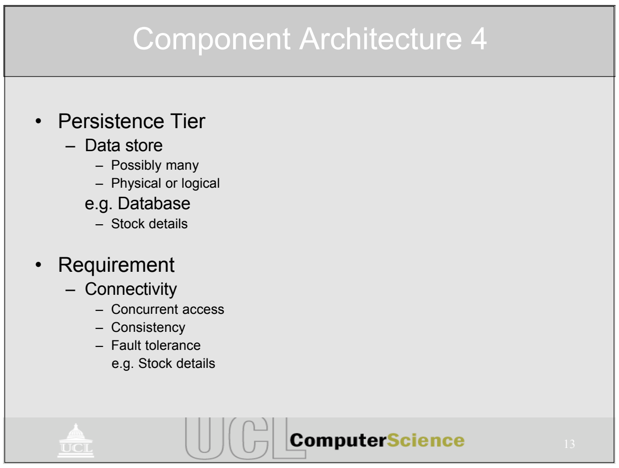
11



Component Architecture 3

- Business Object
  - Business logic
    - Validate data requests
    - Limit connectivity to the persistent storage
    e.g. EJB
    - Searching for an item
- Requirement
  - Data processing
    - Process non-persistent storage requests
    - Load balancing
    e.g. Stock search, credit card validation, gather data from multiple databases

12

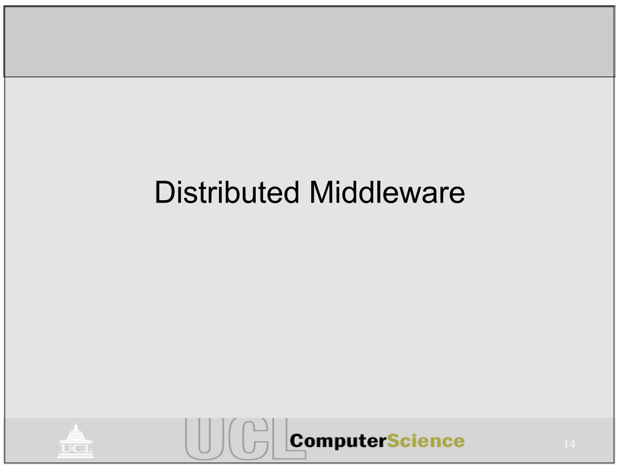


Component Architecture 4

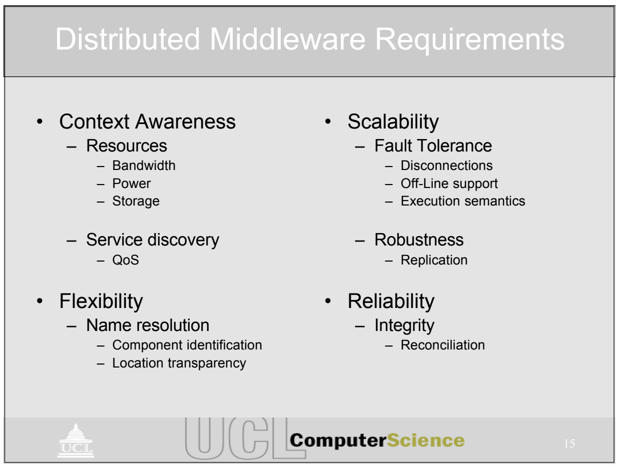
- Persistence Tier
  - Data store
    - Possibly many
    - Physical or logical

  e.g. Database
    - Stock details
- Requirement
  - Connectivity
    - Concurrent access
    - Consistency
    - Fault tolerance

    e.g. Stock details

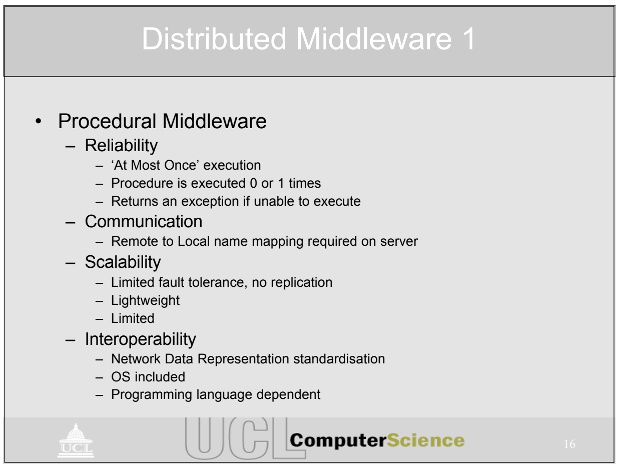


Distributed Middleware



Distributed Middleware Requirements

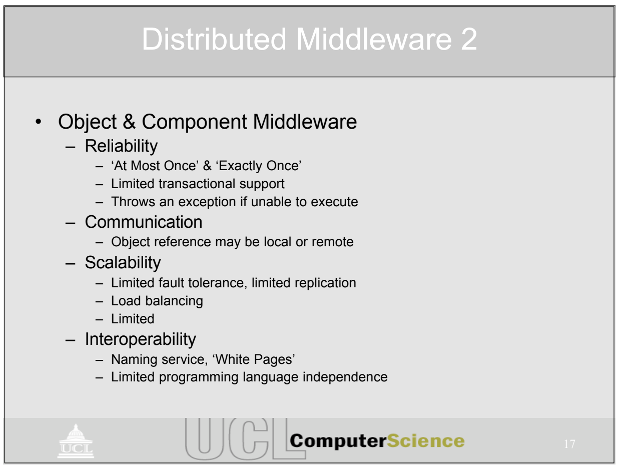
- Context Awareness
  - Resources
    - Bandwidth
    - Power
    - Storage
  - Service discovery
    - QoS
- Flexibility
  - Name resolution
    - Component identification
    - Location transparency
- Scalability
  - Fault Tolerance
    - Disconnections
    - Off-Line support
    - Execution semantics
  - Robustness
    - Replication
- Reliability
  - Integrity
    - Reconciliation



Distributed Middleware 1

- Procedural Middleware
  - Reliability
    - ‘At Most Once’ execution
    - Procedure is executed 0 or 1 times
    - Returns an exception if unable to execute
  - Communication
    - Remote to Local name mapping required on server
  - Scalability
    - Limited fault tolerance, no replication
    - Lightweight
    - Limited
  - Interoperability
    - Network Data Representation standardisation
    - OS included
    - Programming language dependent

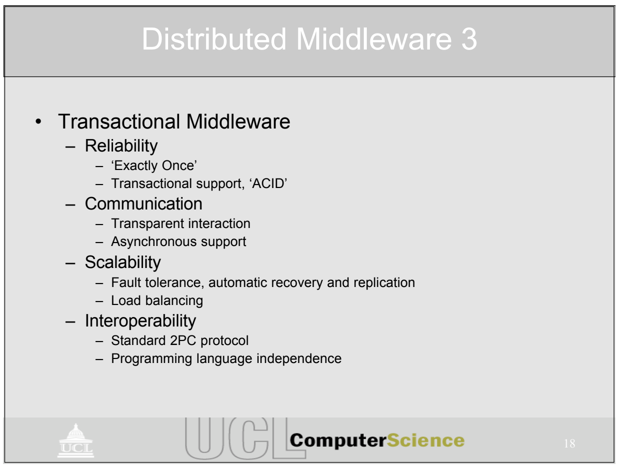
16



Distributed Middleware 2

- Object & Component Middleware
  - Reliability
    - ‘At Most Once’ & ‘Exactly Once’
    - Limited transactional support
    - Throws an exception if unable to execute
  - Communication
    - Object reference may be local or remote
  - Scalability
    - Limited fault tolerance, limited replication
    - Load balancing
    - Limited
  - Interoperability
    - Naming service, ‘White Pages’
    - Limited programming language independence

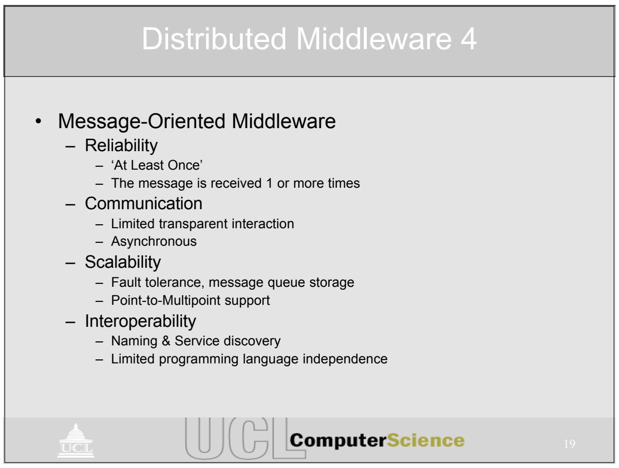
17



Distributed Middleware 3

- Transactional Middleware
  - Reliability
    - ‘Exactly Once’
    - Transactional support, ‘ACID’
  - Communication
    - Transparent interaction
    - Asynchronous support
  - Scalability
    - Fault tolerance, automatic recovery and replication
    - Load balancing
  - Interoperability
    - Standard 2PC protocol
    - Programming language independence

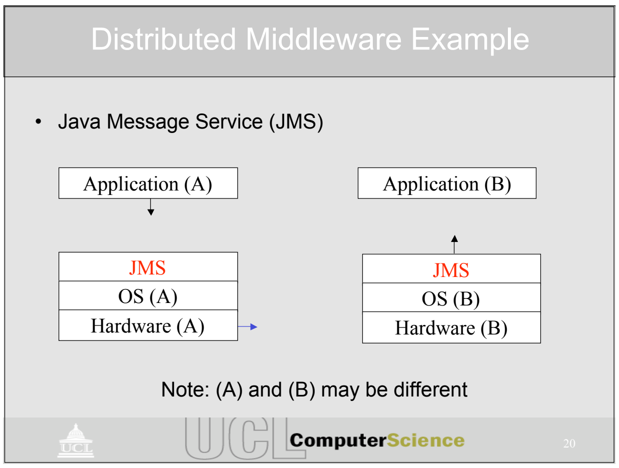
18



Distributed Middleware 4

- Message-Oriented Middleware
  - Reliability
    - 'At Least Once'
    - The message is received 1 or more times
  - Communication
    - Limited transparent interaction
    - Asynchronous
  - Scalability
    - Fault tolerance, message queue storage
    - Point-to-Multipoint support
  - Interoperability
    - Naming & Service discovery
    - Limited programming language independence

UCL ComputerScience 19

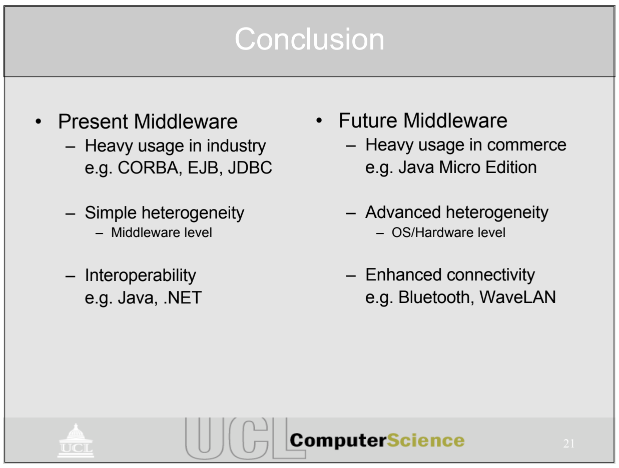


Distributed Middleware Example

- Java Message Service (JMS)

Note: (A) and (B) may be different

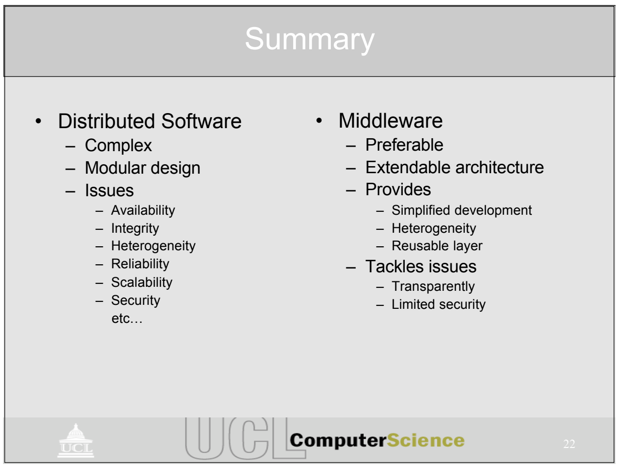
UCL ComputerScience 20



Conclusion

- Present Middleware
  - Heavy usage in industry
    e.g. CORBA, EJB, JDBC
  - Simple heterogeneity
    - Middleware level
  - Interoperability
    e.g. Java, .NET
- Future Middleware
  - Heavy usage in commerce
    e.g. Java Micro Edition
  - Advanced heterogeneity
    - OS/Hardware level
  - Enhanced connectivity
    e.g. Bluetooth, WaveLAN

UCL ComputerScience 21

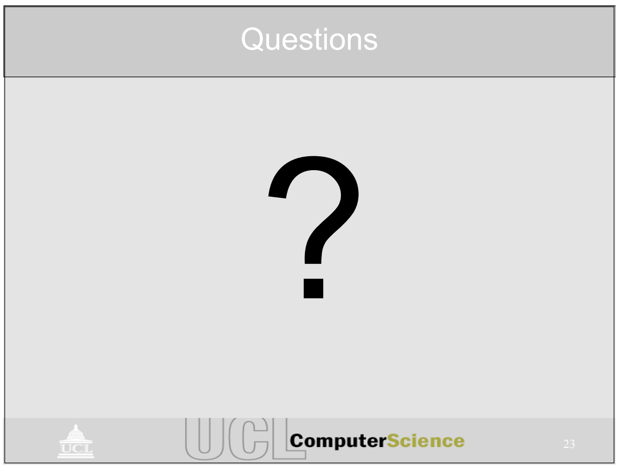


Summary

- Distributed Software
  - Complex
  - Modular design
  - Issues
    - Availability
    - Integrity
    - Heterogeneity
    - Reliability
    - Scalability
    - Security

    etc…
- Middleware
  - Preferable
  - Extendable architecture
  - Provides
    - Simplified development
    - Heterogeneity
    - Reusable layer
  - Tackles issues
    - Transparently
    - Limited security

UCL ComputerScience 22



Questions

?

UCL ComputerScience 23