

## Distributed Computing: Chapter 6

Doug Tygar  
IS 206  
21 September 1999

## Complexity

- A system that cannot be understood in all its detail by a single person or small group of people is **complex**
- The intricacy of the logic embodied in software
  - suffers no physical limitations
  - complexity is a primary limitation
  - advances allow us to extend that complexity

## Some sources of complexity

- Problem domain is complex
- Top-down design
  - as opposed to independent actors in the economy
- Software is not adaptable like people
- Large team efforts required
- Integration of heterogeneous suppliers

## Caution

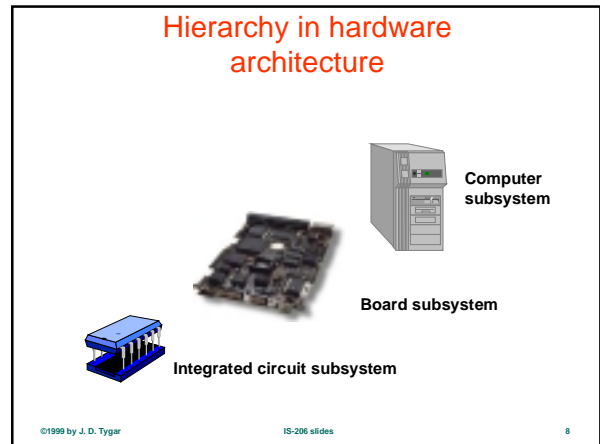
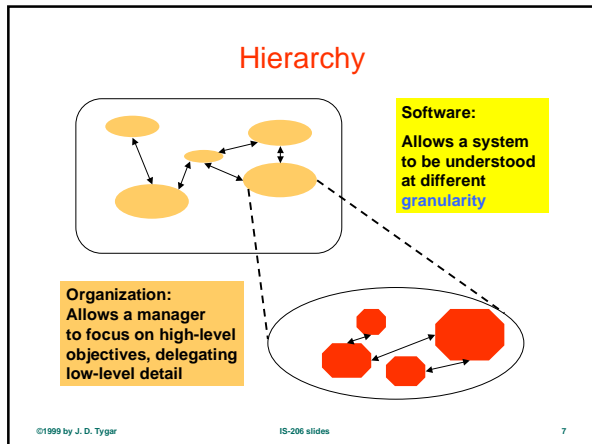
- Applications in course are relatively simple
- We have addressed
  - only the top of the hierarchy
  - ignored details
  - essence of hierarchical design: make complex appear simple

## Some solutions to complexity

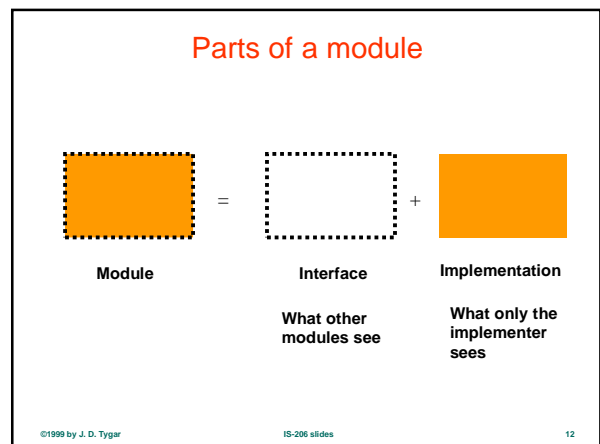
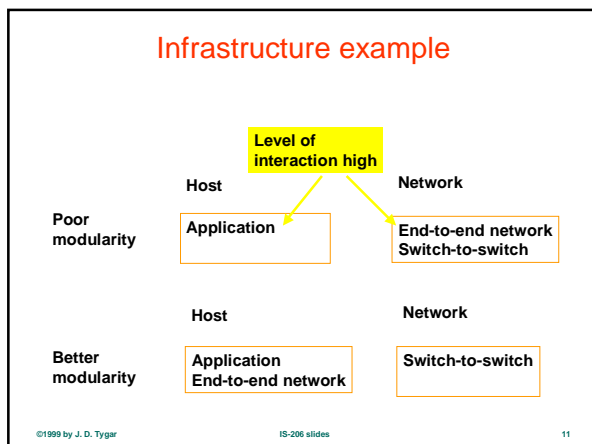
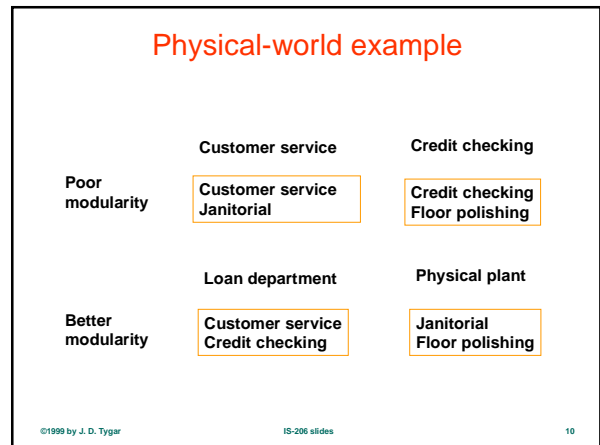
- Modularity properties
  - separation of concerns
  - reuse
- Interoperability through interfaces
  - abstraction
  - encapsulation

## Modularity

- Modular system is divided into subsystems (called modules) with known properties
  - Modules have distinct functional groupings
  - Hierarchy supports views at different granularity and scale
  - Separation of concerns among modules
  - Reusability of some modules



- ### Separation of concerns
- **Module desiderata:**
  - **Designed and implemented independently as possible**
  - **The level of interaction**
    - may be internally high
    - should be externally low
  - **They can then be assigned to different groups or companies for design**
    - minimum coordination costs
- ©1999 by J. D. Tygar IS-206 slides 9



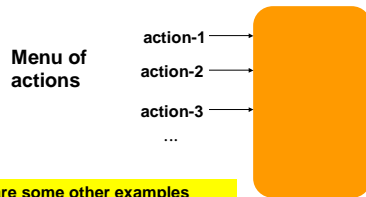
## Interfaces

- Focus of module interaction and interoperability
- Two purposes:
  - Informs other modules how to interact
  - Informs implementer about what has been promised to other modules

## Hardware interface

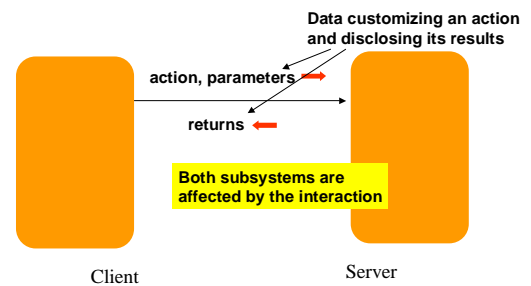
- Physical connection
- Electrical properties
- Formats of data passing through the interface (structure and interpretation)

## Possible software interface



What are some other examples of types of interaction at interfaces?

## Module interaction through interfaces



## Data types

- Data passing an interface is often specified in terms of a limited number of standard **data types**
- Data type = range of values and allowable manipulation
- Data type does *not* presume a specific representation, to allow heterogeneous platforms
  - Representation must be known when data passes a specific module interface

## Example data types

- Integer
  - "natural number between -32,767 and +32,768"
  - Could be represented (in many ways) by 16 bits
    - since  $2^8 = 65,536$
- Float
  - "number of the form  $m \cdot 10^{\frac{n}{32768}}$ , where  $m$  is in the range -32,767 to +32,768 and  $n$  is in the range -255 to +256"
  - Could be represented by  $16+8 = 24$  bits

## More data types

- **Character**
  - “values assuming a-z and A-Z plus space and punctuation marks”
    - could be represented by 7 or 8 bits
- **Character string**
  - “collection of  $n$  characters, where  $n$  is customizable”
    - could be represented by  $7 * n$  bits

## Compound data types

- **Programmer-defined composition of basic data types**

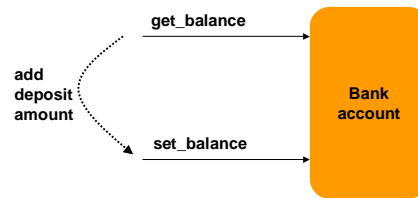
- **Example:**

```
Employee {
  String name;
  String address;
  Integer year_of_birth;
  etc.
}
```

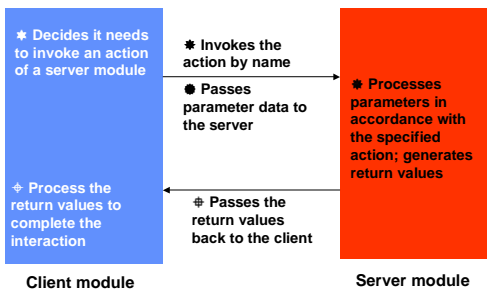
## Protocols

- A defined **sequence** of actions between/among two or more subsystems required to achieve some higher-level functionality
- Interface specification focuses on **actions** (including formats of parameters and returns) and **protocols**

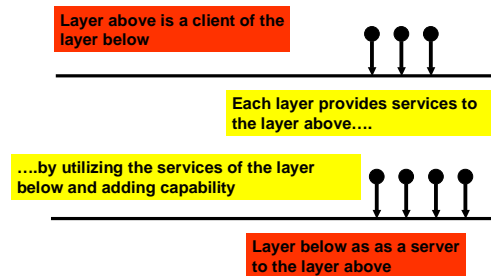
## Example protocol: deposit

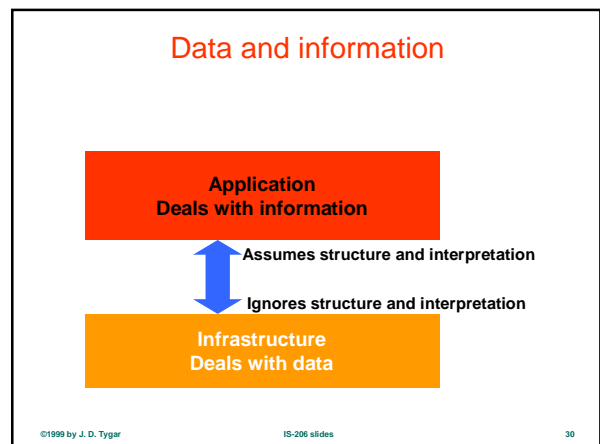
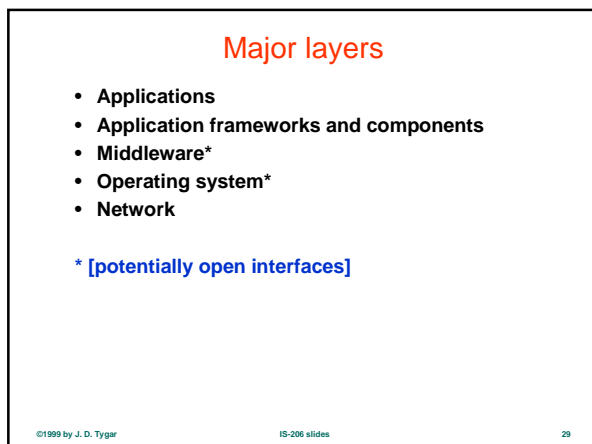
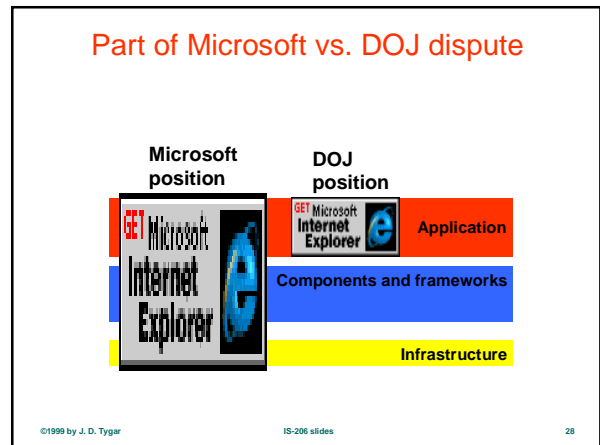
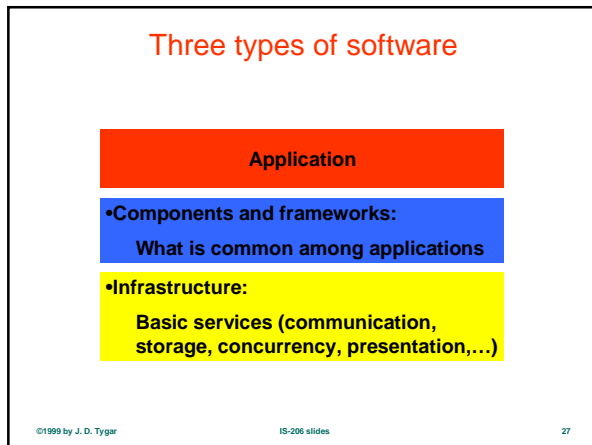
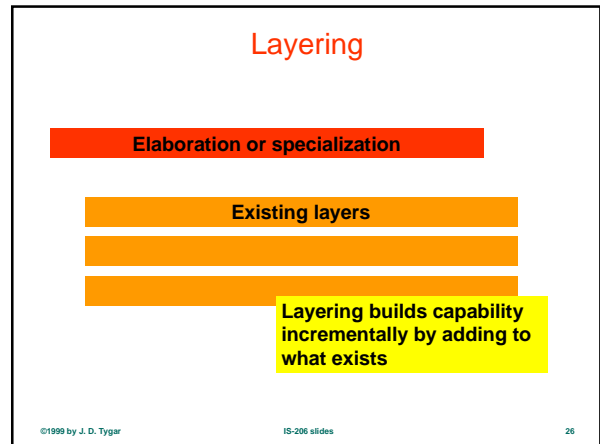
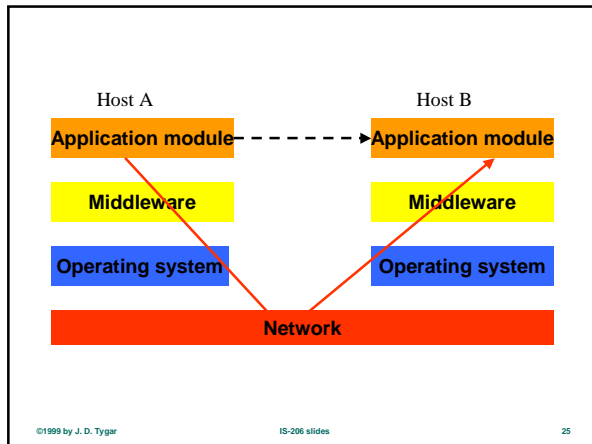


## Anatomy of an action invocation



## Interaction of layers





## Data and information in layers

- **Infrastructure deals with data**
  - Or at most minimal structure and interpretation of data suitable for a wide range of applications
- **Application adds additional structure and interpretation**
- **Yields a separation of concerns**

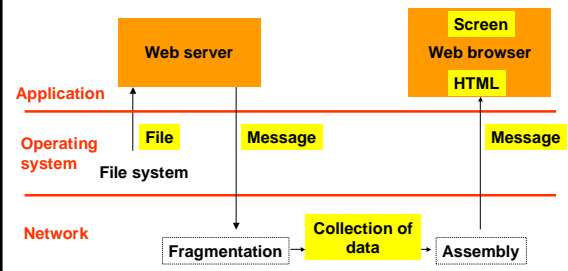
## Data

- In the simplest case, the infrastructure deals with a set of **data**
  - collection of bits
  - specified number and ordering
- **Objective of infrastructure: store and communicate packages while maintaining data integrity**
- **File for storage, message for communication**

## Data integrity

- **Retain the**
  - values
  - order
  - number**of bits in a package**

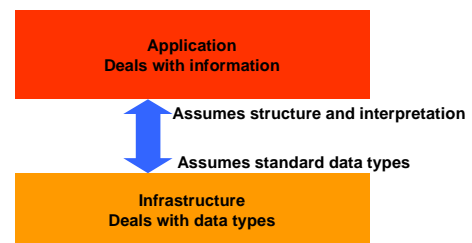
## Example

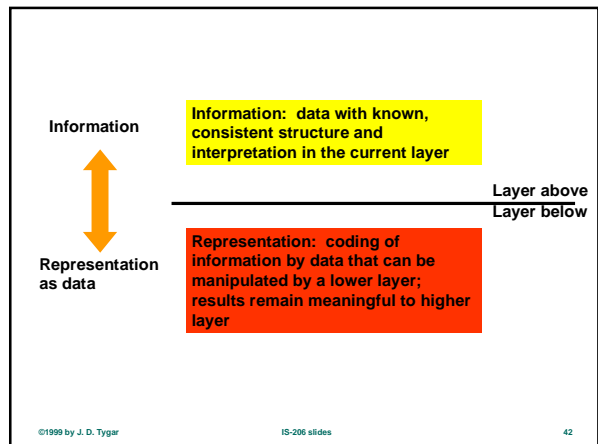
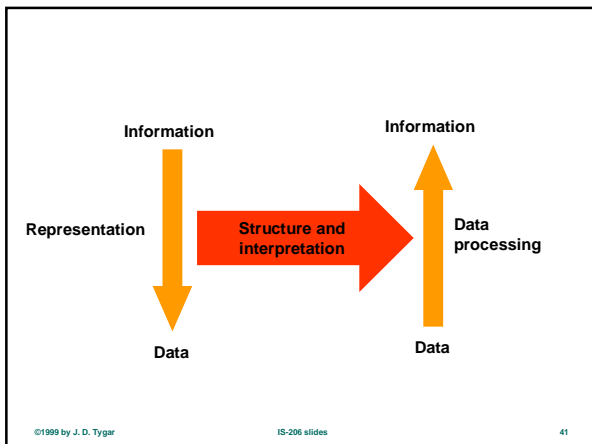
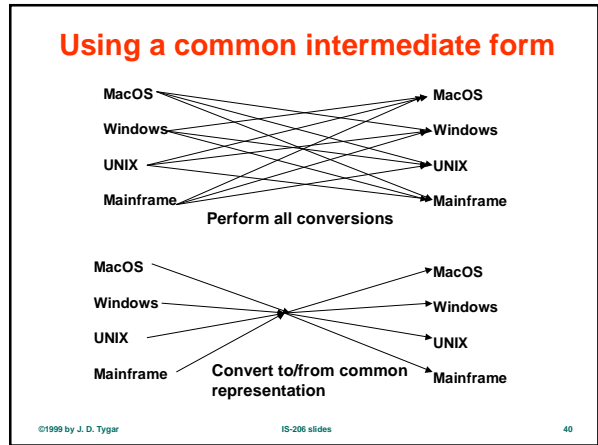
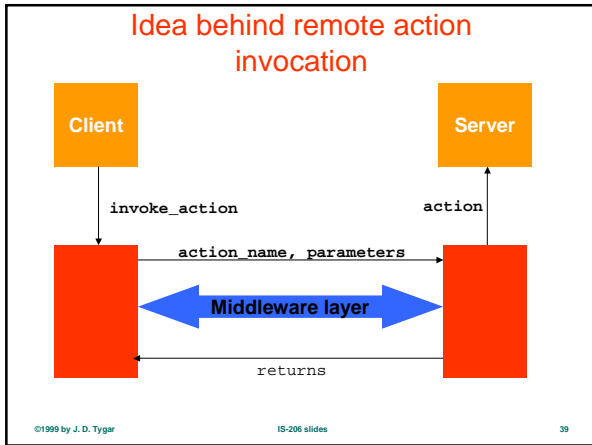
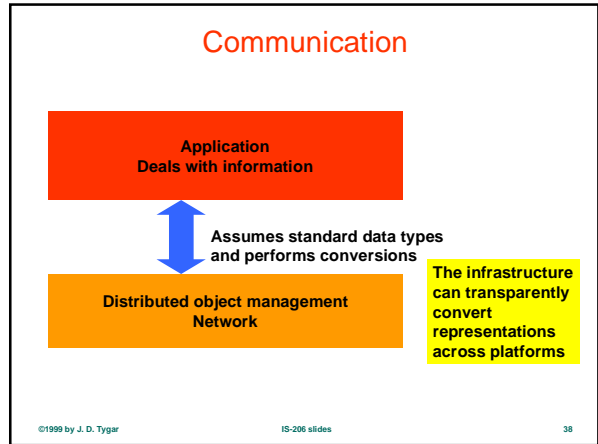
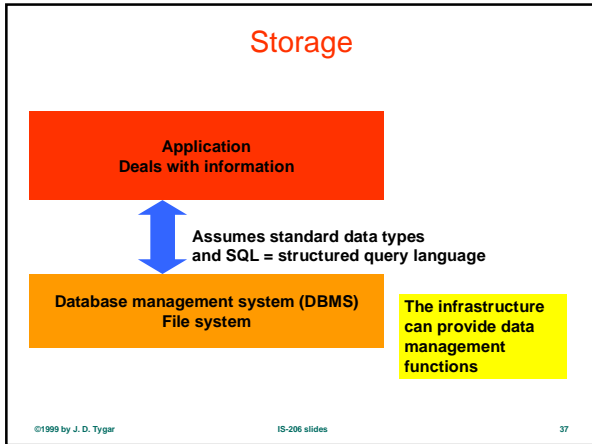


## Information in the infrastructure

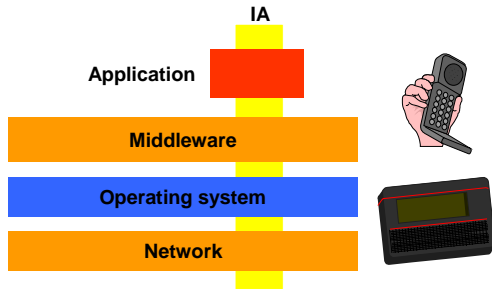
- **Sometimes appropriate for infrastructure to assume structure and interpretation**
  - to add capabilities widely useful to applications
  - to help applications deal with heterogeneous platforms, where representations differ
- **At most, data types**

## Data and information





## Information appliances



©1999 by J. D. Tygar

IS-206 slides

43

## Question

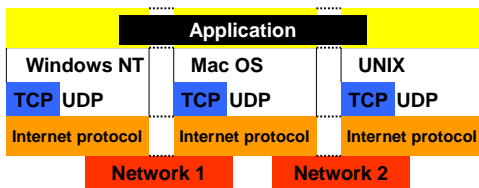
- What advantages and disadvantages do you see for the information appliance?

©1999 by J. D. Tygar

IS-206 slides

44

## Horizontal structure in layers

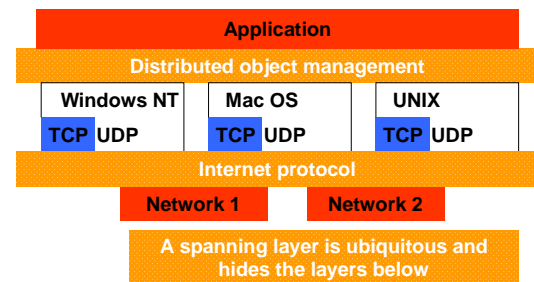


©1999 by J. D. Tygar

IS-206 slides

45

## Spanning layer



©1999 by J. D. Tygar

IS-206 slides

46

## Abstraction

- A property of well-designed interfaces to modules
- Hide detail, displaying only what is necessary
- Simplify, displaying only what is meaningful to the outside
- Important for complexity management

©1999 by J. D. Tygar

IS-206 slides

47

## Encapsulation

- Module implementation details (anything not explicit at interface) should be inaccessible from the outside
  - So other modules cannot become inadvertently dependent on implementation
  - In the case of components, for proprietary or security reasons

©1999 by J. D. Tygar

IS-206 slides

48

## Modularity key concepts

- **Divide and conquer:** decomposition of the system into modules with well-defined functional groupings
- **Separation of concerns:** great dependency internally, little dependency externally
- **Abstraction:** hide detail and simplify
- **Encapsulation:** make internal implementation inaccessible
- **Reusability:** meet generalized needs, configurable