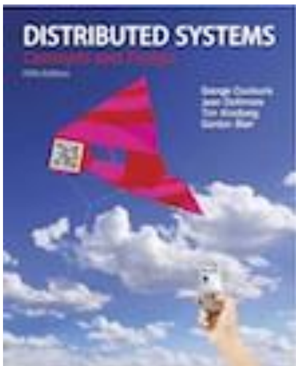
Slides for Chapter 2:
System Models (Part 1 of 2)



From **Coulouris, Dollimore, Kindberg and Blair**

Distributed Systems: Concepts and Design

Edition 5, © Addison-Wesley 2012

Text extensions to slides © David E. Bakken, 2012

Introduction [2.1]

- Real-world systems should (ideally) be designed to function in widest possible range of circumstances (incl. difficulties and threats)
- Chap2: how properties and design issues of DSs can be captured and analyzed with descriptive models
  - **Physical models:** HW composition of computers (and devices) and networks that interconnect them
  - **Architectural models**: describe w.r.t. computational tasks done by computational elements (single or aggregate) connected by networks
  - **Fundamental models**: abstract perspective examining an individual aspect of a distributed system
    - **Interaction models** (struct+seq of elements' comms), **failure models**, **security models**

© 2012 David E. Bakken

Difficulties and Threats for DSs

- Many problems face designers of DSs!
- Widely varying modes of use
  - Workload
  - Some parts disconnected or with flaky connectivity
  - Some need high bandwidth and/or low latency
- Wide range of system environments
  - Heterogenieties discussed earlier
  - Networks vary widely in performance (statically and dynamically)
  - Scale from tens to millions of computers

© 2012 David E. Bakken

Difficulties and Threats for DSs (cont.)

- Internal problems
  - Non-synchronized clocks
  - Conflicting data updates
  - *Many* modes of HW+SW failure for individual components
- External threats: attacks on
  - **C**onfidentiality
  - **I**ntegrity
  - **A**vailabiilty (incl. DoS attacks)

© 2012 David E. Bakken

Physical Models [2.2]

- **Physical model**: representation of underlying HW in a DS that abstracts away specific details of techs (comp+net)
  - Baseline model (minimal): extensible set of computer nodes interconnected by a network that passes messages
  - Beyond this, 3 generations of DSs: early, internet-scale, contemporary
- Early DSs:
  - Late 70s and early 80s, when Ethernet came
  - Typically 10-100 nodes connected by a LAN, sharing files+printers
  - Internet: limited connectivity, low bandwidth; email, file transfer
  - Mostly homogeneous, openness not a concern (or known!)
  - QoS in its infancy (lotsa research started)

© 2012 David E. Bakken

Internet-Scale DSs

- Emerged in 1990s (google 1996): dramatic growth of Internet (broadband)
- Early DSs model extended to systematically expoit "network of networks" (internet)
- Large # nodes, global reach and use
- Significant heterogeneity
- Lead to open standards and middleware (started late 70s)
- QoS greatly improved
- Nodes typcailly
  - Desktop computers
  - Discrete (not embedded within other physical entities)
  - Autonomous: endependent of other computers largely

© 2012 David E. Bakken

Contemporary DSs

- Mobile computing, ergo need service discovery and spontaneous interoperation
- Ubiquitous computing, ergo handle where computers are embedded in everyday objects and in surroundings
- Cloud computing and clusters: autonomous nodes ➔ cluster that provides a given service
- Result: huge increase in heterogeneity (all types)

© 2012 David E. Bakken

Figure 2.1
Generations of Distributed Systems

Distributed systems:	*Early*	*Internet-scale*	*Contemporary*
Scale	Small	Large	Ultra-large
Heterogeneity	Limited (typically relatively homogenous configurations)	Significant in terms of platforms, languages and middleware	Added dimensions introduced including radically different styles of architecture
Openness	Not a priority	Significant priority with range of standards introduced	Major research challenge with existing standards not yet able to embrace complex systems
Quality of service	In its infancy	Significant priority with range of services introduced	Major research challenge with existing services not yet able to embrace complex systems

© Pearson Education 2012

Distributed System-of-Systems (SoS)

- System (esp. software) organized into system of systems (analogy to internet: network of networks)
- Subsystems subsystems are almost independent systems (architecturally) assembled for a particular task
- Composition issues for QoS are huge (DARPA 90s, EC 2012)
- **Emergent properties**: when simple(r) subsystems form complex collective behaviors
  - Biological examples: flock of birds or school of fish
  - New and subtle behaviors emerge
  - Observable in many structures: hierarchies, decentralized (e.g., marketplace)
  - Key problem in SOSs (EC 2012)

© 2012 David E. Bakken

Architectural Models [2.3]

- Structure a system in terms of separately specified components and their relationships
- Goal: ensure structure meets present & (likely) future req.
- Concerns: reliability, manageability, adaptability, cost-effectiveness
- Three-phase buildup of concepts (*long* sub-chapter!)
  - Core underlying architectural elements [2.3.1]
  - Composite arch. patterns usable in isolation or combination [2.3.2]
  - Middleware platforms supporting programming styles emerging from [2.3.1] and [2.3.3]

© 2012 David E. Bakken

Architectural Elements [2.3.1]

Need to consider 4 key questions:

1. What **entities** are communicating in the DS?
2. What **communication paradigm**/pattern do entities use?
3. What **roles and responsibilities** do entities have
   - May change!
4. How are entities mapped onto physical infrastructure (**placement**)

© 2012 David E. Bakken

Communicating Entities

- System perspective: processes are communicating
  - Simple environments (sensors): no processes, so entities≡nodes
  - Most environments: threads, so technically the endpoints
- Programming perspective: more problem-oriented abstr.
  - Objects: coherent packaging of code+data, multiple instances
    - Problem-oriented abstractions, units of decomposition
    - Access via interfaces (spec. in IDL)
    - Distributed objects more in Chap 5, 8
  - Components
    - Similar to objects: code+data, interfaces
    - Also specify assumptions made (needed external components/interfaces) … i.e., dependencies made explicit … better "contract" for constructing systems
  - Web services (access objects/components via WWW)
    - Rather ugly underlying technologies at time

© 2012 David E. Bakken

Communication Paradigms

- 3 kinds: interprocess comm., remote invoc., indirect comm.
- **Interprocess communication** (IPC)
  - Low-level support for communication
  - Usually socket API

© 2012 David E. Bakken

Remote Invocation

- Most common (arguably), two-way exchange; buildup…
- **Request-reply protocols** (application level)
  - Pattern imposed on underlying message passing to support client-server
  - Client app code sends message with operation, params, bookeeping in request message
  - Server sends msg with bookkeeping, params in reply message
  - Low-level, typically simple embedded systems w/strong RT needs
- **Remote procedure call** (RPC)
  - Make a remote call look (almost) like a local call
  - Supports many transparencies and heterogeneities
  - Directly supports client-server computing at higher level than RRPs

© 2012 David E. Bakken

Remote Invocation

- **Remote method invocation (RMI)**
  - Extends procedural RPC to object-oriented programming
  - Multiple object instances: can pass object refs/IDs as params
  - Tighter integration than RPC into the language

© 2012 David E. Bakken

Decoupled communication

- IPC, RRP, RPC, RMI all have explicit receivers/endpoints for each direction of comm
  - Senders must know receivers IDs; receivers often know senders
  - Sender and receiver must both exist at same time
  - Can be less flexible than desirable for some apps
- **Space uncoupling**: senders do not need to know who sending to
- **Time uncoupling**: senders and receivers don't have to have overlapping lifetimes (exist at same time)
- Uncouplings support **indirect communication** (Chap 6)

© 2012 David E. Bakken

Overview of Indirect Communication Techniques

- **Group communication**
  - 1:many comms with group ID
  - Recipients join group, senders send to group
  - Groups often maintain membership, handle member failures
  - IP multicast trivial example, but many more fancier ones
- **Publish-subscribe**
  - Producers (publishers) send out info, publishers get it
  - Intermediate service is in between
  - Can subscribe based on data: topics

© 2012 David E. Bakken

Overview of Indirect Communication Techniques (cont.)

- **Message queues**
  - Senders send to a specific queue, point-to-point
  - Consumers can get from queue (or be notified if new items)
- **Tuple spaces**
  - Structured data: (int, float, string, …) with a given signature
  - Processes can read or remove tuples, can match values of some/all fields in tuple
- **Distributed shared memory (DSM)**
  - Abstraction of a shared address space or data structures therein
  - Lots of research in the late 80s and 90s, died out mostly

© 2012 David E. Bakken

Figure 2.2
Communicating entities and communication paradigms

Communicating entities (what is communicating)		*Communication paradigms (how they communicate)*		
System-oriented entities	*Problem-oriented entities*	*Interprocess communication*	*Remote invocation*	*Indirect communication*
Nodes	Objects	Message passing	Request-reply	Group communication
Processes	Components	Sockets	RPC	Publish-subscribe
	Web services	Multicast	RMI	Message queues
				Tuple spaces
				DSM

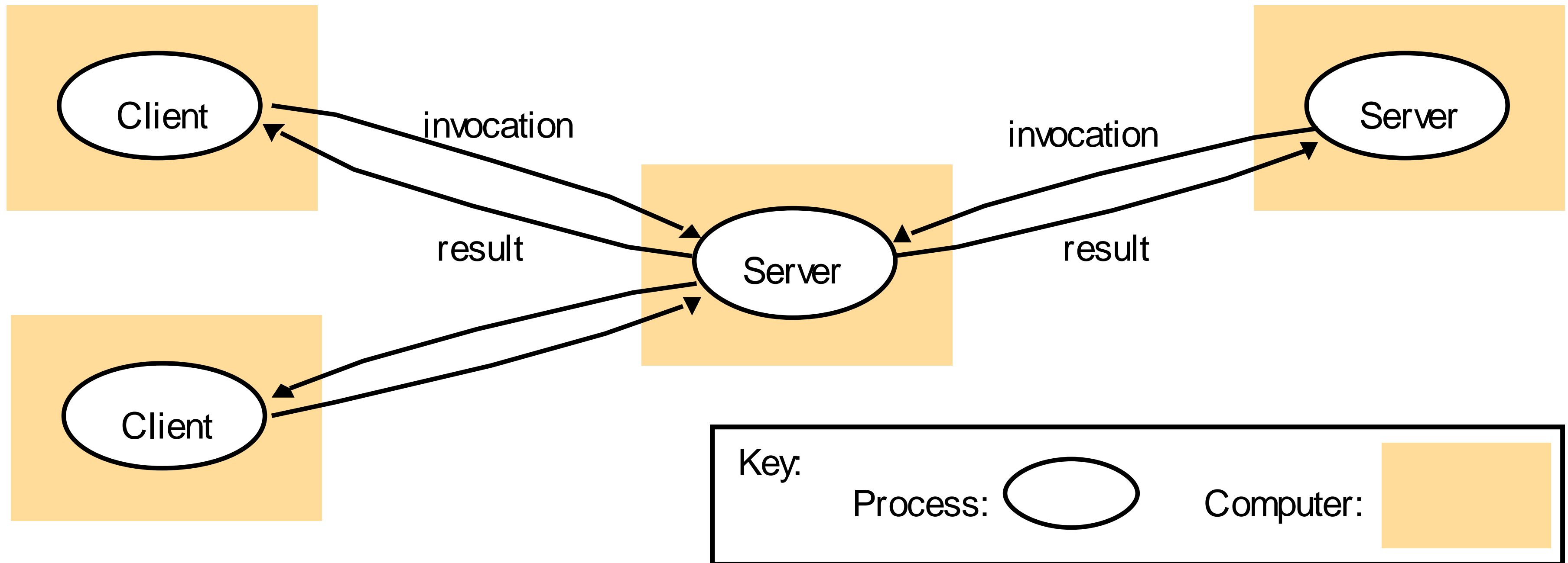
© Pearson Education 2012

Roles and Responsibilities

- Issue: what role does a given entity take
- **Client-server**
  - Most widely studied and deployed
  - Client sends request to server, which replies
  - Can be either RPC or RMI
  - C/S w.r.t a given interaction: A→B→C means B client and server
- **Peer-to-peer** (P2P): scales better, no centralized service
  - Observation: use not (just) centralized servers from a service, but end user can support that service (plenty of resources at edges!)
  - All entities are equals (and none/few “more equal than others”)
  - Entities run same program with same interfaces
  - Examples: BitTorrent, Skype, ..

© 2012 David E. Bakken

Figure 2.3
Clients invoke individual servers



© Pearson Education 2012

Placement

- How to map entities (objects, services, …) onto physical infrastructure
- Must take into account many things:
  - Patterns of communication
  - Reliability and current load of given machines
  - (Often) strong knowledge of application/service
- No optimal solutions, only strategies that help
  - Mapping services onto multiple servers
  - Caching
  - Mobile code
  - Mobile agents

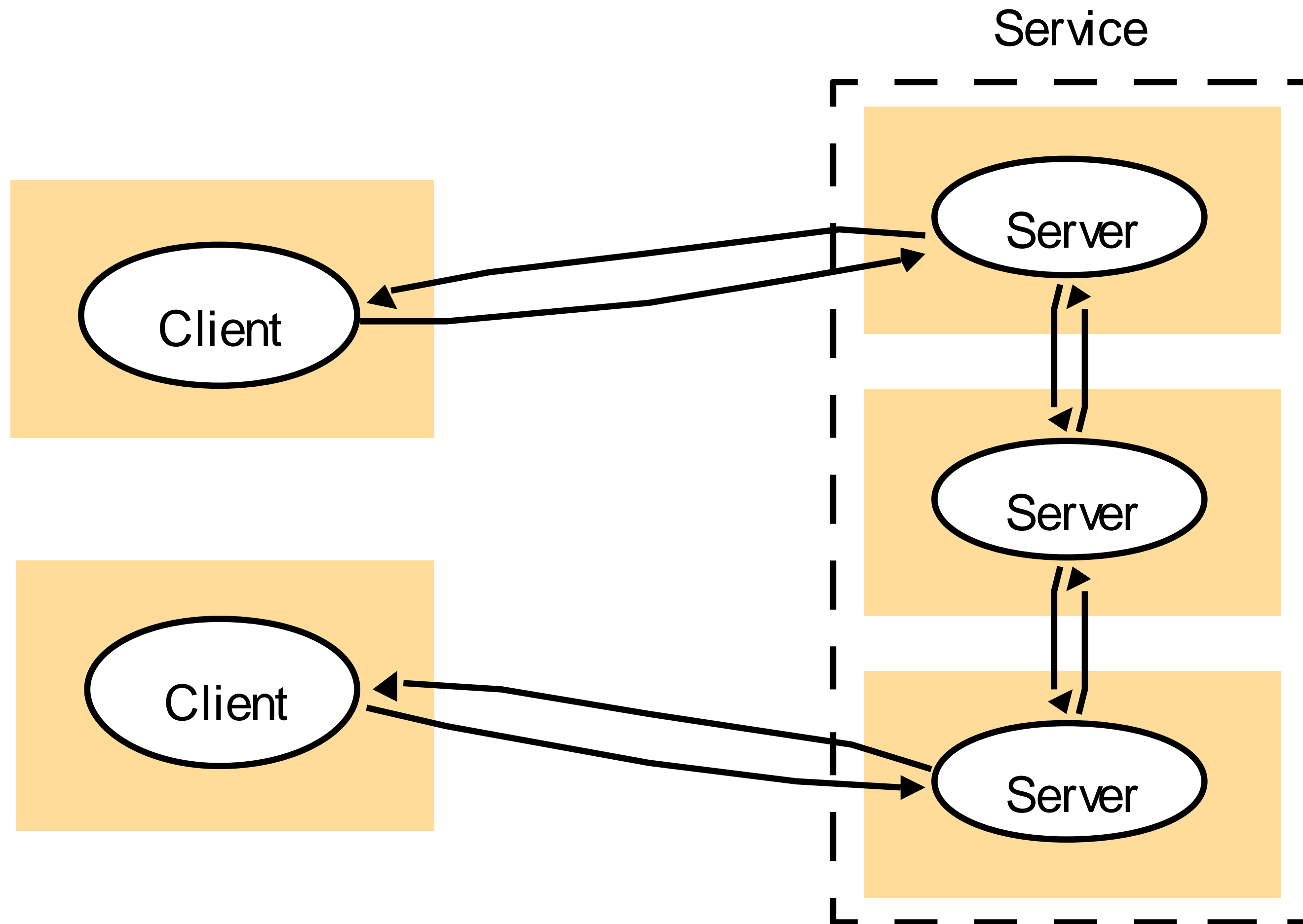
© 2012 David E. Bakken

Placement (cont)

- Mapping services to multiple servers (Fig 2.4)
- Caching
  - Cache: a store of recently used data objects closer or at a client
  - Examples?
  - Lotsa bookkeeping passed around to track updates/staleness/etc
  - If client requests stale object, it is fetched
- Mobile code
  - Applets …. And client-side (edge) resources usually plentiful
- Mobile agents
  - Agent: a running program (code+data) that travels to carry out a task for some entity, and returns results
  - Difference from mobile code?

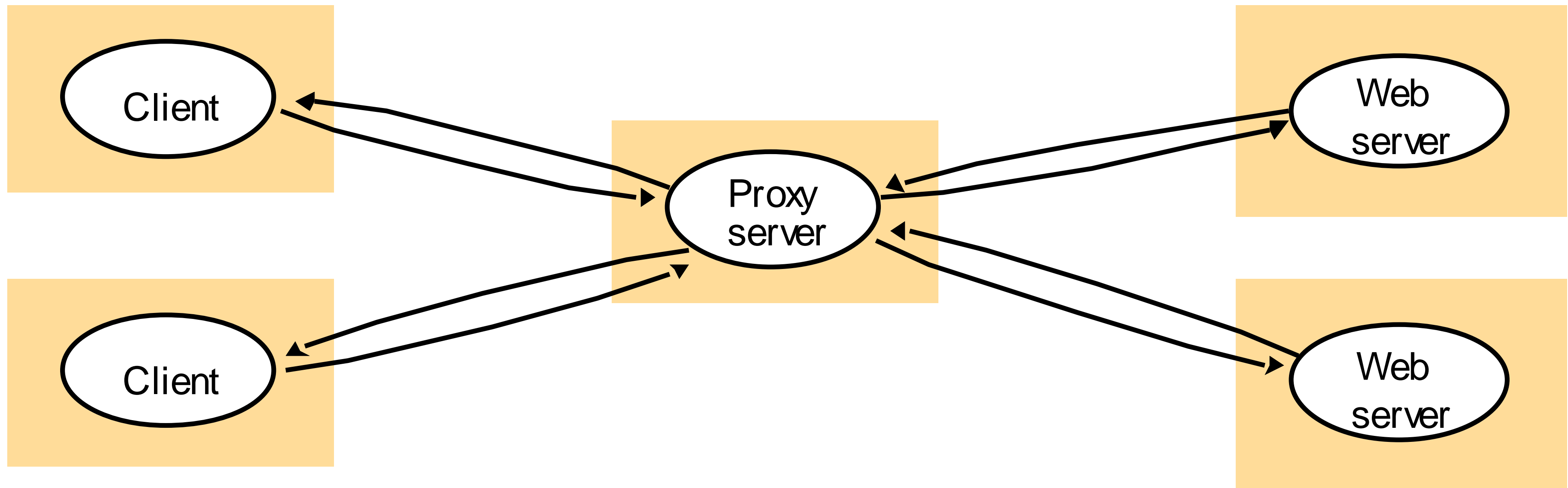
© 2012 David E. Bakken

Figure 2.4
A service provided by multiple servers (servers are P2P)



© Pearson Education 2012

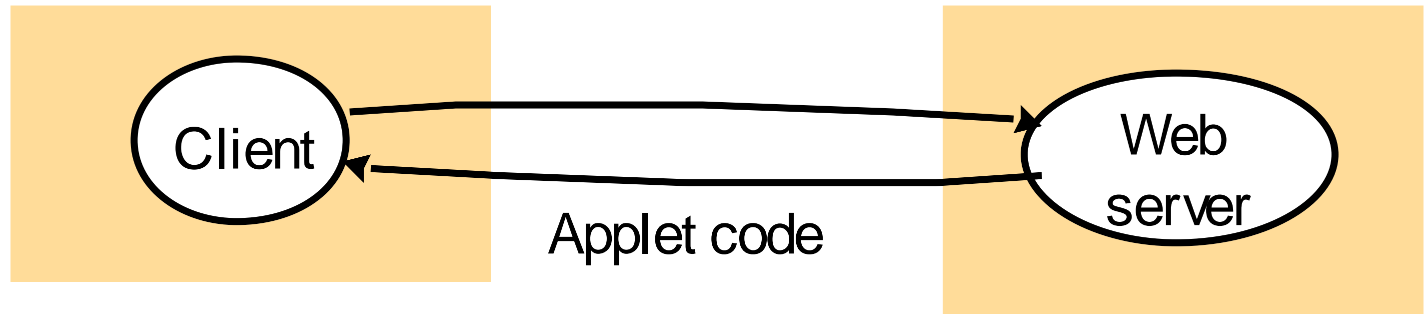
Figure 2.5
Web proxy server



© Pearson Education 2012

Figure 2.6
Web applets

a) client request results in the downloading of applet code



b) client interacts with the applet

© Pearson Education 2012

Architectural Patterns [2.3.2]

- Build on more primitive architectural elements in [2.3.1] and before
- “not themselves necessarily complete solutions but rather offer partial insights that, when combined with other patterns, lead the designer to a solution for a given problem domain”.
  - Extremely nice definition, lots of issues behind it!
- Patterns we cover
  - Layering
  - Tiered architectures
  - Thin clients
  - Other misc: proxy, brokerages, reflection

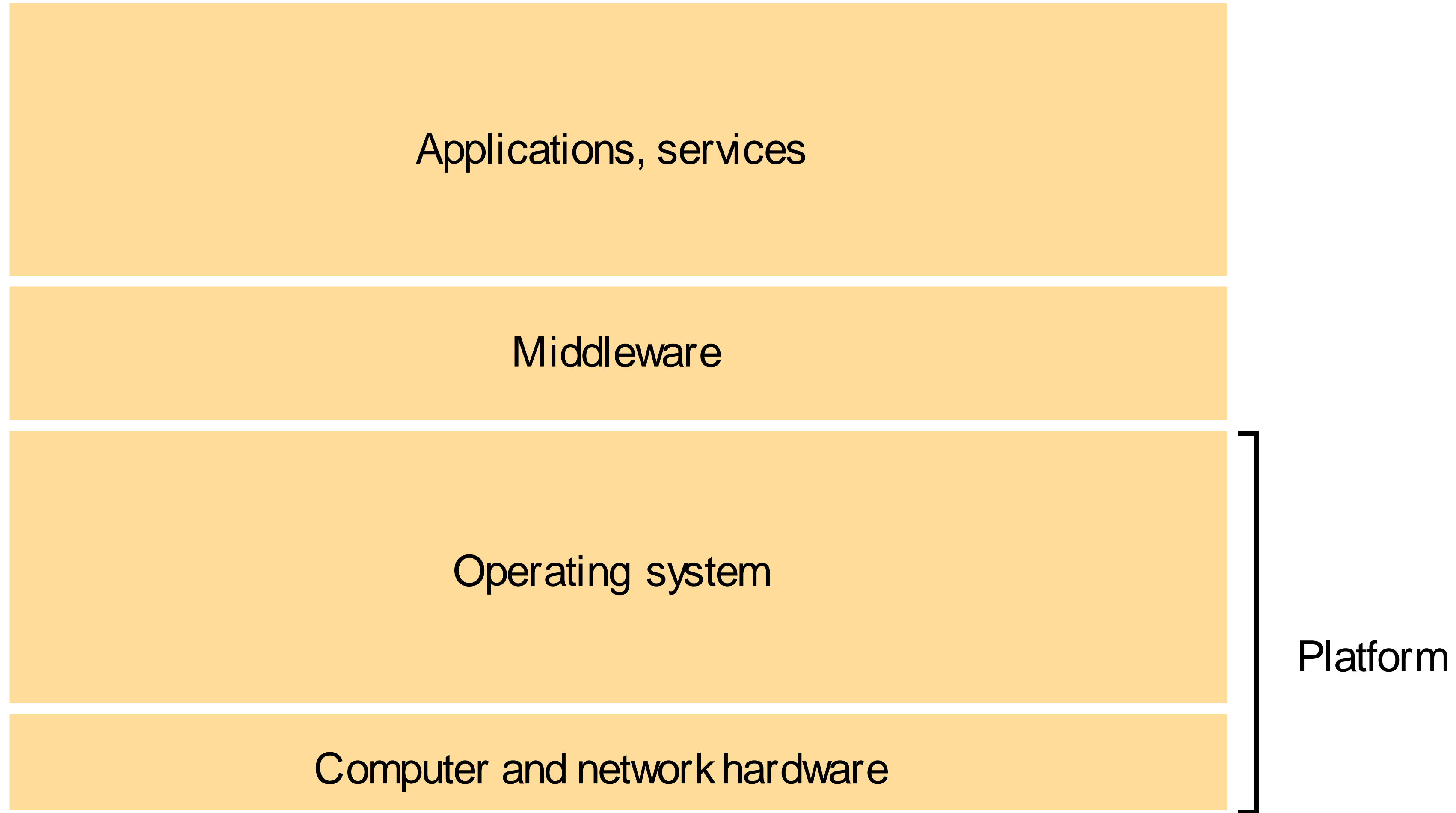
© 2012 David E. Bakken

Layering

- Familiar from networking design
- In a DS, means a vertical organization of services into service layers
- **Platform**: lowest-level HW and SW layers
- **Middleware**: layer(s) of software above platform
  - masking heterogeneities
  - Providing higher-level programming abstraction
    - much closer to application's items of domains than the platform
  - Supports different kinds of interactions: RCP, RMI, pub-sub, …

© 2012 David E. Bakken

Figure 2.7
Software and hardware service layers in distributed systems



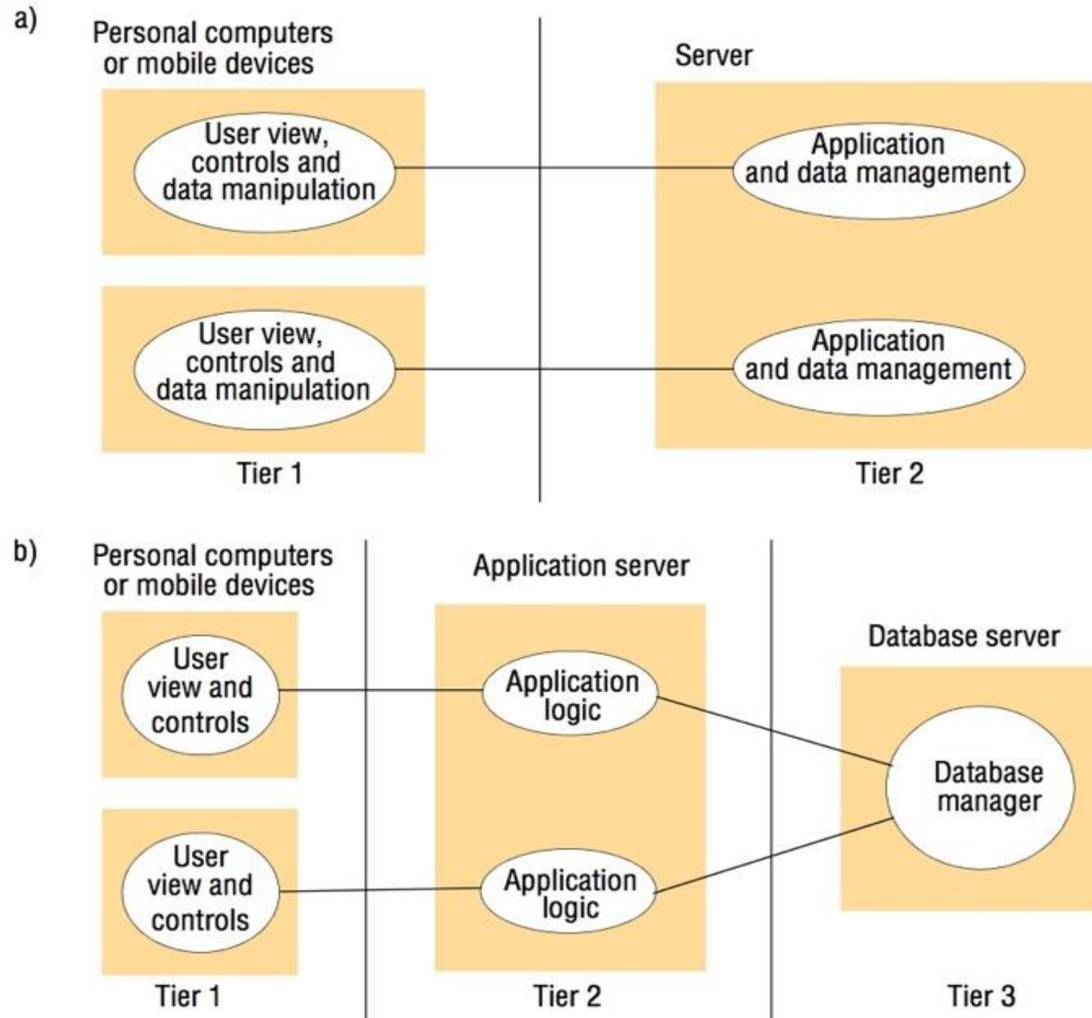
© Pearson Education 2012

Tiered Architectures

- Horizontal organization of application/service functionality across different servers
- Typical **three-tiered architecture**:
  - **Presentation logic**: user interactions and visualization
  - **Application logic**: app-specific processing (AKA **business logic**)
  - **Data logic**: persistent storage of data (e.g., database)
  - Above on separate processes
- Two-tiered can split above functionality across client-server in different ways
- (Read about AJAX, testable but not lecturing on)
- Q: tiered architectures contradictory or complimentary to layering?

© 2012 David E. Bakken

Figure 2.8
Two-tier and three-tier architectures



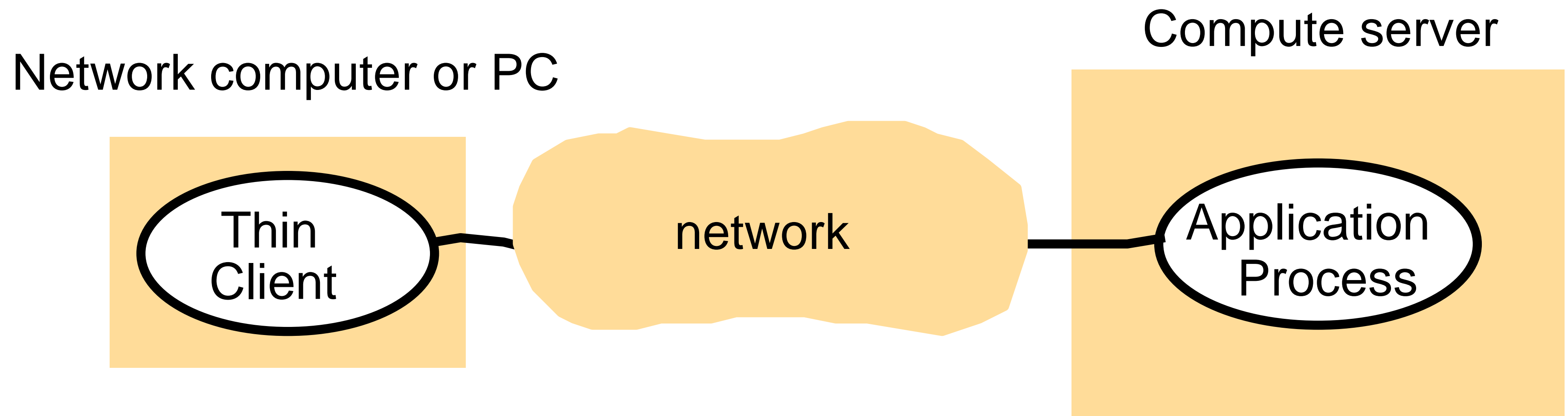
© Pearson Education 2012

Thin Clients & Other Patterns

- General-purpose desktop computer can be a pain to manage
- **Thin client**: SW layer supporting a window-based UI accessing remote programs and servers
- X-Windows early example
- Other architectural patterns
  - **Proxy**: intermediate in local address space (MW, web proxies)
  - **Brokerage**: **service broker** helps **service requester** find the right **service provider**
  - **Reflection**: applicaition/service utilizes knowledge of its internal structure; very very useful (Blair research)
    - **Introspection**: dynamic discovery of properties (read-only)
    - **Intercession**: dynamically modifying structure or behavior

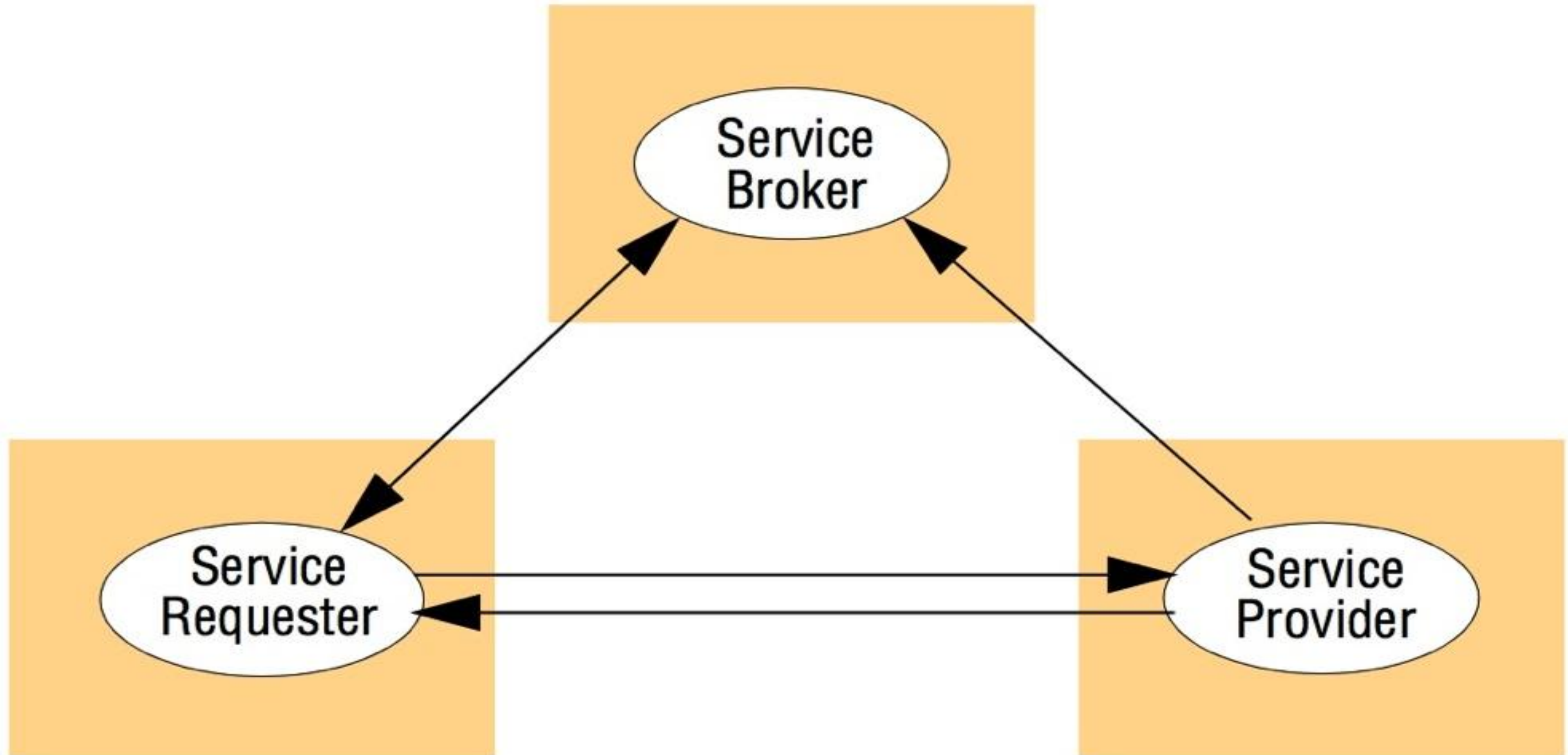
© 2012 David E. Bakken

Figure 2.10
Thin clients and compute servers



© Pearson Education 2012

Figure 2.11
The web service architectural pattern



© Pearson Education 2012

Associated Middleware Solutions [2.3.3]

- Categories: RPC, group communication, client-server, publish-subscribe, …..
- Limitations of middleware
  - Sometimes need application-specific knowledge for performance and reliability reasons
    - E.g., reliable email delivery on top of TCP/IP
  - Classic paper: end-to-end argument in system design [Saltzer et al 1984]: required for 564
    - Some comms-related functions can only be done right with app knowledge
    - So don't push those functions into the comms layer
  - Authors consider this a limitation of MW, I consider it an opportunity for MW (and good research done on it, e.g., DARPA Quorum …)
    - QoS-enabled, adaptive MW can really help here (BBN QuO)

© 2012 David E. Bakken

Figure 2.12
Categories of middleware (some overlap, more in Chap 8)

Major categories:	*Subcategory*	*Example systems*
Distributed objects (Chapters 5, 8)	Standard	RM-ODP
	Platform	CORBA
	Platform	Java RMI
Distributed components (Chapter 8)	Lightweight components	Fractal
	Lightweight components	OpenCOM
	Application servers	SUN EJB
	Application servers	CORBA Component Model
	Application servers	JBoss
Publish-subscribe systems (Chapter 6)	-	CORBA Event Service
	-	Scribe
	-	JMS
Message queues (Chapter 6)	-	Websphere MQ
	-	JMS
Web services (Chapter 9)	Web services	Apache Axis
	Grid services	The Globus Toolkit
Peer-to-peer (Chapter 10)	Routing overlays	Pastry
	Routing overlays	Tapestry
	Application-specific	Squirrel
	Application-specific	OceanStore
	Application-specific	Ivy
	Application-specific	Gnutella

© Pearson Education 2012