Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Chapter 1 – Motivation

Middleware

- Middleware
  - supports the development, deployment, and execution of complex information systems
  - facilitates interaction between and integration of applications

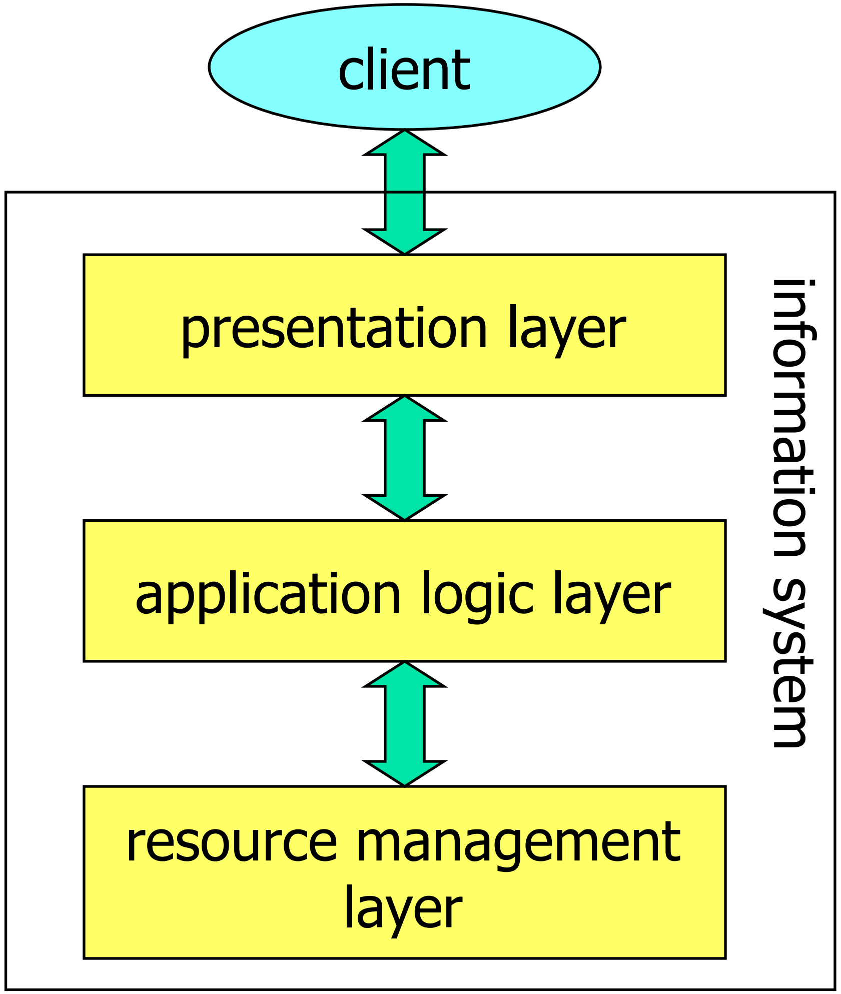
  across multiple distributed, heterogeneous platforms and data sources
- Two major aspects
  - middleware as a programming abstraction
  - middleware as infrastructure
- Principles
  - make distribution transparent
  - support standardized APIs/languages/data formats to overcome platform heterogeneity
  - application logic independent from infrastructure code
  - powerful programming abstractions

Transaction Processing (TP)

- TP application
  - collection of transaction programs
  - provides functions to automate a given business activity
  - typically interacts with an on-line user (on-line TP, OLTP)
- Transaction program
  - executes a number of steps/operations to implement a business function
    - accesses shared data (e.g., using a DBS)
    - may communicate with other programs/components
  - example: order processing on the internet
    1. user submits order request using a web browser
    2. web server routes the request to a transaction server
    3. transaction program is executed on the server process the order (involves accessing catalog tables, inserting into an order table and billing a credit card)
- Transaction
  - (effects of) executing a transaction program
  - with expected properties/guarantees for its steps/operations: ACID

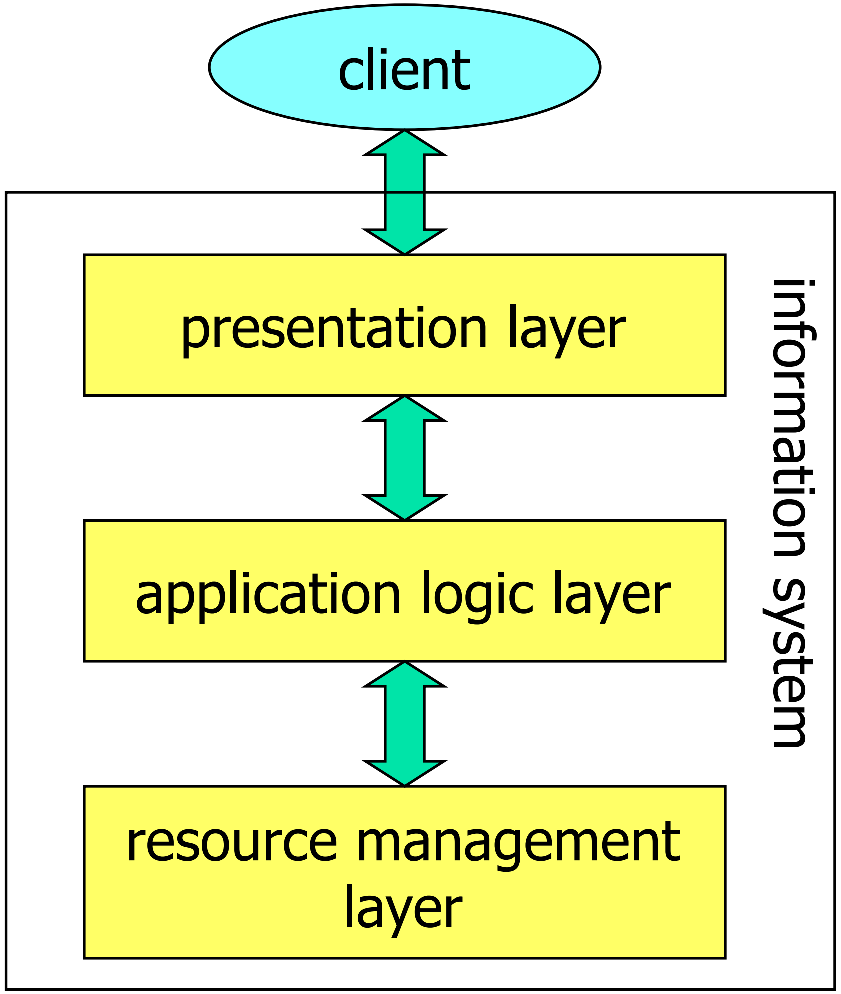
Layers of an Information System

- Separation of functionality into three conceptual layers
  - presentation
  - application logic
  - resource (e.g., data) management
- Architecture of an IS
  - layers can be combined and distributed in different ways
  - 1-tier, 2-tier, 3-tier, n-tier
- Challenges
  - distribution
  - autonomy
  - heterogeneity
  - performance & scalability
  - high availability
  - complexity
  - …



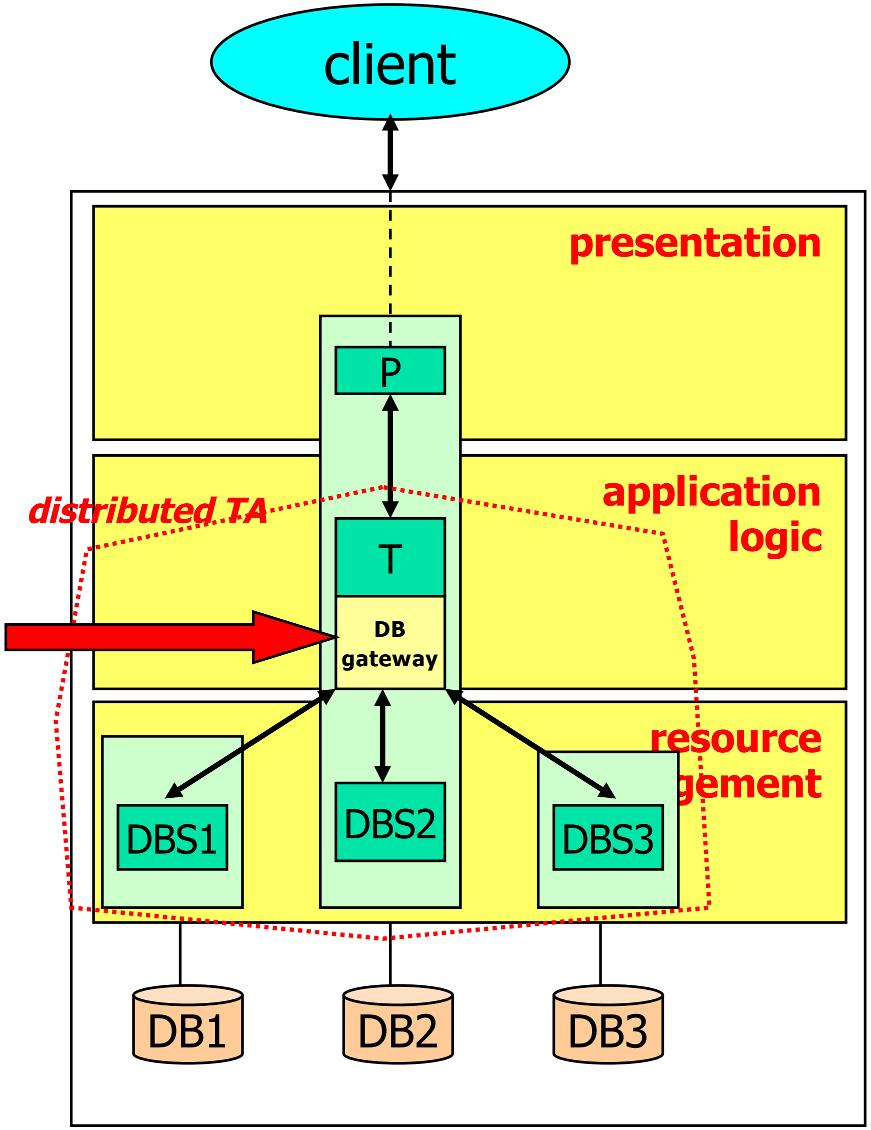
Chapter 2 – Distributed Information System

- Distributed Transactions for achieving global atomicity
  - 2PC, hierarchical 2PC
- Logical layers of an information system
  - presentation, application logic, resource management
- Design strategies
  - ideally top-down, but usually bottom-up (out of necessity)
- Architectures
  - 1-tier, 2-tier, 3-tier, n-tier
  - flexibility, distribution options vs. performance, complexity, manageability
- Distribution alternatives
  - units of distribution, pros and cons
- Communication
  - synchronous, asynchronous



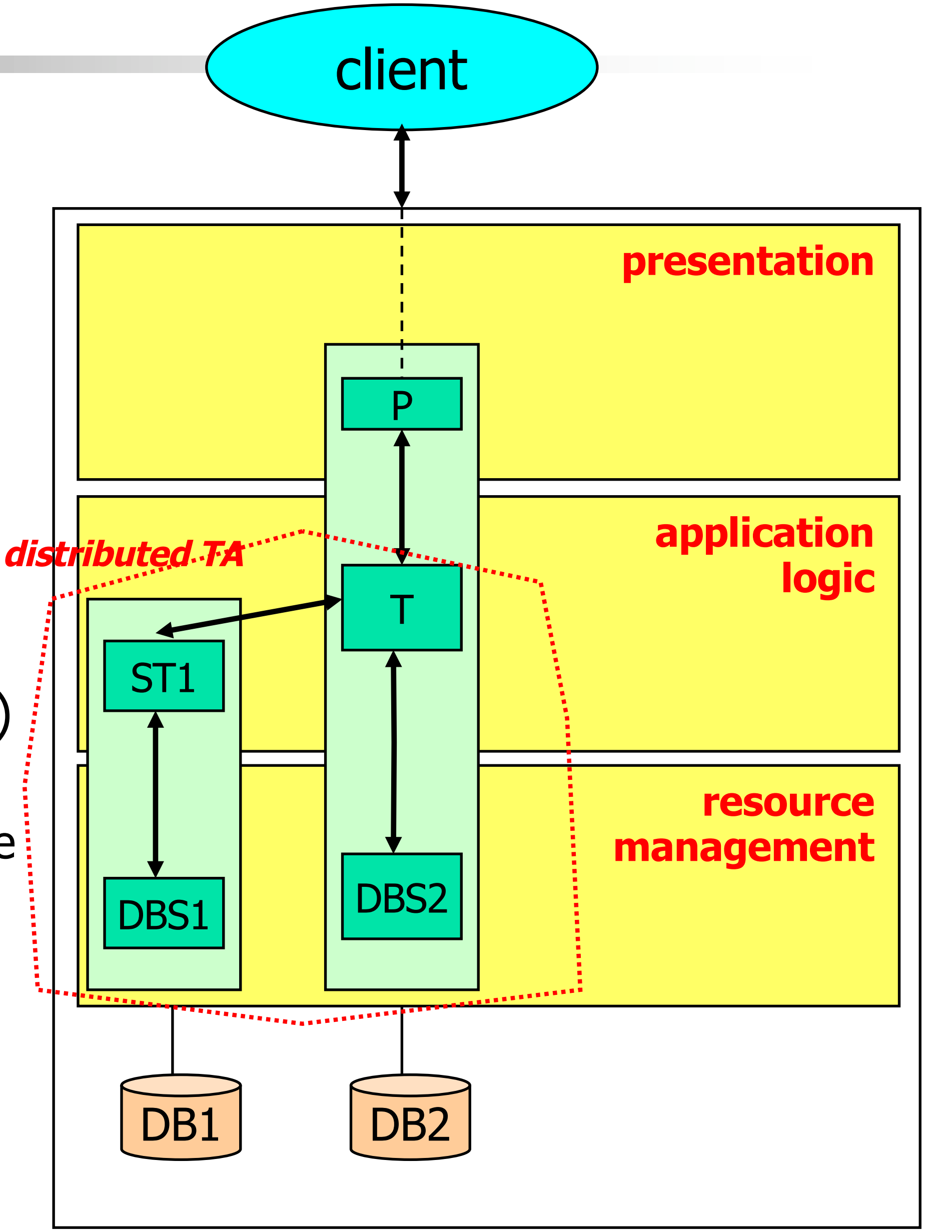
Chapter 3 - Database Gateways

- Goal: uniform database access in a distributed and heterogeneous environment
- Important requirements
  - Uniform Database Access API and language (SQL + ODBC/JDBC/…)
  - Dynamic, late binding to specific DB/DBS
  - Simultaneous access to multiple DB/DBMS
    - to the same DB
    - to different DBs
    - within the same (distributed) transaction
    - to multiple DBMS products
  - Support for DBMS vendor extensions



Chapter 4 – Remote Procedure Calls and Distributed Transactions

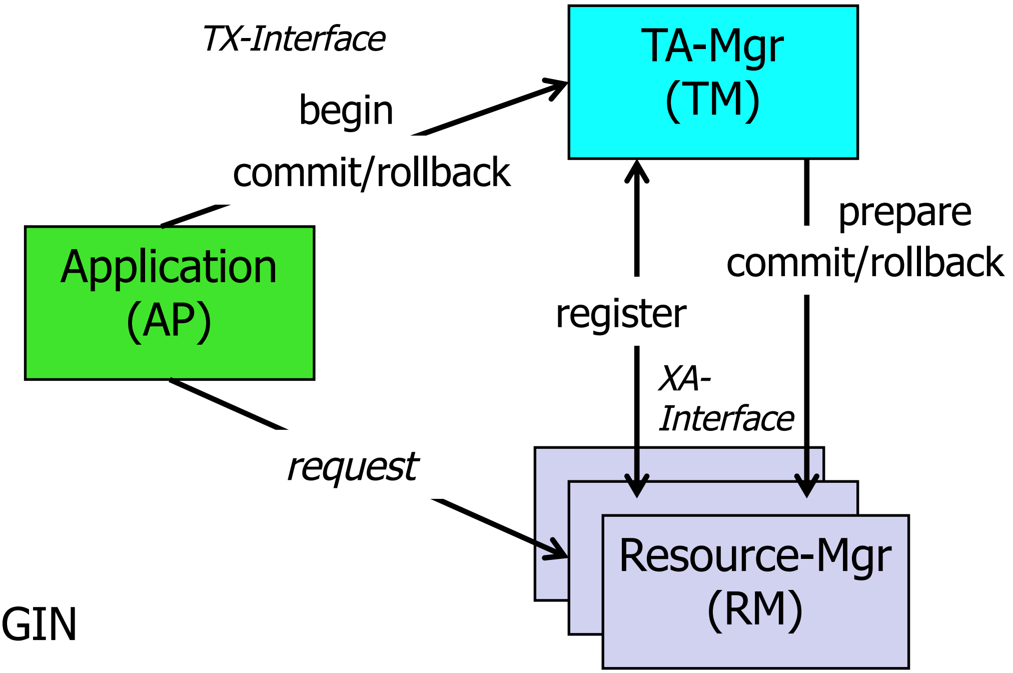
- Distributed (Information) System
  - consists of (possibly autonomous) subsystems
  - jointly working in a coordinated manner
- How do subsystems communicate?
  - **Remote Procedure Calls (RPC)**
    - transparently invoke procedures located on other machines
  - Messaging, Message Queuing (later)
- Transactional Support (ACID properties) for distributed processing
  - Server/system components are Resource Managers
  - (Transactional) Remote Procedure Calls (TRPC)
  - Distributed Transaction Processing



X/OPEN – Standard for Distributed TA Processing

- Resource Manager
  - recoverable
  - supports external coordination of TAs using 2PC protocol (XA-compliant)
- TA-Mgr
  - coordinates, controls RMs
- Application Program
  - demarcates TA (TA-brackets)
  - invokes RM services
    - e.g., SQL-statements
  - in distributed environment: performs (T)RPCs
- Transactional Context
  - TRID generated by TA-Mgr at BEGIN
  - established at the client
  - passed along (transitively) with RM-requests, RPCs

local environment



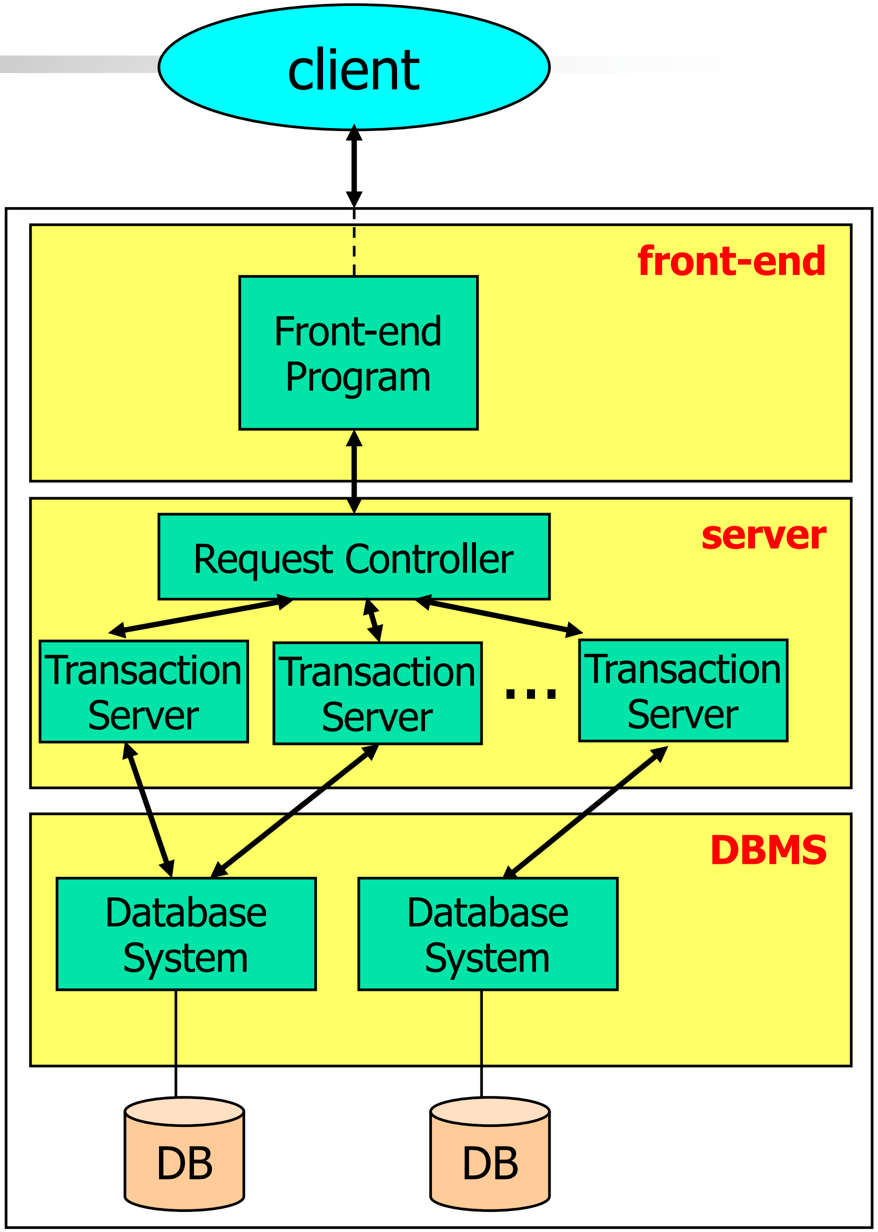
Application Middleware – Main Tasks

- Distributed computing infrastructure (RPC, RMI)
- Transactional capabilities
  - programming abstractions (demarcation)
  - distributed transaction management
- Security services
  - authentication, authorization, secure transmission, …
- Unified access to heterogeneous information sources and application systems
- Scalable and efficient application processing
  - large number of client applications or end users
- Reliability, high availability

Programming model abstractions that allow the developer to focus on application logic (i.e., ignore infrastructure as much as possible)

TP Applications

- Front-end program
  - interacts with display devices
  - gathers and validates input, displays output
  - constructs and forward request
- Request controller
  - guides the request execution
  - determines required steps, then executes them by invoking transaction servers
  - usually runs as part of an ACID transaction
- Transaction server
  - process that runs application programs doing the actual work of the request
  - almost always runs within the scope of an ACID transaction
  - typically interacts with a DBMS
  - simple applications can be composed into more complex ones
    - composition of transactions



Application servers are crucial for supporting development and execution of TP application programs to build scalable TP systems

Chapter 5 – Application Server Middleware

- Architecture of transaction processing applications
  - front-end programs, request controller, transaction server
- Different types of application server middleware
  - TP monitors, object brokers, object transaction monitors, component transaction monitors
- Transactional capabilities in application servers
  - address the transaction composability problem
    - transaction demarcation/bracketing approaches (explicit vs. implicit demarcation)
    - nested transactions
  - transaction processing standards and interoperability
- Support for shared state
  - sessions, stateless vs. stateful applications/servers
- Mapping application components to processes and threads
  - multithreading, server classes, process structure of request controller, TA server
- Scalability (caching, pooling, partitioning and replication)

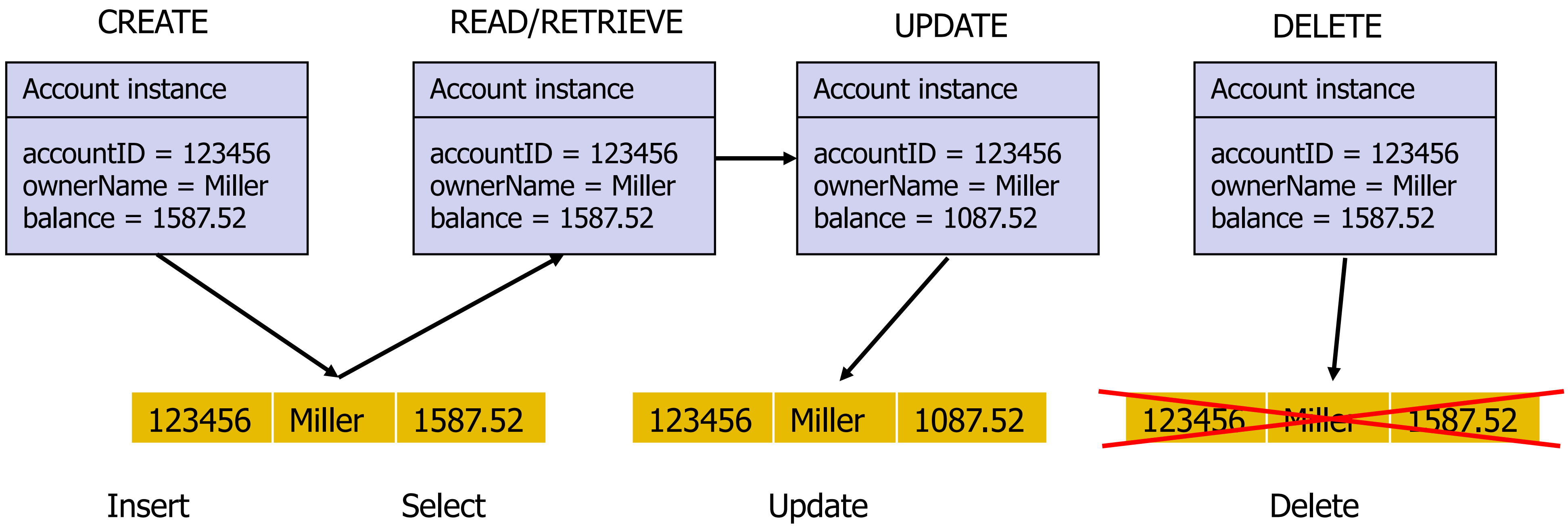
Object/Relational Impedance Mismatch

- Object-oriented programming/design is increasingly used for building information systems
  - general approach: design a domain object model that represents the data, structure and common behavior of the business objects
  - domain object state has to be retrieved from and written to an underlying DBS (usually a relational DBS)
- Problem: object-oriented and relational models have severe differences
  - ➔ impedance mismatch

	objects	**relations**
structure	•complex values, collections •class hierarchies (inheritance)	•flat tables
relationships	•binary •1:1, 1:n, n:m (using collections) •uni-/bi-directional references	•binary •1:1, 1:n •value-based, symmetric
behavior	•methods	
access paradigm	•object navigation (follow references)	•declarative, set-oriented (queries)

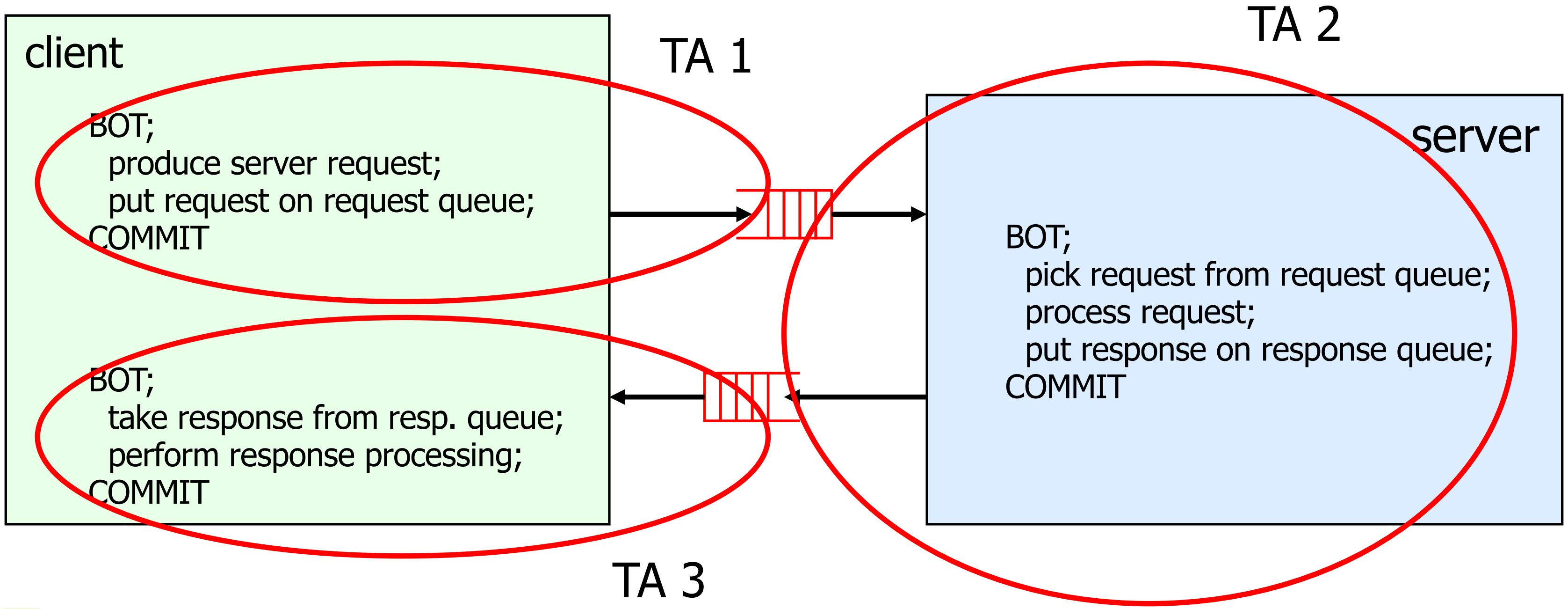
Chapter 6 – Object Persistence Services

- The impedance mismatch needs to be resolved in the application program
  - requires detailed knowledge of the DB-schemas, involves coding SQL statements
- Middleware to help with this task
  - object/relational mappers (ORM), object persistence services/frameworks
  - shield the application from existing data stores, simplify programming model



Chapter 7 - Message-Oriented Middleware

- Queued, asynchronous transaction processing
- Decoupling Request Entry, Request Processing, and Response Delivery, use separate TAs for each task
  - optimize for throughput
  - avoid resource contention of single-transaction (TRPC) approach
  - can be generalized to multi-transaction requests

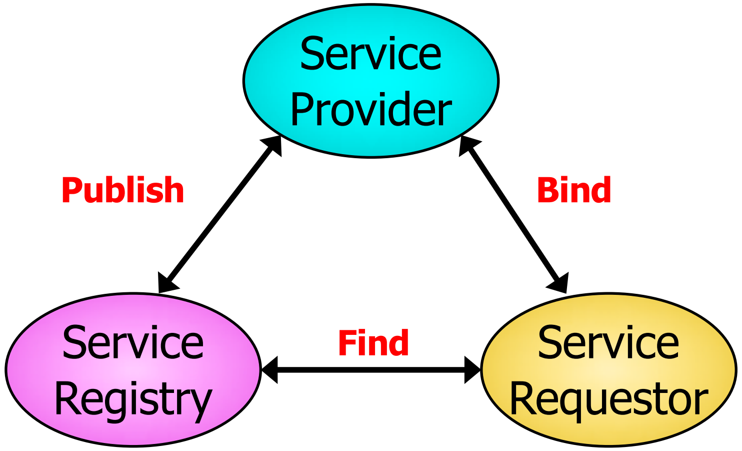

client
BOT;
produce server request;
put request on request queue;
COMMIT
BOT;
take response from resp. queue;
perform response processing;
COMMIT
TA 1
TA 2
TA 3
server
BOT;
pick request from request queue;
process request;
put response on response queue;
COMMIT

Service-Oriented Computing (SOC)

- Service-oriented architecture (SOA)
  - models an application as a composition of reusable services
    - focus is on functions, not things (in contrast to OO-design)
  - services are characterized by
    - the messages they exchange
    - the interface contracts defined between service requester and provider
- TP system based on SOA may include
  - multiple reusable services offered by a single TA-program, or multiple distributed services
  - both synchronous and asynchronous communication mechanisms
    - service is invoked by sending a message to the service
  - service can implement a TA or a step within a TA (request controller or TA server)
- Increased popularity of SOC
  - service-oriented access to functions of large-scale web sites (search, social networking, e-commerce)
  - advent of standard web service protocols

Chapter 8 – Foundations of Web Services

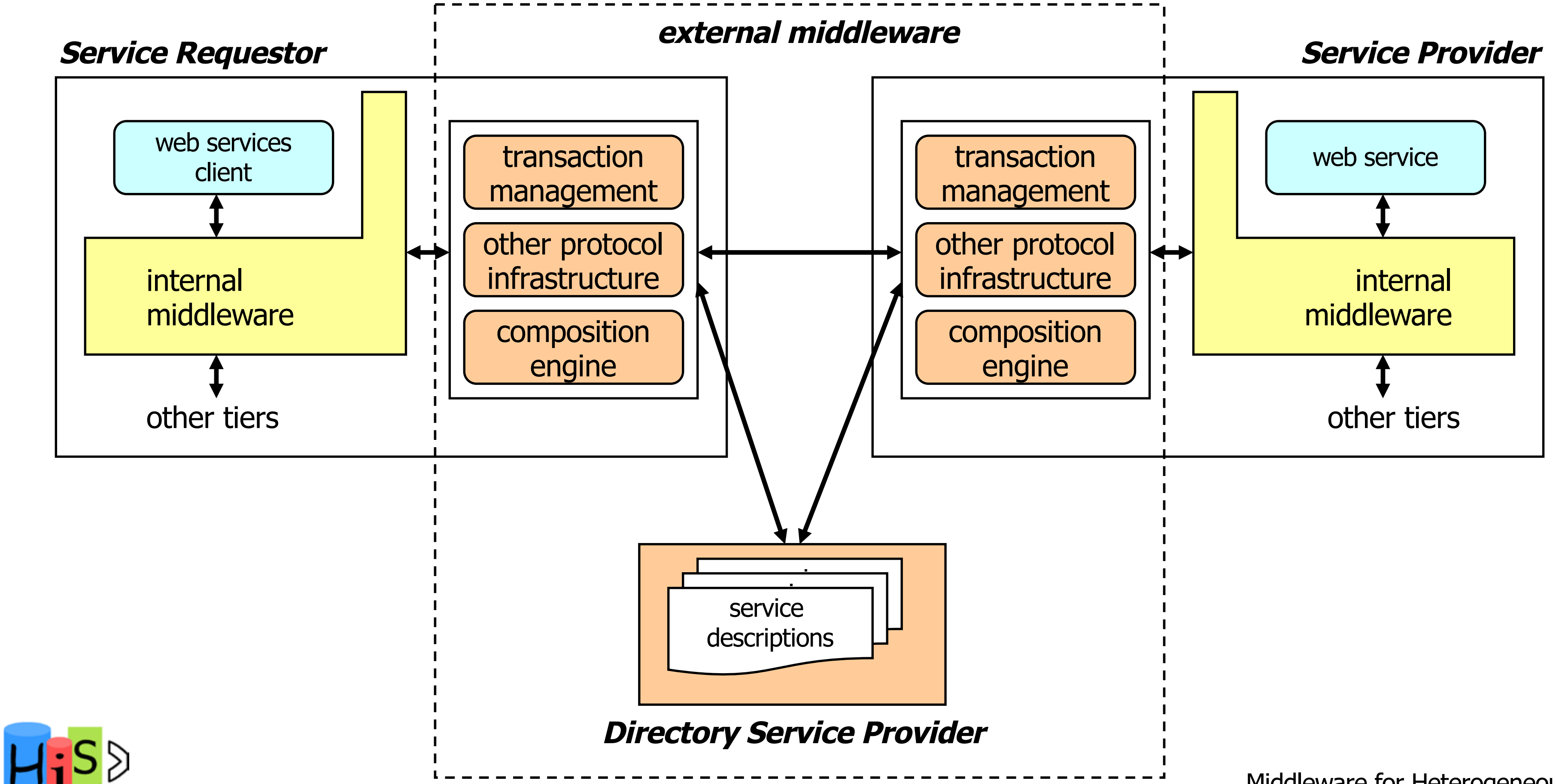
- SOA-implementation based on Web Services
  - application function is mapped to a specific service interface (e.g., *AddCustomer*)
  - standards for SOA: invocation (SOAP), interface description (WSDL), registry (UDDI), all based on XML
  - interoperability: interfaces are available for appl. servers, ORBs, MOM, DBMS, …
    - includes transaction interoperability
  - service assembly & composition: tools and techniques available
- Implementation based on Representational State Transfer (REST)
  - architectural style for building large-scale distributed hypermedia systems
  - application function is mapped to a specific resource providing a generic interface
    - resource identification through URI; uniform interface: HTTP GET, PUT, POST, DELETE
    - example: *HTTP POST* on *www.company-xyz.com/customers* to add customer



Web services are wrappers for existing IS-functionality

Chapter 9 – Web Service Coordination and Transactions

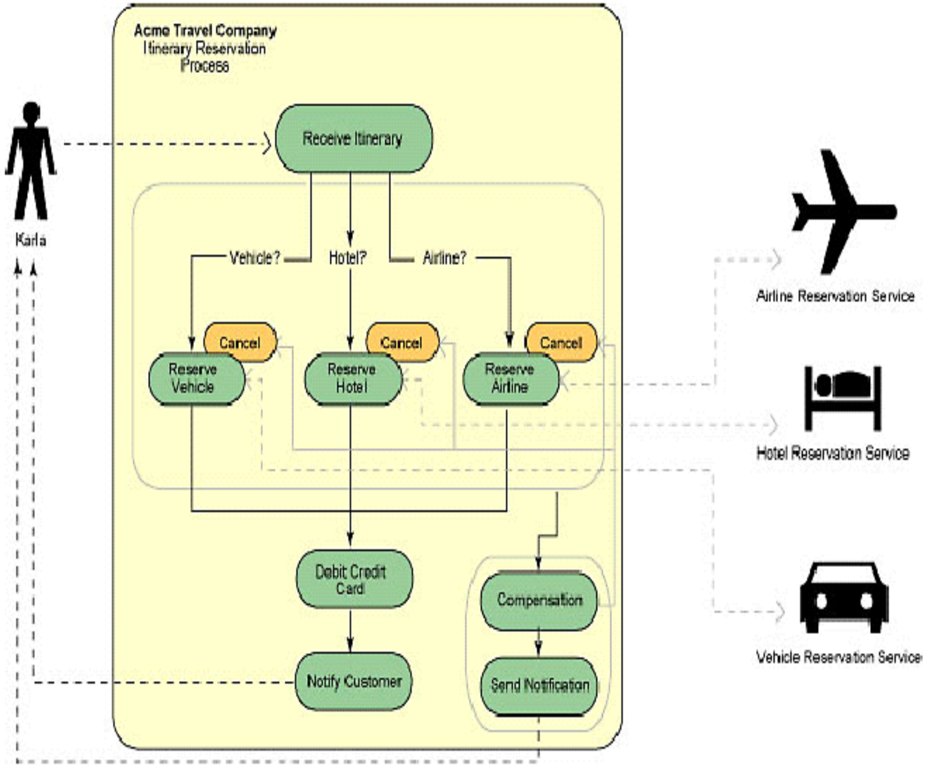
- Standardized web service infrastructure for
  - conversation control
  - coordination protocols and transaction management



© Prof.Dr.-Ing. Stefan Deßloch

Chapter 10 - Business Processes Modelling and Workflow Management

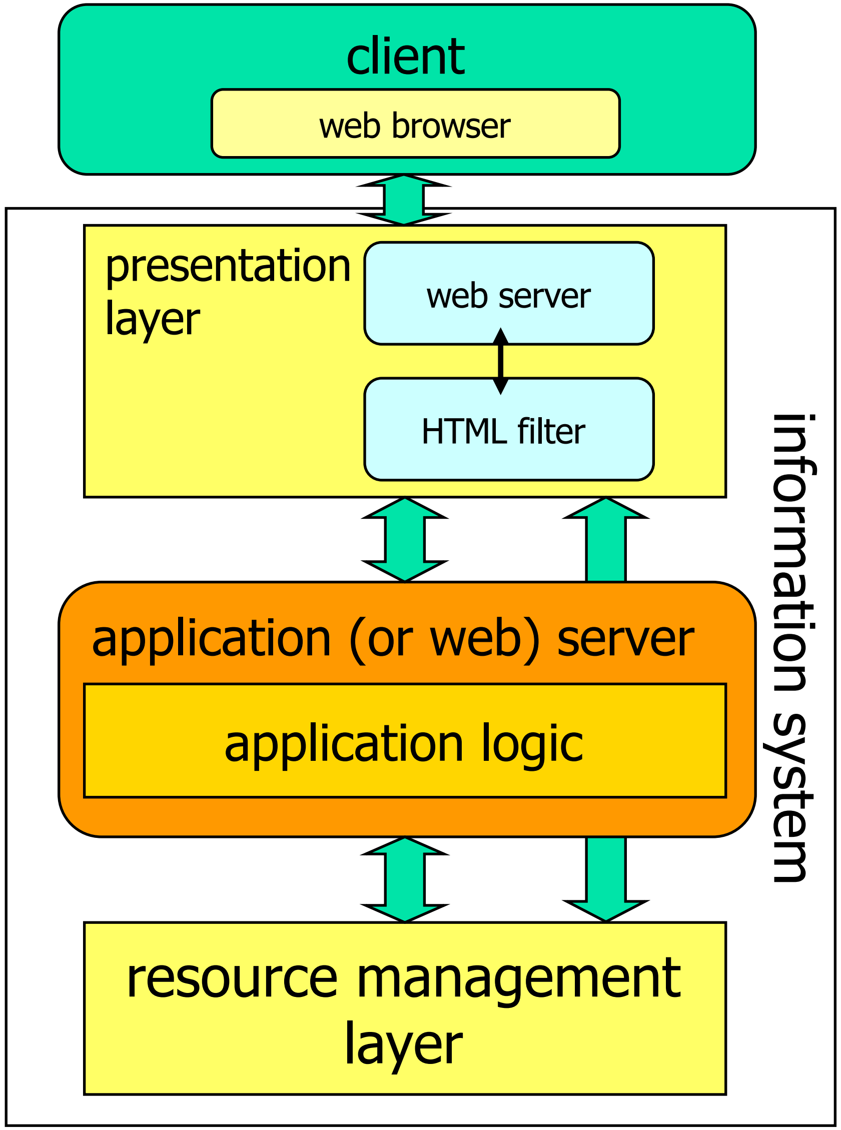
- Goal: efficient execution of core business processes (optimization & analysis)
- Workflow management systems for process control and execution
  - advanced transaction concepts (-> long-running, compensation)
- Web service composition
  - business process making use of web services
  - business process externalized as a web service
  - correlation
  - dynamic binding of business partners and web services
- BPEL – standard for web service composition



Role of the WWW for IS

- Initial purpose: sharing information on the internet
  - technologies
    - HTML documents
    - HTTP protocol
  - web browser as client for internet information access
- For Information Systems: connecting remote clients with applications across the internet/intranet
  - "web-enabled" applications
    - extend application reach to the consumer
    - leverage advantages of web technologies
  - web browser as a universal application client
    - "thin client"
    - no application-specific client code has to be installed
  - requirements
    - content is coming from dynamic sources (IS, DBS)
    - request to access a resource has to result in application invocation
    - session state: tracking repeated interactions of the same client with a web server

- Presentation layer may be realized in separate tiers
  - client-side presentation using browser, *client components (optional)*
  - server-side presentation done by web server, dynamic HTML generation (HTML filter)
- Presentation components interact with application logic components
  - managed by appl. server, or *run within web server environment*
- Access to RM layer
  - "encapsulated" in appl. logic component
  - may also be performed directly *within presentation logic component*
    - special case, if there is no application logic (only data access logic)



Data Integration Middleware

- Traditional Middleware (shortcomings)
  - supports access to multiple data sources within the same application, transaction
    - directly (using DB-gateways)
    - indirectly (by invoking distributed application components)
  - but fails to provide data integration
    - no means to analyze/query data from multiple sources within the same statement

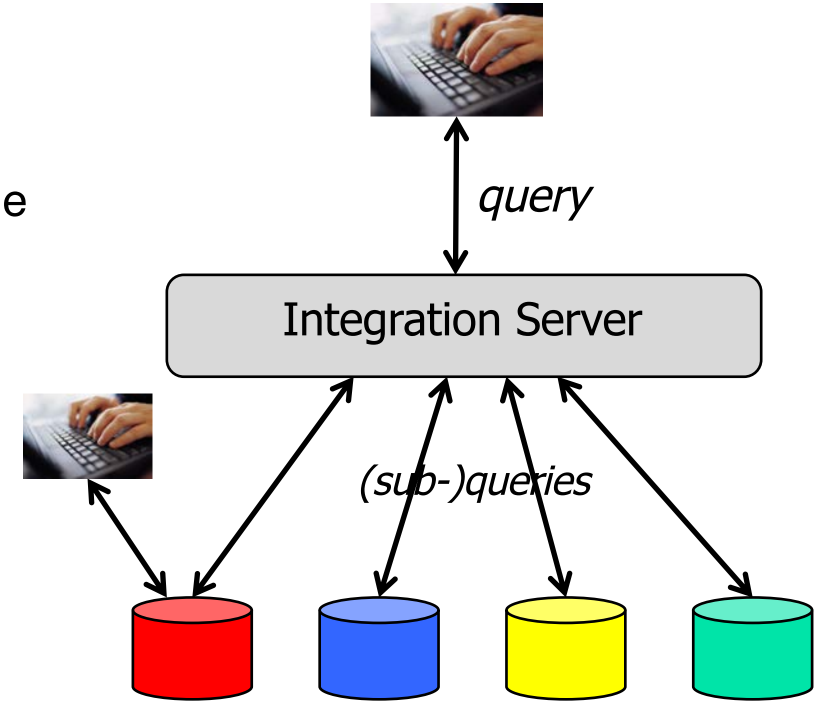
      ```
      SELECT *
      FROM Source1-table T1, Source2-table T2
      WHERE T1.a1 = …
                AND
                T1.a2 = T2.a1
      ```
    - does not help to overcome data heterogeneity

Chapter 12 – Information Systems Integration

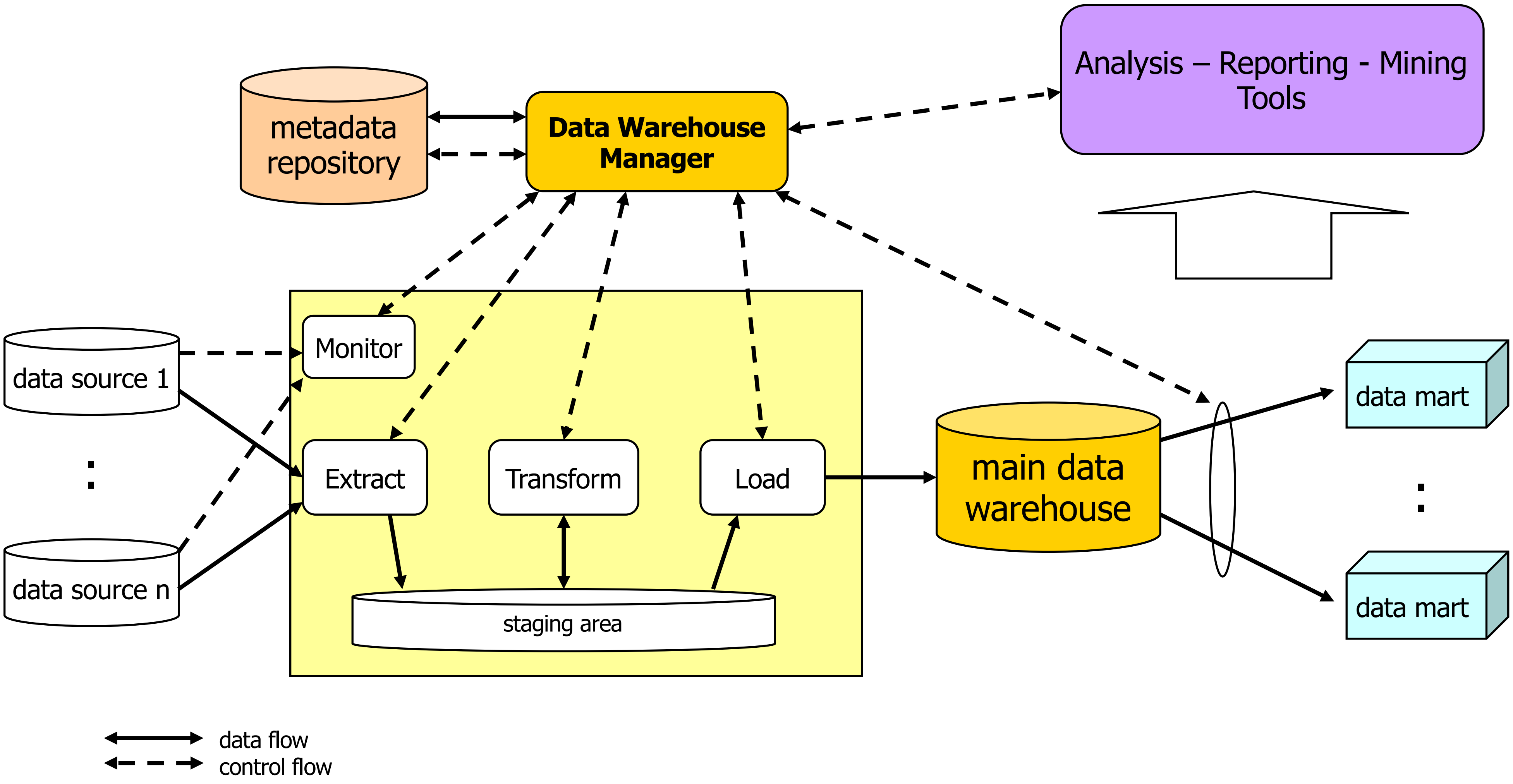
- Introduces fundamental concepts for middleware that facilitates interaction between and integration of applications across multiple distributed, heterogeneous platforms and data sources
- Major challenges: distribution, autonomy, heterogeneity
  - different forms of (data) heterogeneity
- Data/Information Integration
  - goal: integrated access to (heterogeneous) data originating from multiple sources
  - materialized integration extracts data from sources and stores it in an integrated database for query processing, data analysis
  - virtual integration leaves data in the sources and performs complex, distributed query processing for data access
  - both approaches have pros and cons
- Information integration process
  - general techniques, independent of middleware plattforms

Chapter 13 – Virtual Data Integration

- Goal: homogeneous, integrated view of data from multiple sources
  - illusion of a single (logical) database
  - a single query may collect (or join) data from multiple sources
- "On-Demand" Data Integration
  - data stays where it is (in the sources)
    - not copied into a new DB
  - data is transformed/integrated at query time
    - integration server combines results from data source queries
- Architectures for virtual data integration
  - distributed DBMS, federated DBMS, mediator-based systems
    - based on a global schema, can support location and distribution transparency
  - multi-database systems
    - no global schema, only support location transparency



Chapter 14 - Data Replication and Materialized Integration



Chapter 15 – Information Integration

- Schema Matching
  - Find inter-schema correspondences
- Schema Mapping
  - Based on correspondences
  - Define how to "translate" one schema into another
    - implies data transformation
- Schema Integration
  - Based on correspondences (and mapping)
  - Define an integrated, global/federated schema

➔ **Integration Plan!**

- Integration plan can then be “implemented” using middleware for virtual or materialized data integration