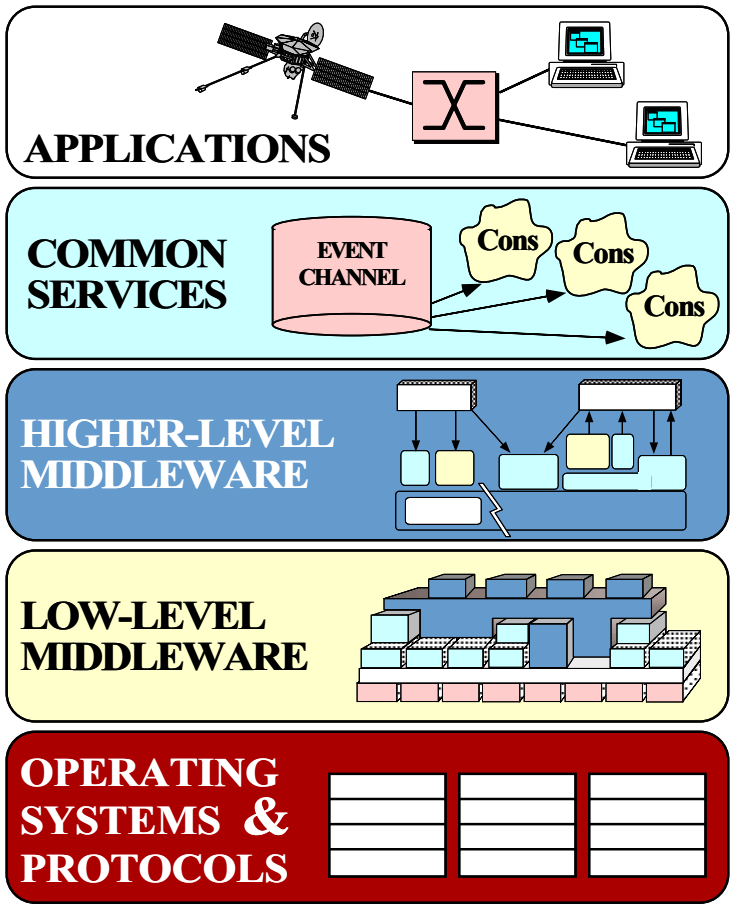
Research Directions for Middleware

Douglas C. Schmidt
schmidt@cs.wustl.edu

Washington University, St. Louis
www.cs.wustl.edu/∼schmidt/

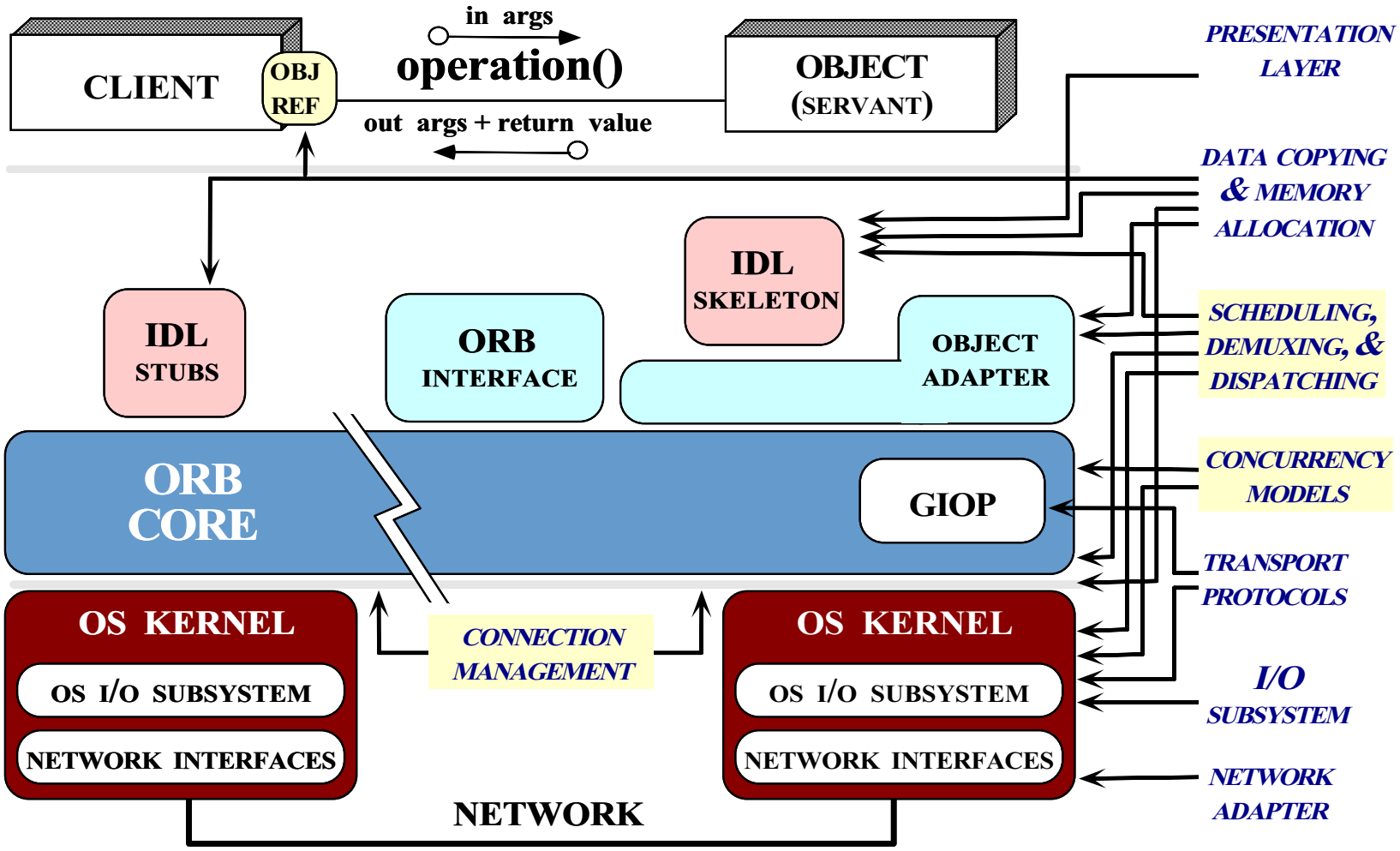
January 23^{rd}, 1999

Overview of Middleware

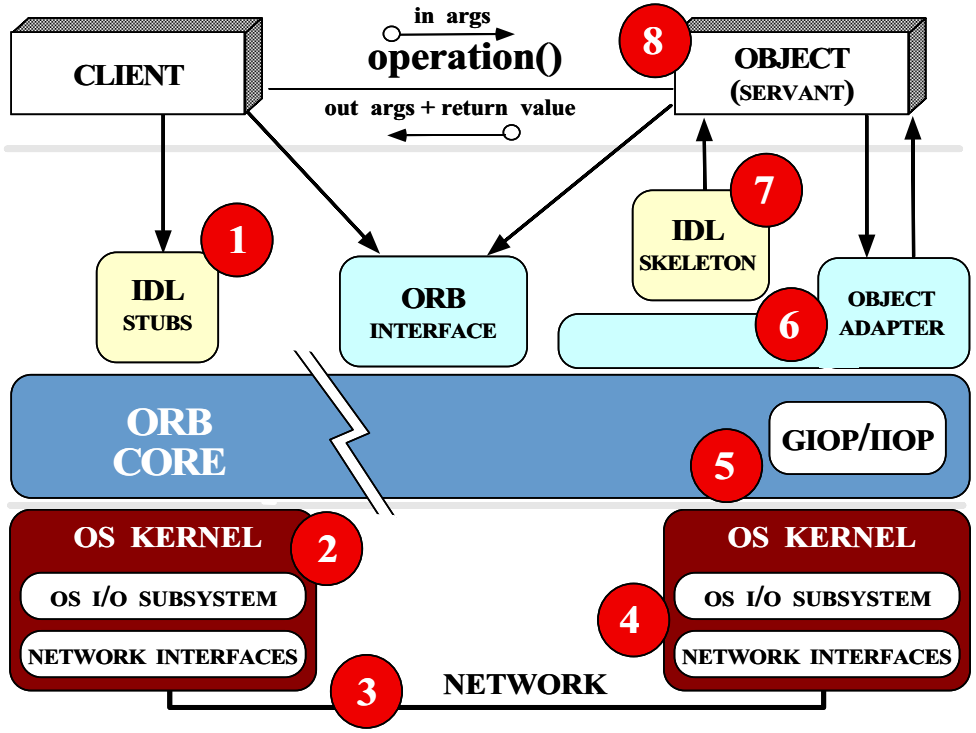


- **Observations**
  - Historically, apps built directly atop OS
  - Today, more and more apps built atop *middleware*
  - Middleware has several layers
- **General Research challenges**
  - Performance optimizations
  - Quality of Service (QoS)
  - Software architecture & patterns

Scope of Performance Optimization Challenges

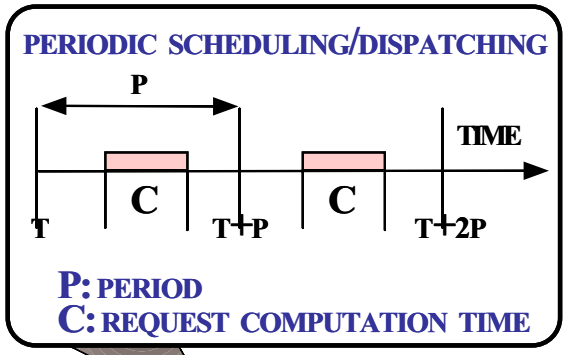


Scope of QoS Challenges



- **Key Challenges**
  - Specifying QoS requirements
  - Determining operation schedules
  - Alleviating priority inversion and non-determinism
  - Reducing latency/jitter for demultiplexing
  - Reducing presentation layer overhead
  - Maintaining small footprint

Example: Providing QoS to Remote Operations

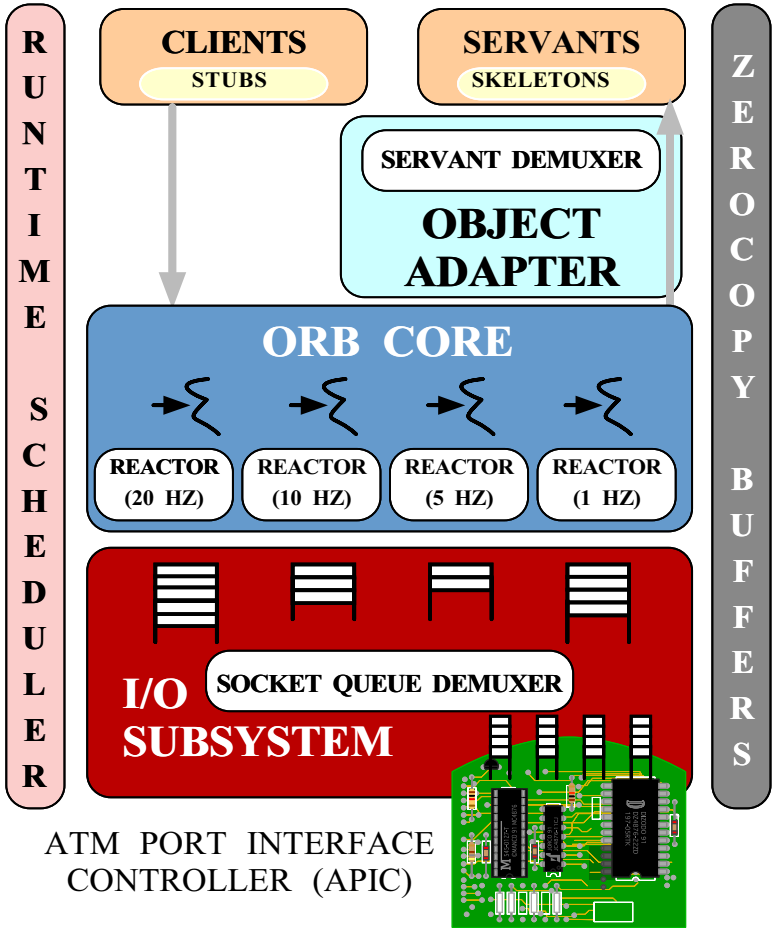


RT Operation

```
struct  RT_Info {
  Time worstcase_exec_time_;
  Period period_;
  Criticality criticality_;
  Importance importance_;
};
```

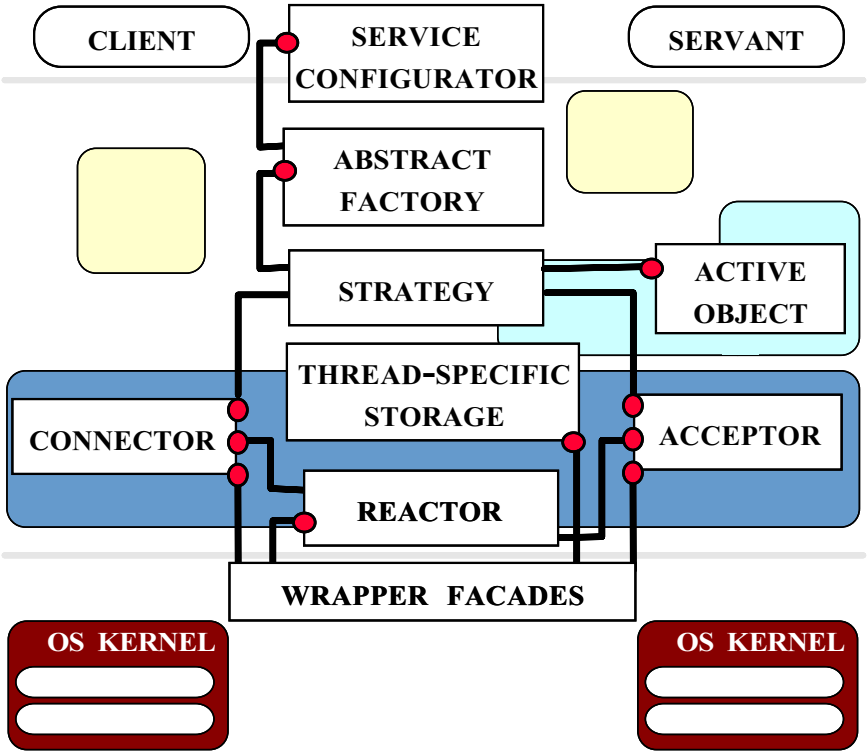
- **Design Challenges**
  - Specifying/enforcing QoS requirements
  - Focus on *Operations* upon *Objects*
    - Rather than on communication channels or threads/synchronization
  - Support static *and* dynamic scheduling
- **Solution Approach**
  - Servants publish resource (*e.g.*, CPU) requirements and (periodic) deadlines
  - Most clients are also servants

Scope of Software Architecture Challenges



- **Solution Approach**
  - Integrate RT dispatcher into ORB endsystem
  - Support multiple request scheduling strategies
    - *e.g.*, RMS, EDF, and MUF
  - Requests ordered *across* thread priorities by OS dispatcher
  - Requests ordered *within* priorities based on *data dependencies* and *importance*

Applying Patterns and Frameworks to Middleware



www.cs.wustl.edu/∼schmidt/ORB-patterns.ps.gz

- **Benefits**
  - Facilitate design and code reuse
  - Preserve crucial design information
  - Guide design and implementation choices
  - Document and alleviate common traps and pitfalls

Summary of Communication Middlware Research

- Current generation: real-time middleware
  - Real-time static scheduling services
  - Minimize ORB priority inversion and non-determinism
  - Reduced end-to-end latency via demuxing optimizations
  - Applied optimizations to IIOP protocol engine
- Future work
  - Dynamic and hybrid scheduling of distributed remote operations
  - Distributed QoS and integration with high-speed networks
  - Optimizing IDL compilers

Concluding Remarks

- Researchers and developers of distributed, real-time applications confront common challenges
  - *e.g.*, service initialization and distribution, error handling, flow control, scheduling, event demultiplexing, concurrency control, persistence, fault tolerance
- Successful solutions apply *design patterns*, *frameworks*, and *components* to resolve these challenges
- Middleware is an effective way to achieve reuse of distributed software components
- Requirements for next-generation communication middleware provide a fertile source of research topics