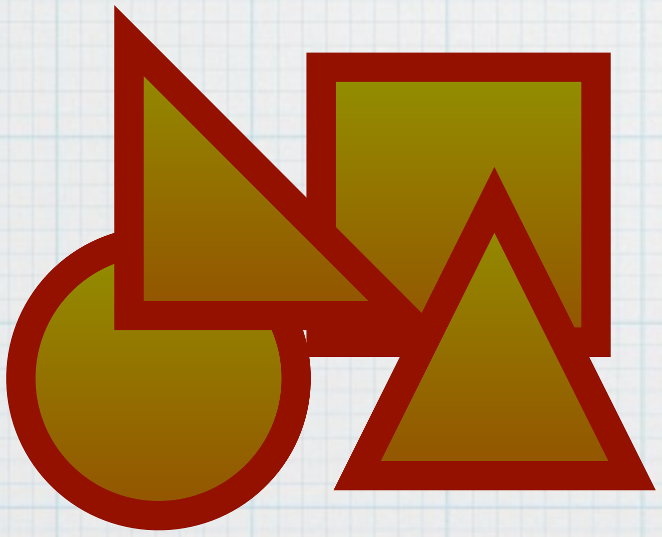
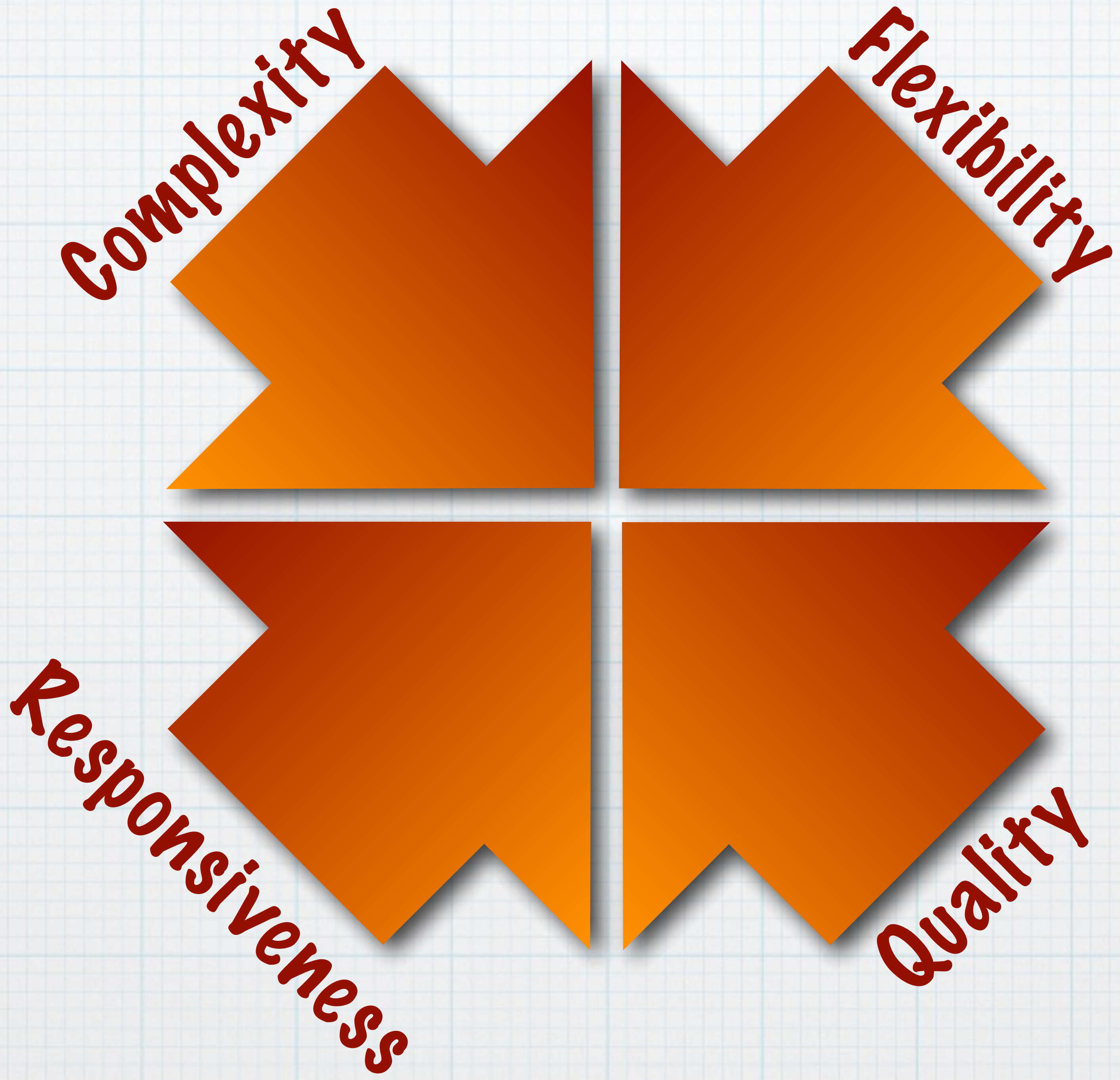
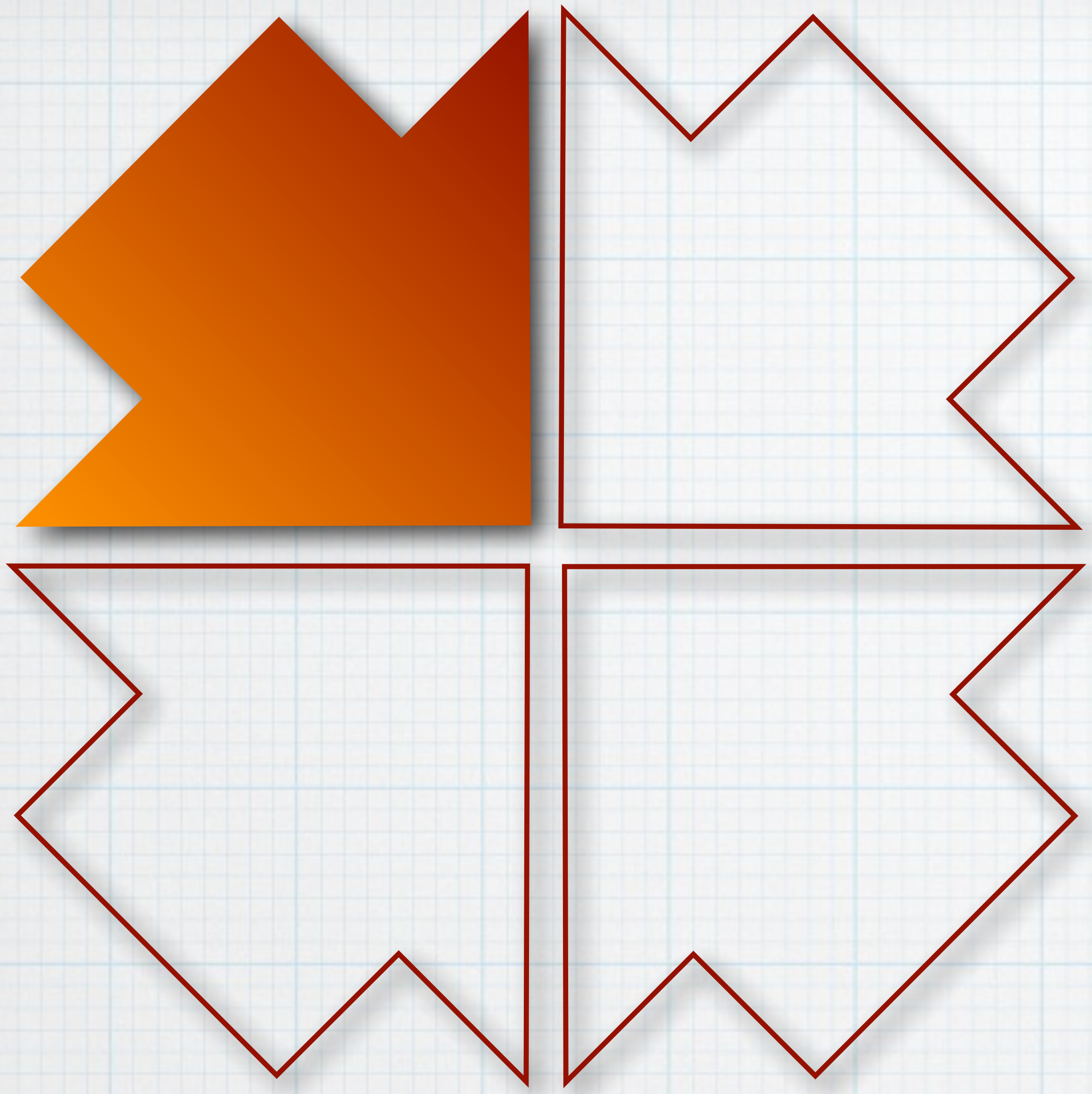
CS630
Object-Oriented Systems Engineering

Les Waguespack, Ph.D.



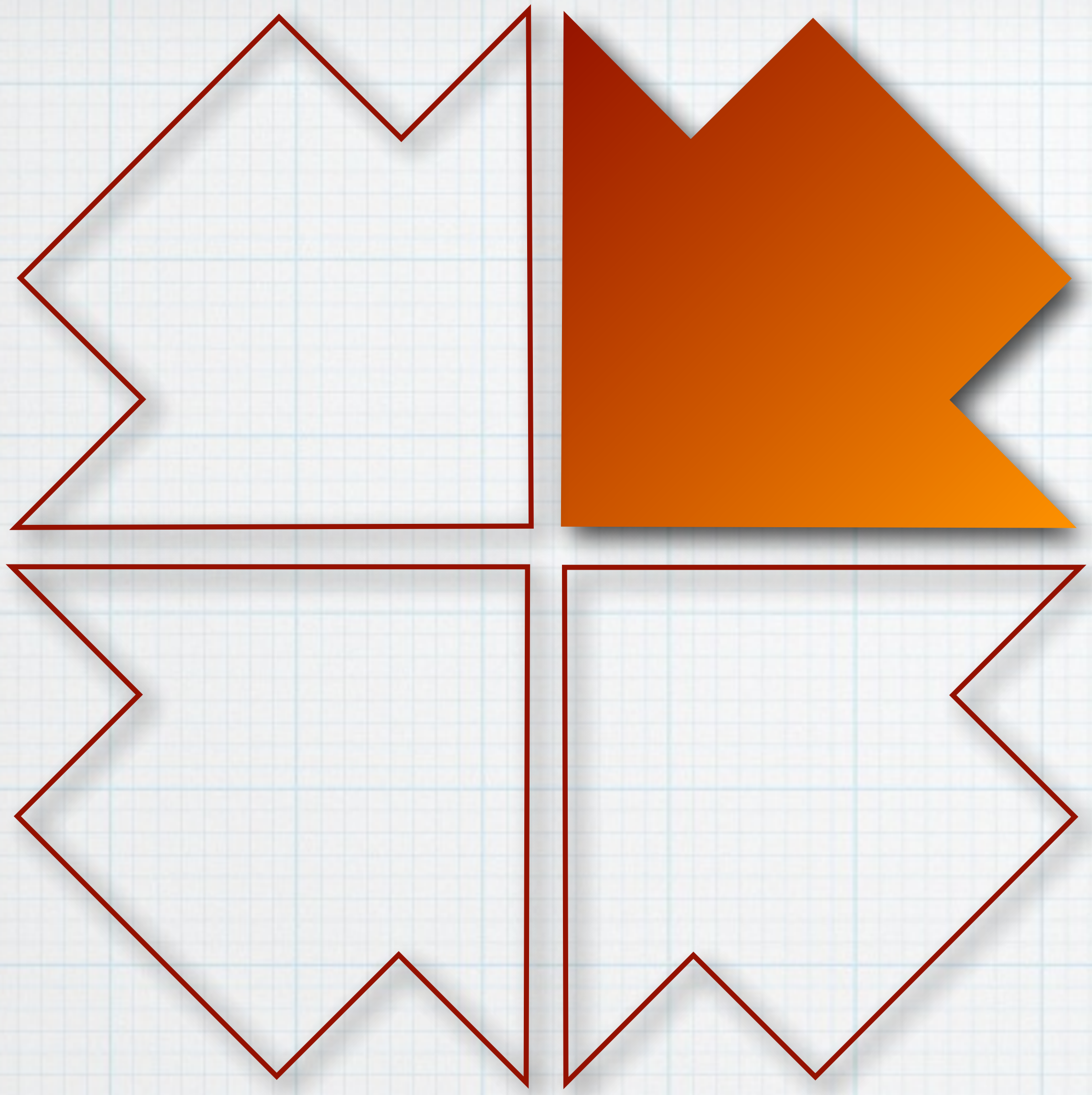
Building Object-Oriented Systems





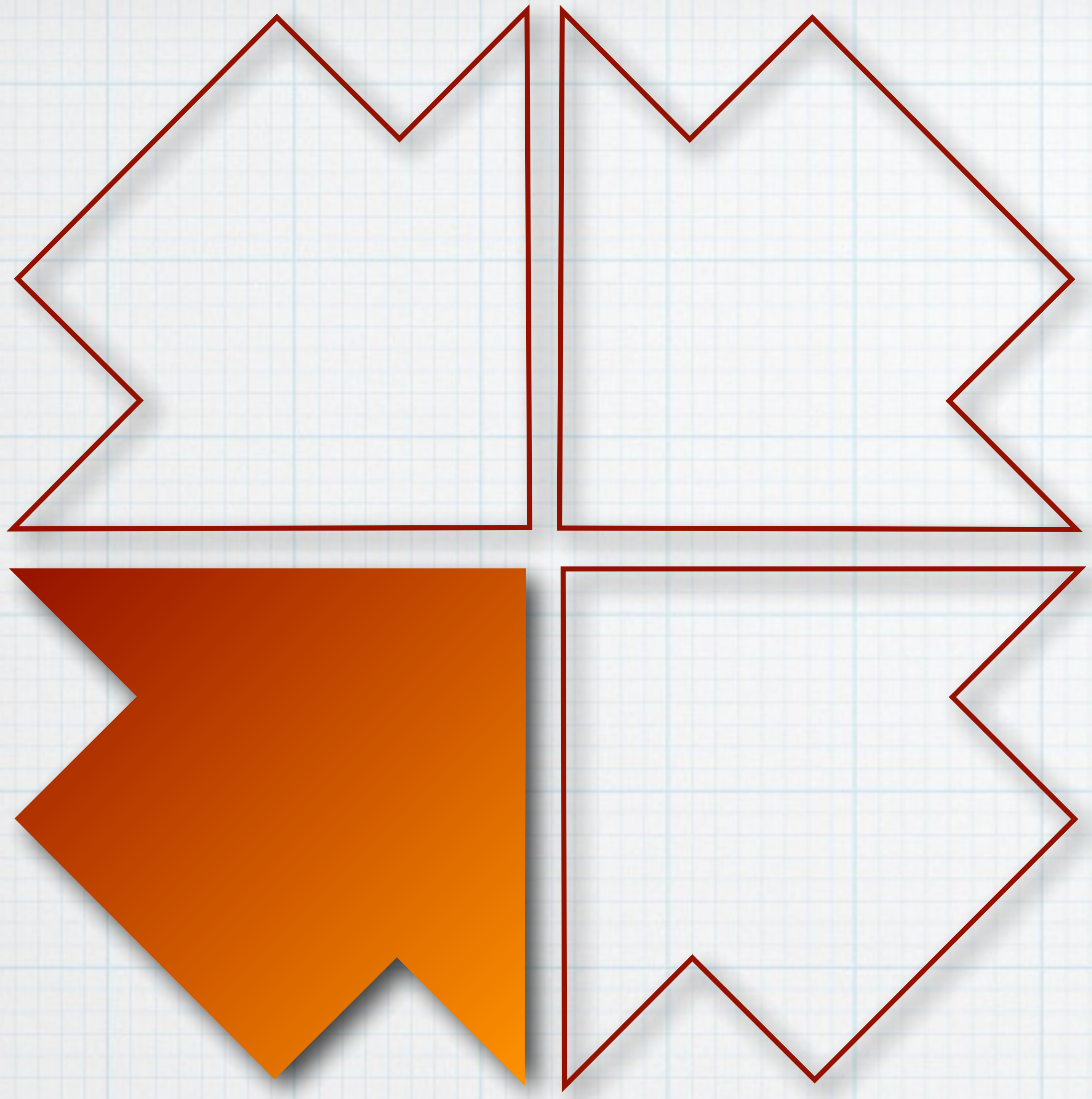
Complexity

- Represent any kind of data in any possible structure
- Encapsulation of related data and procedures hides complexity
- Natural reflection of real world simplifies access and understanding



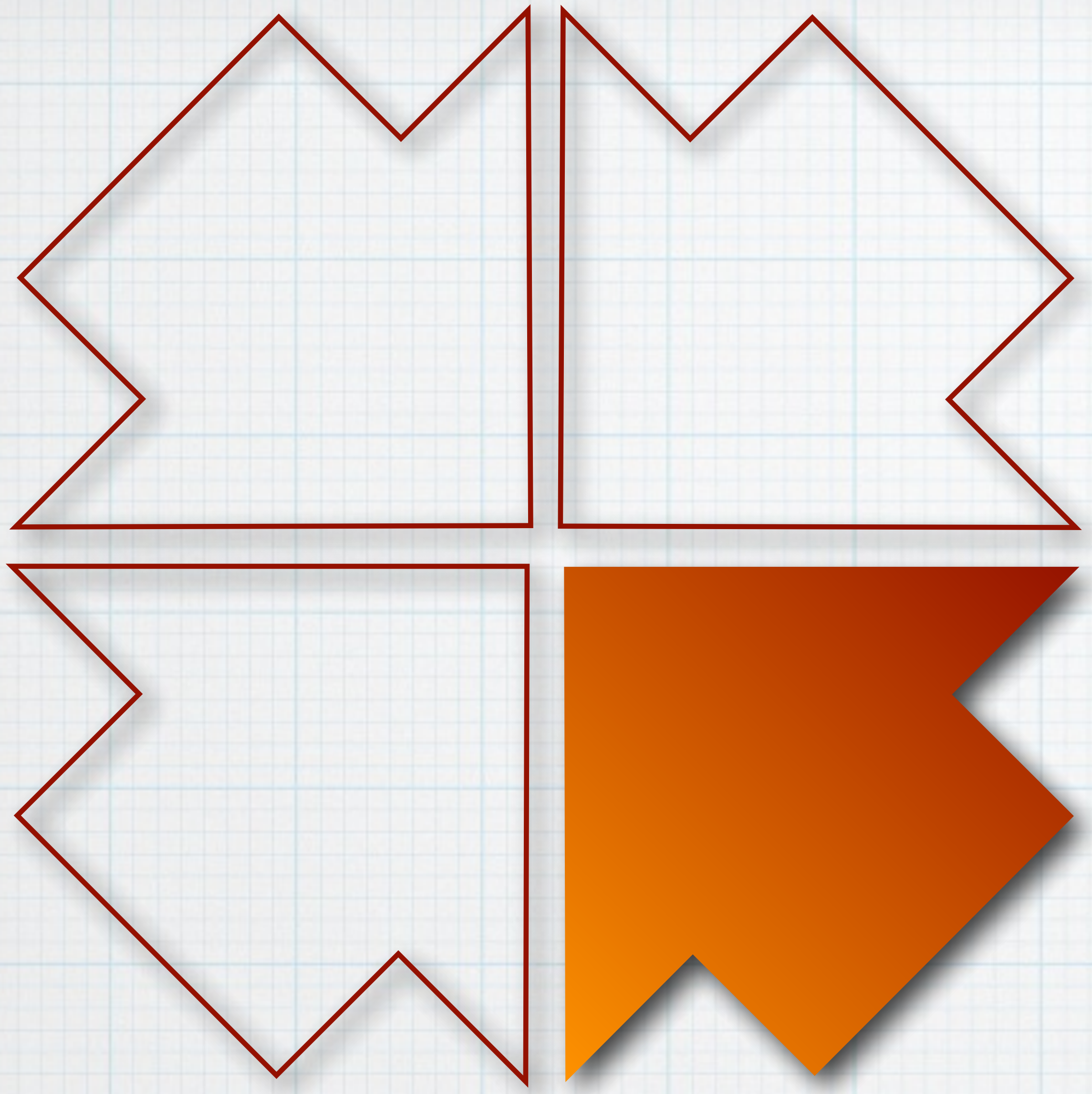
Flexibility

- Modularization isolates changes and minimizeds their impact
- Layered development insulates applications from models and classes
- Inheritance handles generalization, specialization and exceptions



Responsiveness

- Efficient storage, retrieval and processing of complex structures
- Encapsulated procedures react to events immediately
- New applications can be developed an order of magnitude faster



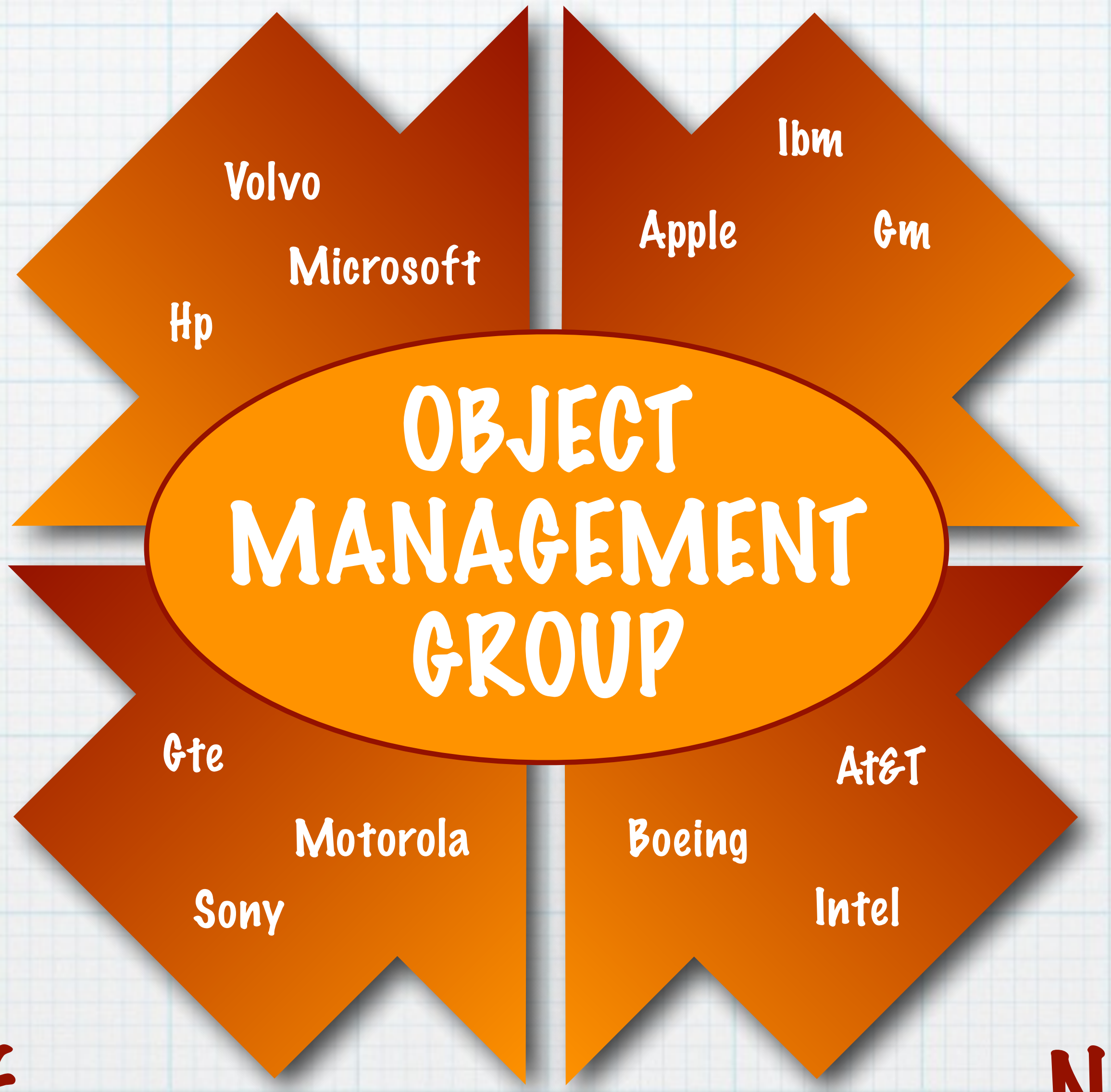
Quality

- Reuse of proven, reliable code greatly reduces defects
- Rapid prototyping with end users maximizes fitness to purpose
- Graphical interface and object visualization increase usability

STANDARDS

Distributed Transactions

Concurrent Execution



Versioning of Objects

Notification of Events

Internationalization

OBJECT MANAGEMENT GROUP

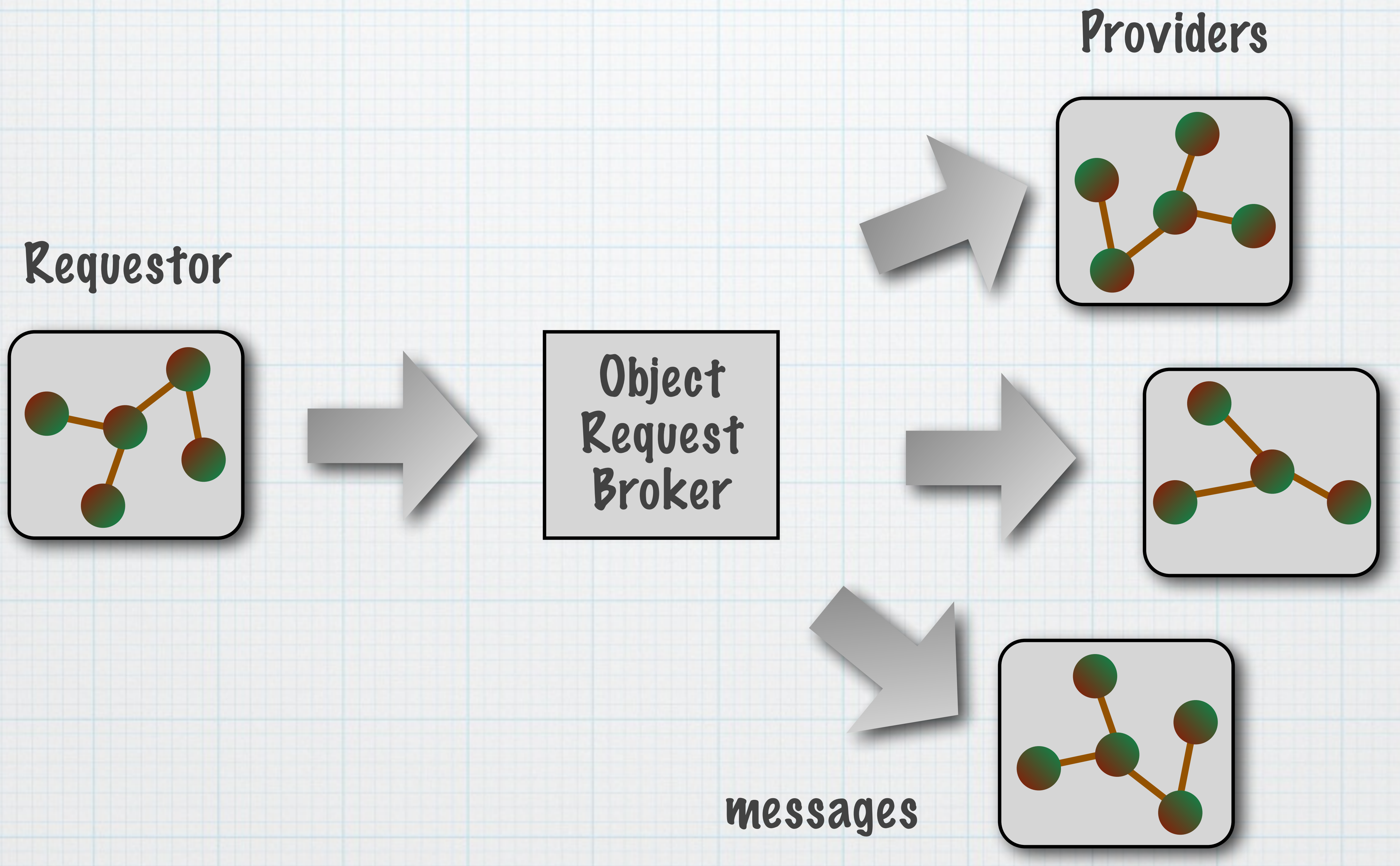
Object Request Broker

The **Object Request Broker** approach attempts to generalize the interaction of objects.

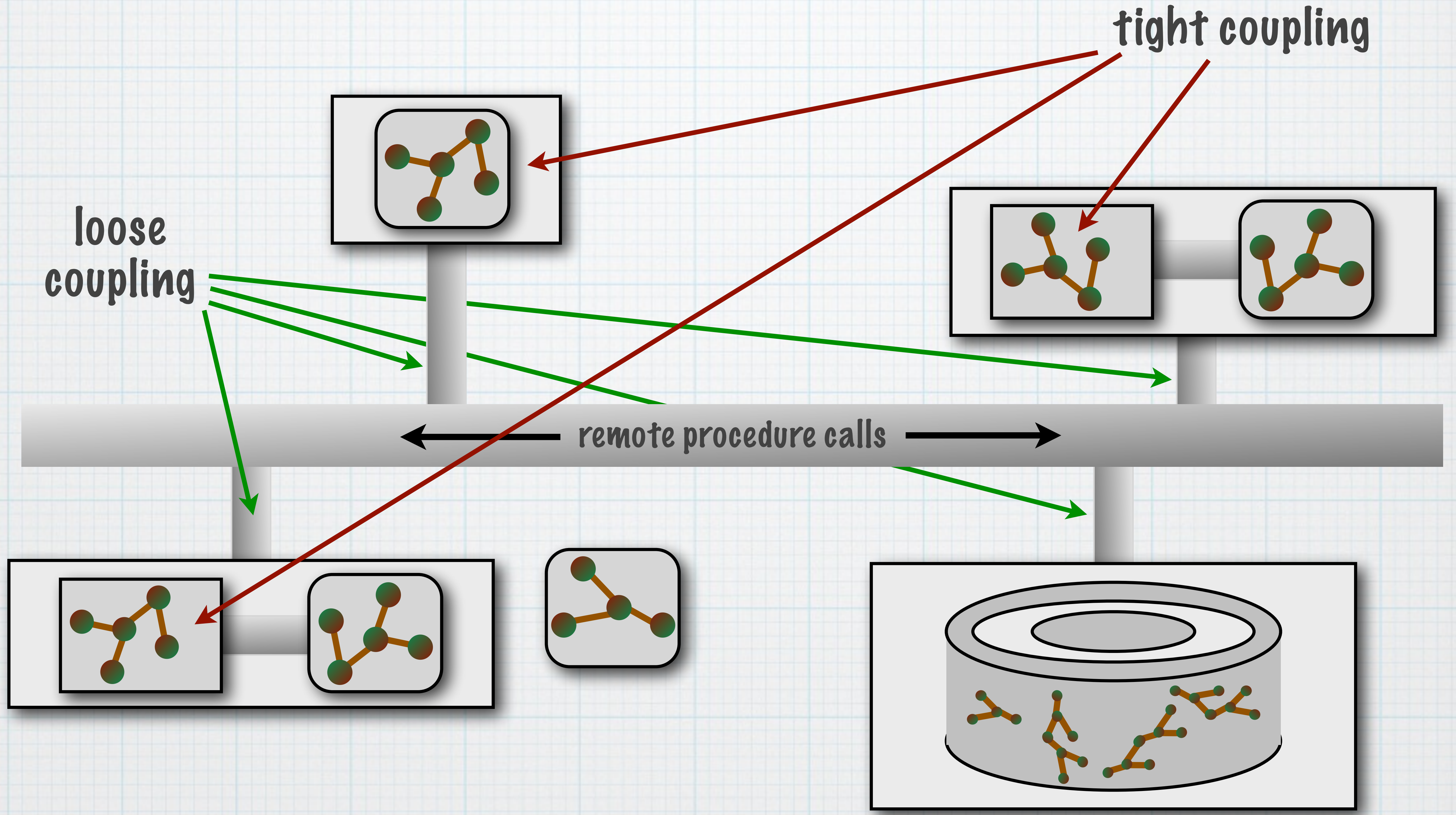
It provides a common interface for objects of different paradigms and it improves the modularity of the software by allowing differing paradigms to coexist.

It attempts to allow the advance of OO without commitment to a particular OO architecture.

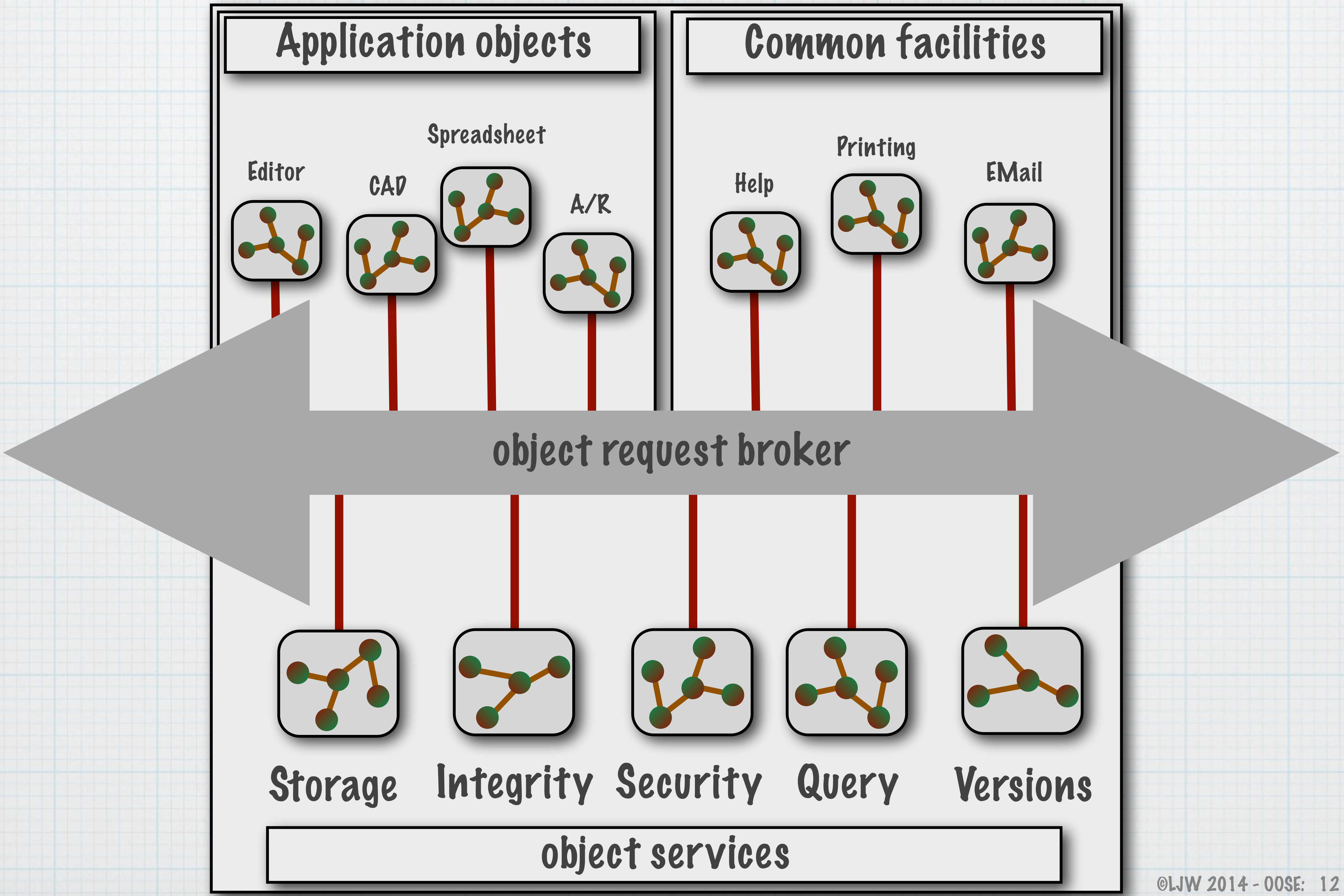
A Generalized Object Model



Object Coupling



OMG Reference Model



Software Component Industry

* The "object interaction platform" will allow "plug and play" object components and boost a new software product segment
* Software component vendors will face an evolution similar to hardware component manufacturers
* Cataloging, sorting, identifying and selecting components for a project will require a new set of standards and practices

Component Quality Assurance

* There is no precedent for accepting "off the shelf" software with confidence in mission critical applications because software has historically been far less reliable than hardware

* Quality standards will need to be developed and administered by an industry "watchdog" to boost confidence and support commercialization

* Quality "stamps of approval" will evolve that will "guarantee" the merchantability of software components

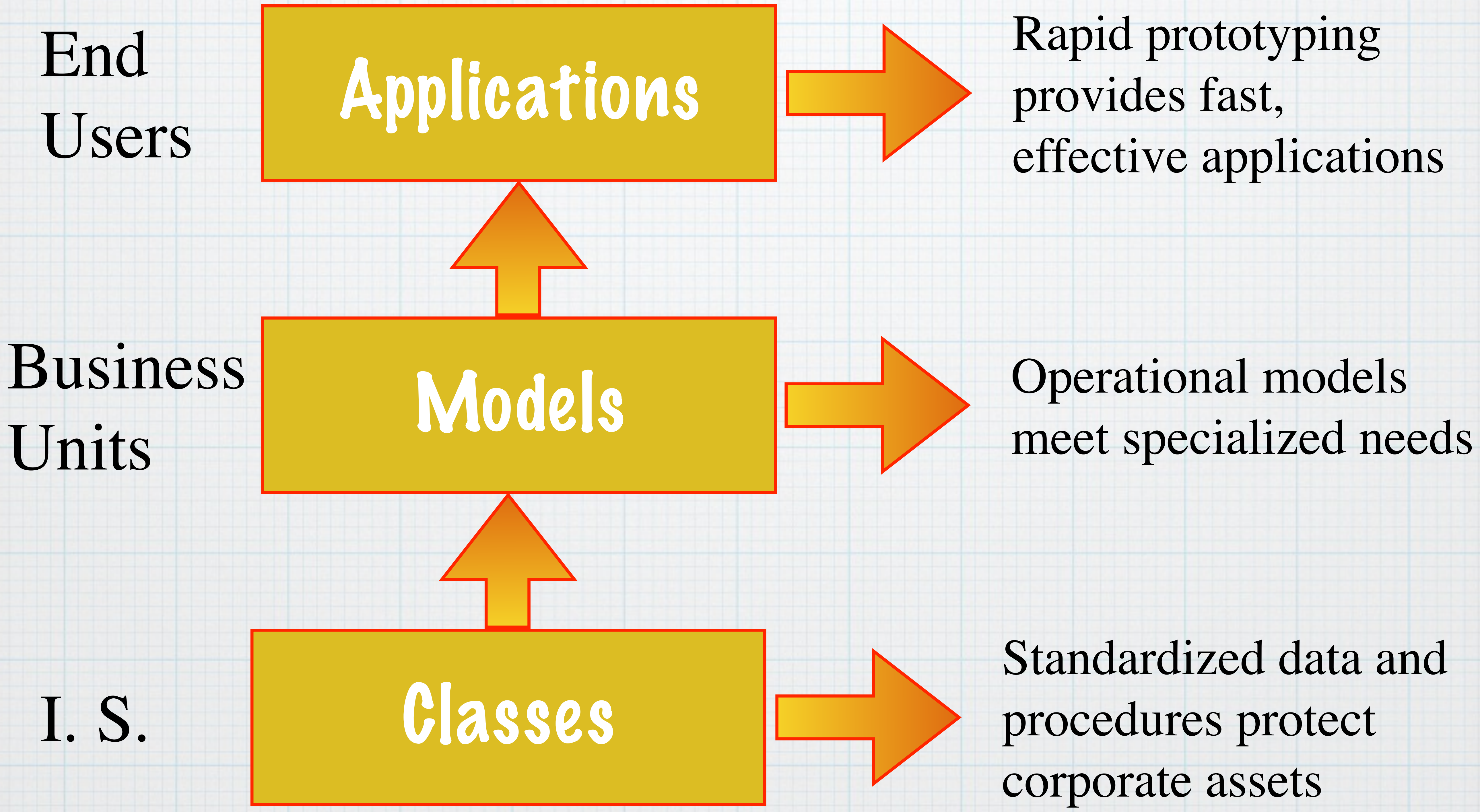
Software Component Commerce

* We are accustomed to licensing packages or products that are "one level"
* Components will be embedded and therefore less visible to the buyer or the seller of the "final" product
* The compensation structure will likely resemble that of commercial music (royalties, performance fees, "suite" fees, etc.)
* In any case how is "inheritance" addressed?

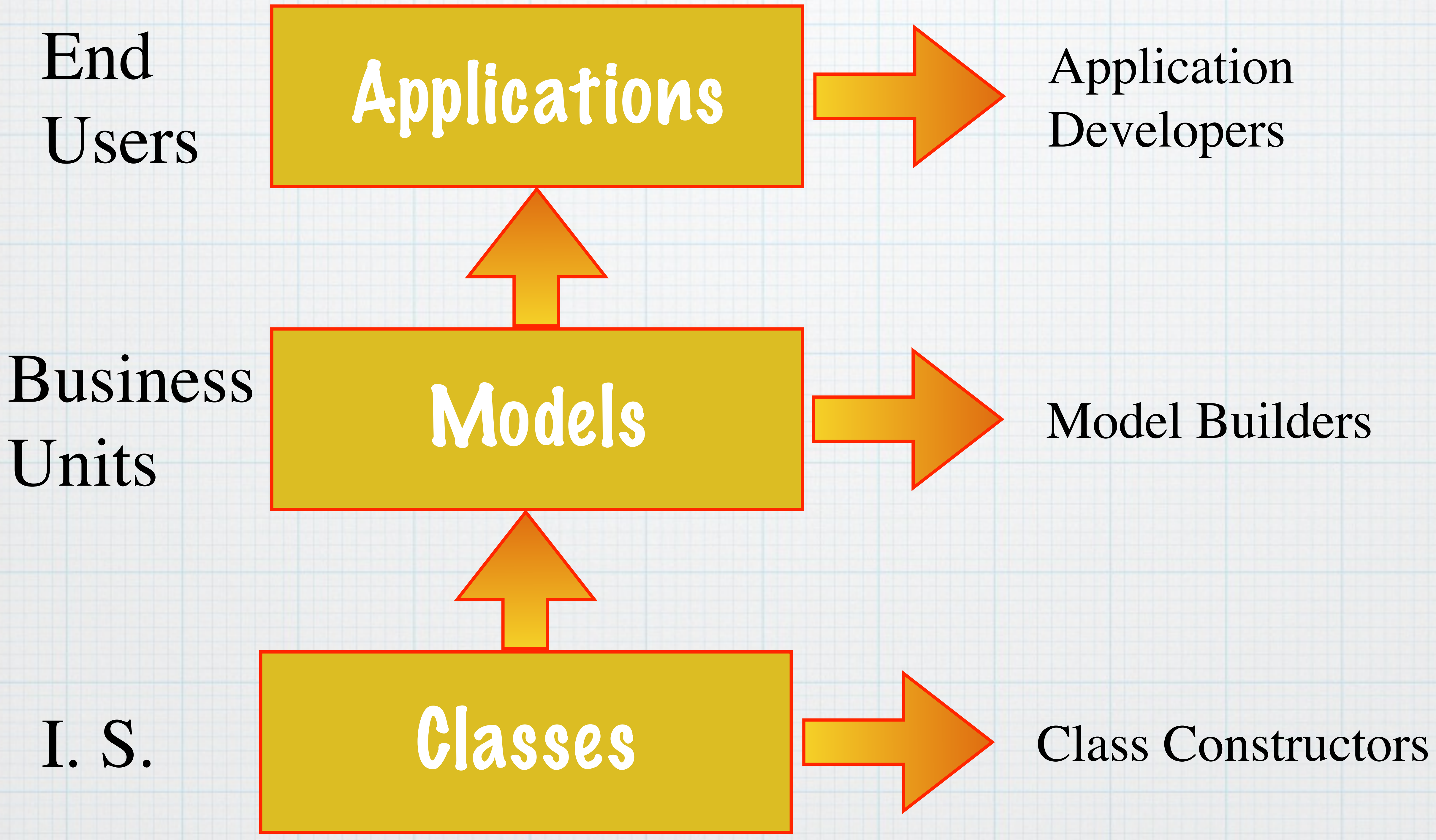
Evolving a Corporate Computing Culture

* Business computing is straddling a "mainframe vs. personal computer" based computing model
* The culture of each is orthogonal to the other.
  * corporate asset vs. one time solution
  * long lead time vs. "instant" gratification
  * global awareness vs. home grown
  * standard vs. convenient and easy to use

A Possible Architectural and Cultural Alternative


End Users
Applications
Rapid prototyping provides fast, effective applications
Business Units
Models
Operational models meet specialized needs
I. S.
Classes
Standardized data and procedures protect corporate assets

Cultural Alternatives Will Require Professional Alternatives



New Job Titles and Job Functions

* Analysts, designers, programmers, quality assurers evolved to drive the "software life cycle" as conceived by the IS culture
* Analysis, design, programming, and quality assurance are intrinsic professional skills needed to manufacture object based systems
* The software life cycle may not be appropriate any more for this type of software m manufacture
* The division must blur between domain specialist and technologist
* "Users" will be come more "technical"

Diffusing Object Orientation

* The application/model/class architecture is appealing, but is not consistent with contemporary patterns of software development (particularly corporate systems)
* Object orientation is intrinsically more "difficult" than any current form of "productivity" paradigm in use in industry.
* The focus on "models" in the object orientation paradigm is an unknown cost/risk in corporate IS development.

Summary of Object Orientation Impacts

* System Modeling
* Object Oriented Analysis
* Object Oriented Design
* Object Oriented Programming
* Reuse
* Object Oriented Database
* Enterprise Management
* Software Merchandising
* Information System Frameworks

Impacts...

* **OO Modeling**: natural for analyst and domain expert; inheritance, polymorphism and message passing is a natural framework identifying "real world" relationships and interconnections; patterns of similarity and difference are defined canonically; class hierarchy captures an "abstract structure" of the domain that is comparable for purposes of "goodness" metrics.

Impacts...

* **OO Analysis:** more closely defining objects found in the "natural" domain promotes more incisive inspection and uncovers more information that is intrinsic to the particular application domain's distinctness within the more general application area; facilitates a high degree of modular independence in domain description; classification as a primary analysis activity opens the door to greater user participation and focus on the "user world" view and improve communication.

Impacts...

- **OO Design:** co-locating the behavior and state definition surpasses abstract data typing as a means of characterizing object behavior; naturally imposes encapsulation and information hiding; coupling and cohesion are manageable; class hierarchies can mirror "user view" objects with "information view" object with little obfuscation; the use of class libraries to "clone" close relatives leverages design experience and normalizes design practice and standards (e.g. Windows, Mac GUI's).

Impacts...

* OO Programming: The modeling power with abstractions in analysis and design are carried through in OOP to organize and partition the software; inheritance and polymorphism directly support modularization and complexity control; class definition allows programmers to extend the base language to meet application specialized coding paradigms or disciplines; class libraries not only organize code for the current task, but archive it for future similar tasks.

Impacts...

* Reuse: OO development naturally deposits abstractions, models, and class definitions that may be reused on the next similar task by defining the next application as a variation on the old (rather than from scratch); OO is particularly interesting because reuse is feasible at each stage in the application development (analysis, design, programming); the prospect of "object normalization" may allow automated reuse.

Impacts...

- OO Database: Still evolving most OODB's are elaborate class libraries primarily accessible via OOP; the class library serves as data dictionary and software copylib in favor of data administration; there is hope that "object normalization" will allow the high degree of semantic integrity analysis in OODB as is now possible in relational database domain.

Impacts...

* **Enterprise Management:** As database management evolved to enable enterprise modeling OO has the potential to extend the tool power of data dictionaries and data repositories to capture policy; OO will allow the degree of information tool integration that may be used to truly "manage the information resource as a whole."

Impacts...

* **Software Merchandising:** Application development frameworks and GUI's are natural prototypes for application domain frameworks (accounting, inventory, office automation, etc.); the "plug-compatible" potential of objects means that "modular" products will be more common as they already are in the Mac and Windows desktop arena.

Impacts...

- **Information Framework:** Although each of object orientation's facets offer benefits (analysis, design, programming, database, even modeling) it is the prospect that an entire system framework may be devised from these technologies that unifies the life cycle, the documentation, the notation, the terminology, and the underlying abstractions. To achieve this framework will require IS reorientation, task structure, organization structure, job structure and titles, IS - user communication paths, and above all an organizational commitment to the object oriented paradigm.