System Construction

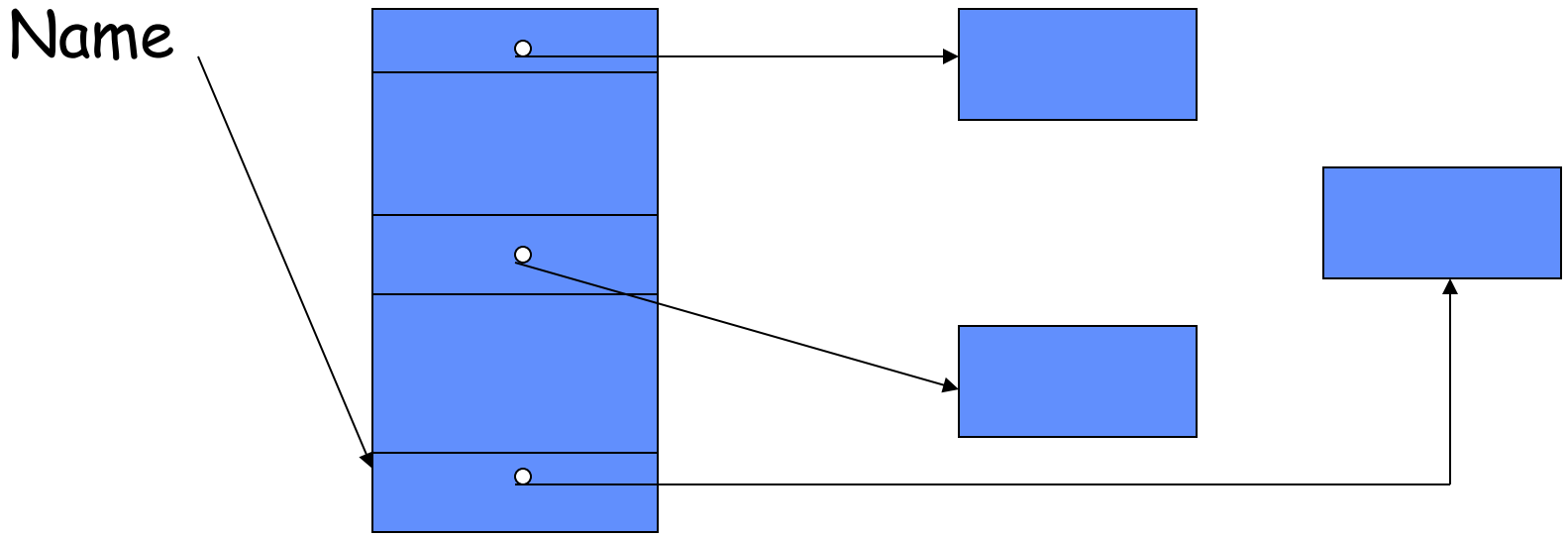
- Assume coding as a basic skill
- Problem: how to assemble systems
  - Statically
  - Dynamically
- Basic issue:
  - linking component references to components
  - This is typically too big a job for a compiler to handle
- Static assembly: build facility
  - Create a dependency graph
  - Determine what in the system has changed (interfaces)
  - Determine what depends on changed (interfaces)
  - Recompile dependencies
  - Link components
  - Check for incompatibilities
  - Resolve incompatibilities – change components
  - Cycle until no more incompatibilities

System Construction

- Build Roles
  - Build owner – coordinates the process
  - Developers – responsible for the components
  - Build administrator – does build according to the guide book
  - Build assistants – problem hackers
- Most automated part of building systems
  - Still need tool support
  - Still human intensive
- Reality
  - In really large systems, can take days, weeks
  - Large number of builds of the same system
    - Different purposes: local use, system test, etc
    - Faults discovered at build time
  - Large amount of time to eliminate faults
    - Isolate fault, determine responsibility, negotiate solution
  - Often lack sufficient resources

System Construction

- Dynamic assembly
  - Typical structure: indirection (*late binding*)
    - (name, link) (link, component)
    - Name → link structure → component
    - Update by replacing link

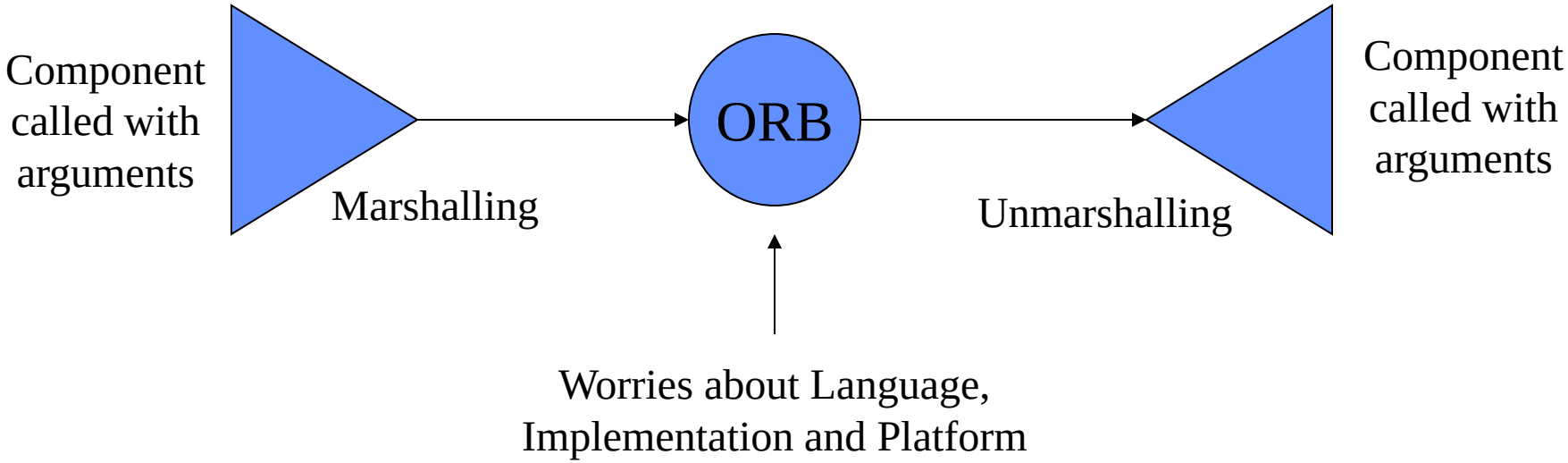


OMG CORBA

- OMG CORBA
  - Common object request broker architecture
  - Extends build/link problem to include
    - Components build in different languages
    - Components running on different platforms
    - In distributed systems
  - Provides basic wiring – ie,
    - a standard connector for arbitrary components
  - Goal: open interconnection
    - Provide high level protocols as standards
    - CORBA compliant: adhering to these standards
    - Problem: costly, not as efficient as "binary" interconnections
      - ✓Ie, shoving bits back and forth as in COM

CORBA

- CORBA's structure: 3 parts
  - Set of invocation interfaces
  - The object request broker (ORB)
  - Set of object adapters



CORBA

- Method invocations and object adapters
  - Various degrees of "late binding"
  - Component called with arguments
  - Data "marshalled" and sent to the ORB
  - ORB worries about language, implementation and platform issues
  - Data "unmarshalled" at appropriate place
  - Desired component called with appropriate arguments
- Internal details
  - Use IDL (intermediate definition language) as intermediary
  - Generate stubs and skeletons
    - Stub looks like local object, forwards to real object
    - Skeleton gets data and invokes target object
  - Works well with standard method invocations

CORBA

- Registration
  - Server programs register with the ORB
    - ORB then knows how to invoke and where
  - Pure applications do not register
    - Not startable by the ORB
- Beyond basic wiring
  - Naming – white pages
  - Security
  - Object trader services – yellow pages
  - Transactions – OTS
    - One of the most important services
    - Maintains a current transaction context
    - Objects must have/implement an interface *TransactionObject*
      - ✓Begin, commit, rollback
    - Resources have to implement interface *Resource* (2 phase commit)

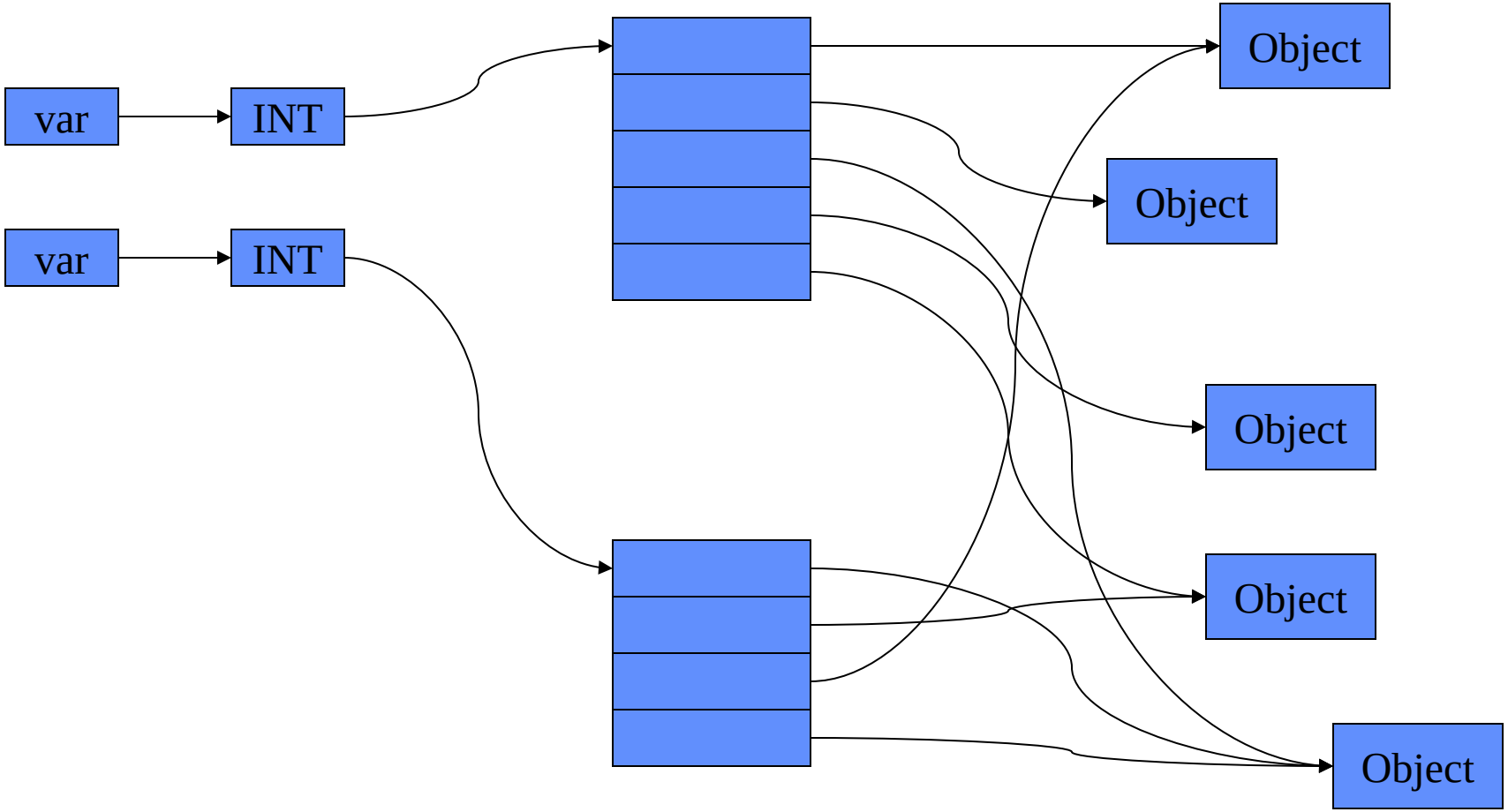
CORBA: Fine Grain Services

- Change management services – versioning
- Concurrency services – locks
- Event notification
- Externalization
- Licensing
- Life cycle
- Objects collection (of standard library objects)
- Object query service (OQL & SQL)
- Persistent object services
- Properties services
- Relationship services
- Time service

Microsoft COM

- OLE + Active X
- A binary standard
  - Specifies nothing about how a particular programming language may be bound to it
  - Just shoves bits from one place to another
- Fundamental entity: Interface
  - A pointer to an interface node
  - Interface node points to a table of procedure variables
  - Hence, uses *double* indirection
  - Methods need notion of *self* or *this*
  - Can have multiple interfaces implemented by the same set of objects

Microsoft COM



Microsoft COM

- How does a client learn about other interfaces?
- How does one compare the identity of COM objects?
  - QUERY_INTEFACE
    - checks for named interfaces
    - Gets a unique ID
  - IUNKNOWN
    - Used to identify the entire COM object
- Objects are reference counted to keep track of referring interfaces

Microsoft COM

- 2 forms of composition supported
- Containment
  - One object has exclusive reference to another
  - Outer object conceptually contains the inner object
  - Inner object transparent to client
  - Enables reuse of components contained in the outer implementation
- Aggregation
  - Use case hierarchies and forward are expensive
  - Exports aggregated interfaces so directly callable
  - Problems
    - If need filtering, interpretation etc
    - Dependencies on specific object
    - Lose transparency of containment
  - Requires inner objects to collaborate
  - Can be used to construct efficient generic wrappers

Microsoft COM

- Interfaces and "polymorphism"
  - Actually overloading
    - Distinct implementations for each signature (ie parameters)
    - Different on the basis of the argument types
  - Polymorphism
    - One single implementation
    - Language handles distinct types of arguments
- Other COM services
  - Distribution – requires proxies and stubs
  - Uniform data transfer
  - Dispatch interfaces
  - Outgoing interfaces and connectable objects
  - COM+ - provides transactional services

Side Note on the Deployment Paper

- ACM SIGSOFT Impact Award
- To be awarded at ACM SIGSOFT FSE
- *A Design Framework for Internet-scale Event Observation and Notification* by David Rosenblum (UCL) and Alexander Wolf (ICL) – ESEC/FSE 1997
  - This widely cited paper has been very influential in promoting the publish/subscribe coordination paradigm for large, Internet-scale distributed systems. Publish/subscribe middleware can provide the necessary run-time support to evolvable and adaptable architectures, which are becoming more and more important to support modern service-based applications.
- Extension of the work we will talk about today
- Mentored both at Bell Labs
- Alex co-author on the SW Architecture paper