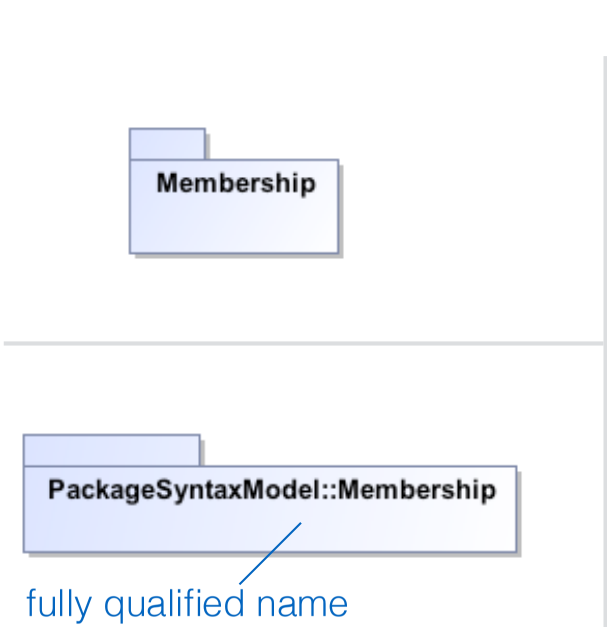
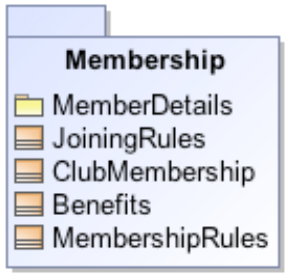
Analysis - packages

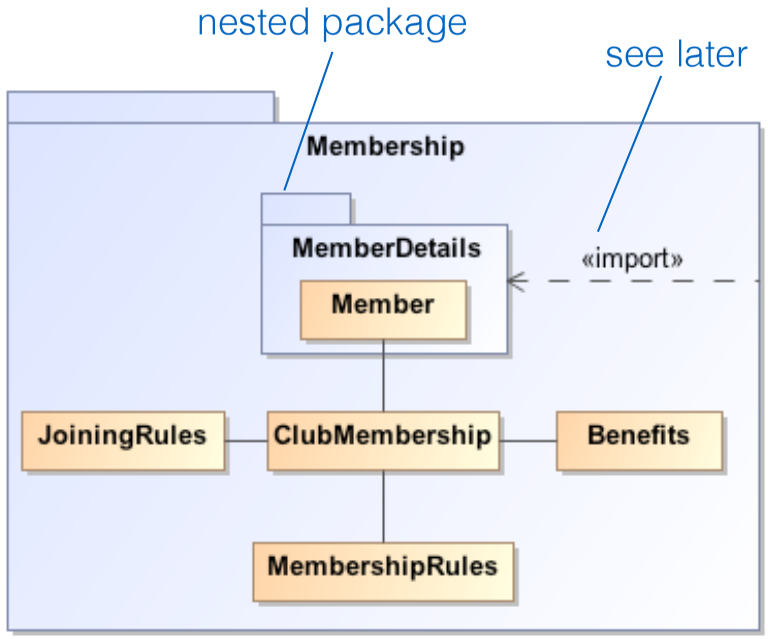
Packages

- A package is a general purpose mechanism for organizing model elements into semantically related groups. Packages define “semantic boundaries” in the model and can provide units for parallel working and configuration management
- Each package defines an encapsulated namespace, i.e. all names must be unique within the package
- In UML 2 a package is a purely logical grouping mechanism - use components for physical grouping
- Every model element is owned by exactly one package, and this forms a hierarchy rooted in a top level package that can be stereotyped «topLevel»
- Analysis packages contain use cases, analysis classes, use case realizations, analysis packages

Package syntax







Standard package stereotypes	
«Framework»	A package that contains model elements that specify a reusable architecture
«ModelLibrary»	A package that contains elements that are intended to be reused by other packages. Analogous to a class library in Java, C# etc.

Package element visibility

- Package elements may be given a visibility of public, private, protected or package. If an element is visible within a package then it is *visible within all nested packages,* e.g. on the previous slide Benefits is visible within MemberDetails

- To make an element of a nested package visible to its containing package we need to use an «import», «access» or «merge» dependency as shown on the previous slide

Package element visibility semantics

Visibility	Semantics
public	public contents of a containing package are visible externally and within contained packages
private	private contents of a package are only visible within that package
protected	protected contents of a package are only visible when the package is merged with another package
package	Contents with package visibility are only visible within the package and within contained packages

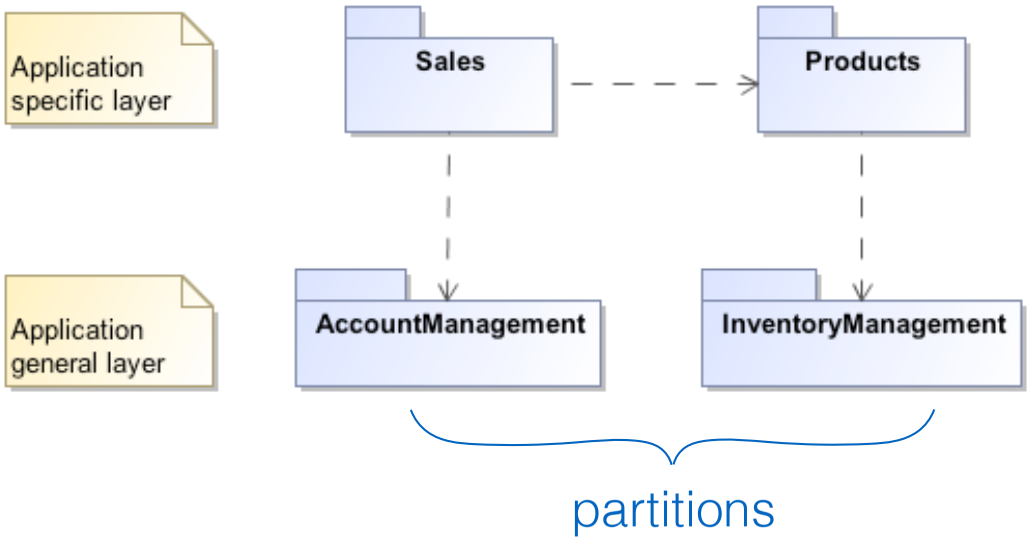
- Note: nested packages can't have protected or package visibility

Package dependencies

Stereotype	Semantics	Transitive
«use»	An element in the client uses an element in the supplier in some way. The client depends on the supplier	Yes
«import»	Public elements of the supplier namespace are added as public elements to the client namespace	Yes
«access»	Public elements of the supplier namespace are added as private elements to the client namespace	No
«trace»	«trace» usually represents an historical development of one element into another more refined version. It is often an extra-model relationship	Yes
«merge»	The client package merges the public contents of its supplier packages. This is a complex relationship only used for metamodeling - you can generally ignore it	Yes

Transitivity: if A depends on B, and B depends on C, then A depends on C i.e. there is an implicit dependency between A and C

Architectural analysis



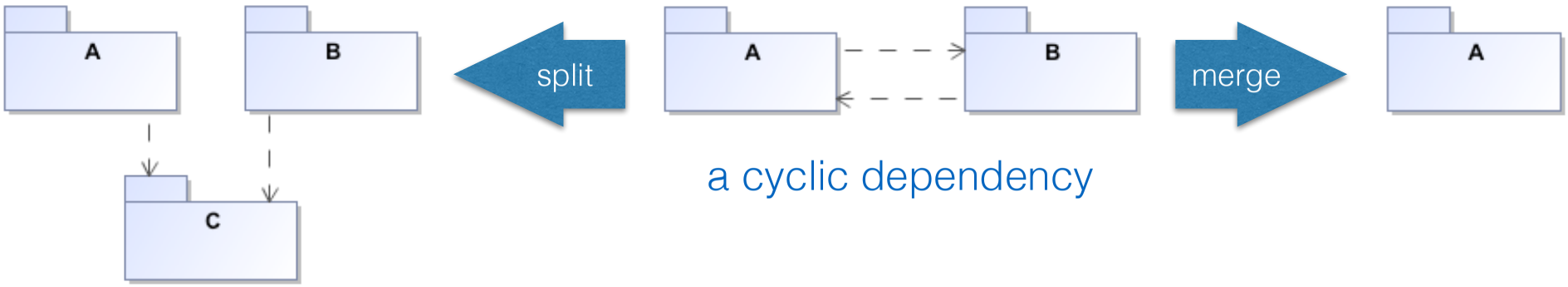
- Organize the analysis classes into a set of cohesive packages
- The architecture should be layered and partitioned to separate concerns, e.g. it's useful to layer analysis models into application specific and application general layers
- To minimize coupling between packages, each package should have the minimum number of public or protected elements

Finding analysis packages

- These are often discovered as the model matures
- We can use the natural groupings in the use case model to help identify analysis packages:
  - One or more use cases that support a particular business process or actor
  - Related use cases
- Analysis classes that realize these groupings will often be part of the same analysis package
- Be careful, as it is common for use cases to cut across analysis packages! One class may realize several use cases that are allocated to different packages

Analysis package guidelines

- A cohesive group of closely related classes or a class hierarchy and supporting classes. Refine packages as analysis progresses
- Minimize all dependencies between packages, including nesting
- Localize business processes in packages where possible
- Don't worry about dependency stereotypes
- 4 to 10 analysis classes per package is usually about right
- Avoid cyclic dependencies!



Summary

- Packages are the UML way of grouping modeling elements. Packages may be nested
- There are «use», «import», «access», «trace»and «merge» dependency relationships between packages
- The package structure of the analysis model defines the logical system architecture