# BoBs: Breakable Objects

## Building blocks for flexible application architectures

**Ph.D. *Viva-Voce***
**July 23rd 2007**

Vikram Jamwal
Advisor: Prof. Sridhar Iyer
IIT Bombay, Mumbai

# Contents

- Problem and Motivations
- Breakable Object (BoB) basics
- BoB for application partitioning
- BoB as elements of reuse
- Related work comparisons
- Discussion and conclusions

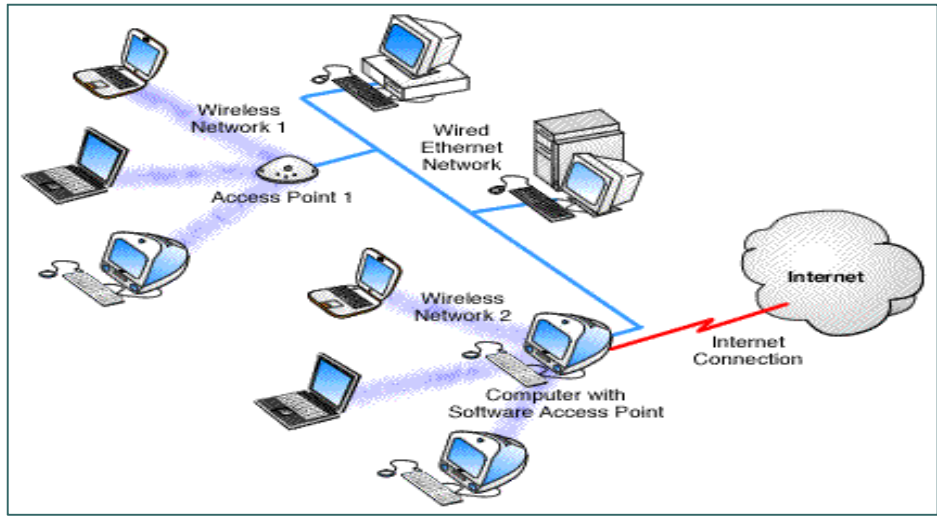
# Contents

- **Problem and Motivations**
- BoB basics
- BoBs for application partitioning
- BoBs as elements of reuse
- Related work comparisons
- Discussion and conclusions

# Given

- Computing devices with various capabilities
- Networks which are diverse
- **Applications need to adapt**
  - **to different**

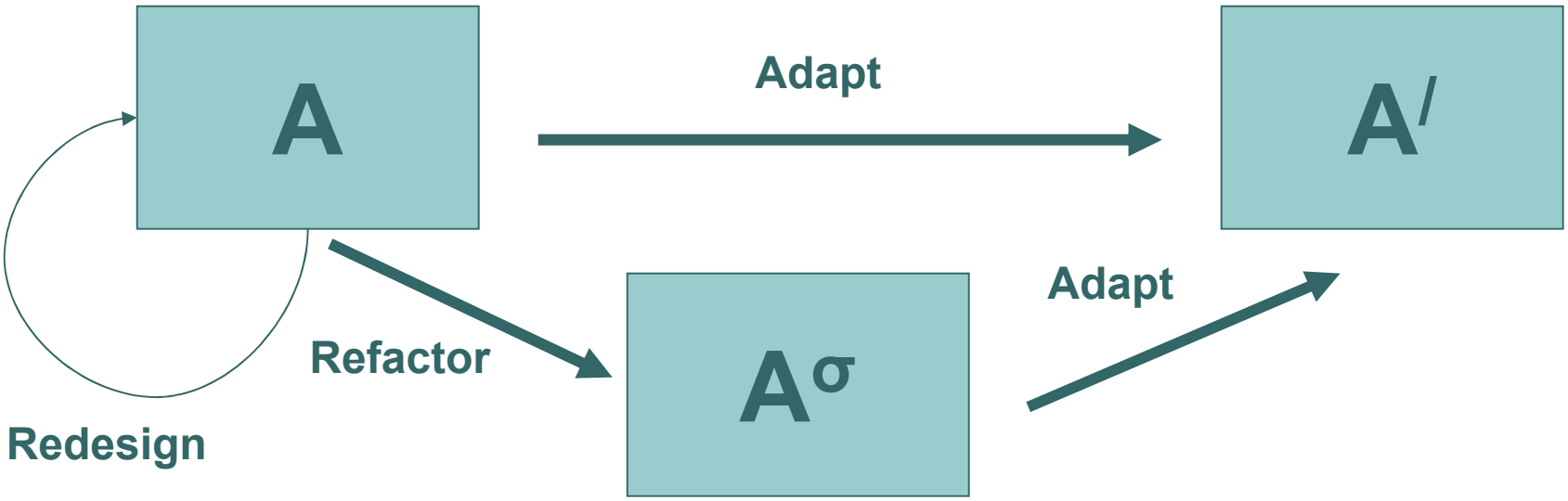
**deployment scenarios**



# Furthermore…

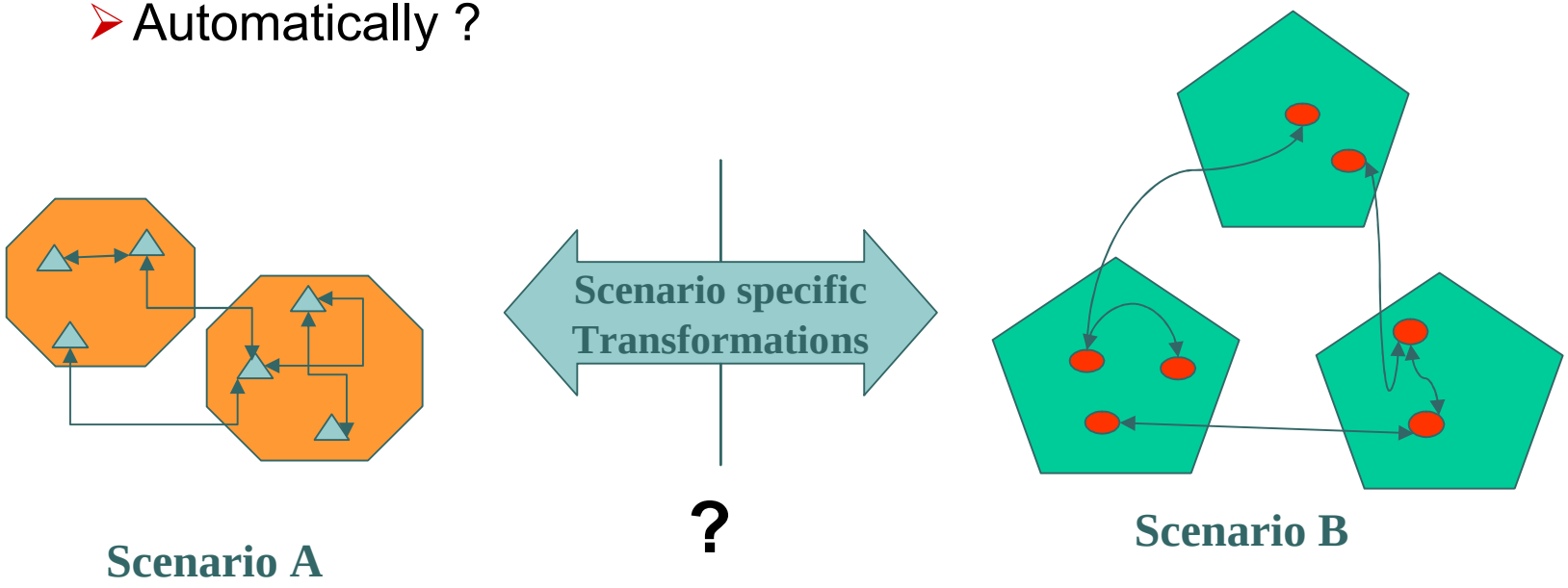
- **Applications need to evolve**
  - Change in user requirements
  - Different sets of user requirements
  - Restructuring for
    - better maintenance
    - better performance

# Application adaptation



# Broader Problem

- Design and implement an application
- Such that
  - Given the **software** for one scenario, we can generate a version of application for a new scenario
    - Through refactoring ( and / or ) adaptation
    - Easily
    - Automatically ?



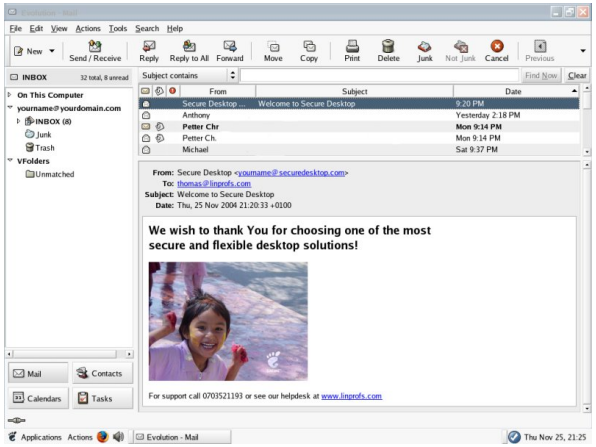
# Problems:

1. Environmental Heterogeneity
2. Distribution of Components
3. Functionality Partitioning

- (1) and (2) active areas of research, many solutions exist
- **(3) still requires adequate attention**

# Motivating Example

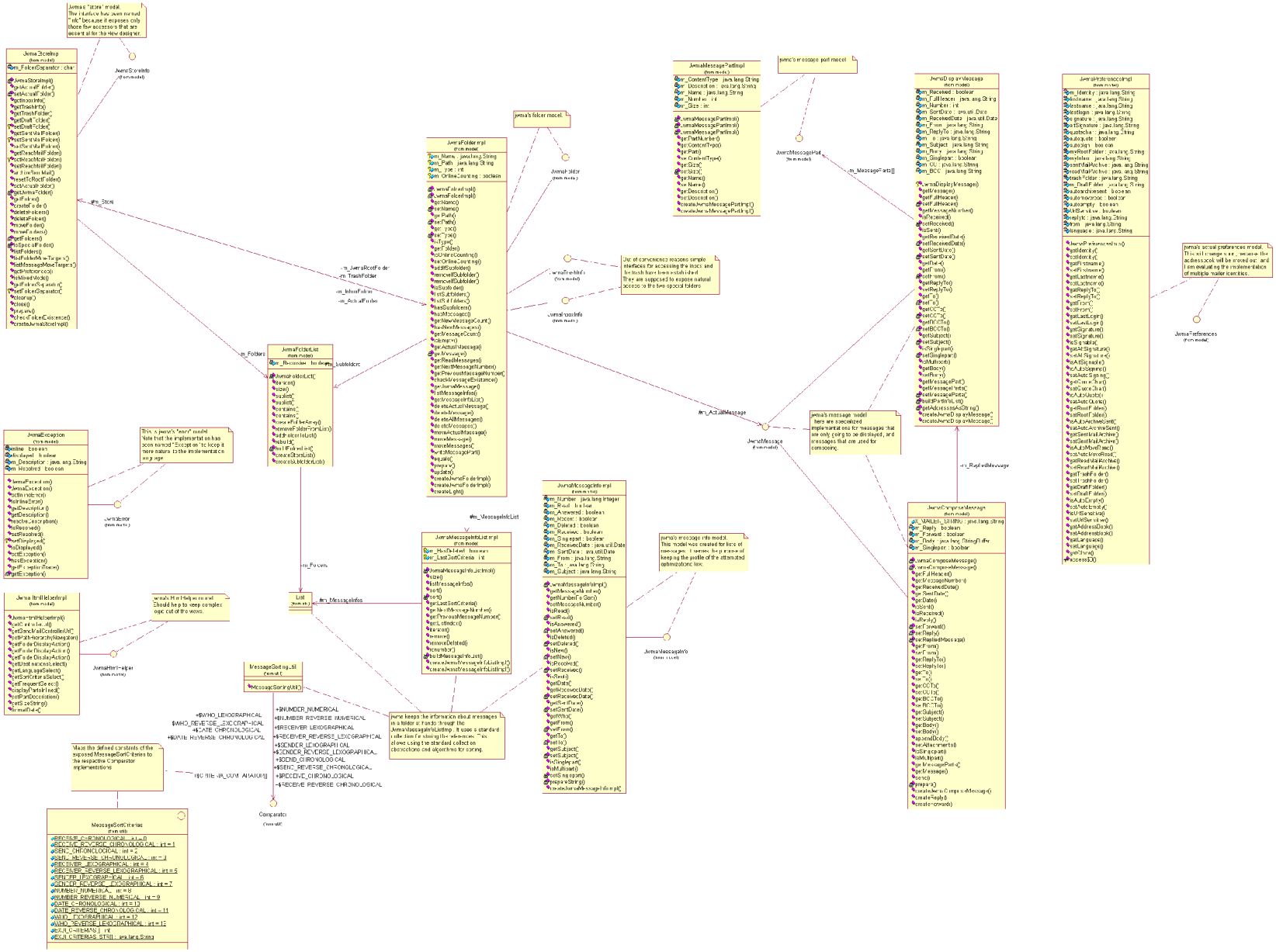
- E-mail Application
- Different devices
  - PC, Web-Client, Mobile Device
- Different Modes / Scenarios
  - On-line
  - Disconnected
  - Off-line




# Functionality Partitioning

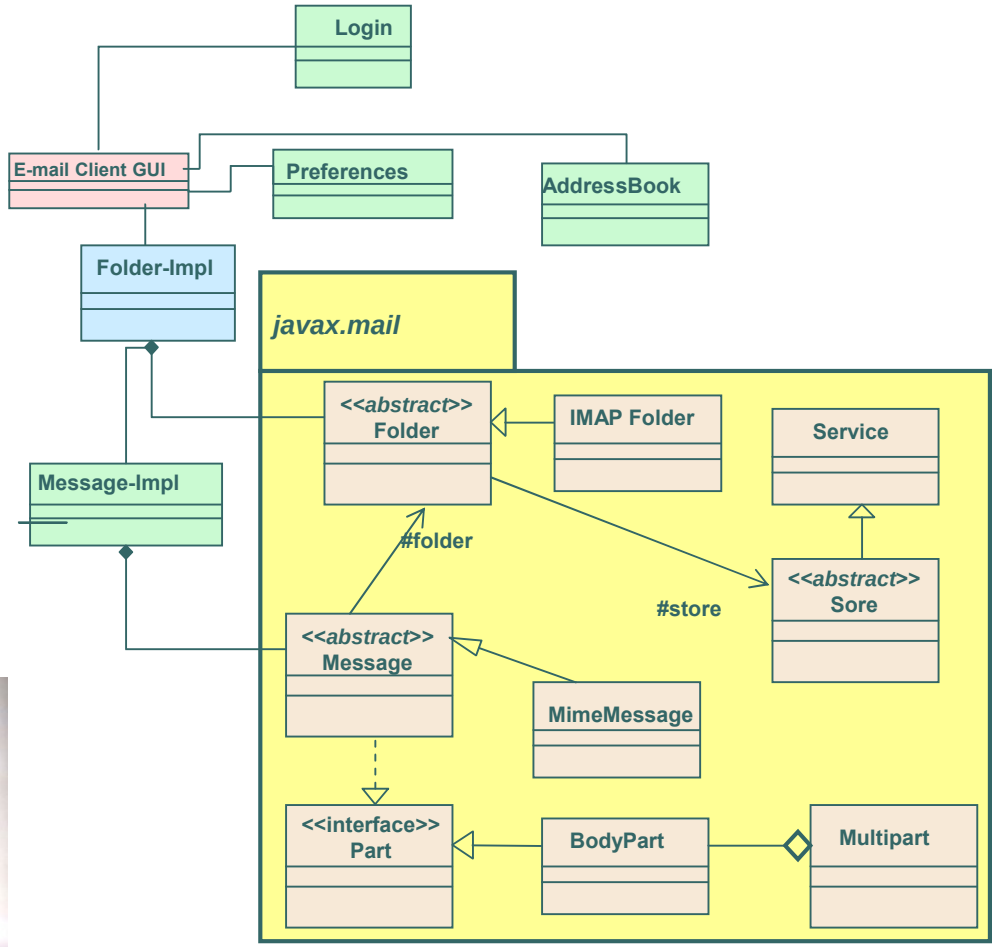
- Apportioning application functionality into deployment specific component sub-sets
- Difficult to achieve in practice because we cannot draw **clean lines of separation**
  - Some functionality may span across multiple classes
  - A single class may contain multiple functionality

# E-mail Internals(jwma)



# E-mail-design

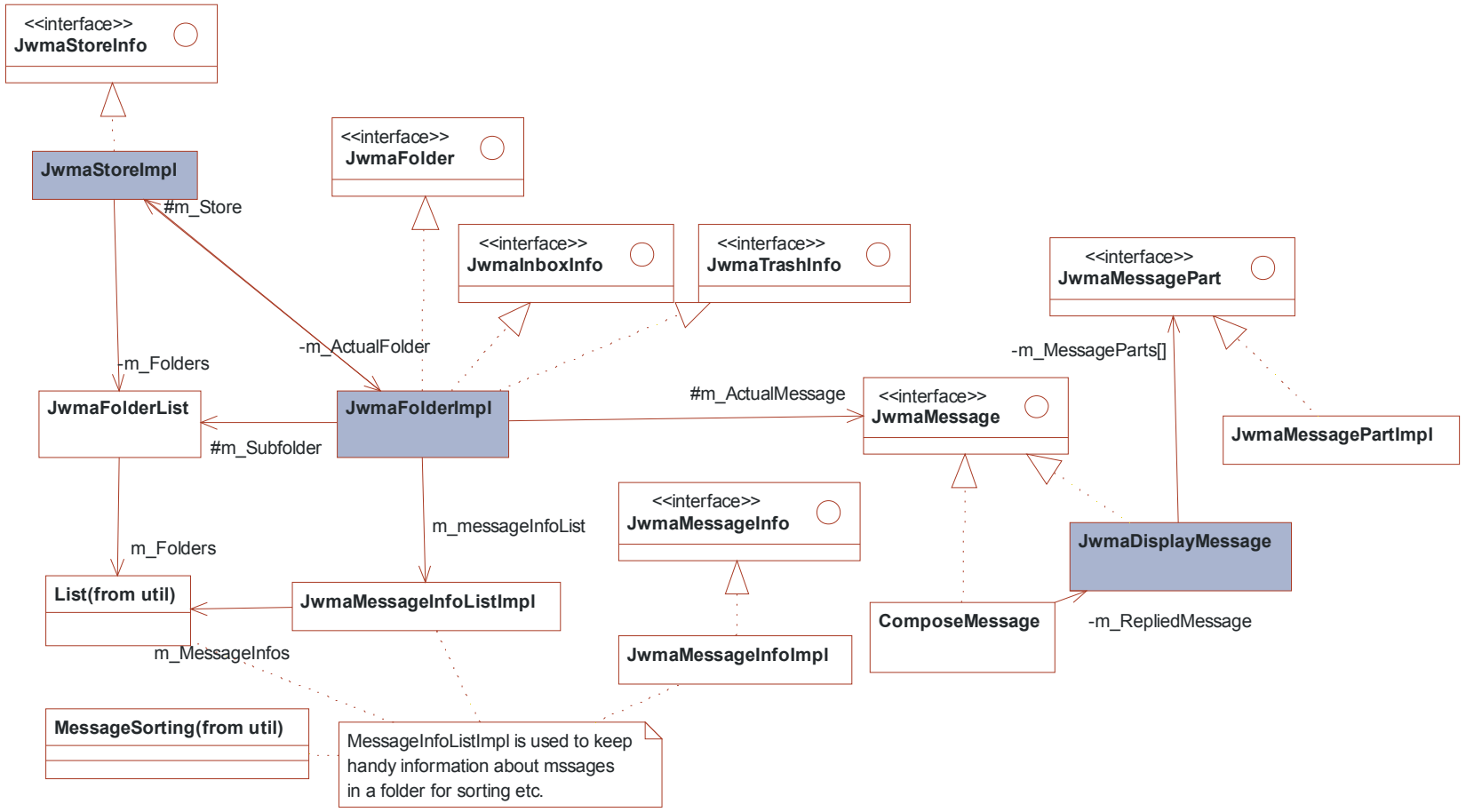
- Different **partitioning** for different versions
- **`Store`** class on server
- **`Folder`** class partitioned between [illegible] and serve[illegible]



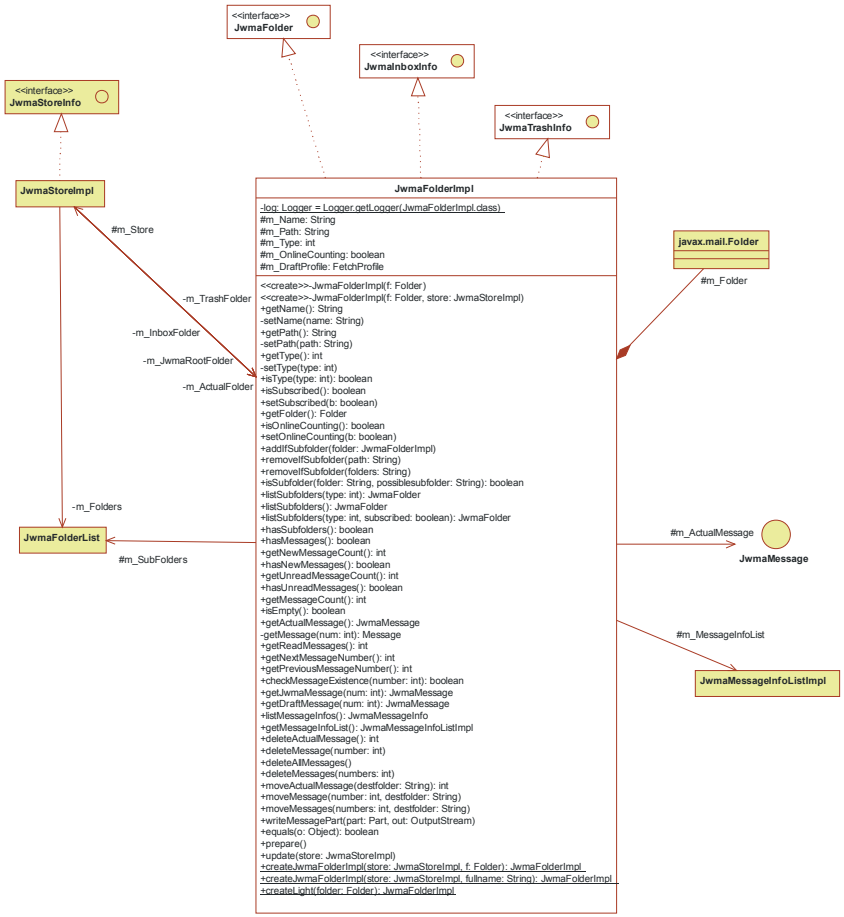
# More specific problem:

- Design and implement an OO application such that
  - functionality of constituent objects is factorable
- The part-units
  - are of desired granularity
  - can be easily extracted
  - are reusable
- **How?**

# Jwma Classes



# Jwma Folder

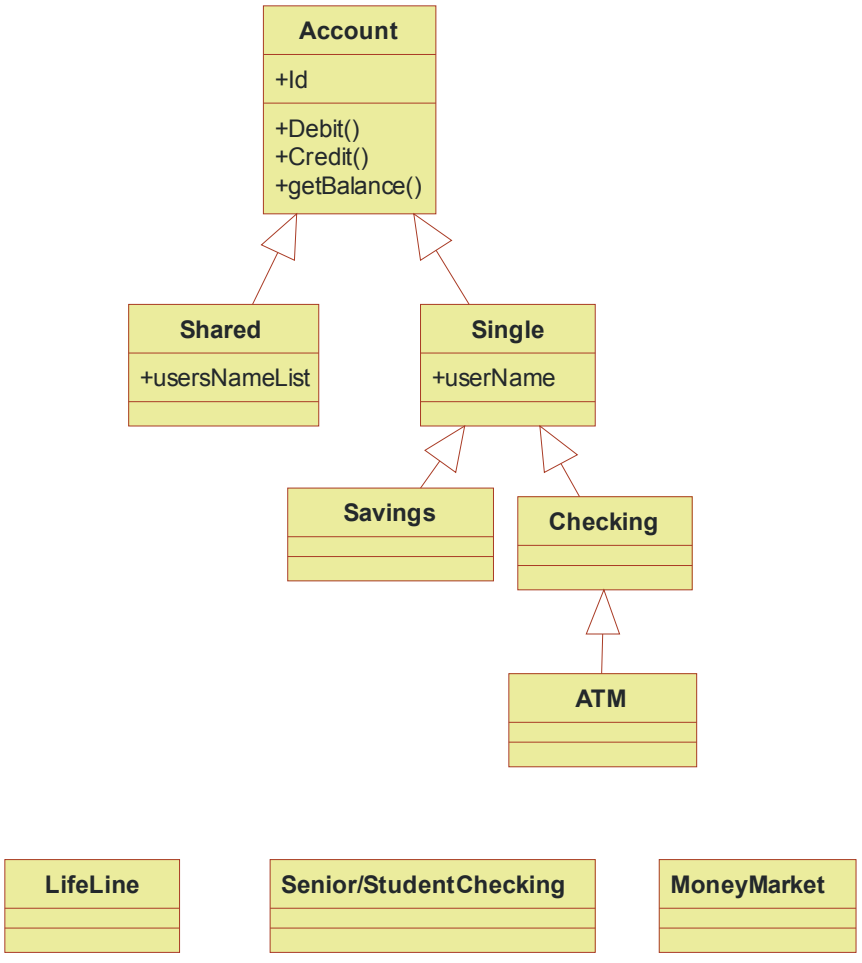

<<interface>>
JwmaFolder
<<interface>>
JwmaInboxInfo
<<interface>>
JwmaStoreInfo
<<interface>>
JwmaTrashInfo
JwmaStoreImpl
JwmaFolderImpl
-log: Logger = Logger.getLogger(JwmaFolderImpl.class)
#m_Name: String
#m_Path: String
#m_Type: int
#m_OnlineCounting: boolean
#m_DraftProfile: FetchProfile
<<create>>-JwmaFolderImpl(f: Folder)
<<create>>-JwmaFolderImpl(f: Folder, store: JwmaStoreImpl)
+getName(): String
-setName(name: String)
+getPath(): String
-setPath(path: String)
+getType(): int
-setType(type: int)
+isType(type: int): boolean
+isSubscribed(): boolean
+setSubscribed(b: boolean)
+getFolder(): Folder
+isOnlineCounting(): boolean
+setOnlineCounting(b: boolean)
+addIfSubfolder(folder: JwmaFolderImpl)
+removeIfSubfolder(path: String)
+removeIfSubfolder(folders: String)
+isSubfolder(folder: String, possiblesubfolder: String): boolean
+listSubfolders(type: int): JwmaFolder
+listSubfolders(): JwmaFolder
+listSubfolders(type: int, subscribed: boolean): JwmaFolder
+hasSubfolders(): boolean
+hasMessages(): boolean
+getNewMessageCount(): int
+hasNewMessages(): boolean
+getUnreadMessageCount(): int
+hasUnreadMessages(): boolean
+getMessageCount(): int
+isEmpty(): boolean
+getActualMessage(): JwmaMessage
-getMessage(num: int): Message
+getReadMessages(): int
+getNextMessageNumber(): int
+getPreviousMessageNumber(): int
+checkMessageExistence(number: int): boolean
+getJwmaMessage(num: int): JwmaMessage
+getDraftMessage(num: int): JwmaMessage
+listMessageInfos(): JwmaMessageInfo
+getMessageInfoList(): JwmaMessageInfoListImpl
+deleteActualMessage(): int
+deleteMessage(number: int)
+deleteAllMessages()
+deleteMessages(numbers: int)
+moveActualMessage(destfolder: String): int
+moveMessage(number: int, destfolder: String)
+moveMessages(numbers: int, destfolder: String)
+writeMessagePart(part: Part, out: OutputStream)
+equals(o: Object): boolean
+prepare()
+update(store: JwmaStoreImpl)
+createJwmaFolderImpl(store: JwmaStoreImpl, f: Folder): JwmaFolderImpl
+createJwmaFolderImpl(store: JwmaStoreImpl, fullname: String): JwmaFolderImpl
+createLight(folder: Folder): JwmaFolderImpl
javax.mail.Folder
#m_Folder
#m_Store
-m_TrashFolder
-m_InboxFolder
-m_JwmaRootFolder
-m_ActualFolder
-m_Folders
JwmaFolderList
#m_SubFolders
#m_ActualMessage
JwmaMessage
#m_MessageInfoList
JwmaMessageInfoListImpl

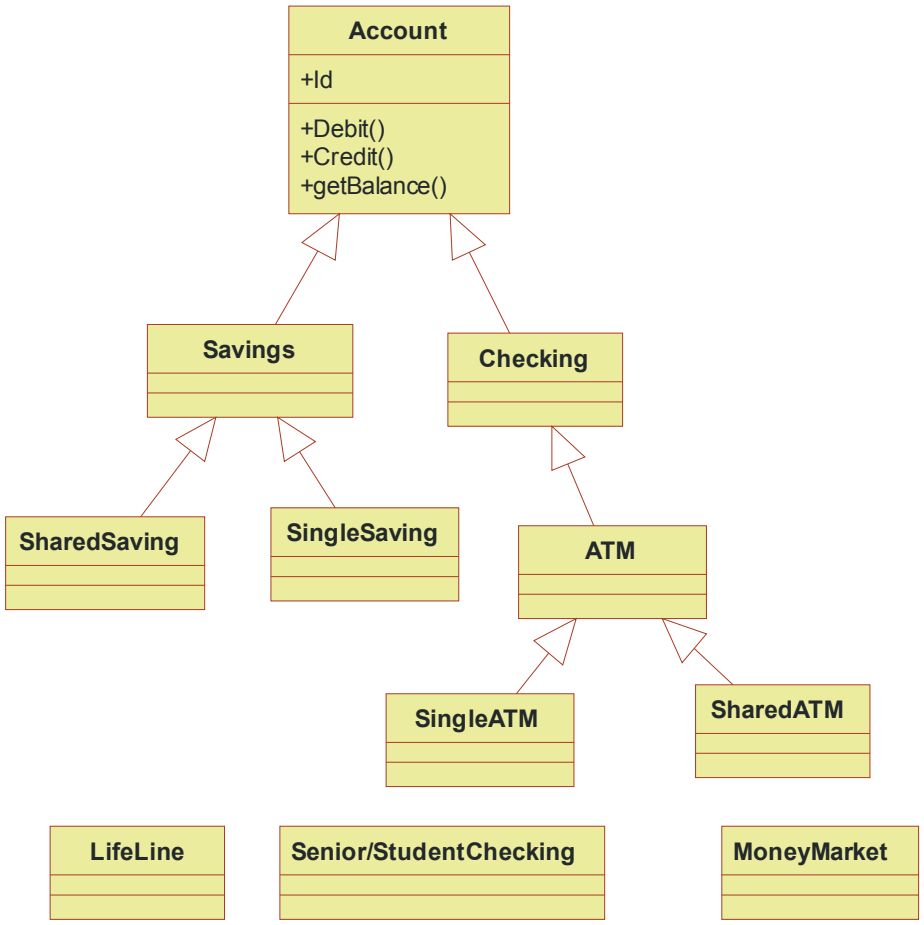
# Our solution:

- Reformulate basic component of the application so that it can be readily partitioned into sub-components
- Hence **Breakable Objects (BoBs)**

- Main Advantage*:*
  - Designing and implementing applications using BoBs makes them *more flexible;* specifically, more *amenable to partitioning*.

# Other issues:



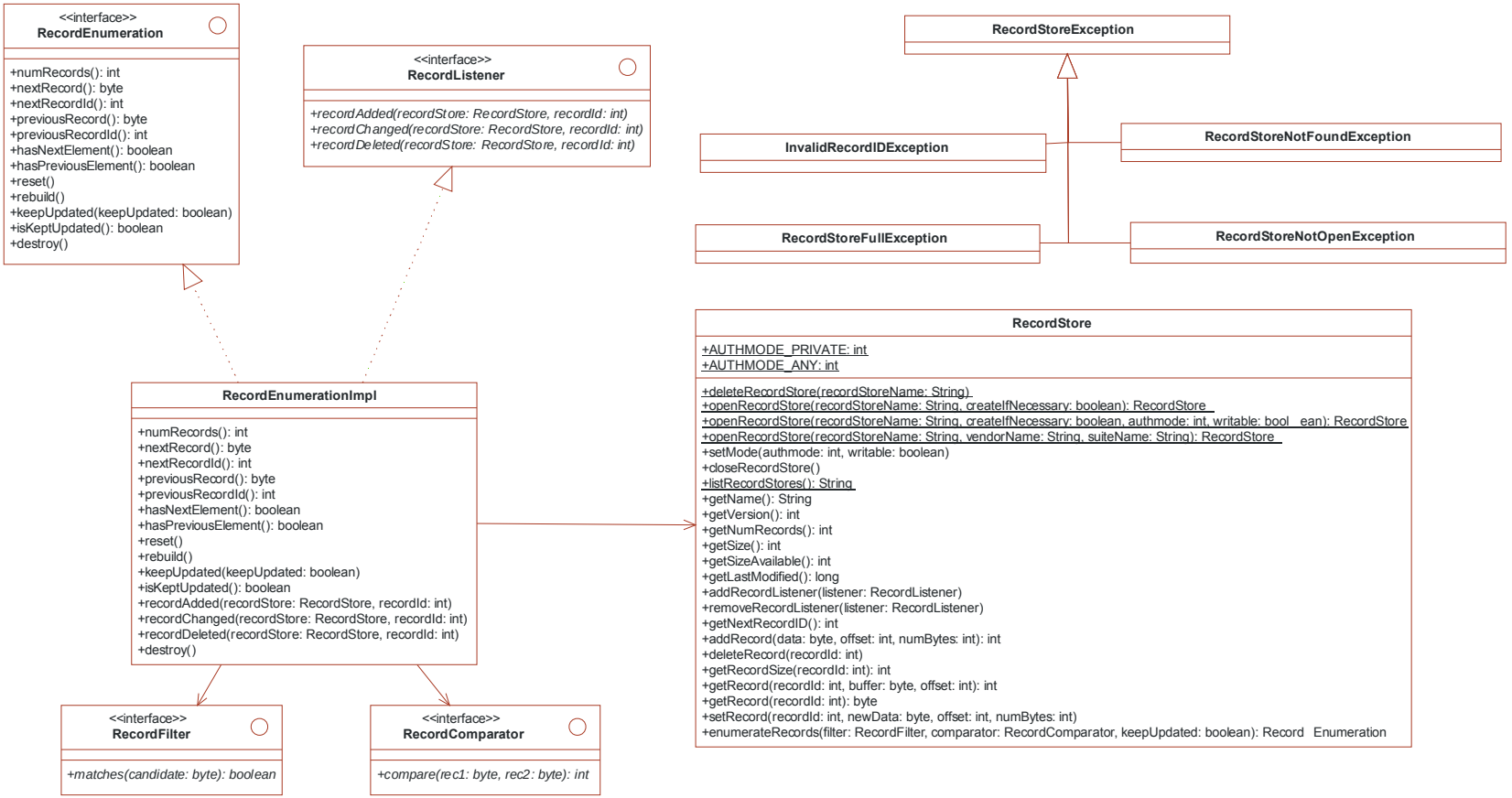
# Problems with inheritance and composibility

- Decomposition
  - Duplicated Features
  - Inappropriate hierarchies
  - Duplicated wrappers
- Composition
  - Conflicting Features
  - Fragile Hierarchies

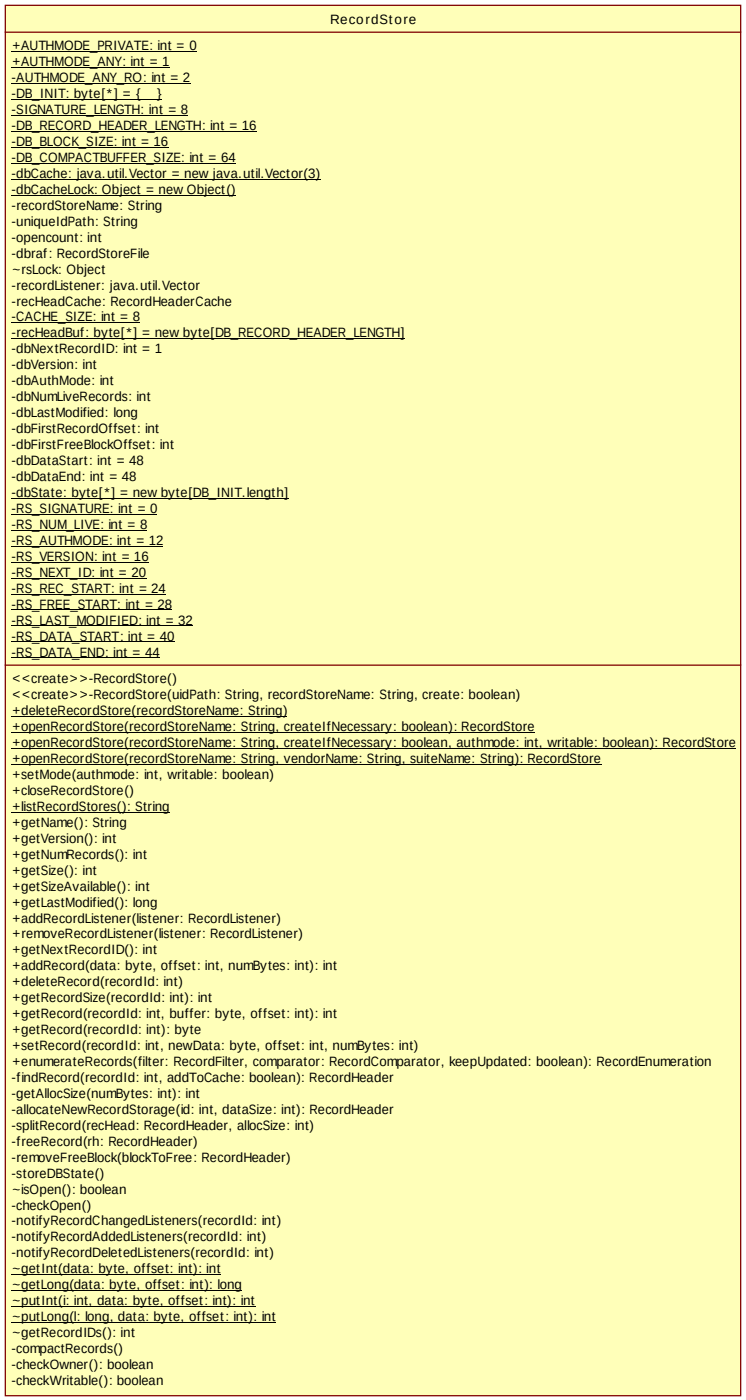
# Other issues:

- Inheritance based composition mechanisms
- Problem of software contraction
- Large software sizes
  - Heavier and more complex versions

# Javax Record Store



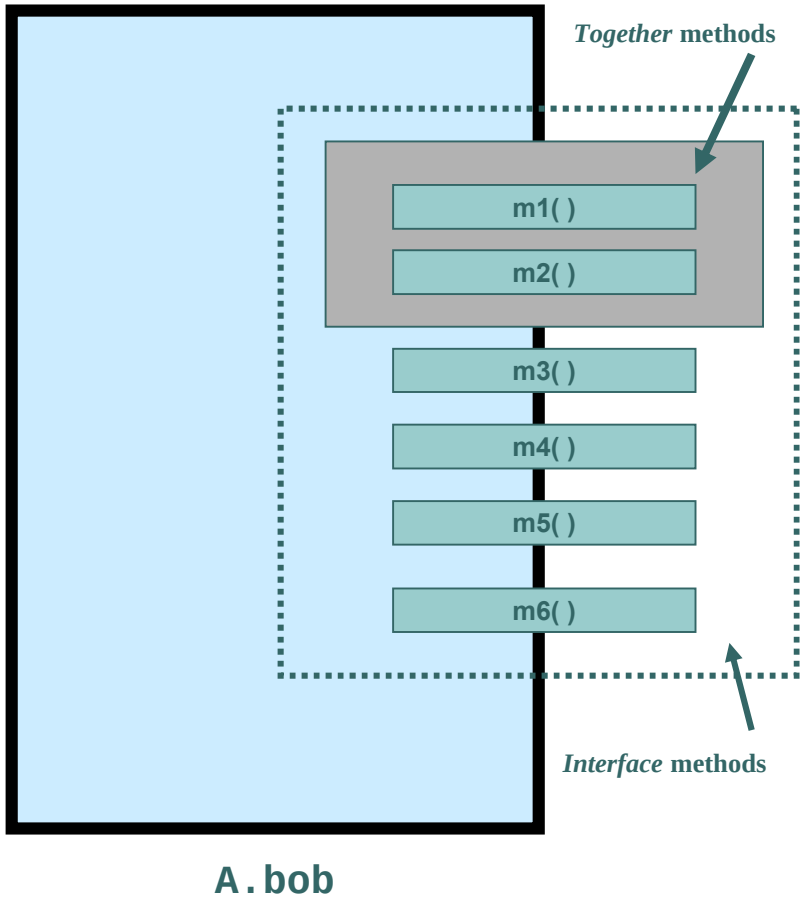
# Implementation



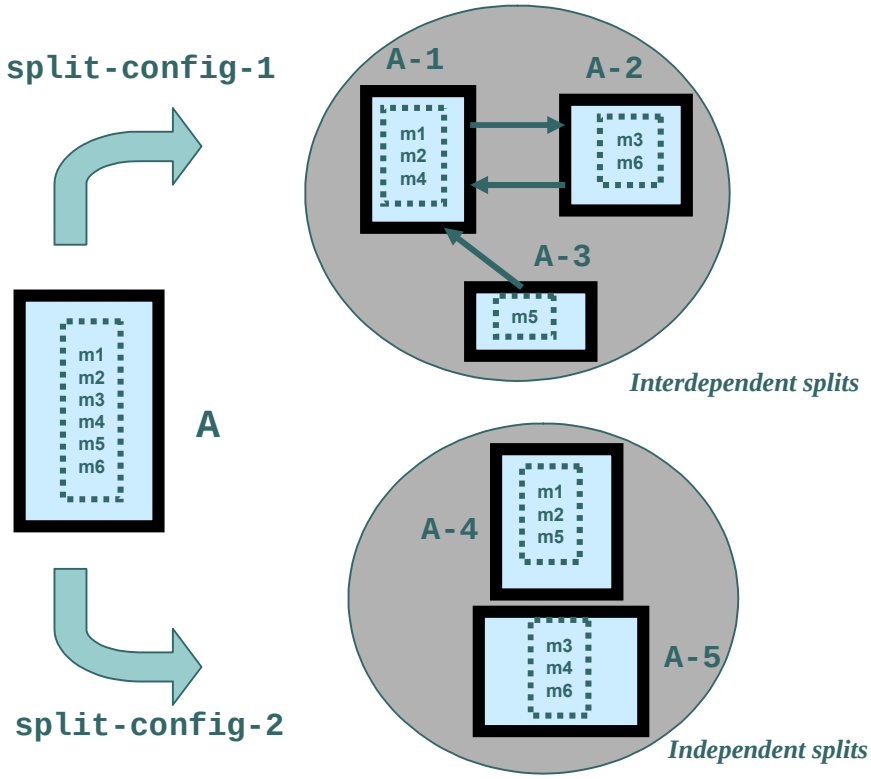
# Contents

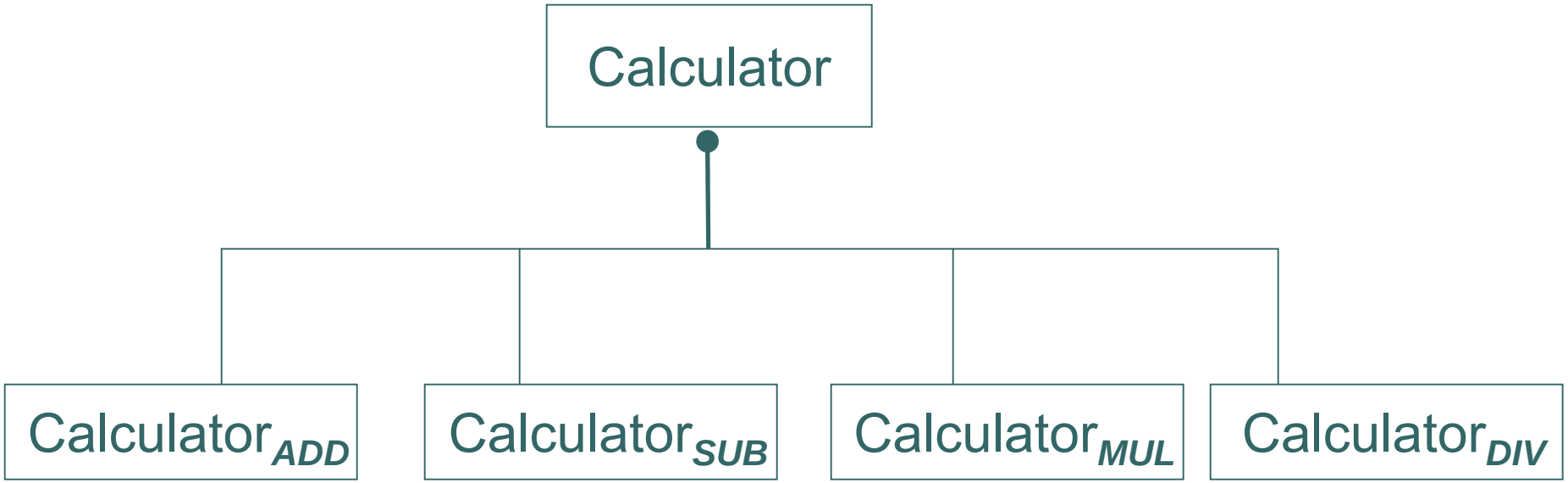
- Problem and Motivations
- **BoB basics**
- BoBs for application partitioning
- BoBs as elements of reuse
- Related work comparisons
- Discussion and conclusions

# BoB



# BoB Splits


split-config-1
A-1
A-2
m1
m2
m4
m3
m6
A-3
m5
Interdependent splits
m1
m2
m3
m4
m5
m6
A
A-4
m1
m2
m5
m3
m4
m6
A-5
Independent splits
split-config-2

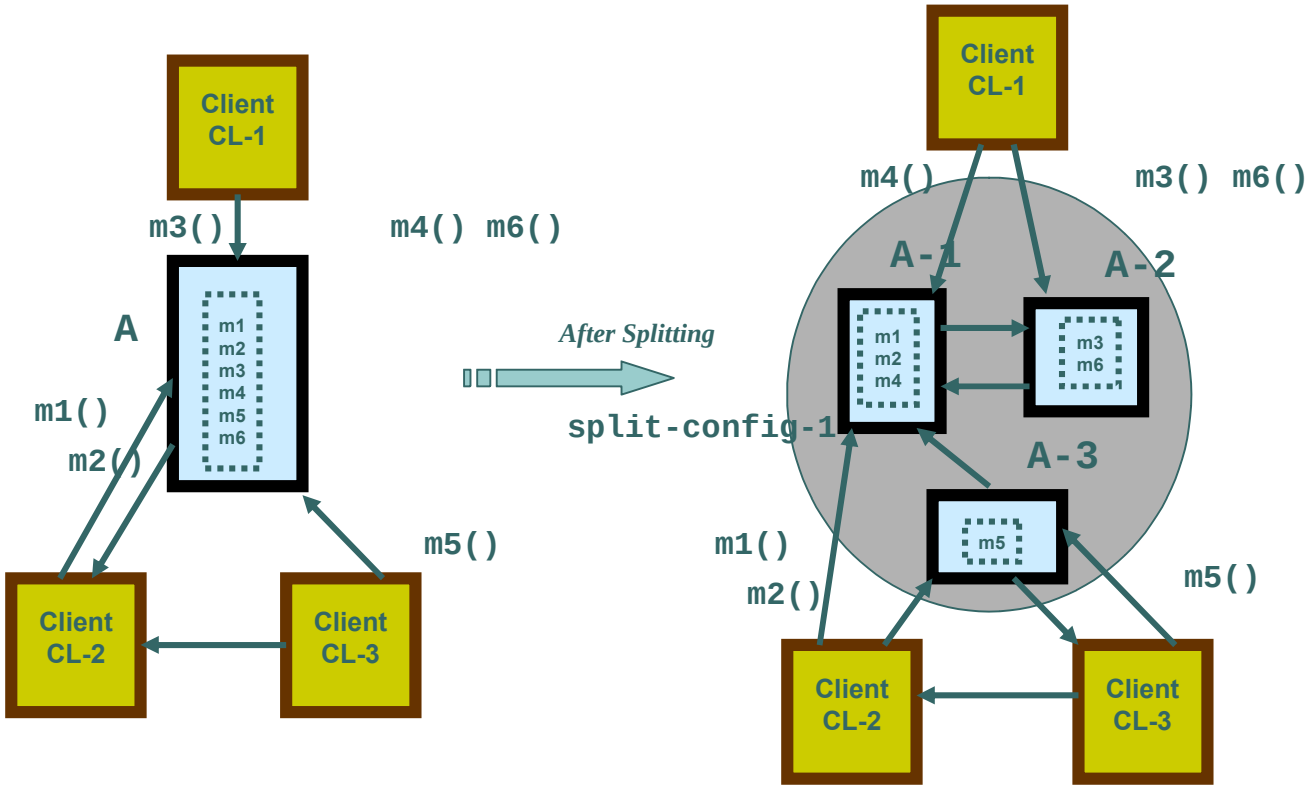



Denotes *is-split-of* and *is-principal-of* relationships. The thick head lies towards the principal class's side.

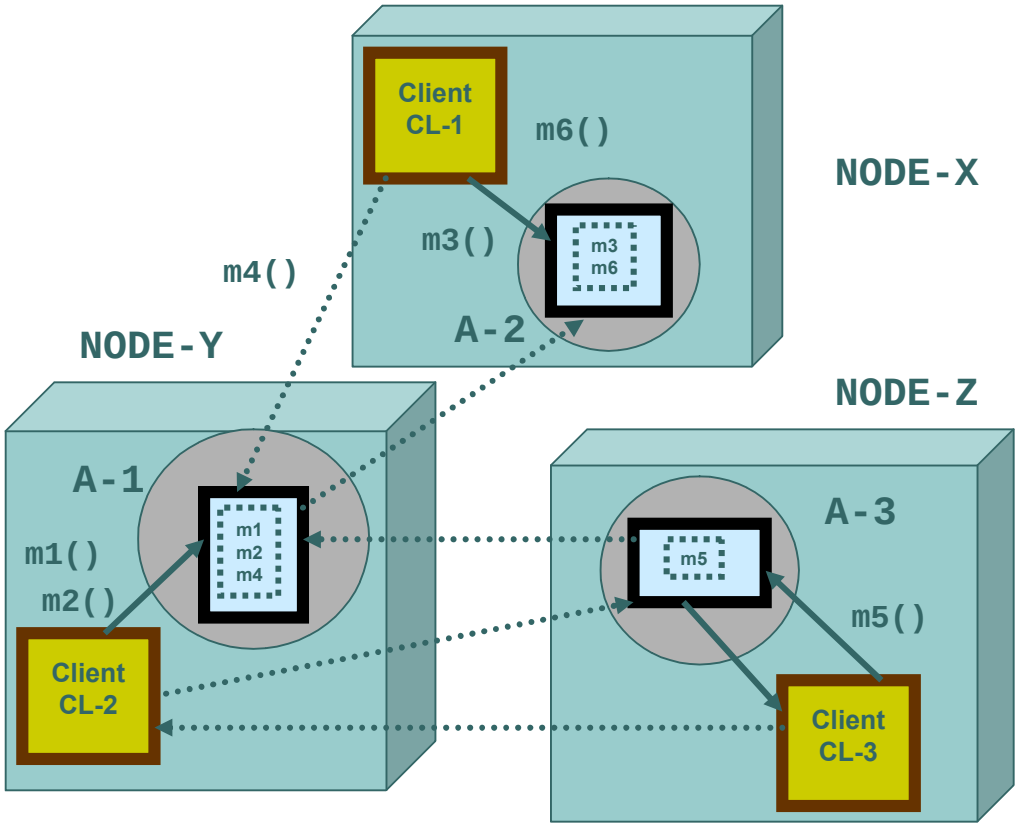
# Contents

- Problem and Motivations
- BoB basics
- **BoBs for application partitioning**
- BoBs as elements of reuse
- Related work comparisons
- Discussion and conclusions

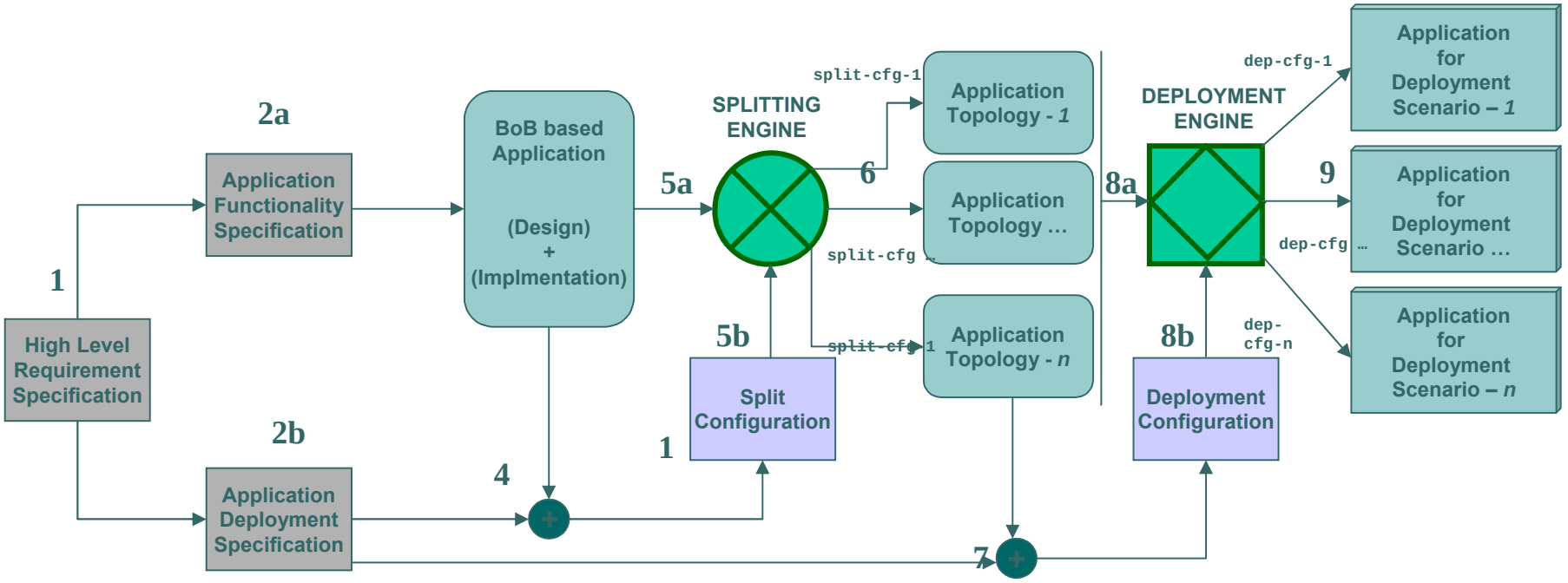
# Program Reorganization



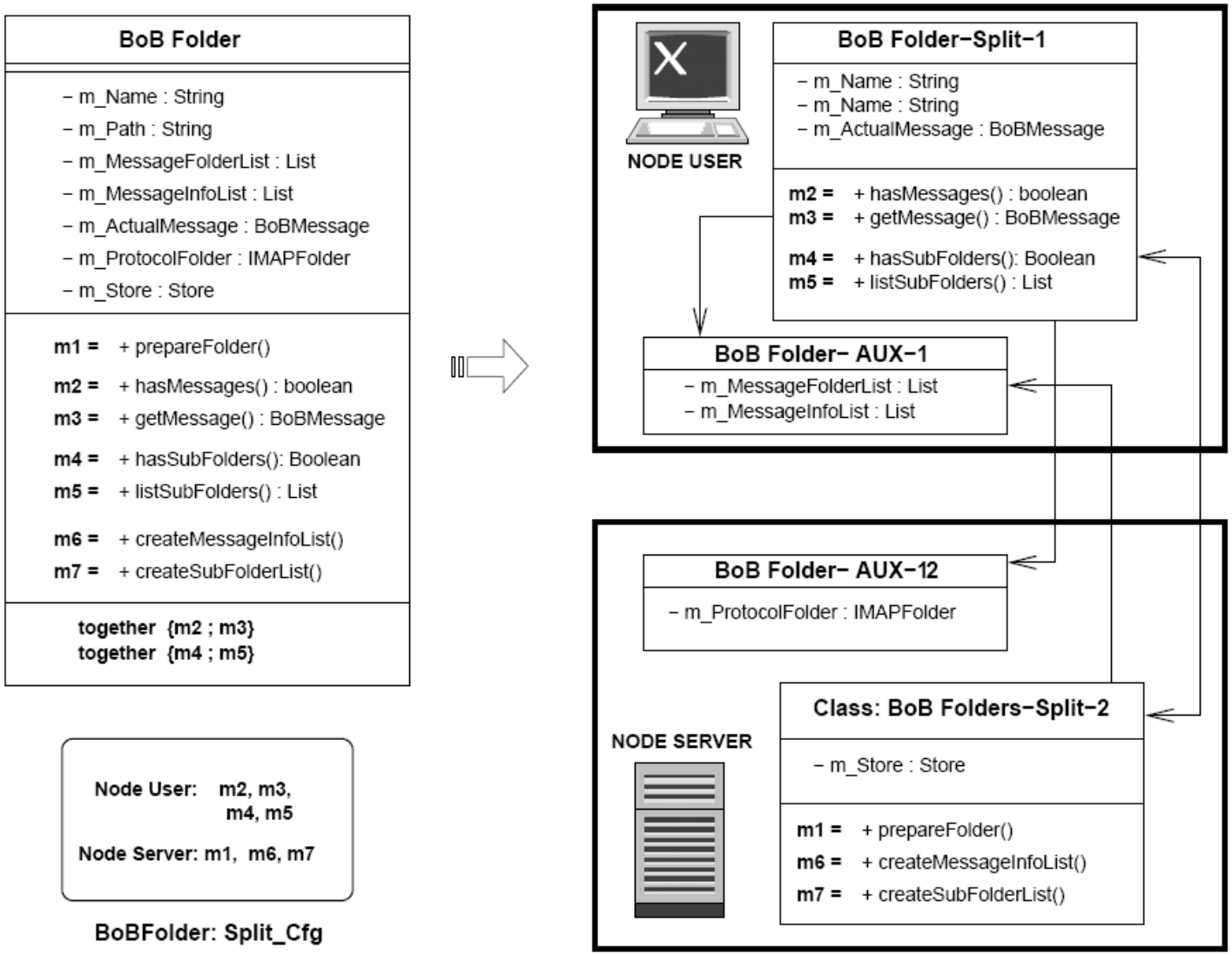
# Redeployment



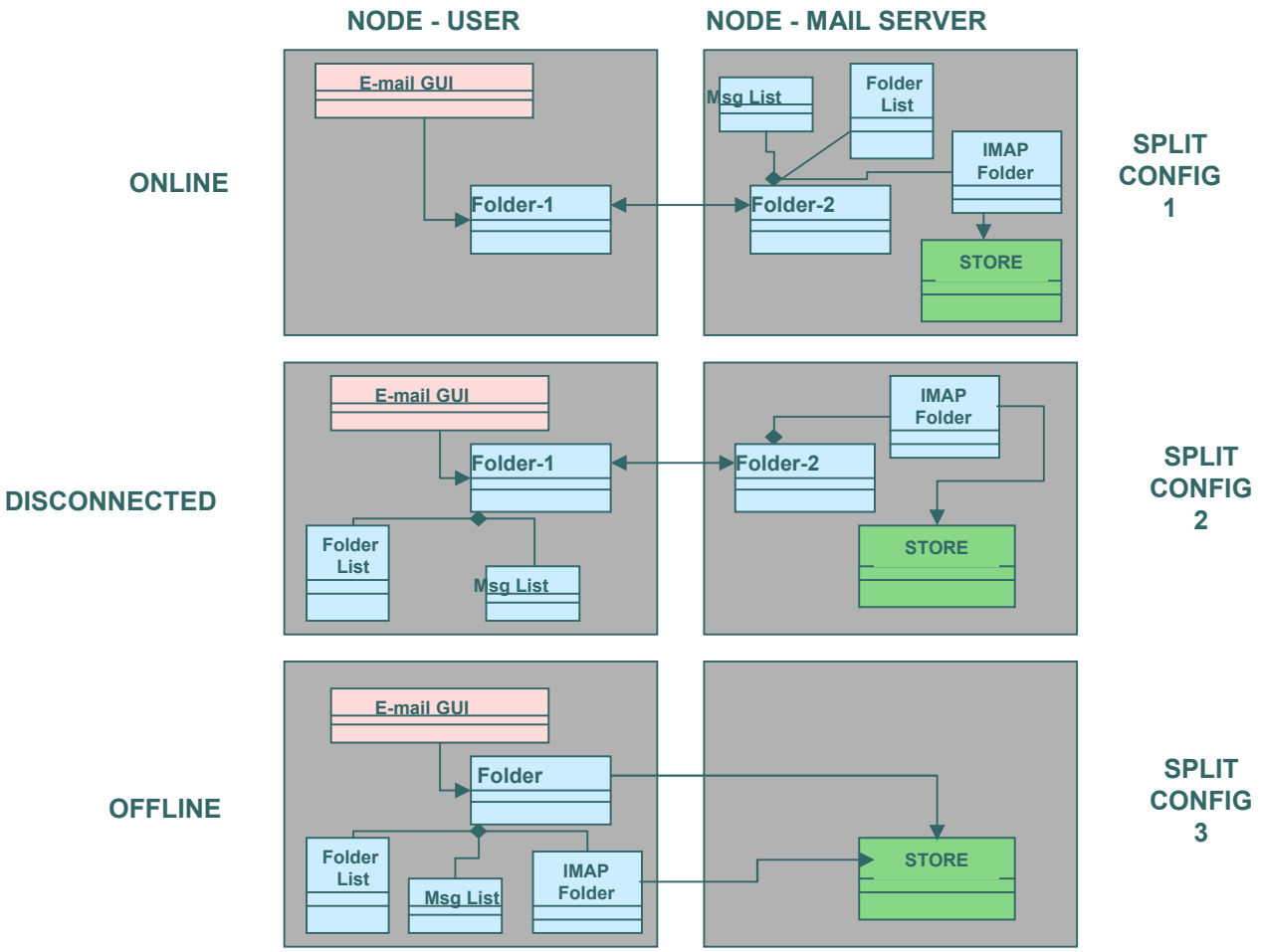
# BODA: Breakable Object Driven Architecture



# Folder BoB


BoB Folder
– m_Name : String
– m_Path : String
– m_MessageFolderList : List
– m_MessageInfoList : List
– m_ActualMessage : BoBMessage
– m_ProtocolFolder : IMAPFolder
– m_Store : Store
m1 = + prepareFolder()
m2 = + hasMessages() : boolean
m3 = + getMessage() : BoBMessage
m4 = + hasSubFolders(): Boolean
m5 = + listSubFolders() : List
m6 = + createMessageInfoList()
m7 = + createSubFolderList()
together {m2 ; m3}
together {m4 ; m5}
Node User: m2, m3, m4, m5
Node Server: m1, m6, m7
BoBFolder: Split_Cfg
NODE USER
BoB Folder–Split–1
– m_Name : String
– m_Name : String
– m_ActualMessage : BoBMessage
m2 = + hasMessages() : boolean
m3 = + getMessage() : BoBMessage
m4 = + hasSubFolders(): Boolean
m5 = + listSubFolders() : List
BoB Folder– AUX–1
– m_MessageFolderList : List
– m_MessageInfoList : List
BoB Folder– AUX–12
– m_ProtocolFolder : IMAPFolder
NODE SERVER
Class: BoB Folders–Split–2
– m_Store : Store
m1 = + prepareFolder()
m6 = + createMessageInfoList()
m7 = + createSubFolderList()

# Online-Disconnected-Offline

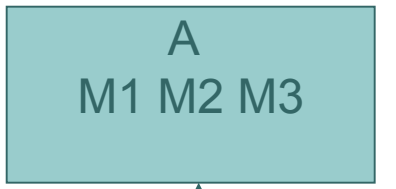


# Programming Model

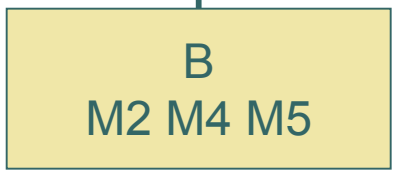
| Construct in JAVA | Status in $JAVA_{BoB}$ |
|---|---|
| **Class declarations** | |
| `public` | Allowed |
| `abstract` | Not Allowed |
| `final` | Allowed (default) |
| `class` *Name of Class* | Allowed |
| `extends` *Super* | Not Allowed |
| `implements` Interface | Allowed |
| **Field Declarations** | |
| `public` | Not Allowed |
| `private` | Allowed |
| `protected` | Not Allowed |
| `package` | Not Allowed |
| `static` | Allowed |
| `final` | Allowed |
| `transient` | Not Allowed |
| `volatile` | Allowed |

| Construct in JAVA | Status in |
|---|---|
| **Method Declarations** | |
| `public` | Allowed |
| `private` | Allowed |
| `protected` | Not Allowed |
| `package` | Not Allowed |
| `static` | Allowed |
| `abstract` | Not Allowed |
| `final` | Allowed |
| `native` | Not Allowed |
| `synchronized` | Allowed |
| **Miscellaneous** | |
| Constructors | Allowed |
| Exceptions | Allowed |
| Threads | Not Allowed |
| Nested Class / Inner Class | Not Allowed |

# Inheritance and BoBs



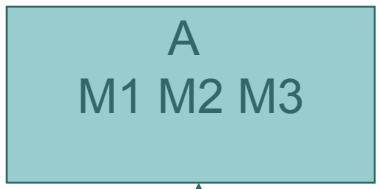

BOB

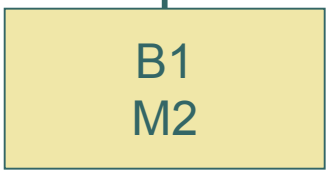


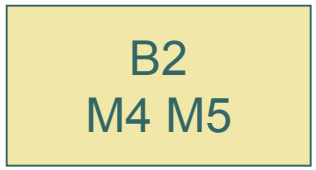

Split Config-1

S1 = M1, M2, M3

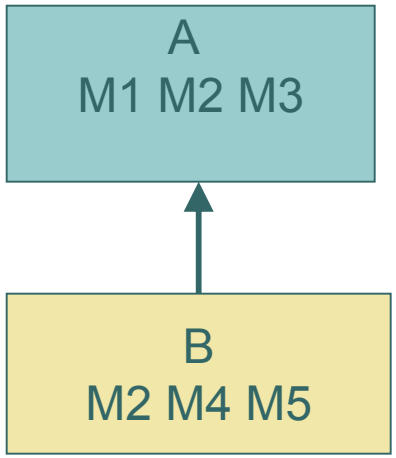
S2 = M4, M5



# Inheritance and BoBs

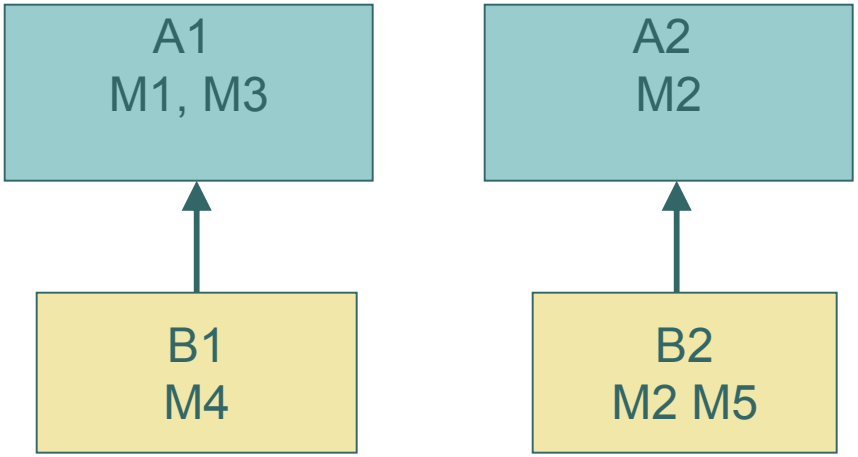



BOB

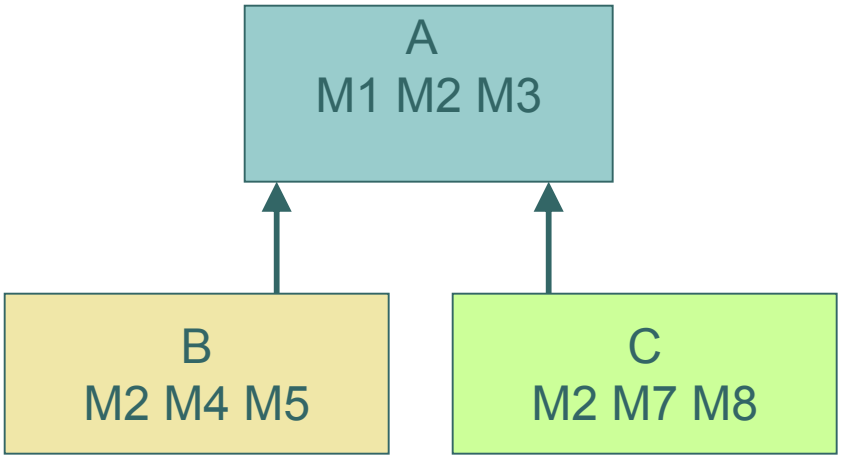
Split Config-2

S1 = M1, M3, M4

S2 = M2, M5



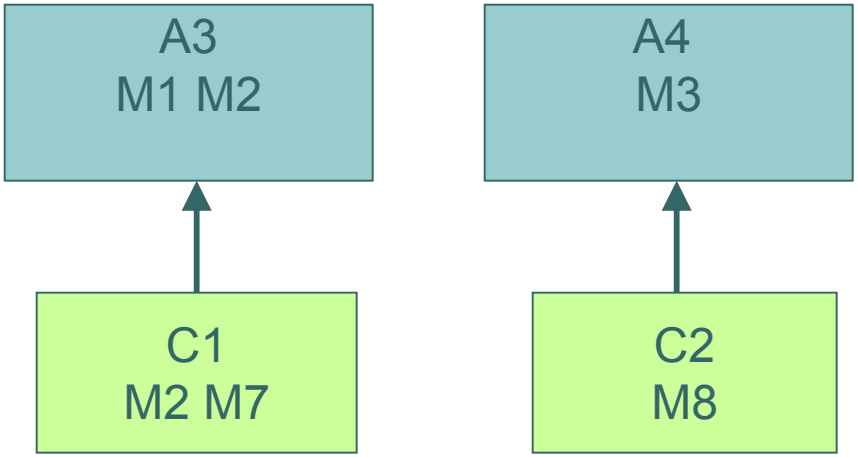
# Inheritance and BoBs




Split Config-3

S3 = M1, M2, M7

S4 = M3, M8




A a = new A;

A b = new B; ?

A c = new C; ?

Each child can split A in a unique way

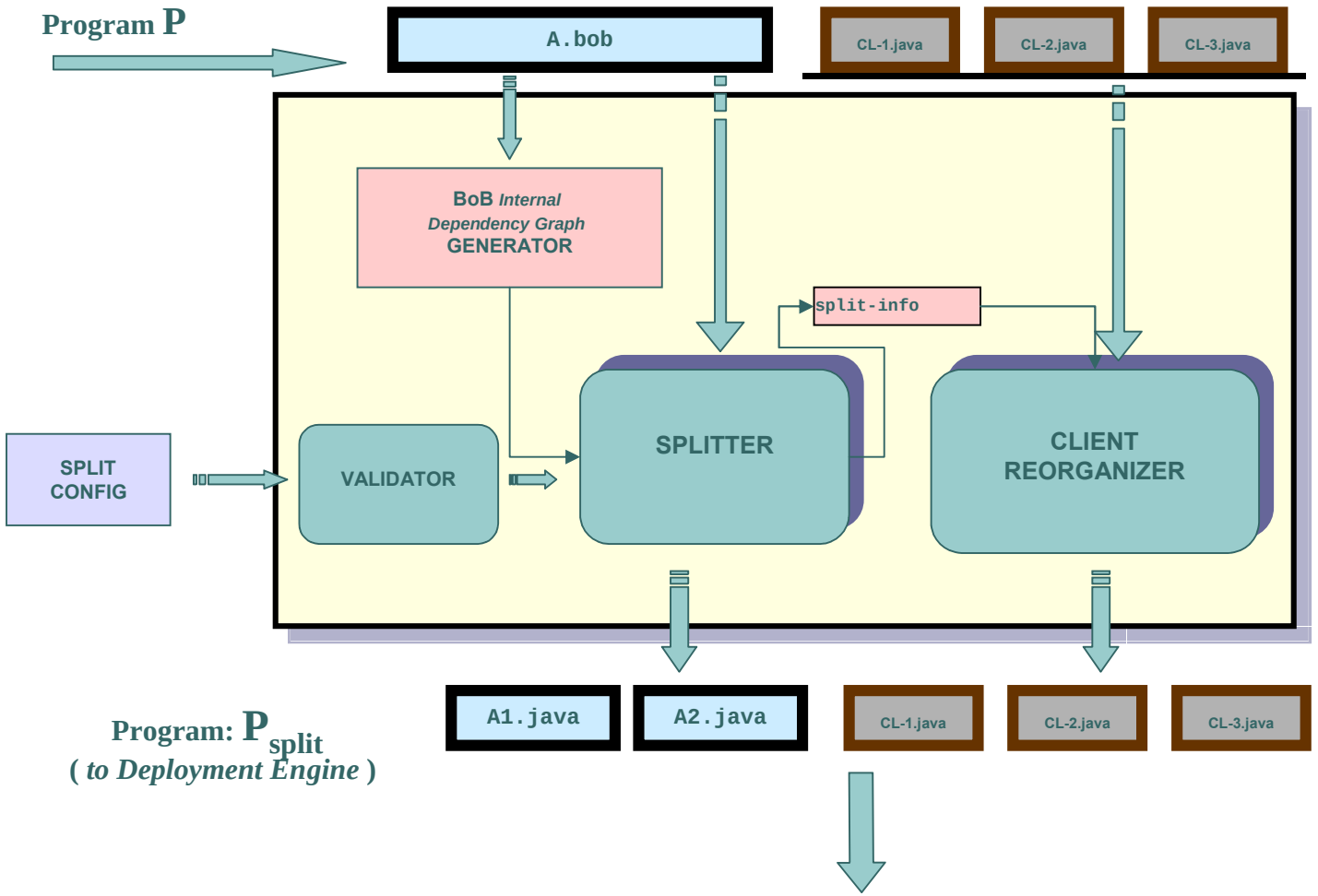
# Retaining old type?

- Case 1, A remains along with A1, A2
- Case 2, only A1 and A2 remain
- We prefer case 2, and
  - allow only interface inheritance with the condition that
    - all the methods of an interface are designated **`together`**
- **Recommended**
  - Aggregation or delegation as the principal composition mechanisms for BoBs.
    - Neater design, reduces complexity

# Class-level/object level

- We do class-level partitioning
- If we allow object-level partitioning
  - i.e. we allow a BoB to be split in more than one way
- For assignment, need to know the <type> of object on RHS and then convert it to the type being assigned (LHS)
  - A ax = new A();
  - A ay = new A();
  - ax split ax1 = m1, m2, split ax2 = m3
  - ay split ay1 = m2, split ay2 = m1, m3;
  - ax = ay ?
- Such a support is not available in the present languages
- **Class level BoB partitioning - sufficient for meaningful applications**

# Splitting Engine



# Split Config

- Format

```
Number of BoBs = n;
BoB 1
BoB Name {
No of splits = k;
Split 1 = (';' separated list of methods specified as
        MethodName (list ArgumentTypes) )
. . .
Split k = . . .
}

BoB n
BoB Name {
. . .
}
```

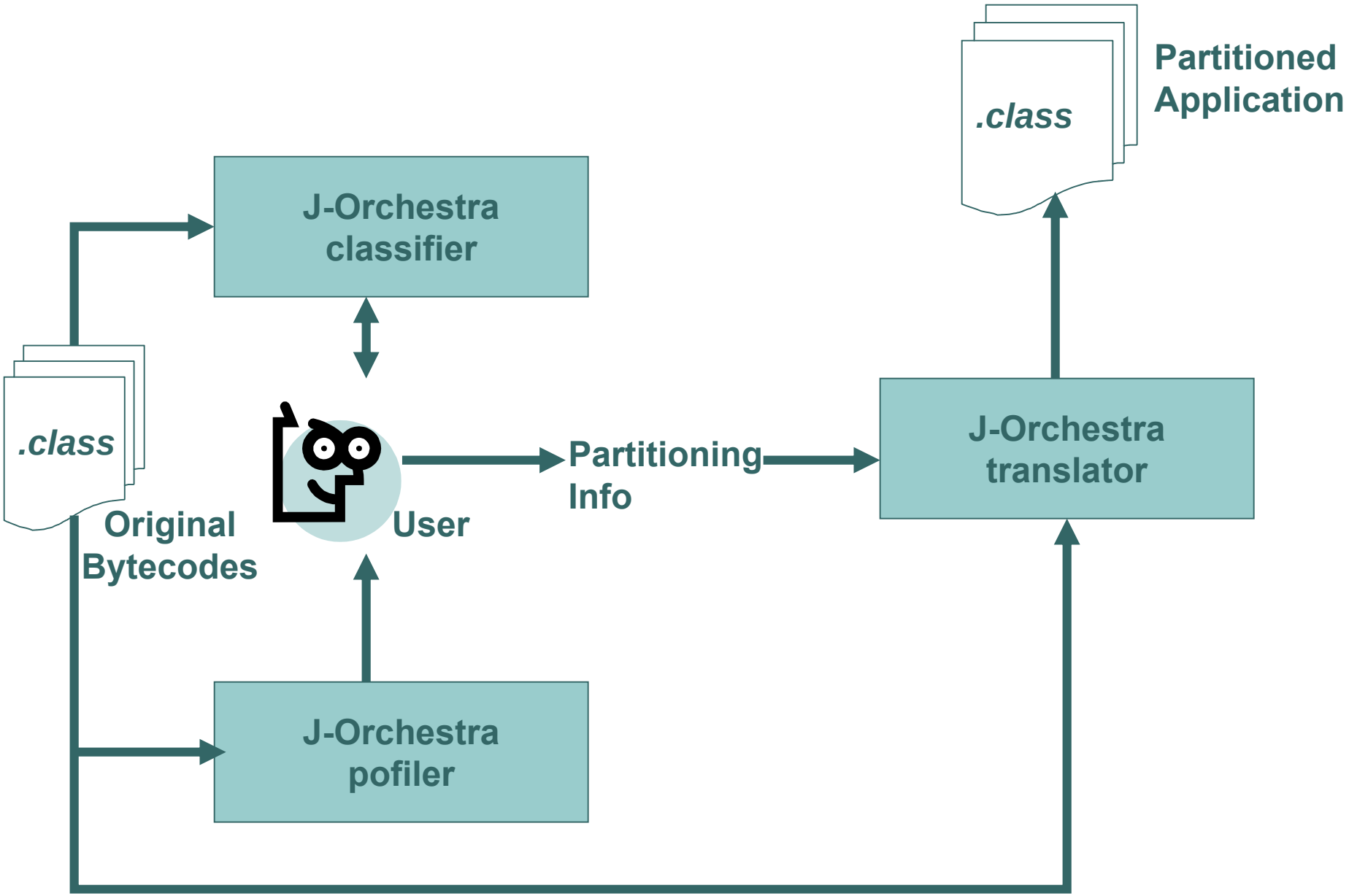
- Properties
  - Only public methods are specified.
  - Every public method in each BoB has to be specified as part of some split.
  - A method cannot belong to more than one split.
  - Clubbed methods (identified by the **`together`** construct) cannot be split.

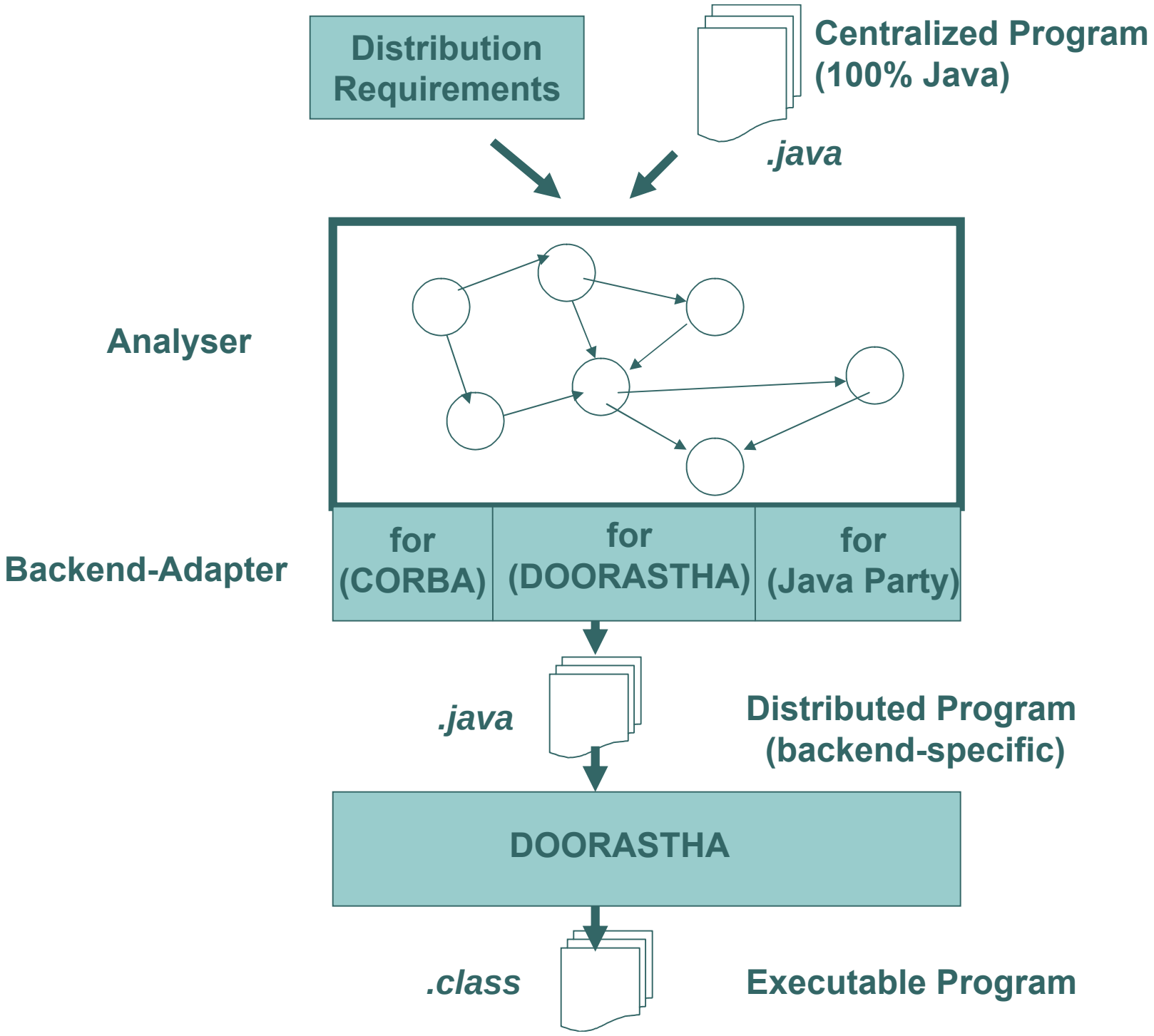
# Splitting Engine Details

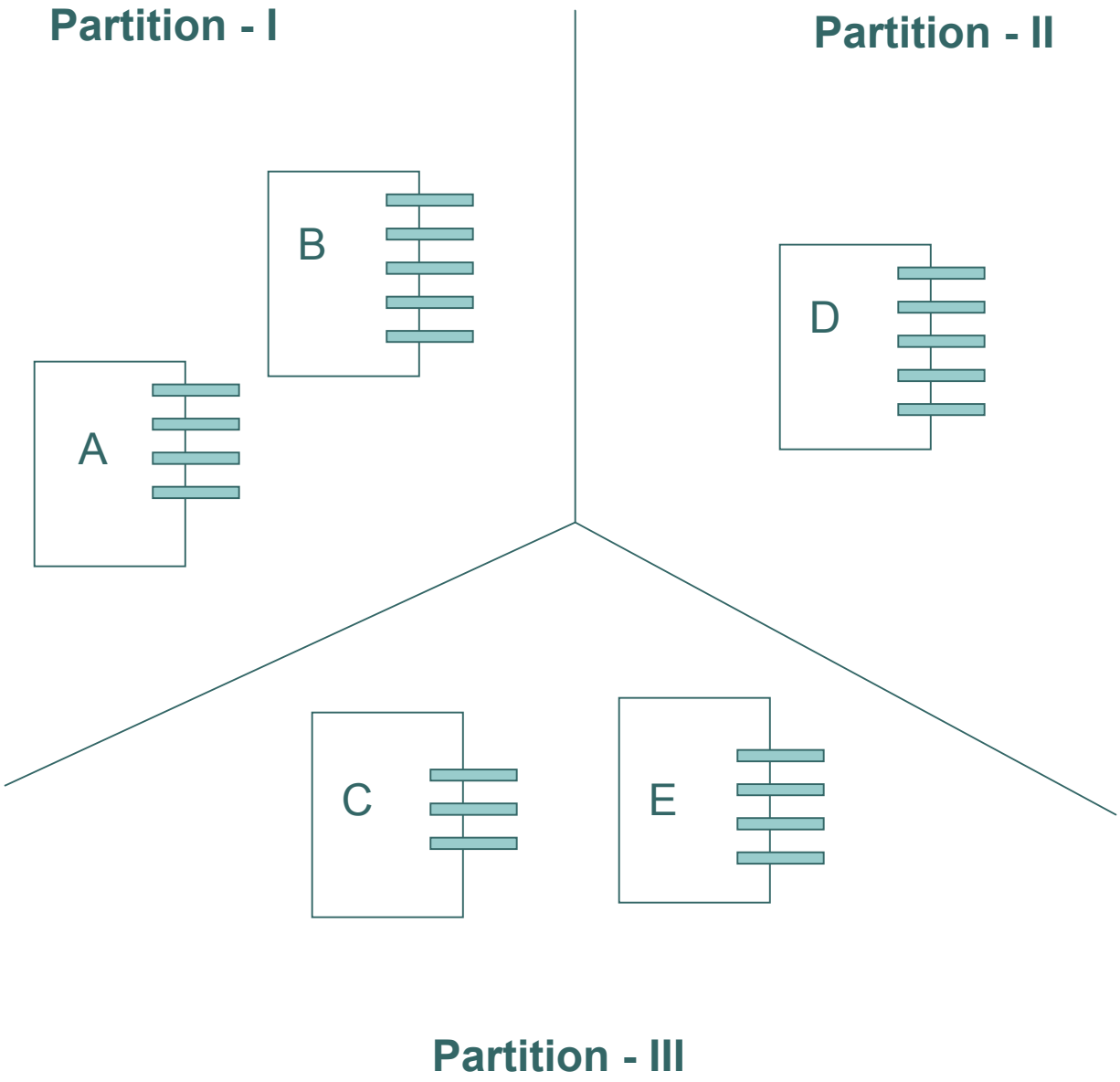
- Algorithms
- Program Equivalence
  - split and non-split program versions
- Equivalency Proofs
- Details

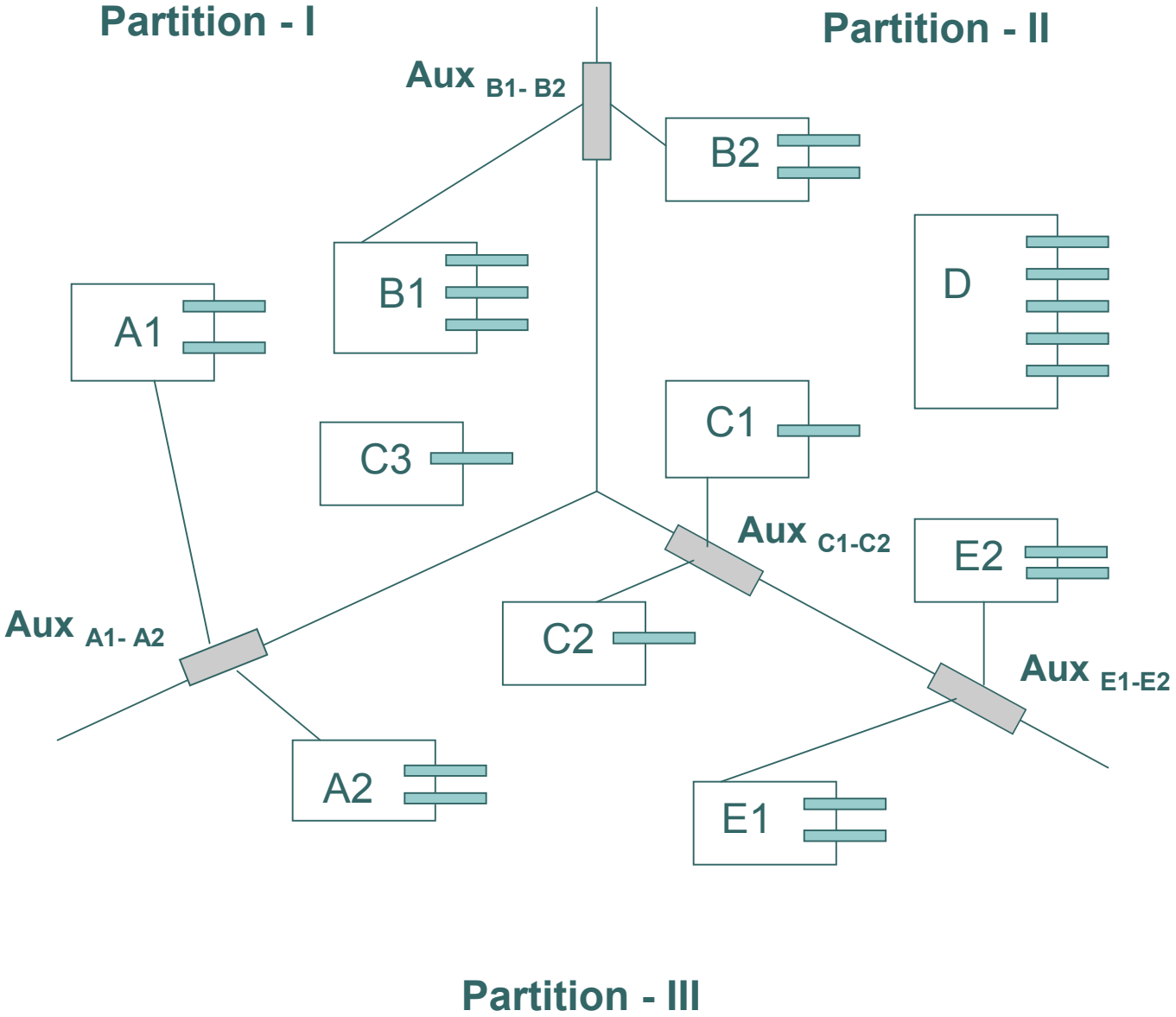
# Deployment

- Presently uses mechanisms by
  - J-orchestra ,
  - Pangaea  for BoB deployments
- Both use RMI as the underlying distribution mechanism


.class
Partitioned Application
J-Orchestra classifier
.class
Original Bytecodes
User
Partitioning Info
J-Orchestra translator
J-Orchestra pofiler

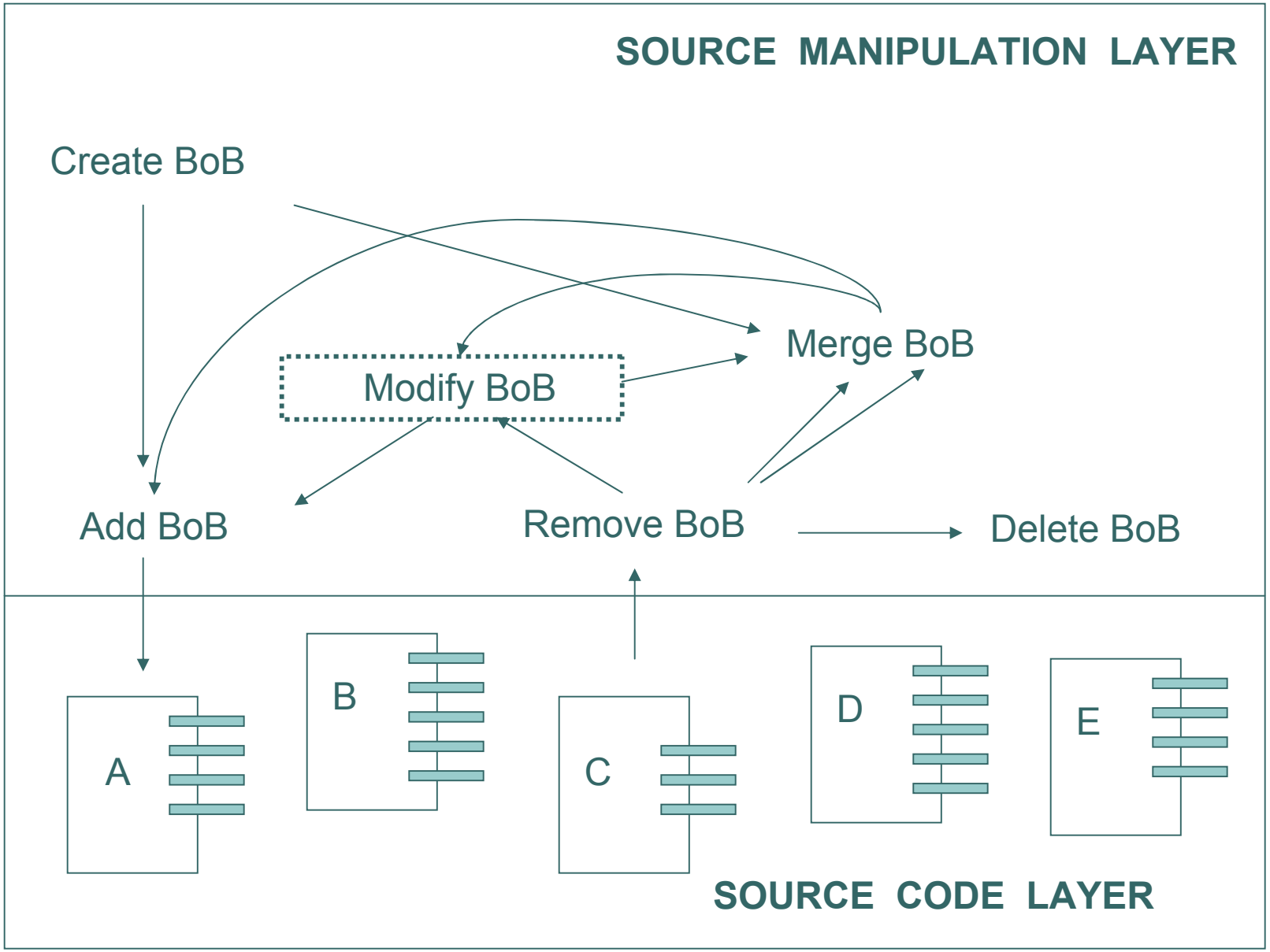

Distribution Requirements
Centralized Program (100% Java)
.java
Analyser
Backend-Adapter
for (CORBA)
for (DOORASTHA)
for (Java Party)
.java
Distributed Program (backend-specific)
DOORASTHA
.class
Executable Program

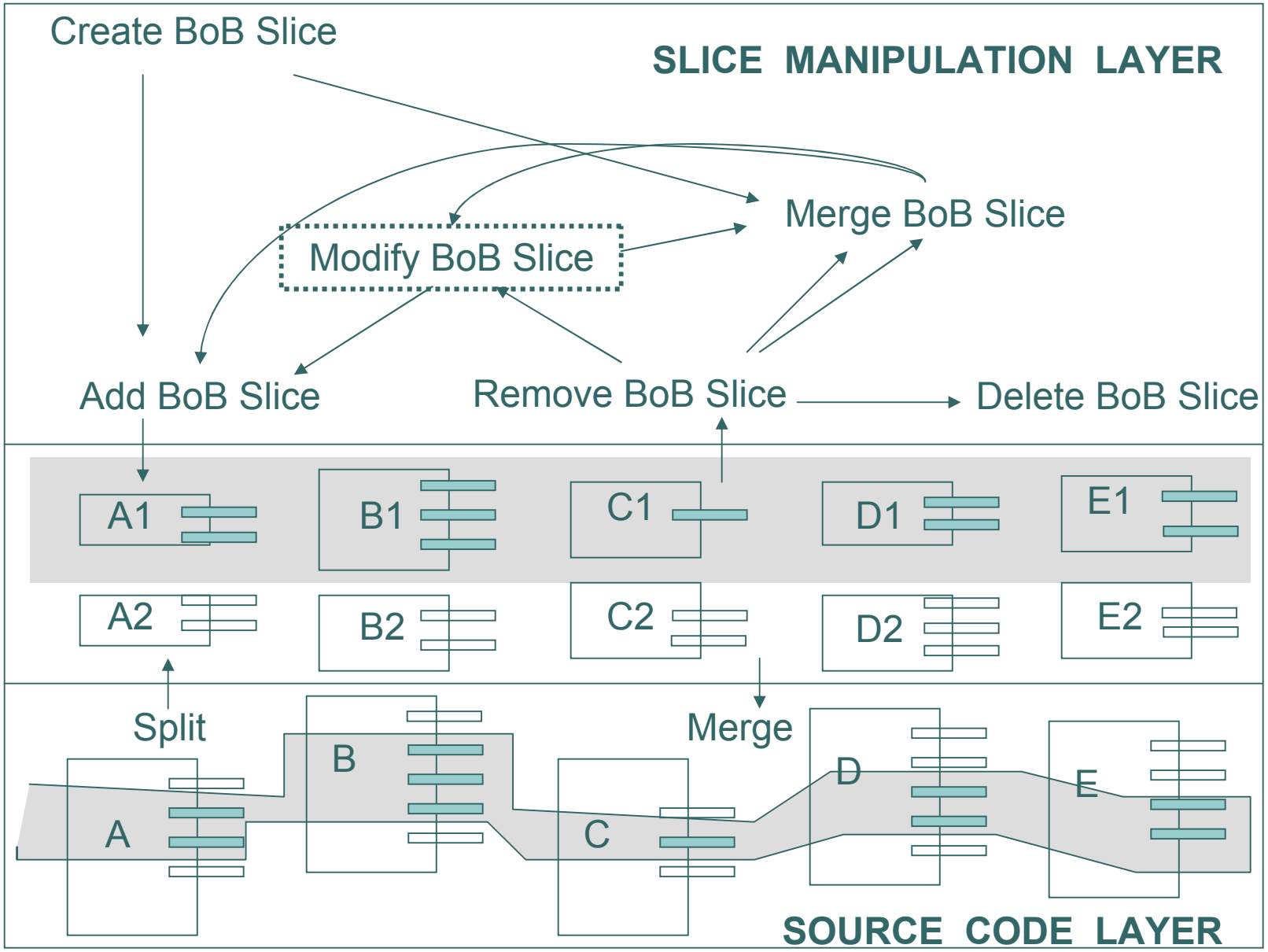

Partition - I
Partition - II
B
D
A
C
E
Partition - III

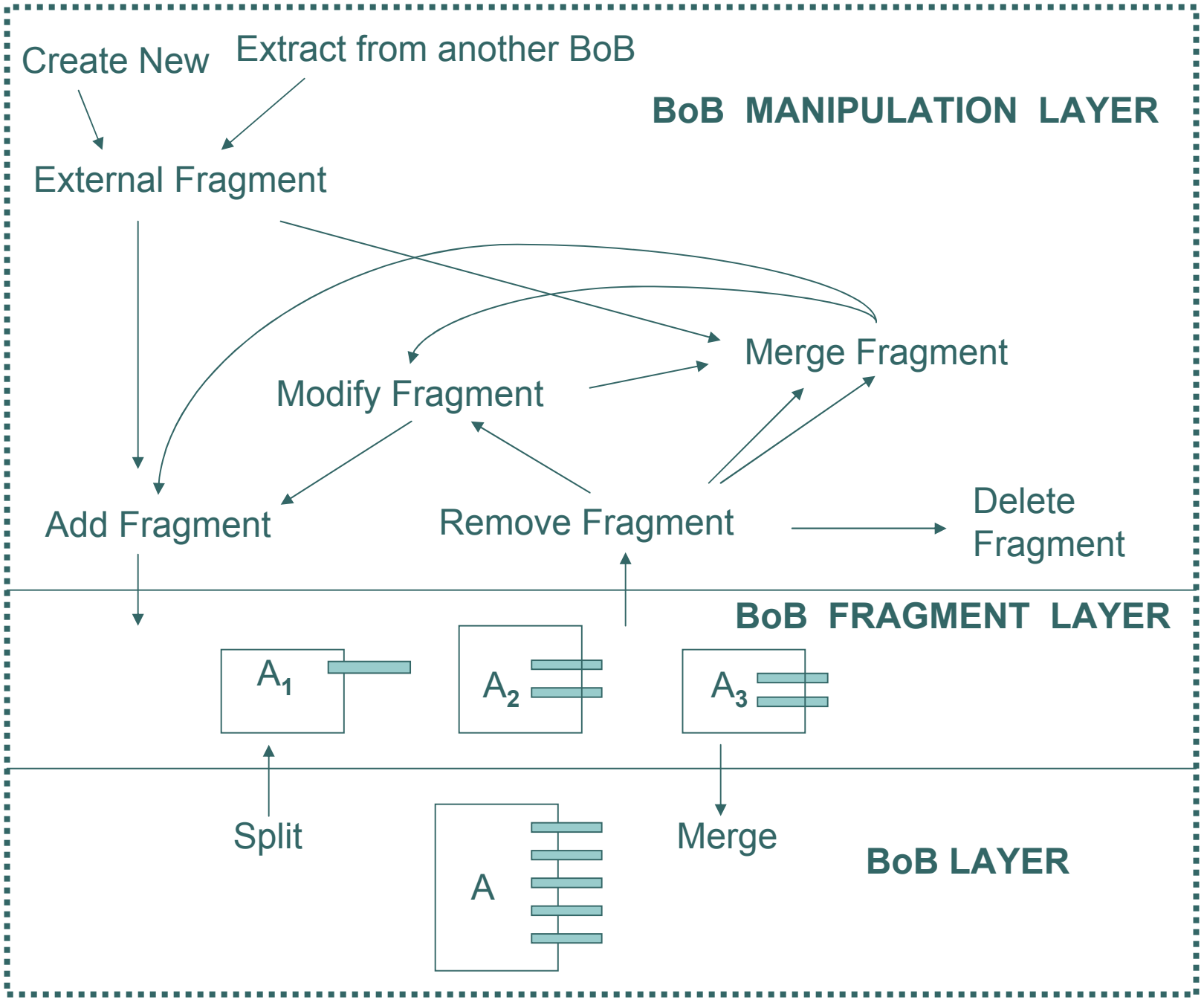

Partition - I
Partition - II
Aux B1- B2
B2
B1
A1
D
C3
C1
Aux C1-C2
E2
Aux A1- A2
C2
Aux E1-E2
A2
E1
Partition - III

# Contents

- Problem and Motivations
- BoB basics
- BoBs for application partitioning
- **BoBs as elements of reuse**
- Related work comparisons
- Discussion and conclusions


SOURCE MANIPULATION LAYER
Create BoB
Merge BoB
Modify BoB
Add BoB
Remove BoB
Delete BoB
A
B
C
D
E
SOURCE CODE LAYER


Create BoB Slice
SLICE MANIPULATION LAYER
Merge BoB Slice
Modify BoB Slice
Add BoB Slice
Remove BoB Slice
Delete BoB Slice
A1
B1
C1
D1
E1
A2
B2
C2
D2
E2
Split
Merge
A
B
C
D
E
SOURCE CODE LAYER

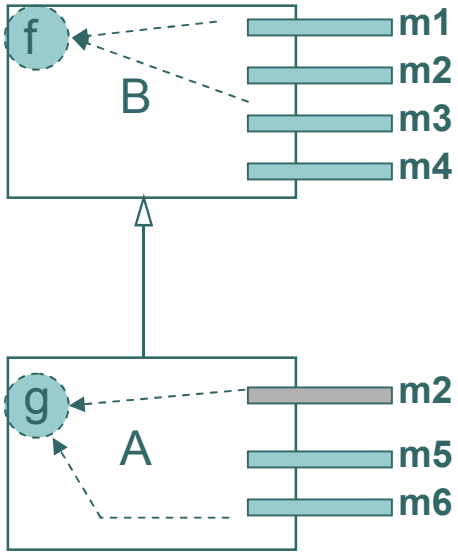

Create New
Extract from another BoB
BoB MANIPULATION LAYER
External Fragment
Merge Fragment
Modify Fragment
Add Fragment
Remove Fragment
Delete Fragment
BoB FRAGMENT LAYER
A1
A2
A3
Split
Merge
A
BoB LAYER

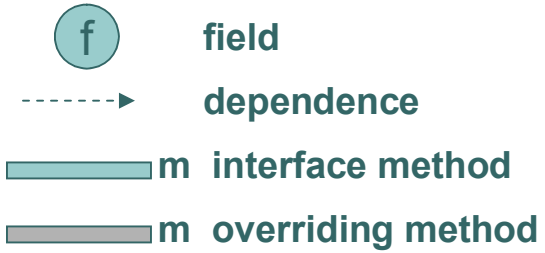
# Problems with inheritance and composibility

- Decomposition
  - Duplicated Features
  - Inappropriate hierarchies
  - Duplicated wrappers
- Composition
  - Conflicting Features
  - Fragile Hierarchies

# BoB Operators

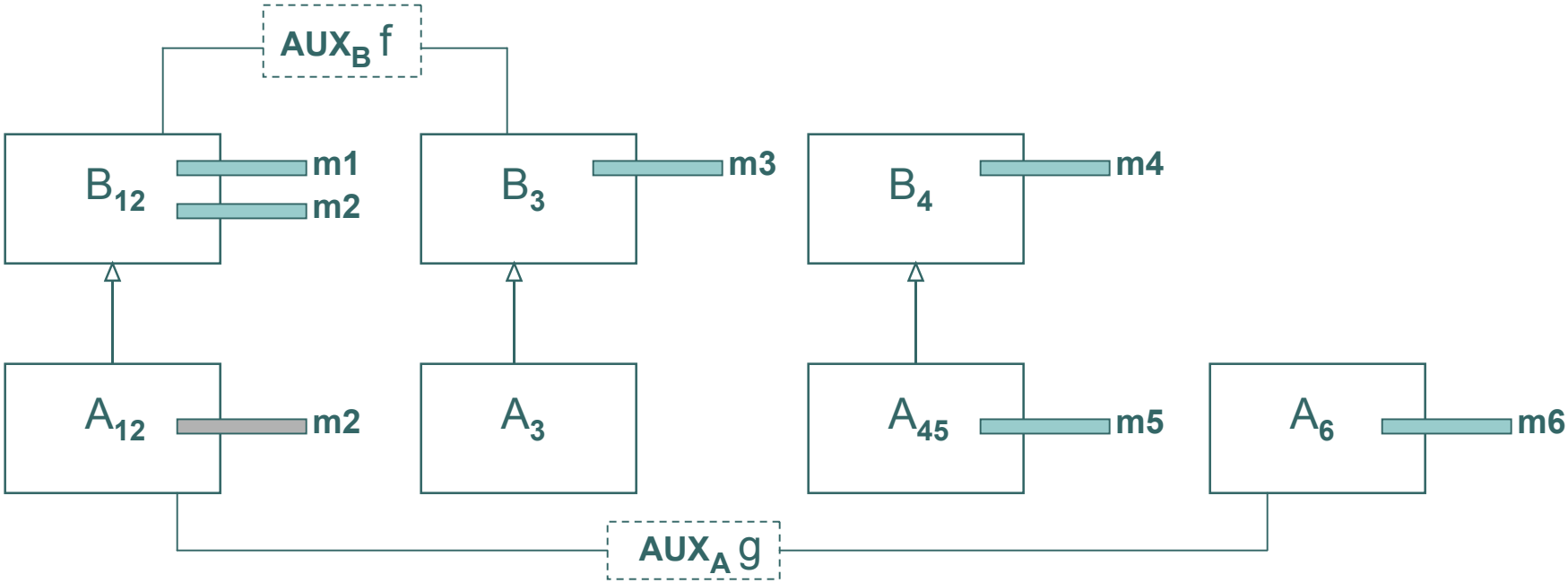
- Split, Merge
- Extract, Remove
- Addition, Subtraction
  - Overwriting
  - Hierarchical
- Replace

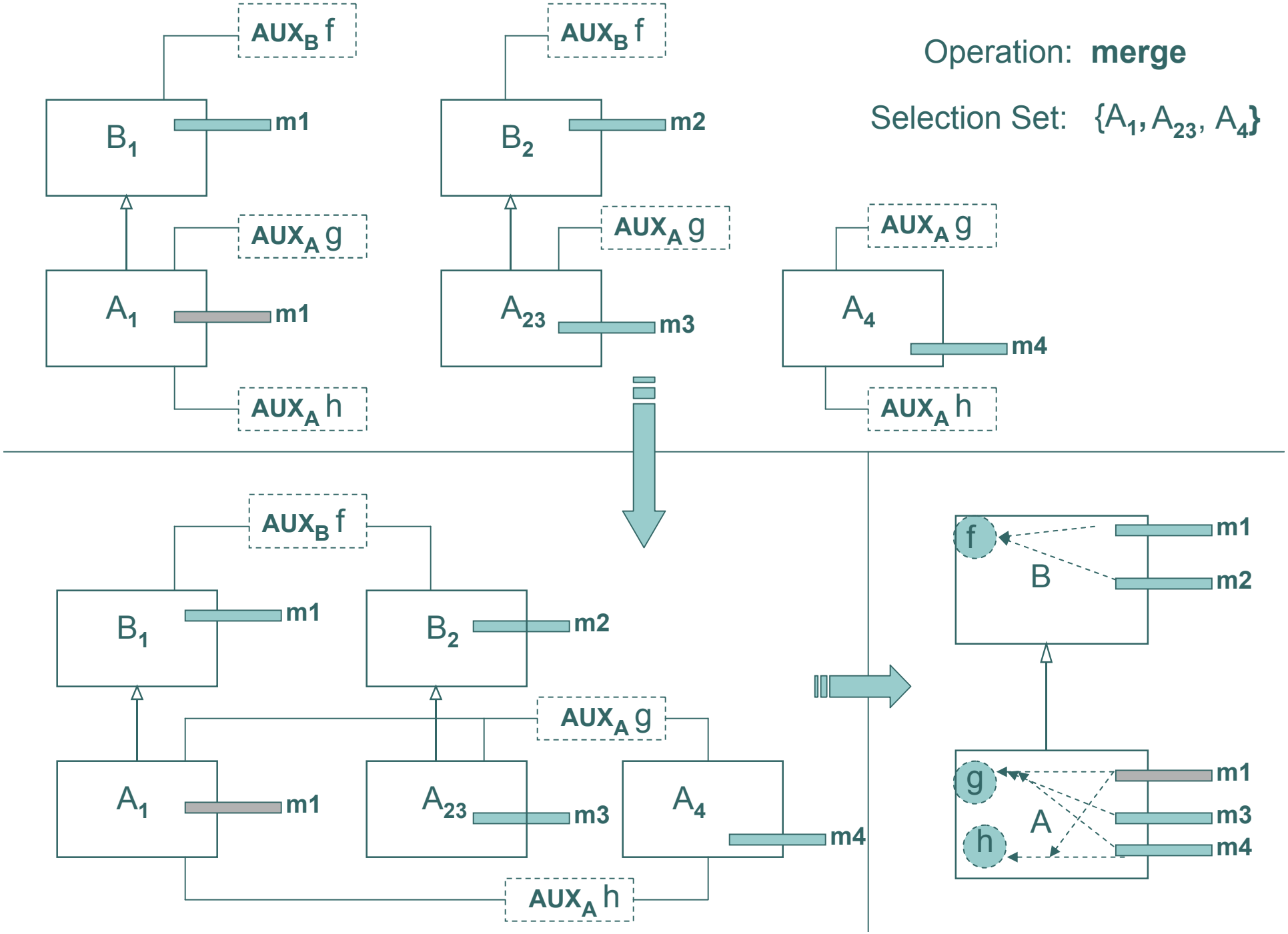


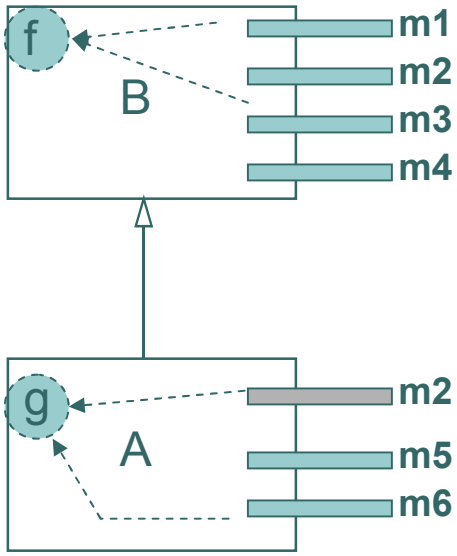




Operation: **split**

Selection Set: A, {{m1,m2}, {m3}, {m4,m5},{m6}}



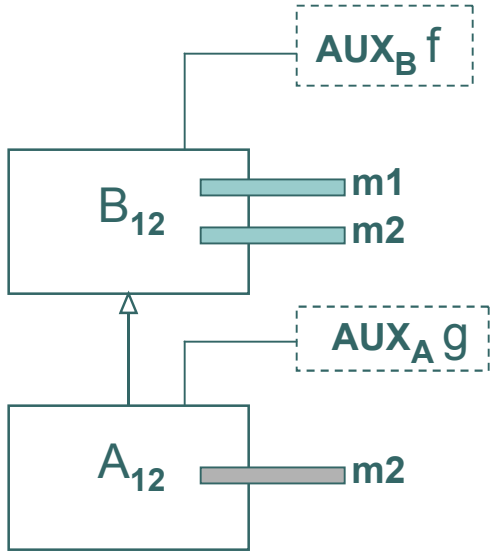

Operation: merge
Selection Set: {A1, A23, A4}
AUXB f
B1
m1
AUXA g
A1
m1
AUXA h
AUXB f
B2
m2
AUXA g
A23
m3
AUXA g
A4
m4
AUXA h
AUXB f
B1
m1
B2
m2
AUXA g
A1
m1
A23
m3
A4
m4
AUXA h
f
B
m1
m2
g
h
A
m1
m3
m4



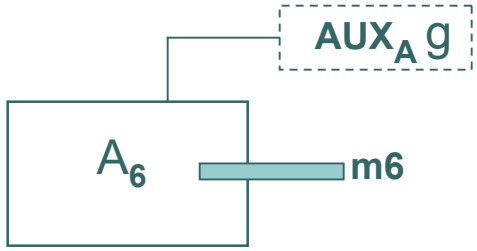

Operation: **extract**

Selection Sets:

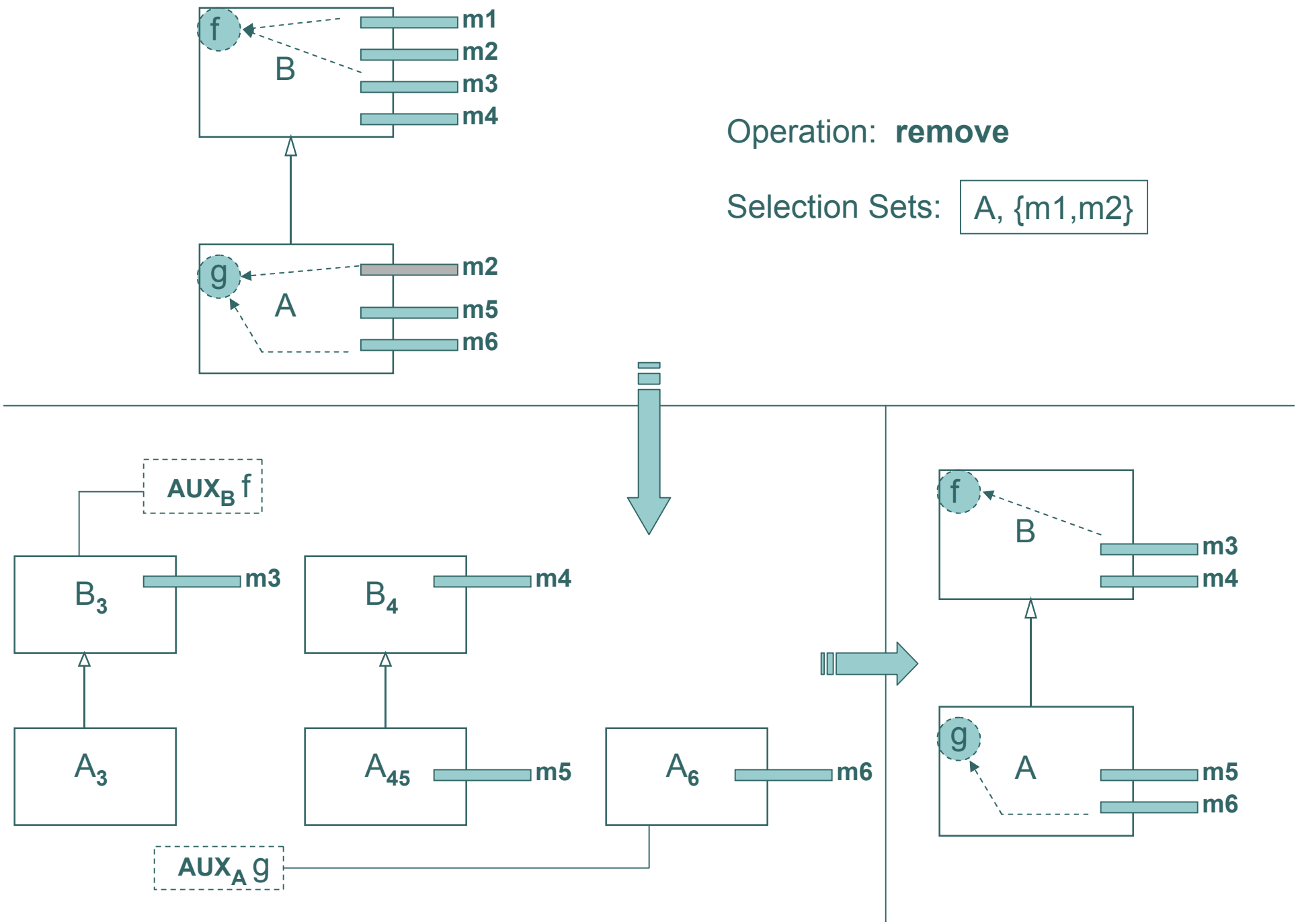
1. A, {m1,m2}
2. A, {m6}




Case (1)

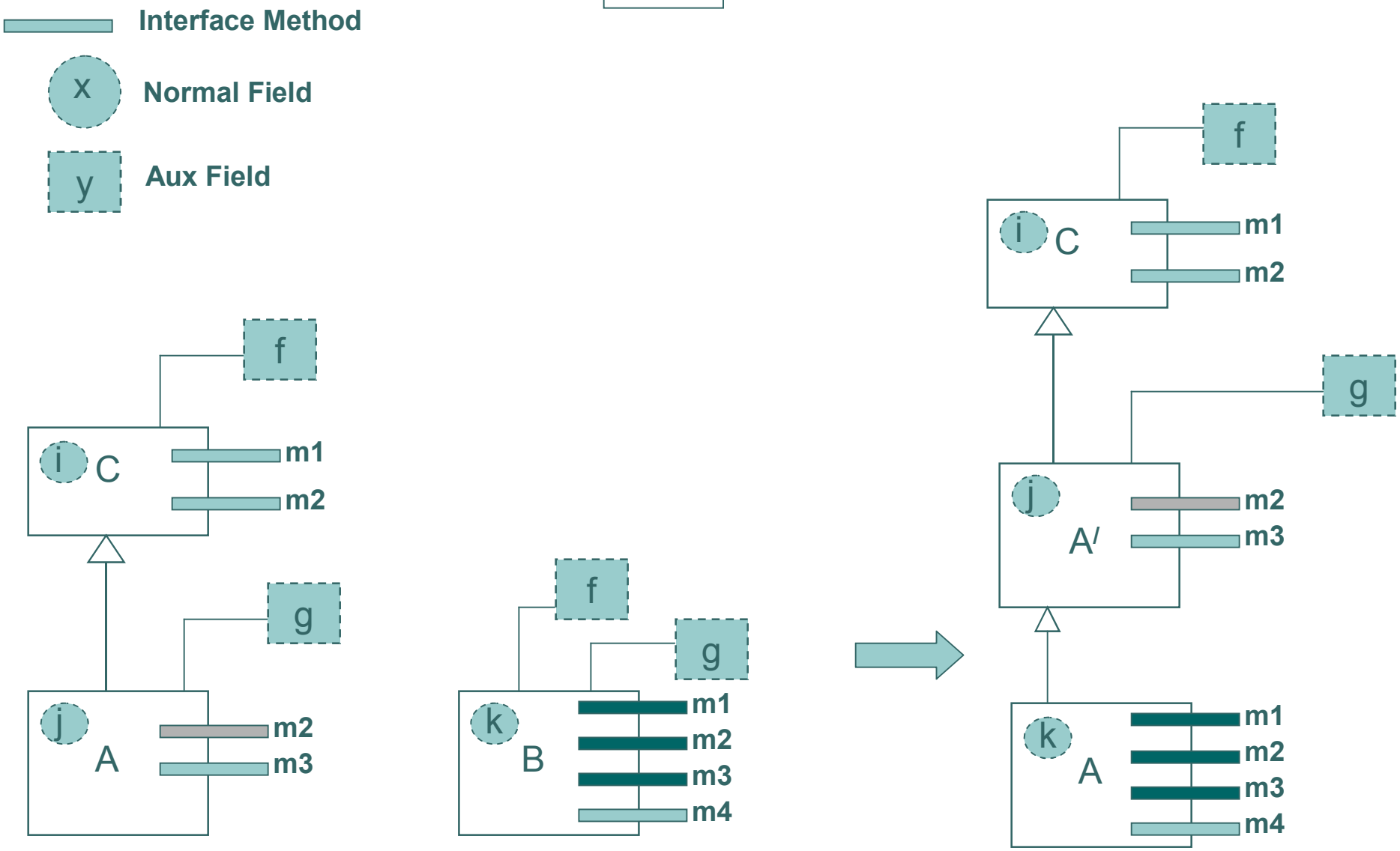



Case (2)


f
B
m1
m2
m3
m4
g
A
m2
m5
m6
Operation: remove
Selection Sets: A, {m1,m2}
AUX_B f
B_3
m3
B_4
m4
A_3
A_45
m5
A_6
m6
AUX_A g
f
B
m3
m4
g
A
m5
m6

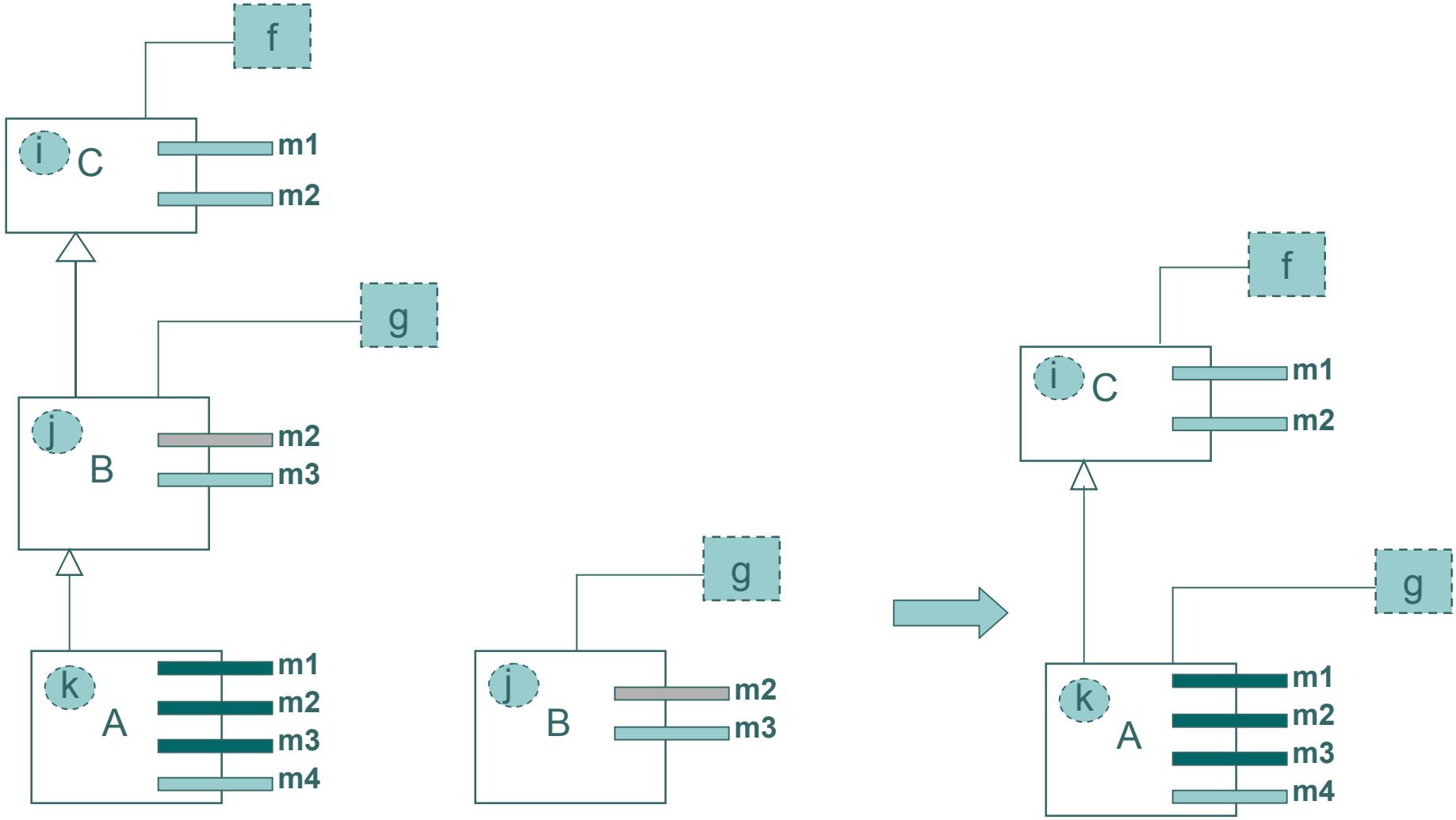
# Operation: **fragment addition –** *hierarchical*

A + B



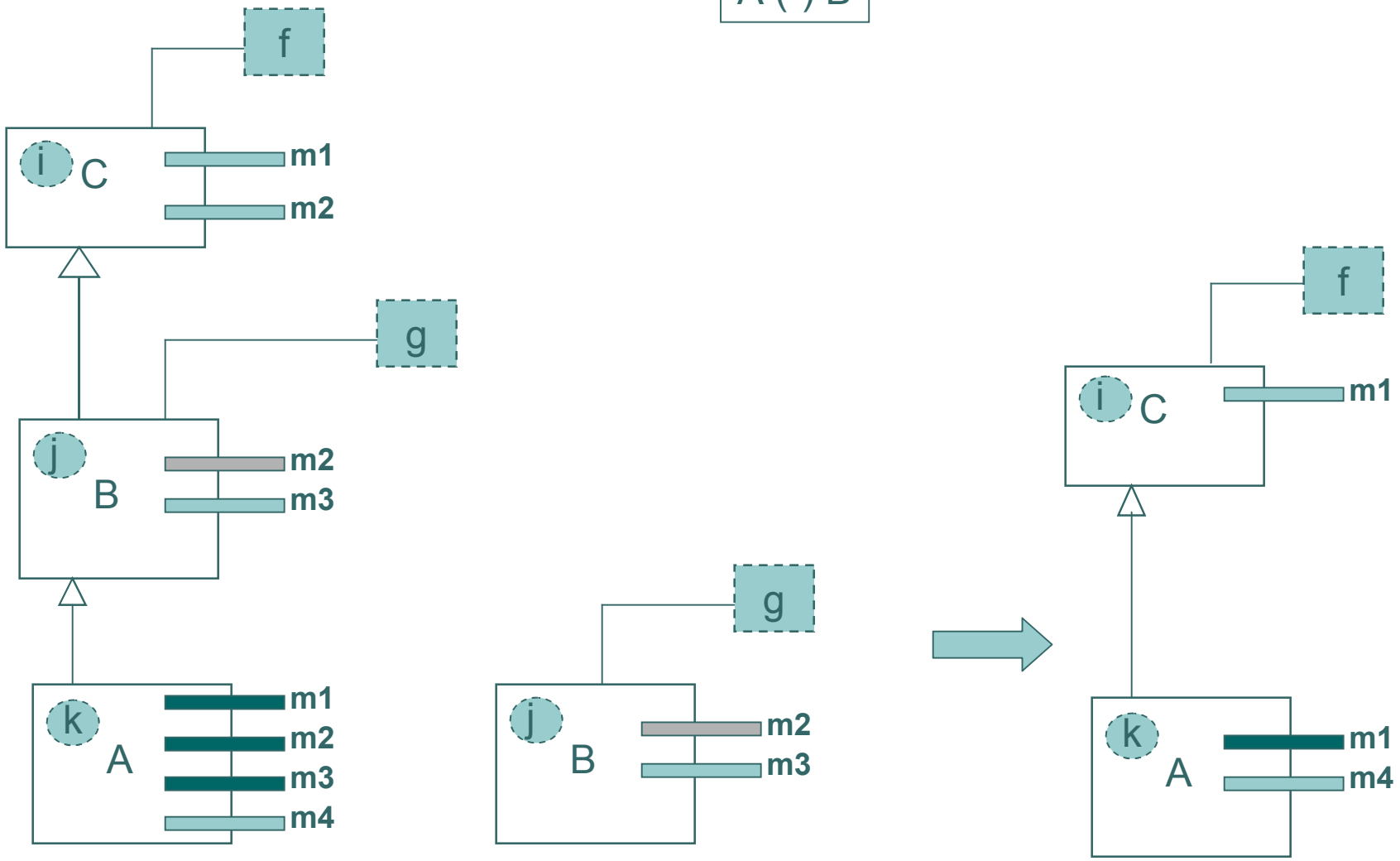
Operation: **fragment subtraction -** *hierarchical*

A - B



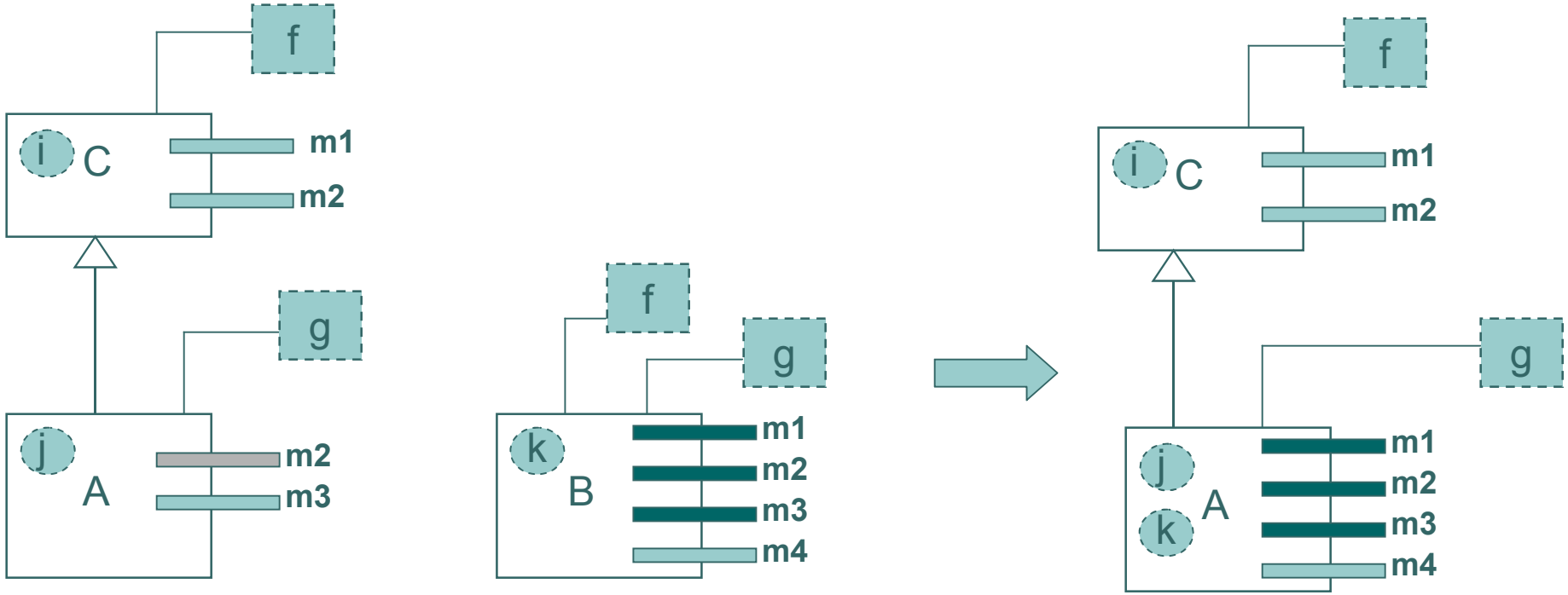
Operation: **fragment subtraction -** *overwriting*

A (-) B



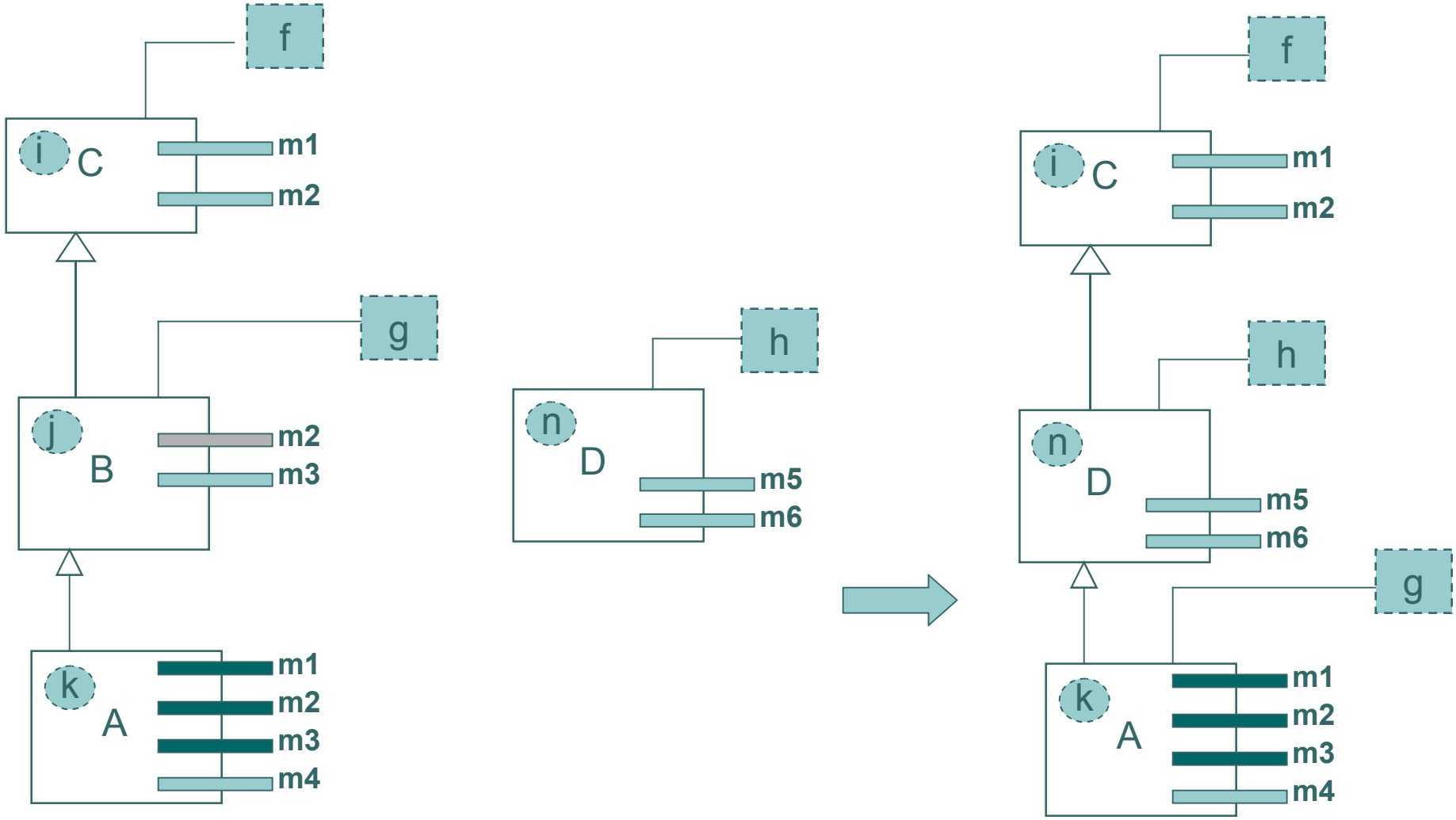
Operation: **fragment addition –** *overwriting*

A (+) B



Operation: **fragment replacement**

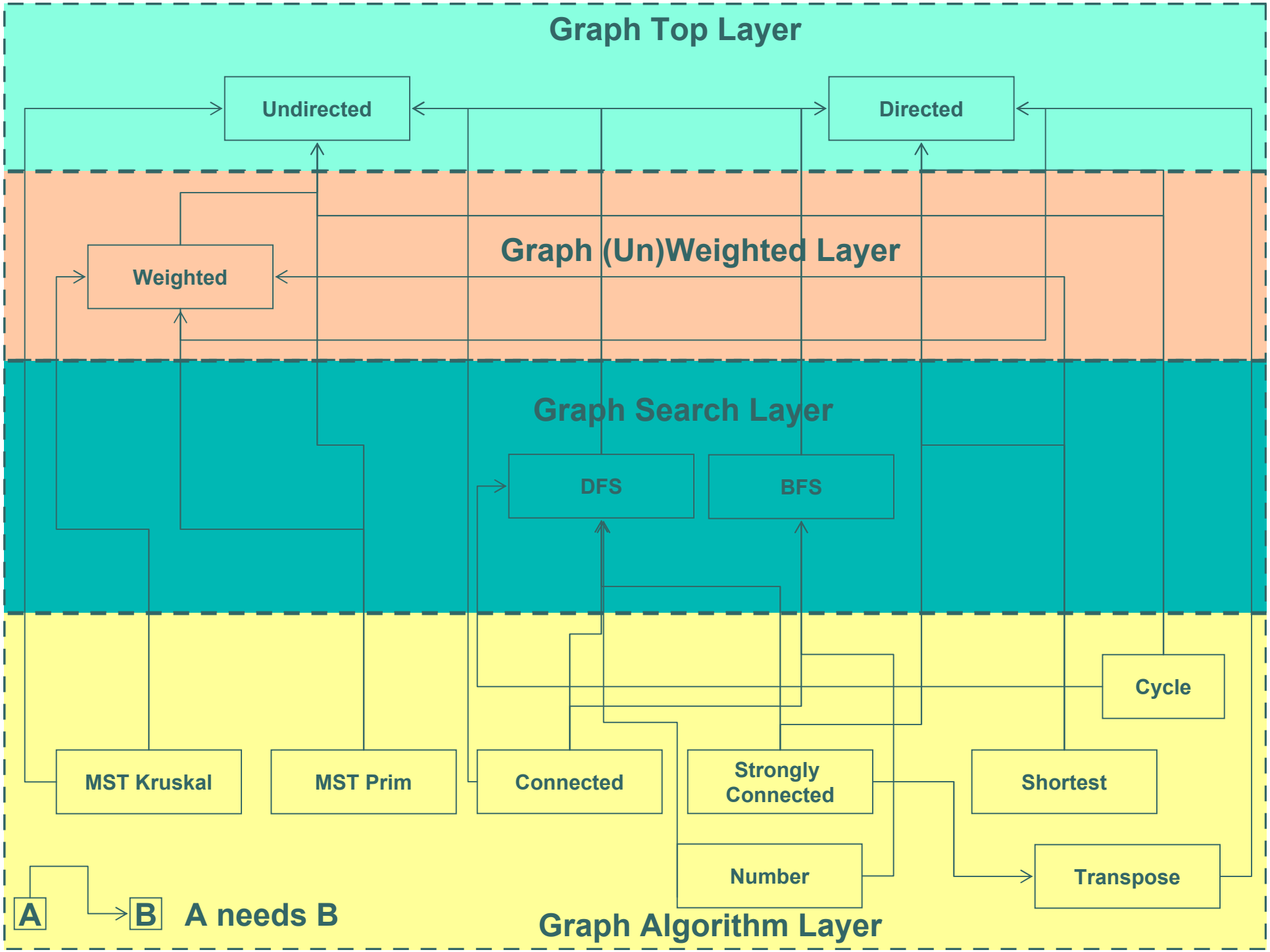
B # D

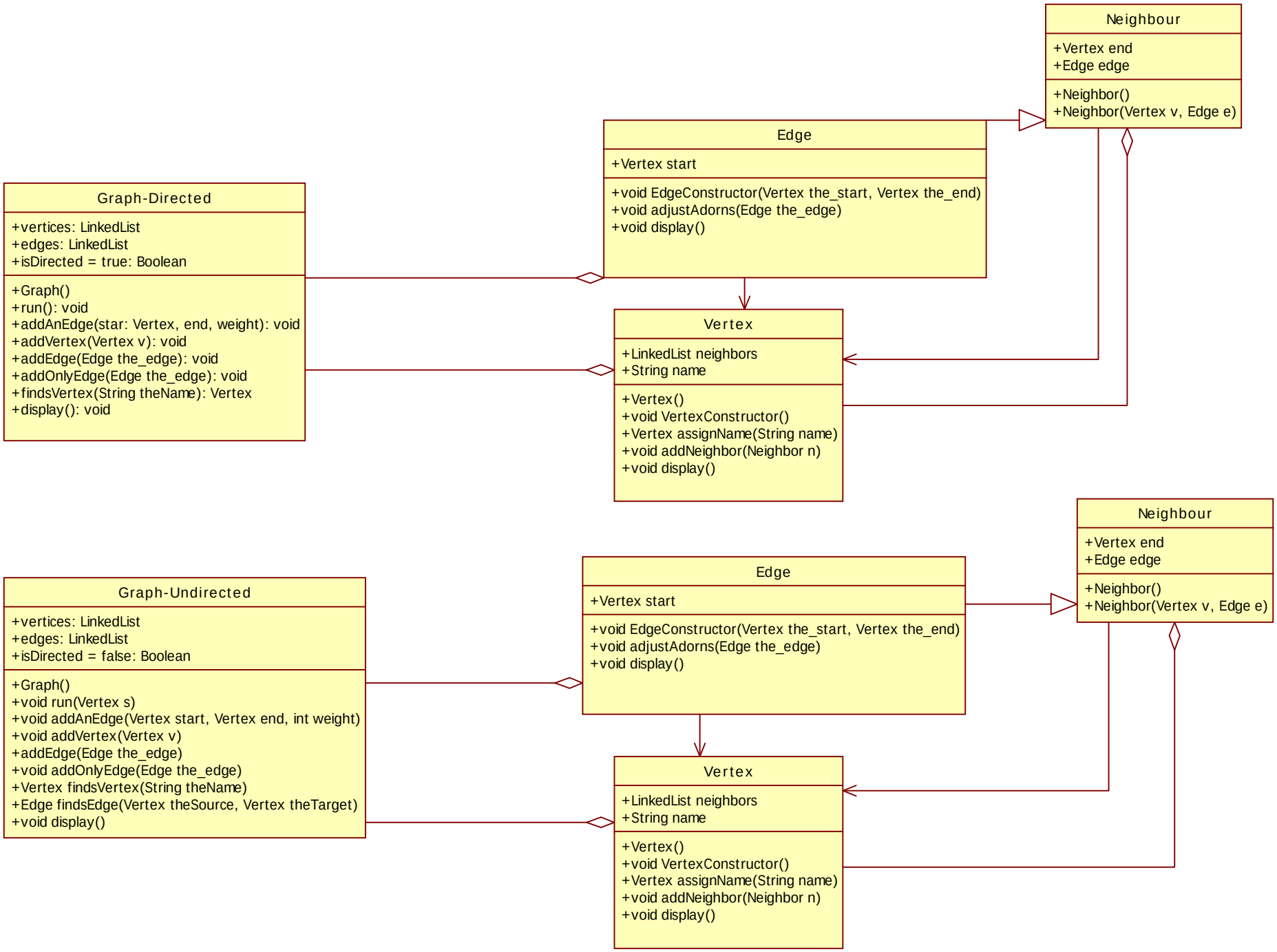


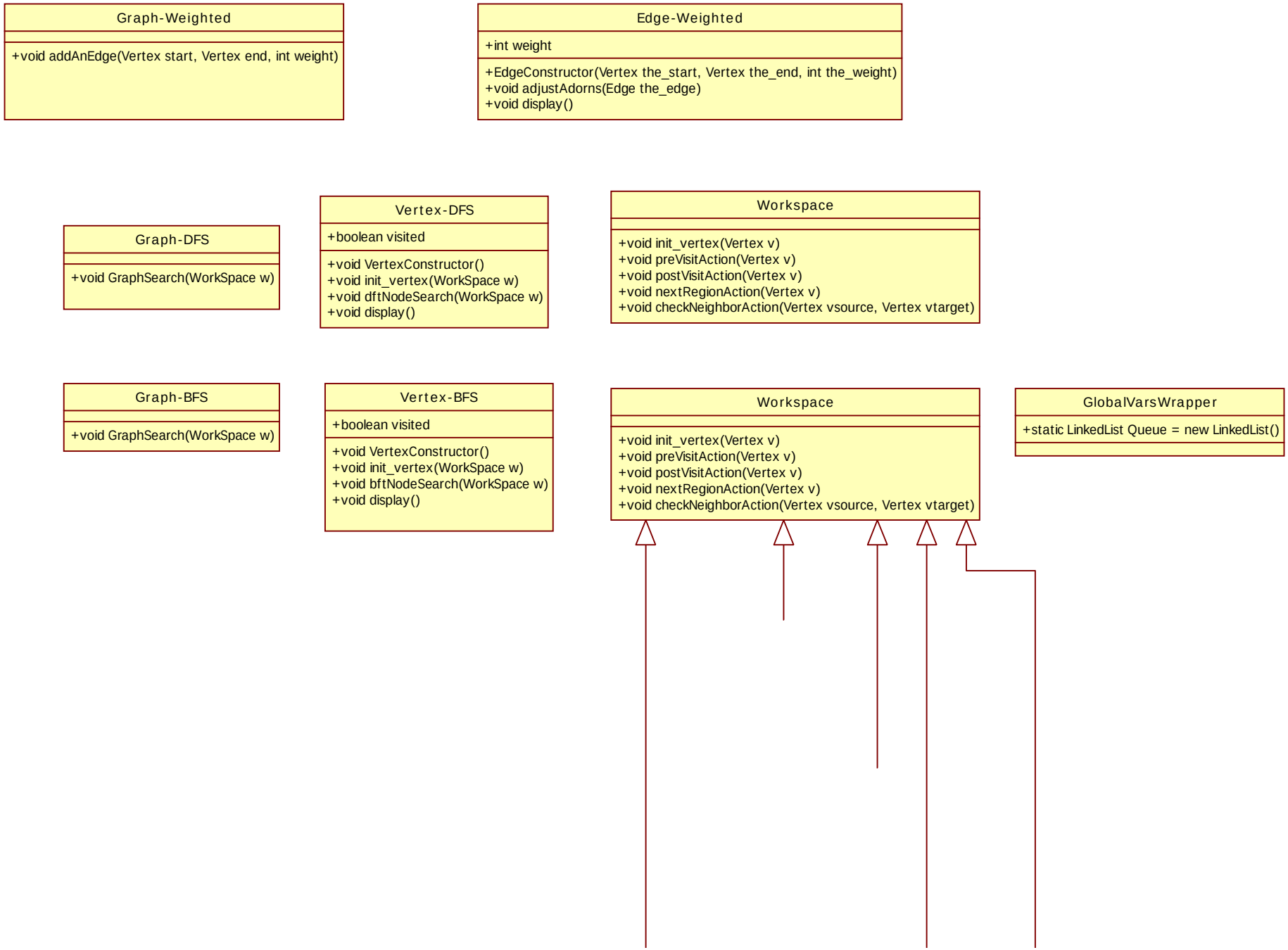
# Example: Graph Product Line

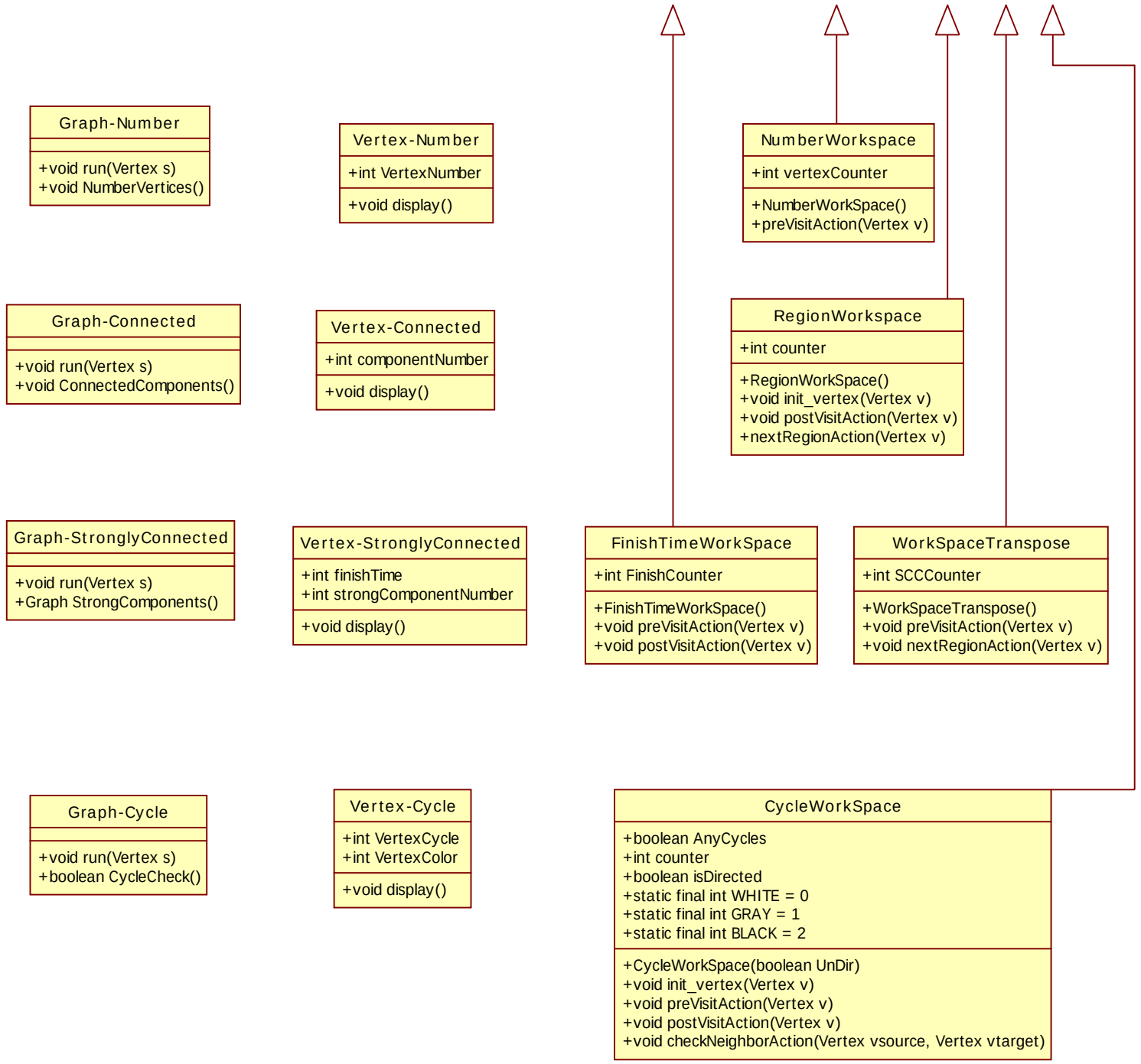
- Family of classical graph applications
- Typical of product lines
  - No two applications will have same set of features
- Features of GPL
  - (Un)Directed, Weight, Search, Algorithm

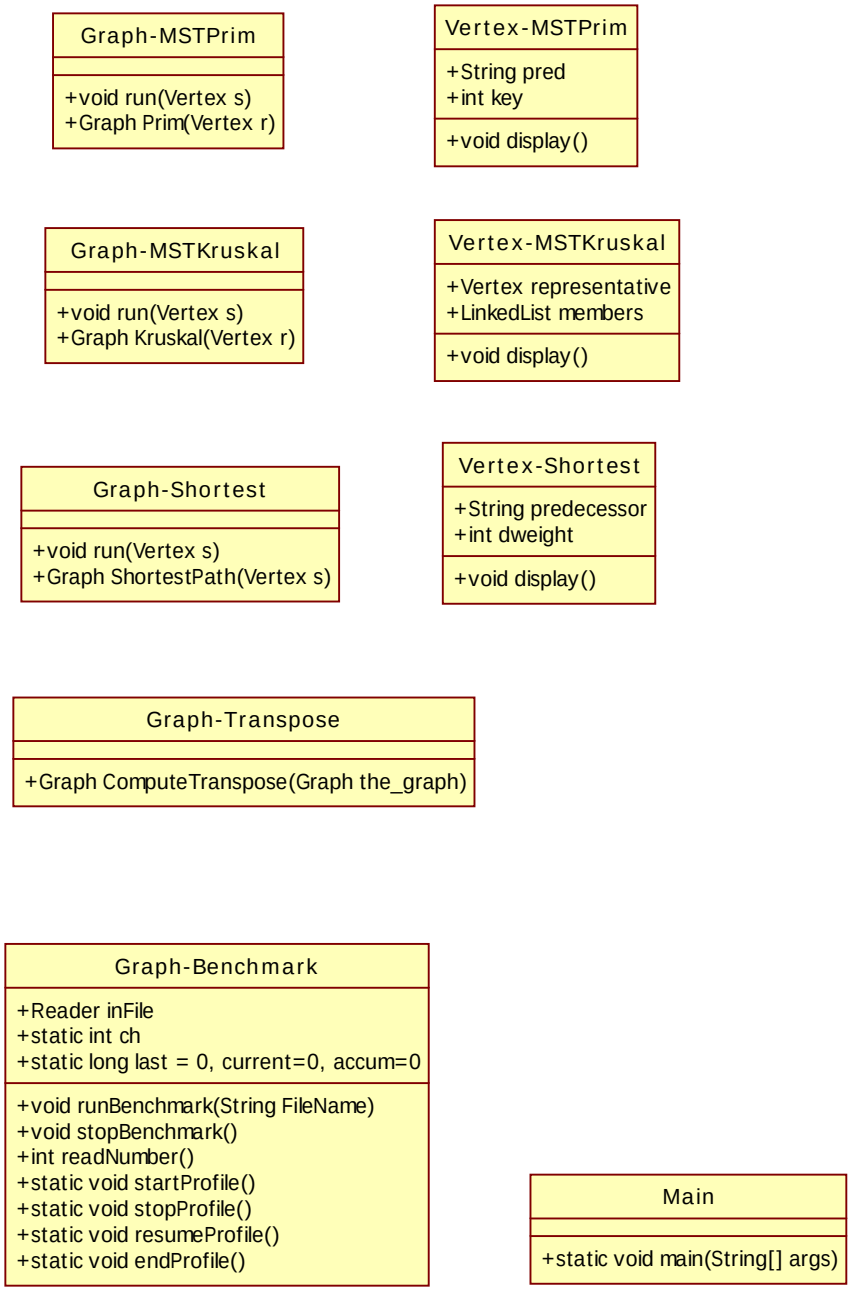
```
GPL := Gtp Wgt Src Alg+;
Gtp := Directed | Undirected;
Wgt := Weighted | Unweighted;
Src := DFS | BFS | None;
Alg := Number | Connected | StronglyConnected
       | Cycle | MST Prim | MST Kruskal | Shortest;
```


Graph Top Layer
Undirected
Directed
Graph (Un)Weighted Layer
Weighted
Graph Search Layer
DFS
BFS
Cycle
MST Kruskal
MST Prim
Connected
Strongly Connected
Shortest
Number
Transpose
A
B
A needs B
Graph Algorithm Layer

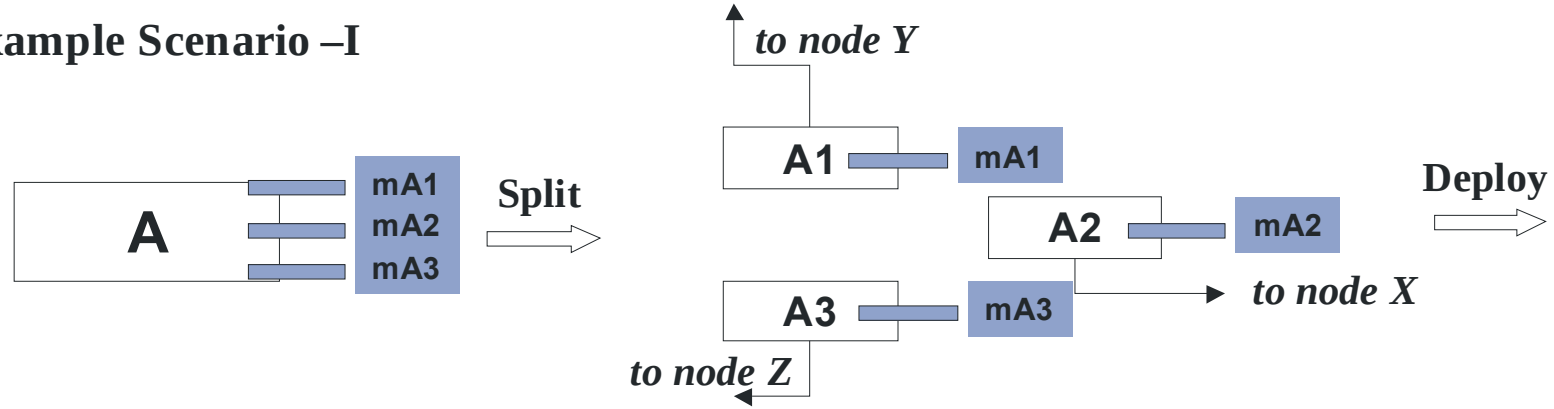

Neighbour
+Vertex end
+Edge edge
+Neighbor()
+Neighbor(Vertex v, Edge e)
Edge
+Vertex start
+void EdgeConstructor(Vertex the_start, Vertex the_end)
+void adjustAdorns(Edge the_edge)
+void display()
Graph-Directed
+vertices: LinkedList
+edges: LinkedList
+isDirected = true: Boolean
+Graph()
+run(): void
+addAnEdge(star: Vertex, end, weight): void
+addVertex(Vertex v): void
+addEdge(Edge the_edge): void
+addOnlyEdge(Edge the_edge): void
+findsVertex(String theName): Vertex
+display(): void
Vertex
+LinkedList neighbors
+String name
+Vertex()
+void VertexConstructor()
+Vertex assignName(String name)
+void addNeighbor(Neighbor n)
+void display()
Neighbour
+Vertex end
+Edge edge
+Neighbor()
+Neighbor(Vertex v, Edge e)
Edge
+Vertex start
+void EdgeConstructor(Vertex the_start, Vertex the_end)
+void adjustAdorns(Edge the_edge)
+void display()
Graph-Undirected
+vertices: LinkedList
+edges: LinkedList
+isDirected = false: Boolean
+Graph()
+void run(Vertex s)
+void addAnEdge(Vertex start, Vertex end, int weight)
+void addVertex(Vertex v)
+addEdge(Edge the_edge)
+void addOnlyEdge(Edge the_edge)
+Vertex findsVertex(String theName)
+Edge findsEdge(Vertex theSource, Vertex theTarget)
+void display()
Vertex
+LinkedList neighbors
+String name
+Vertex()
+void VertexConstructor()
+Vertex assignName(String name)
+void addNeighbor(Neighbor n)
+void display()


Graph-Weighted
+void addAnEdge(Vertex start, Vertex end, int weight)
Edge-Weighted
+int weight
+EdgeConstructor(Vertex the_start, Vertex the_end, int the_weight)
+void adjustAdorns(Edge the_edge)
+void display()
Graph-DFS
+void GraphSearch(WorkSpace w)
Vertex-DFS
+boolean visited
+void VertexConstructor()
+void init_vertex(WorkSpace w)
+void dftNodeSearch(WorkSpace w)
+void display()
Workspace
+void init_vertex(Vertex v)
+void preVisitAction(Vertex v)
+void postVisitAction(Vertex v)
+void nextRegionAction(Vertex v)
+void checkNeighborAction(Vertex vsource, Vertex vtarget)
Graph-BFS
+void GraphSearch(WorkSpace w)
Vertex-BFS
+boolean visited
+void VertexConstructor()
+void init_vertex(WorkSpace w)
+void bftNodeSearch(WorkSpace w)
+void display()
Workspace
+void init_vertex(Vertex v)
+void preVisitAction(Vertex v)
+void postVisitAction(Vertex v)
+void nextRegionAction(Vertex v)
+void checkNeighborAction(Vertex vsource, Vertex vtarget)
GlobalVarsWrapper
+static LinkedList Queue = new LinkedList()

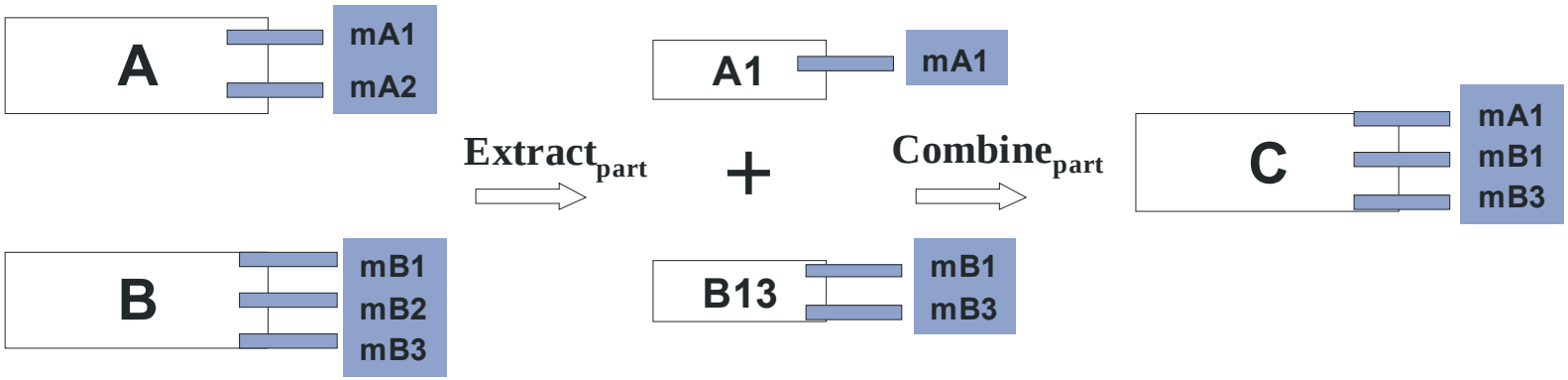

Graph-Number
+void run(Vertex s)
+void NumberVertices()
Vertex-Number
+int VertexNumber
+void display()
NumberWorkspace
+int vertexCounter
+NumberWorkSpace()
+preVisitAction(Vertex v)
Graph-Connected
+void run(Vertex s)
+void ConnectedComponents()
Vertex-Connected
+int componentNumber
+void display()
RegionWorkspace
+int counter
+RegionWorkSpace()
+void init_vertex(Vertex v)
+void postVisitAction(Vertex v)
+nextRegionAction(Vertex v)
Graph-StronglyConnected
+void run(Vertex s)
+Graph StrongComponents()
Vertex-StronglyConnected
+int finishTime
+int strongComponentNumber
+void display()
FinishTimeWorkSpace
+int FinishCounter
+FinishTimeWorkSpace()
+void preVisitAction(Vertex v)
+void postVisitAction(Vertex v)
WorkSpaceTranspose
+int SCCCounter
+WorkSpaceTranspose()
+void preVisitAction(Vertex v)
+void nextRegionAction(Vertex v)
Graph-Cycle
+void run(Vertex s)
+boolean CycleCheck()
Vertex-Cycle
+int VertexCycle
+int VertexColor
+void display()
CycleWorkSpace
+boolean AnyCycles
+int counter
+boolean isDirected
+static final int WHITE = 0
+static final int GRAY = 1
+static final int BLACK = 2
+CycleWorkSpace(boolean UnDir)
+void init_vertex(Vertex v)
+void preVisitAction(Vertex v)
+void postVisitAction(Vertex v)
+void checkNeighborAction(Vertex vsource, Vertex vtarget)


Graph-MSTPrim
+void run(Vertex s)
+Graph Prim(Vertex r)
Vertex-MSTPrim
+String pred
+int key
+void display()
Graph-MSTKruskal
+void run(Vertex s)
+Graph Kruskal(Vertex r)
Vertex-MSTKruskal
+Vertex representative
+LinkedList members
+void display()
Graph-Shortest
+void run(Vertex s)
+Graph ShortestPath(Vertex s)
Vertex-Shortest
+String predecessor
+int dweight
+void display()
Graph-Transpose
+Graph ComputeTranspose(Graph the_graph)
Graph-Benchmark
+Reader inFile
+static int ch
+static long last = 0, current=0, accum=0
+void runBenchmark(String FileName)
+void stopBenchmark()
+int readNumber()
+static void startProfile()
+static void stopProfile()
+static void resumeProfile()
+static void endProfile()
Main
+static void main(String[] args)

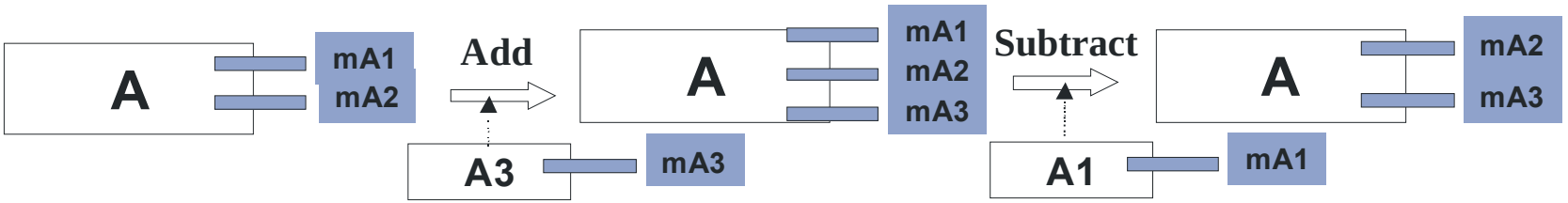
## Example Scenario –I




## Example Scenario –II




## Example Scenario –III



# Flavor

```
public BoBClass GraphOperations
{
   . . .

   public static void main (args[]) {

     BoBClass Graph_UW = Graph_Core + Graph_Undirected + Graph_Weighted;
     Graph_UW g_uw = new Graph_UW();
     g_uw.display();

     BoBClass Graph_DW = Graph_UW - Graph_Undirected + Graph_Directed;
     Graph_DW g_dw = new Graph_dw();
     g_dw.dispaly();

     BoBClass Graph_DuW = Graph_DW - Graph_Weighted + Graph_UnWeighted;
     Graph_DuW g_duw = new Graph_DuW();
     g_duw.display();

}
```

# Contents

- Problem and Motivations
- BoB basics
- BoBs for application partitioning
- BoBs as elements of reuse
- **Related work comparisons**
- Discussion and conclusions

# Related Work/Comparison

- Traditional Objects
- Related Composition Mechanisms
  - Traits, Jigsaw
- Fragmented Objects
- Fragmentation in Databases
- Application Partitioning
- Class Refactoring
- Multi Dimension Separation of Concerns
- Distributed Design Paradigms

# Objects

- Objects can be considered an extreme case of BoB where:
  - all the interface methods are designated as **`together`**
- Other differences come from the programming model chosen for BoBs.

# Operators in JIGSAW [GiladBracha92]

- Merge
  - Yields the concatenation of two modules. Only non confliction attributes are considered
- Modification
  - Overriding (**M1** override **M2**)
- Name conflict resolution
  - **M** rename **a** to **b**
- Select
  - Returns the value of attribute named a in r
- Restrict
  - Removes the attribute a from the module
- Project
  - Retain particular attribute(s) definition(s)
- Freeze
  - Statically binding an attribute **a**
  - Dual operation
    - freeze_all_except **a**
- Attribute Visibility
  - **M** hide **a**, **M** show **a**
- Rename: Access to overriden definitions
  - copy **a** as **b**

# Operators in Traits[Toplas06]

- Composing Classes
  - Class = Superclass + State + Traits + Glue methods
- Trait Composition Operators
  - Sum (+)
  - Aliasing (->)
  - Exclusion (-)

# Fragmented Objects (FOs)

- FO is a
  - Distributed shared object
  - Clients in different address spalce
  - Internally a set of *fragments*
  - Fragments communicate through communication channels
- Comparison
  - Very different concepts; distributed shared object v/s breakable objects
  - BoBs don't carry the notion of distribution per say
  - BoBs can be used to build FOs
    - Possible to achieve dynamic FOs – form and the location of fragments keeps on changin
  - BoBs don't retain a single identity after splitting

# Fragmentation in Databases

- Horizontal and Vertical Fragmentation in Database systems: Class is an ordered relation: $\mathcal{C} = (\mathbf{K}, A, M, I)$
  - Horizontal Fragmentation: $C_h = (\mathbf{K}, A, M, I^{'})$, where, $(I^{'} \subseteq I)$
  - Vertical Fragmentation: $C_v = (\mathbf{K}, A^{'}, M^{'}, I)$, where, $(A^{'} \subseteq A)$ and $(M^{'} \subseteq M)$
  - Hybrid Fragmentation combination of the above two

- BoBs and Fragmentation in Databases
  - horizontal fragmentation and BoB splitting are not related at all
  - vertical fragmentation employs some similar lines, but focus is to find fragments techniques to optimize query.
  - simplified models as no shared fields considered in the latter case
  - BoBs look at splitting from a programming perspective
  - Configuration Files (CFs) make the process of finding the lines of splits external to BoBs

# Application Partitioning

- J-orchestra, Pangaea, Coign, Addistant
- So far focused mainly on
  - finding optimal ways to partition an application among different nodes, and
  - component conversions into distributed components.
- Our focus is:
  - to have an entity which is more suitable for such partitioning
  - declarative approach to application partitioning.
- Granularity level of partitioning
  - objects or components,
  - our case, granularity level
    - finer
    - related to the methods of a BoB.

# Class Refactoring

- Different refactoring methods have been proposed
- Class refactoring method
  - Extract Class
  - Extract Interface [Fowler:catalogue]

- provides a means to create new class by moving the relevant fields *Move Filed* and methods *Move Method* from the old class into a new class.
- The main intent here is to improve the code design by splitting bloated classes and creating new crisper classes.
- No comprehensive techniques exist to provide refactoring of application classes for functional partitioning

# Distributed Design Paradigms

| Paradigm | SA-*initial* | SB-*initial* | SA-*later* | SB-*later* |
|---|---|---|---|---|
| Client-Server | A | know-how<br>resource B | A | know-how<br>resource **B** |
| Remote Evaluation | know-how A | resource B | A | *know-how*<br>resource **B** |
| Code on Demand | resource A | know-how B | resource<br>*know-how* **A** | B |
| Mobile Agent | know-how A | resource | - | know-how<br>resource **A** |
| Breakable Object | know-how A<br>resource | resource | (**A**)-part-1,<br>(knowhow)-<br>part-1 re-<br>source | **(A)-part-1**,<br>(*know-how*)-<br>part-2 re-<br>source |

# Contents

- Problem and Motivations
- BoB basics
- BoBs for application partitioning
- BoBs as elements of reuse
- Related work comparisons
- **Discussion and conclusions**

# Contributions of Work

- The concept of **Breakable Object (or BoB)**
- BoBProgramming Model
  - Structure
- Application Partitioning
  - Splitting, Merging
  - Program Equivalence
    - Proof of equivalence
- BoB Driven Architecture (BODA)
- BoB Composition Mechanisms

# Publications (related to this work)

1. **Automated Refactoring of Objects for Application Partitioning** 12th Asia-Pacific Software Engineering Conference (**APSEC**) ,Taipei , Taiwan, December 15-17,2005. Authors: Vikram Jamwal and Sridhar Iyer

2. **Breakable Objects: Building Blocks for Flexible Application Architectures** 5th Working IEEE/IFIP Conference on Software Architecture(**WICSA**), November 6 - 10, 2005, Pittsburgh, Pennsylvania, USA Authors: Vikram Jamwal and Sridhar Iyer

3. **BoBs: Breakable Objects** (Poster Paper) 20th Object-Oriented Programming, Systems, Languages And Applications (**OOPSLA**) October 16- 20, 2005, San Diego, California, USA Authors: Vikram Jamwal and Sridhar Iyer

4. **Mobile Agent based Realization of a Distance Evaluation System** 2003 International Symposium on Application and the Internet (**SAINT** 2003), Orlando, Florida, USA, Jan 27-31, 2003 Authors: Vikram Jamwal and Sridhar Iyer

5. **Mobile Agents for effective structuring of large-scale distributed applications** Workshop on Software Engineering and Mobility, **ICSE** 2001 at Toronto, Canada Authors: Vikram Jamwal and Sridhar Iyer

# Thank you