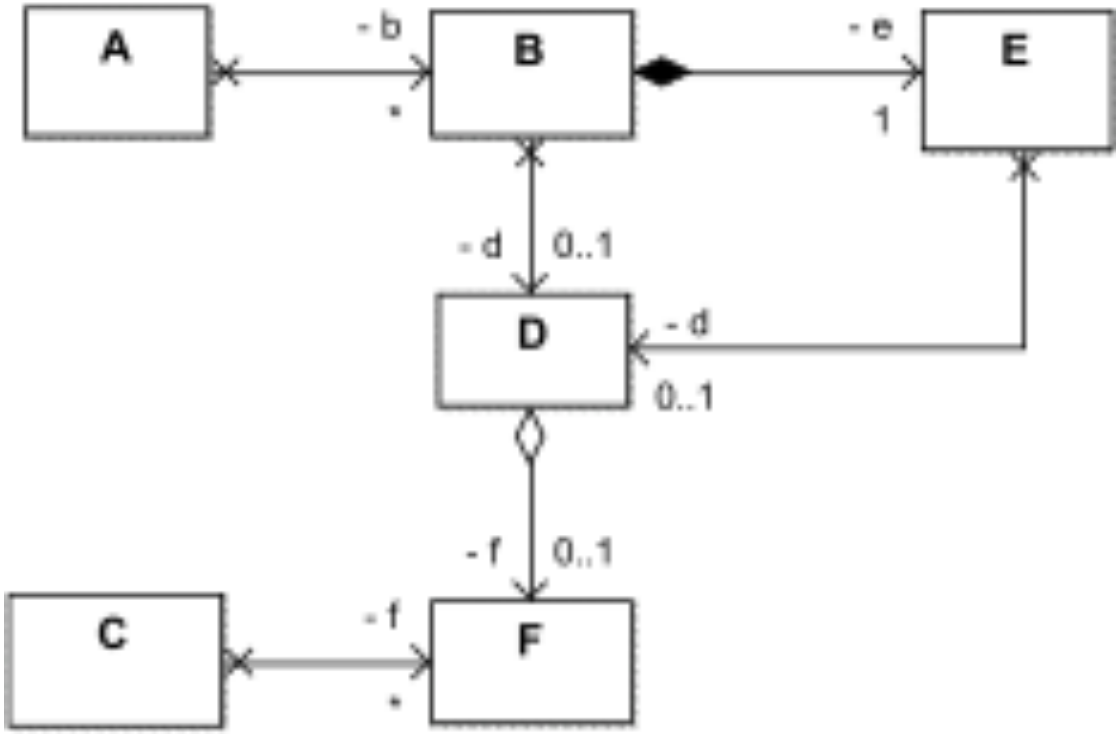
Test d'intégration

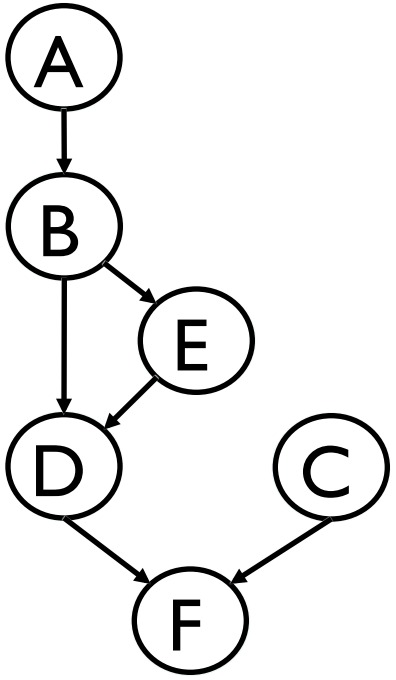
Mottu — Le Traon — Baudry — Sunyé

Intégration

- But : tester les interactions entre classes
- Lien entre test d'intégration et unitaire:
  - il faut ordonner les classes pour le test
- Il faut identifier les dépendances entre classes
  - Problème dans le cas de cycles de dépendances

Cas simple : un graphe acyclique

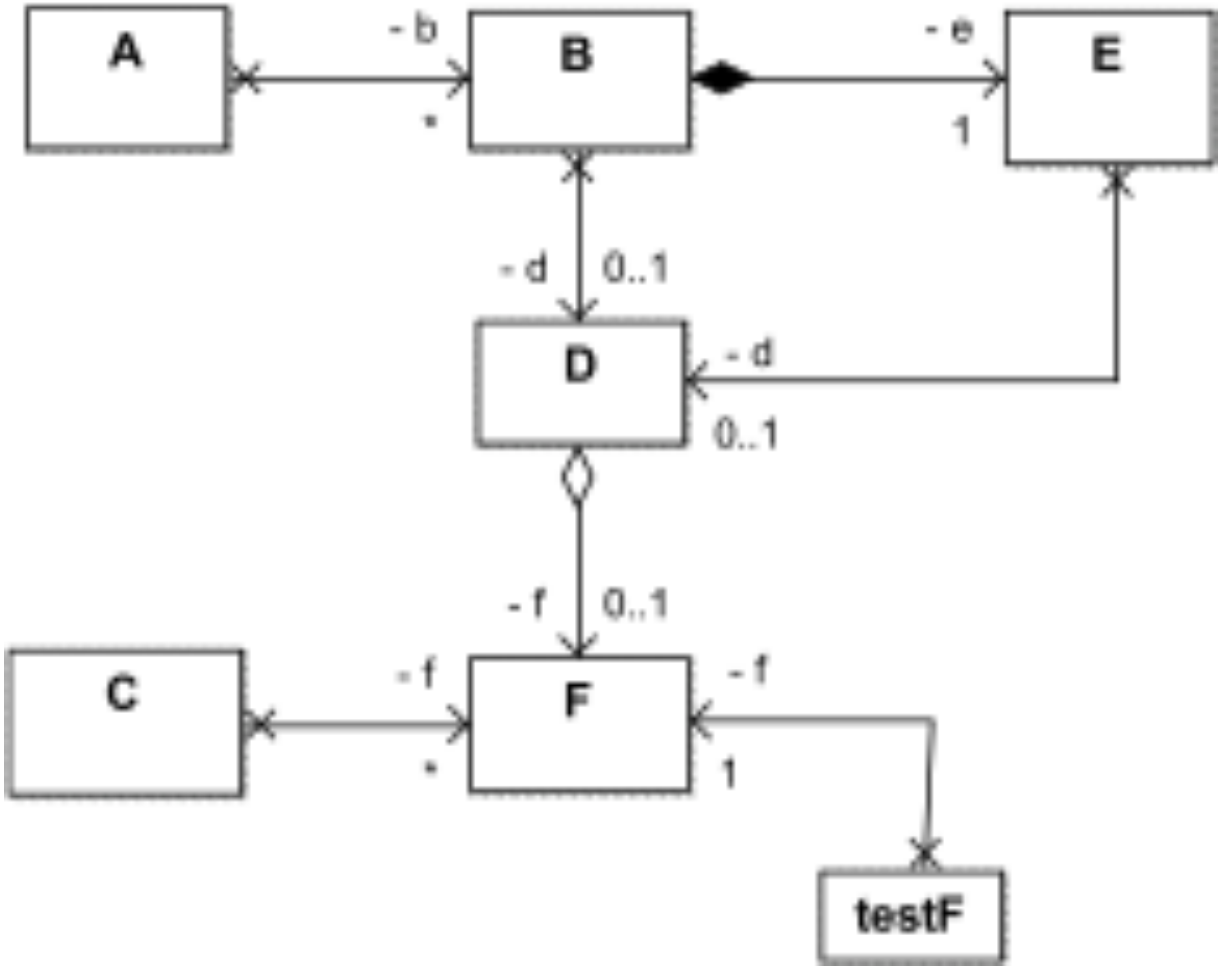




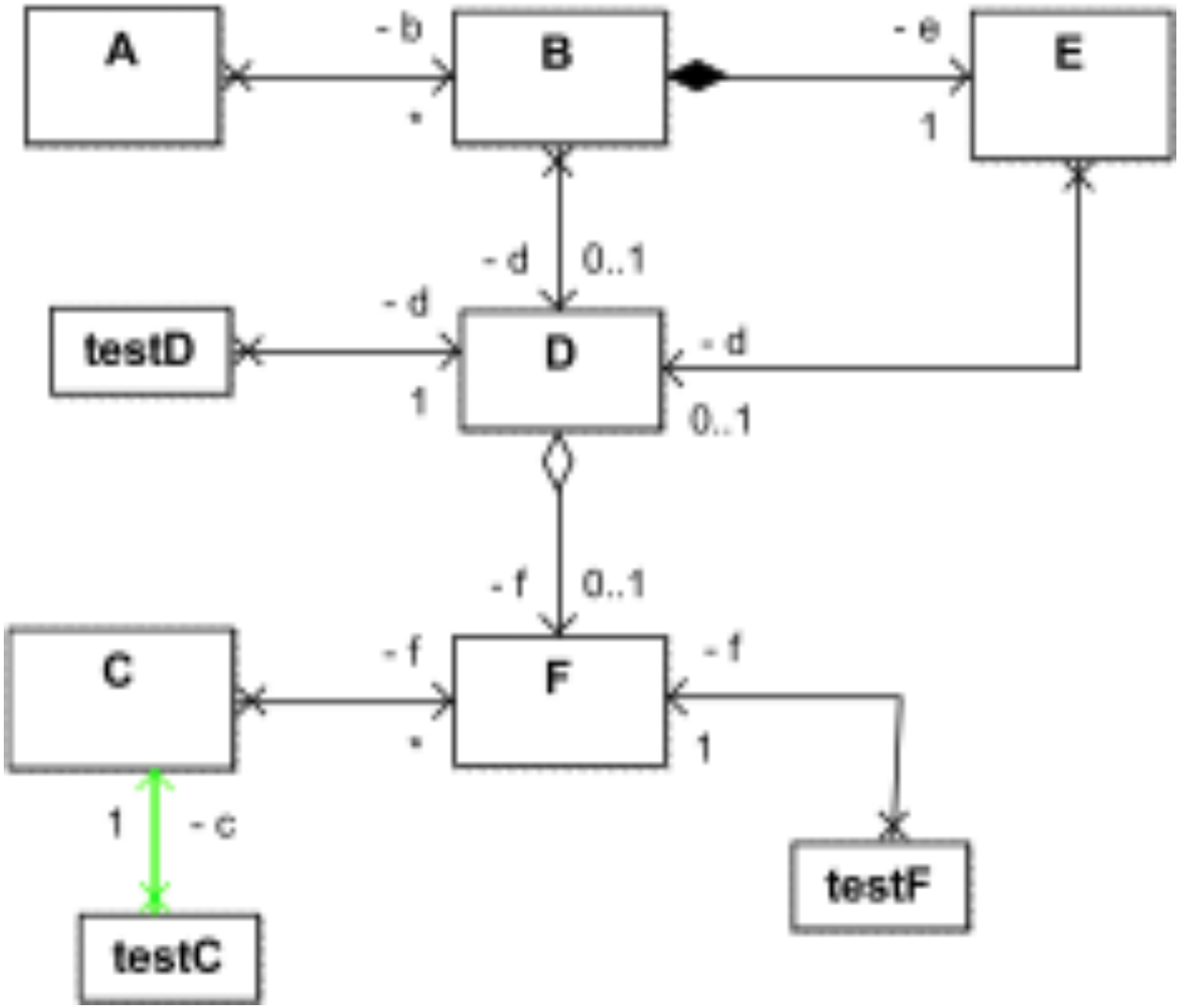
Graphe de dépendances acyclique

Ordre partiel pour le test: F, (C, D), E, B, A

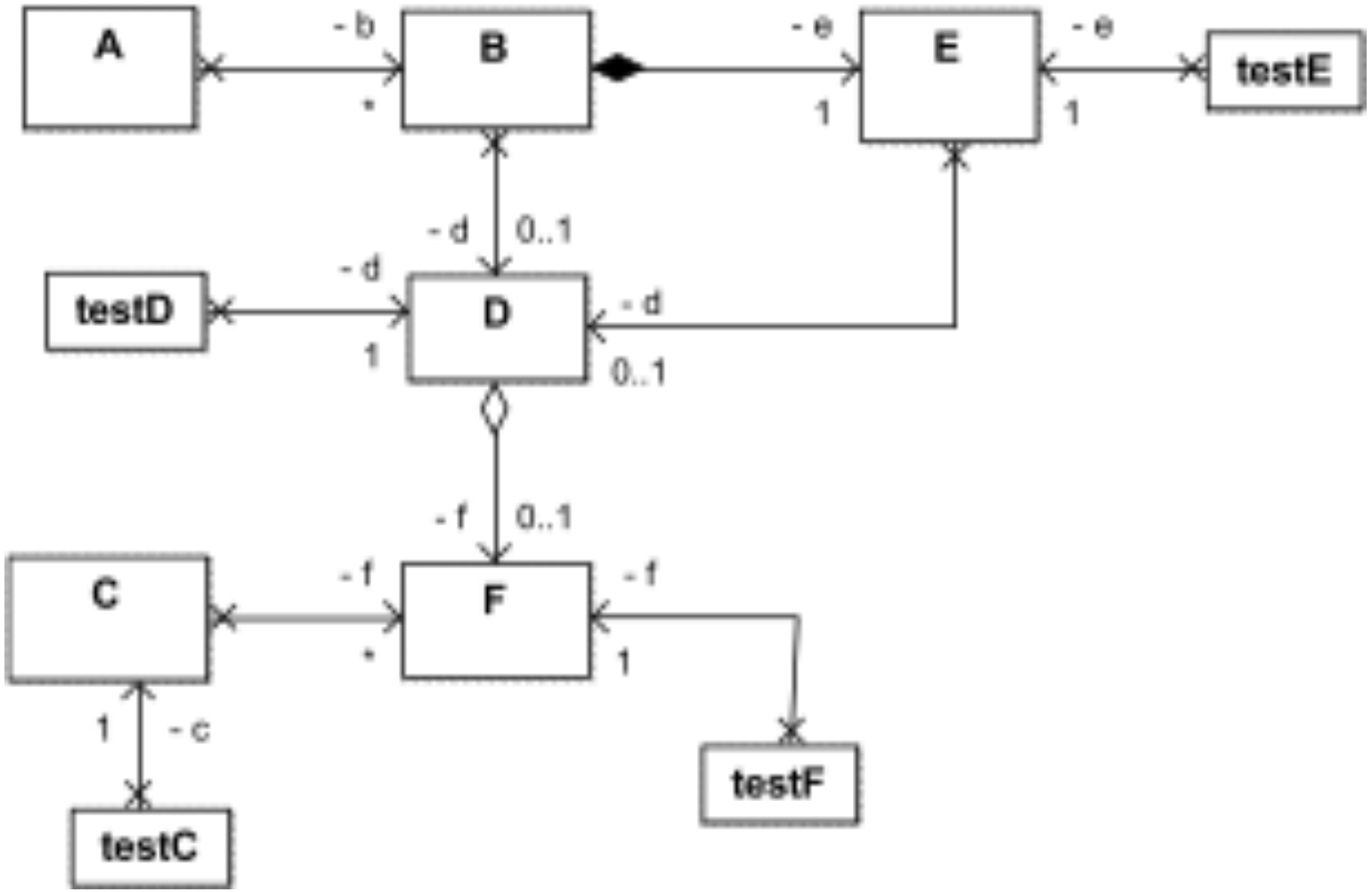
Étape 1



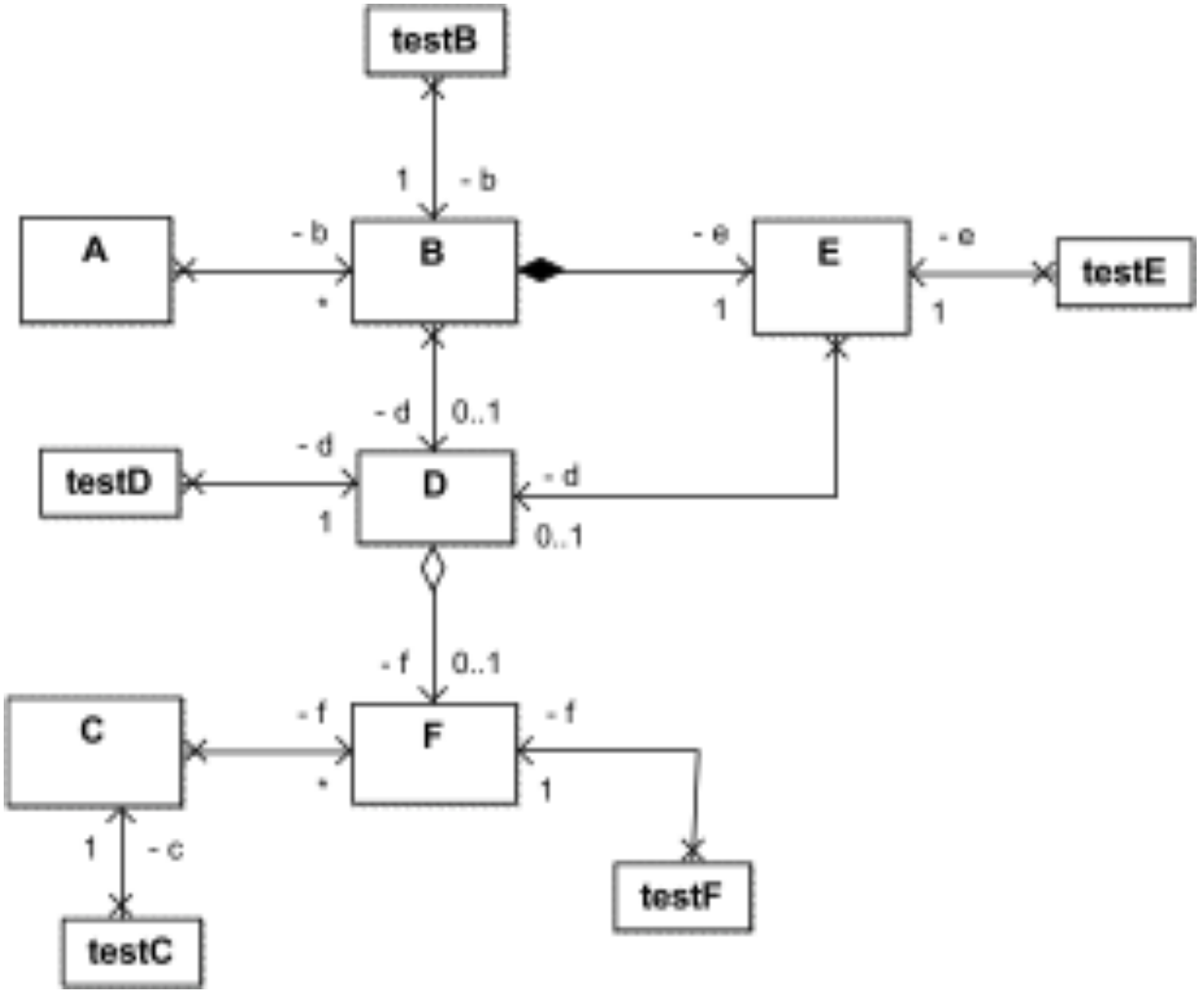
Étape 2



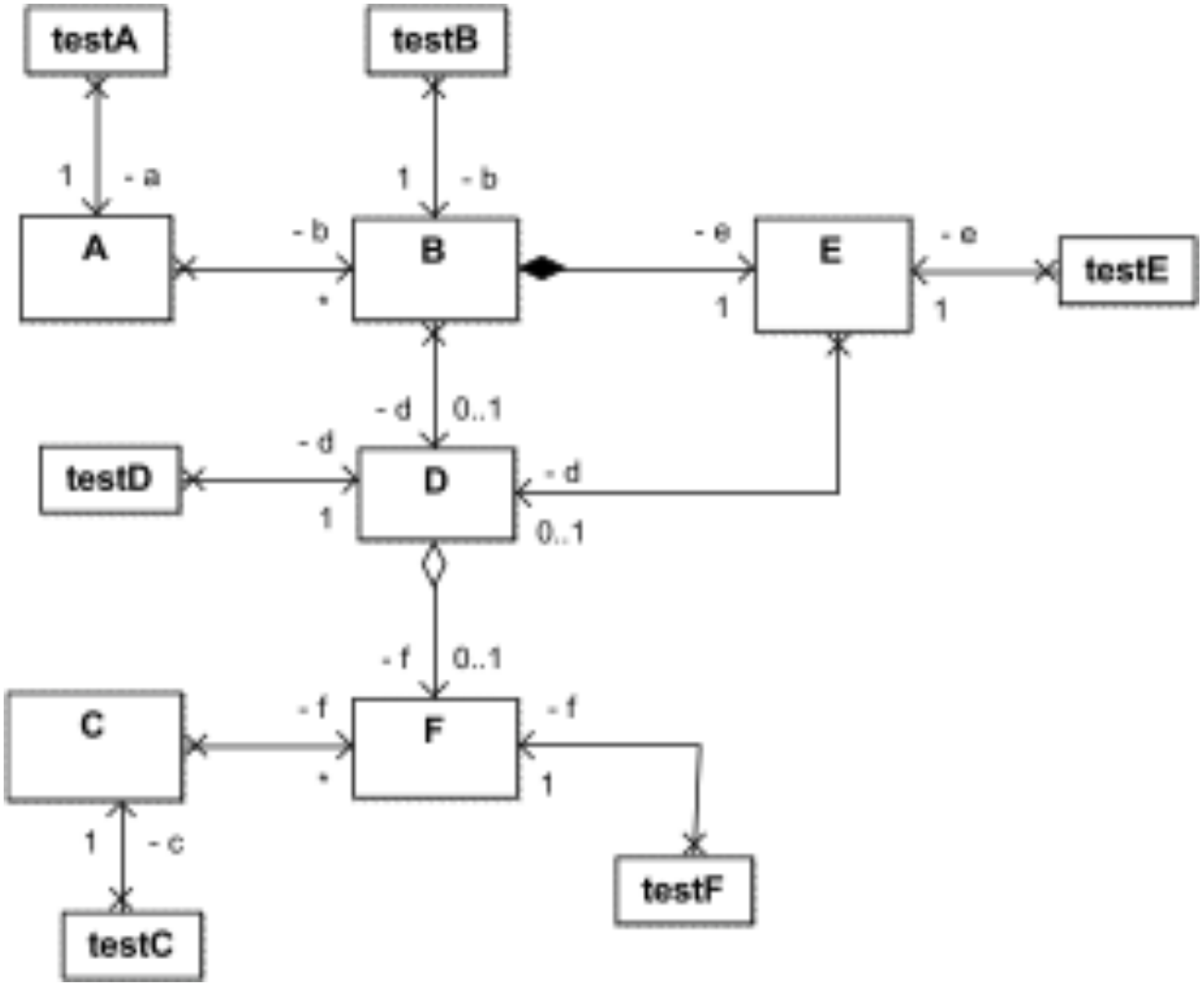
Étape 3



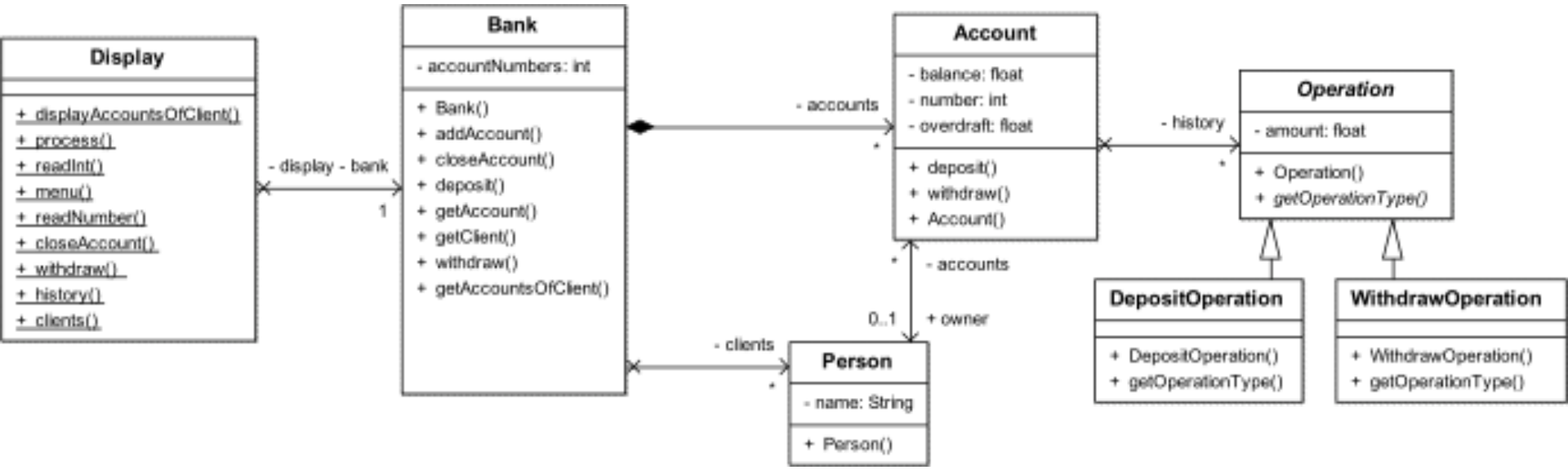
Étape 4

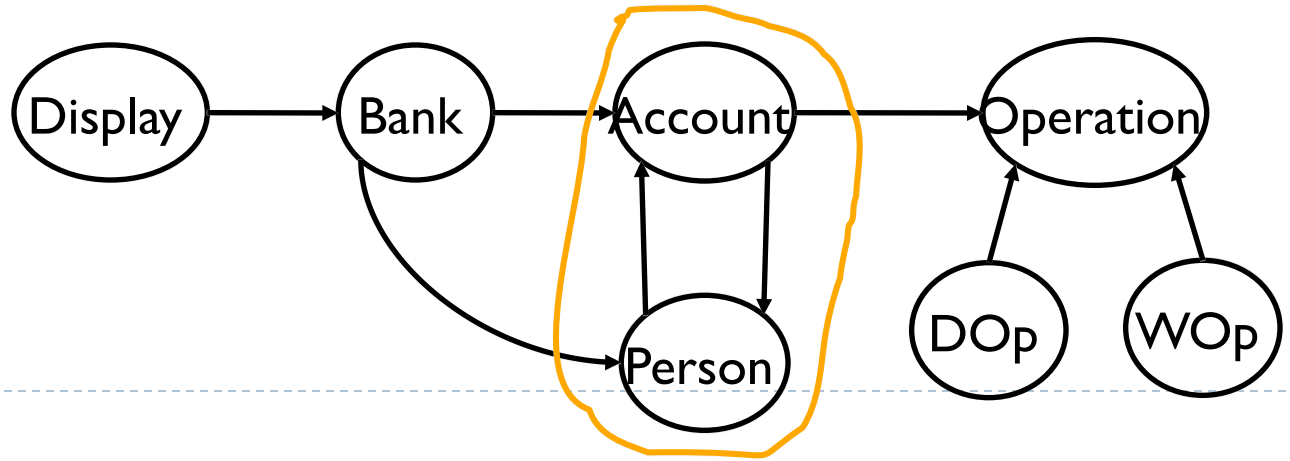

testB
1
- b
A
- b
*
B
- e
1
E
- e
1
testE
- d
0..1
- d
testD
1
D
- d
0..1
- f
0..1
C
- f
*
F
- f
1
1
- c
testC
testF

Étape 5


testA
testB
1
- a
1
- b
A
- b
*
B
- e
1
E
- e
1
testE
- d
0..1
- d
testD
1
D
- d
0..1
- f
0..1
C
- f
*
F
- f
1
1
- c
testC
testF

Cas moins simple: présence de cycles





Intégration avec cycles

- Il faut casser les cycles
  - développer des simulateurs de classes (« bouchon de test »)
  - un simulateur a la même interface que la classe simulée, mais a un comportement contrôlé, cf. cours précédent
- Exemple

Exemples de stub

```
/**
 * Creates an account for the person named name
 * If no client has this name, a new client object is created and is added to the list of clients,
 *  then the account is created
 * If the client exists the account is created, added to the bank's and the client's list of accounts
 */
public int addAccount(String name, float amount, float overdraft) {
		this.accountNumbers++;
		Person p = getClient(name);
		//if a client named name already exists in the bank's set of clients
		if (p!=null){
			Account a = new Account(p, amount, overdraft, accountNumbers);
			p.addAccounts(a);
			this.addAccounts(a);
		}
		//if the client does not exist, add it tp the bank's list of clients and create account
		else{
			Person client = new Person(name);
			this.addClients(client);
			Account a = new Account(client, amount, overdraft, accountNumbers);
			client.addAccounts(a);
			this.addAccounts(a);
		}
		return accountNumbers;
}
```

Exemples de stub

Stub 1

```
/**
 * Creates an account for the person named name
 * If no client has this name, a new client object is created and is
 * added to the list of clients, then the account is created
 * If the client exists the account is created, added to the bank's and the client's list of accounts
 */
public int addAccount(String name, float amount, float overdraft) {
        return 0;
}
```

Stub 2

```
/**
 * Creates an account for the person named name
 * If no client has this name, a new client object is created and is
 * added to the list of clients, then the account is created
 * If the client exists the account is created, added to the bank's and the client's list of accounts
 */
public int addAccount(String name, float amount, float overdraft) {
        return 1;
}
```

Exemples de stub

```
/**
 * Looks for a person named name in the set of clients.
 * Returns the Person object corresponding to the client if it exists
 * Returns null if there is no client named name
 */
public Person getClient(String name) {
        Iterator it = this.clientsIterator();
        while (it.hasNext()){
                Person p = (Person)it.next();
                if(p.getName()==name){
                        return p;
                }

        }
        return null;
}
```

Exemples de stub

Stub 1

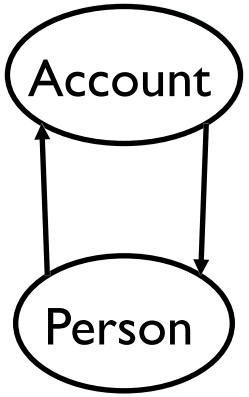
```
/**
 * Looks for a person named name in the set of clients.
 * Returns the Person object corresponding to the client if it exists
 * Returns null if there is no client named name
 */
public Person getClient(String name) {
        return null;
}
```

Stub 2

```
/**
 * Looks for a person named name in the set of clients.
 * Returns the Person object corresponding to the client if it exists
 * Returns null if there is no client named name
 */
public Person getClient(String name) {
        return new Person("toto");
}
```

Exemple Banque

- Exemple, pour tester en présence de ce cycle



Regarder quelles sont les méthodes de Person utilisées par Account

```
public class Person {
   /*
    * Initializes the name of the person with the param n
    * Creates a new vector to intialize the acounts set
    */
   public Person(String n){
      name = n;
      accounts = new Vector(); }

   public String getName(){return name;}
}
```

Stub de la classe Person

```
public class Person {
   /*
    * Initializes the name of the person with the param n
    * Creates a new vector to initialize the accounts set
    */
   public Person(String n){ }

   public String getName(){return (“toto”);}
}
```

Exemple Banque

- Etape 1

  Tester la classe Account avec le stub de Person

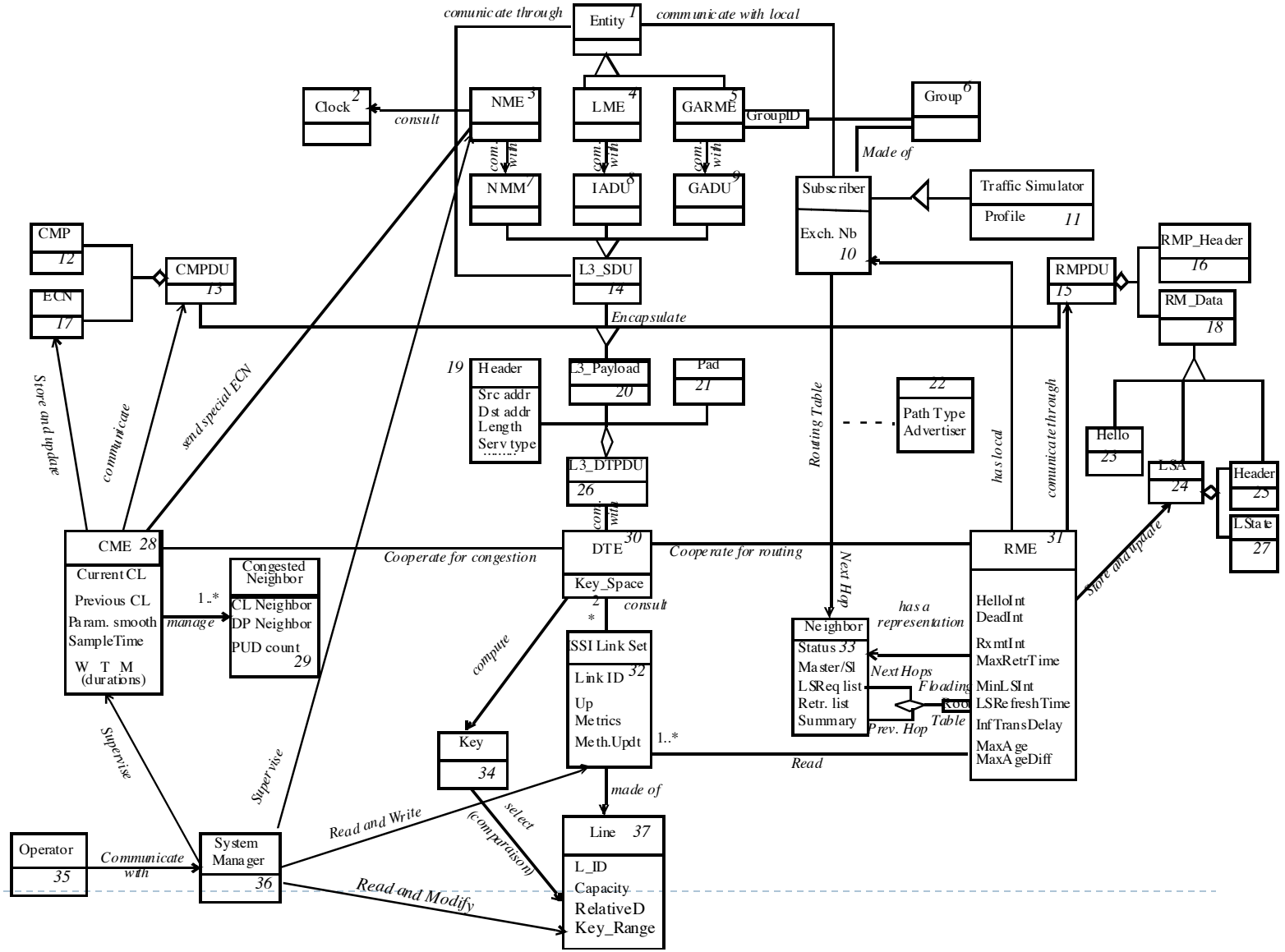
- Etape 2

  Tester la classe Person avec Account

- Etape 3

  Retester la classe Account avec la vraie classe Person

Cas encore moins simple : 37 classes



Cas encore moins simple

- Contraintes sur la conception
  - pas d'interdépendances
  - contrainte forte dans un cadre OO
- Sans contraintes sur la conception
  - on intègre tout d'un coup: stratégie « big bang »
  - heuristique pour prendre en compte les interdépendances au moment de l'intégration

Une stratégie efficace pour l'ordre d'intégration

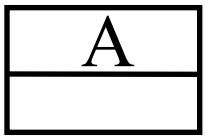
- Basée sur un modèle de graphe:
  - graphe de dépendances de test (GDT)
- Deux types de dépendances
  - Héritage
  - client/serveur
- Dépendances
  - classe – classe
  - méthode – classe

Transformation UML vers GDT

2 types de noeuds

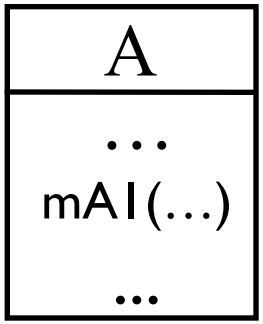
classe

méthode

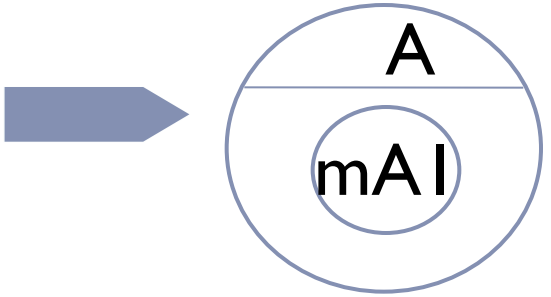


Classe A

Noeud classe



Méthode mA1 de la classe A

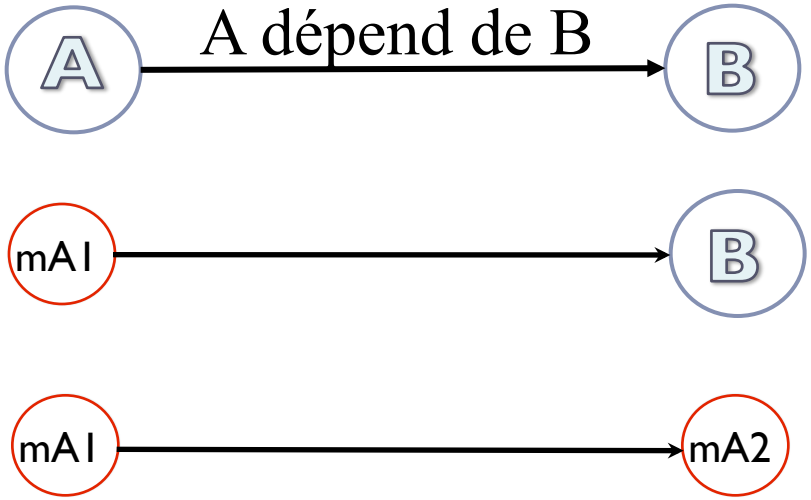


Noeud méthode dans une classe

Transformation UML vers GDT

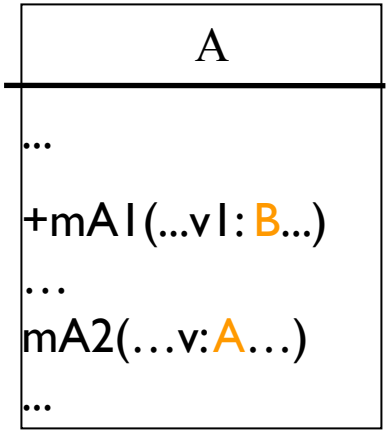
3 types of d'arcs

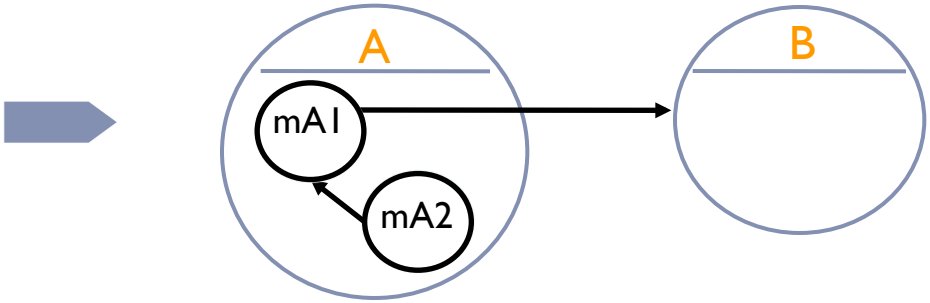
- class_to_class
- method_to_class
- method_to_method



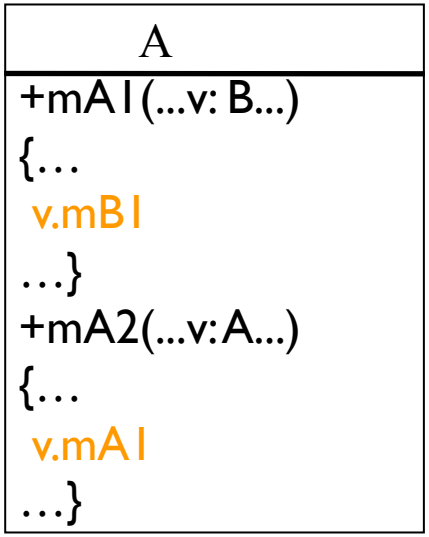
Transformation UML vers GDT

Méthode vers classe

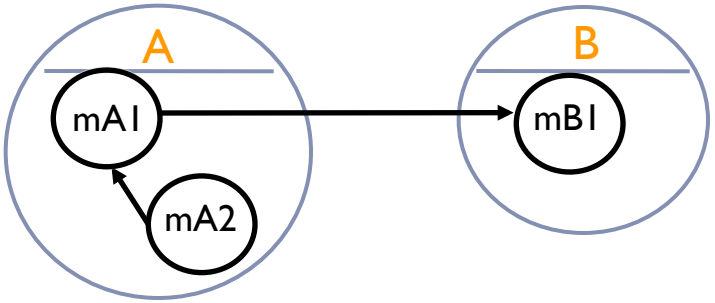




Méthode vers méthode

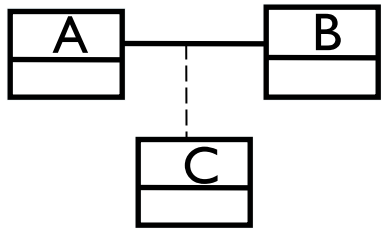


raffinement

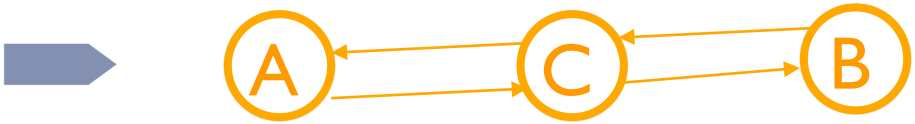


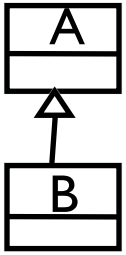
Langage d'action (OCL ...)

Transformation UML vers GDT

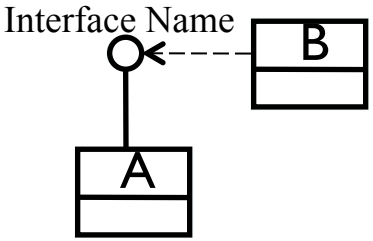


classe d'association

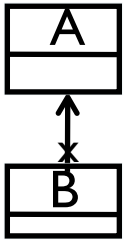




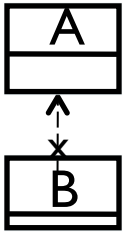
héritage



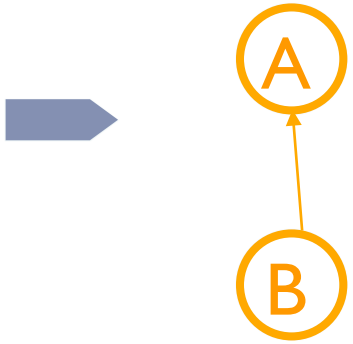
Interface

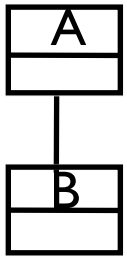


association unidirectionnelle

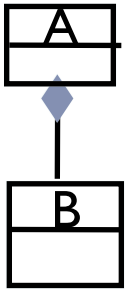


dépendance

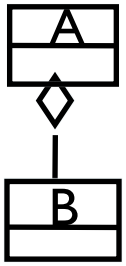




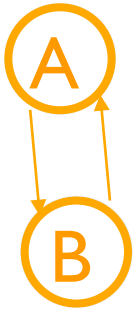
association



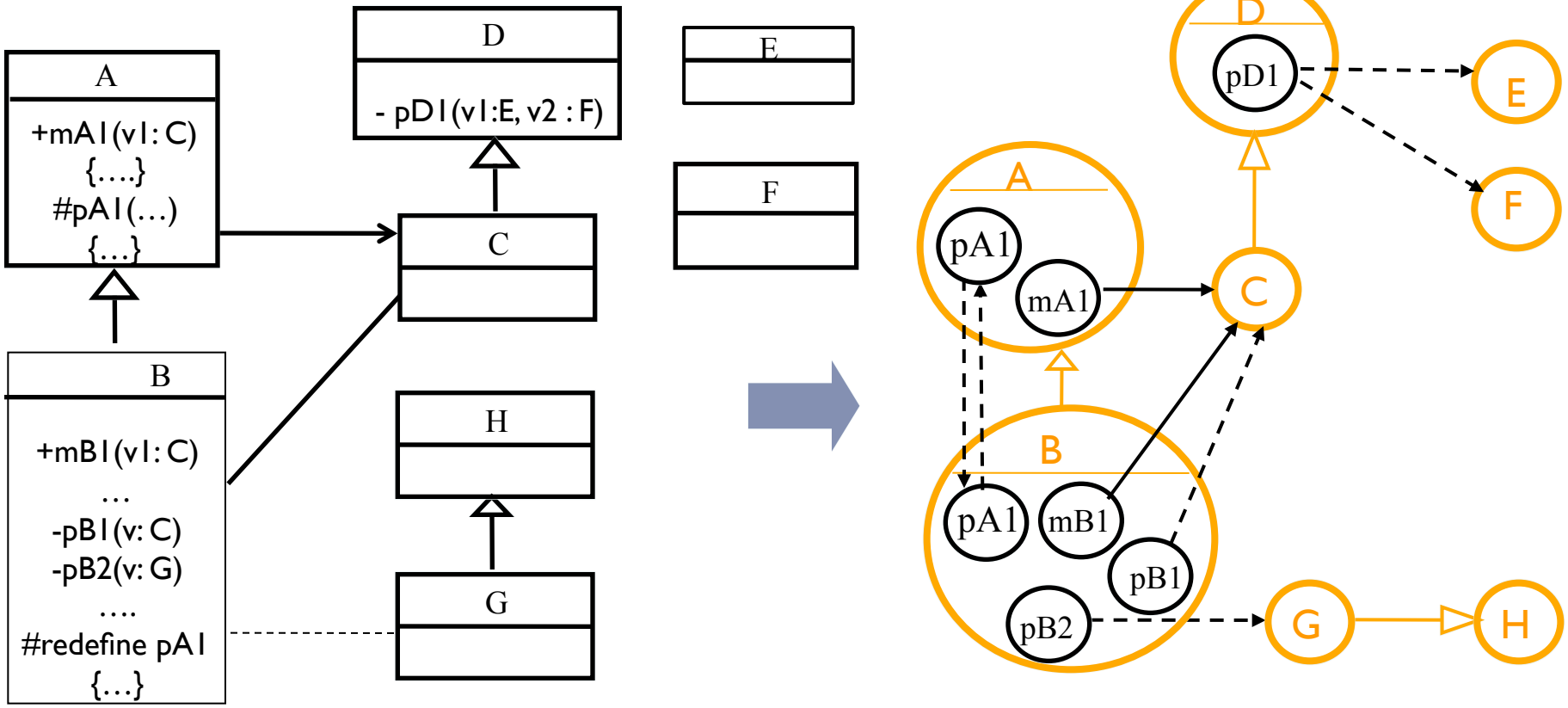
composition



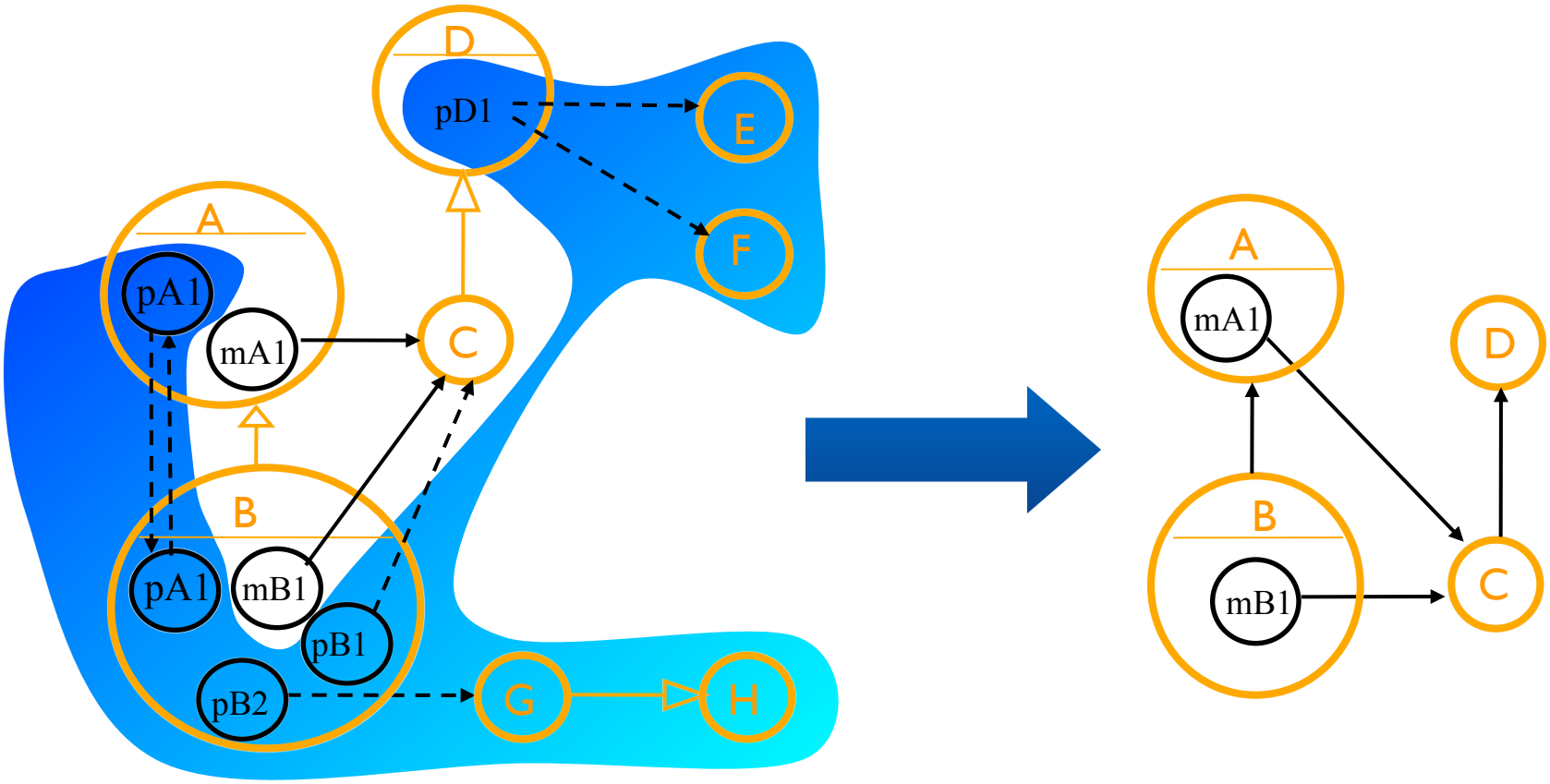
aggregation



Transformation UML vers GDT



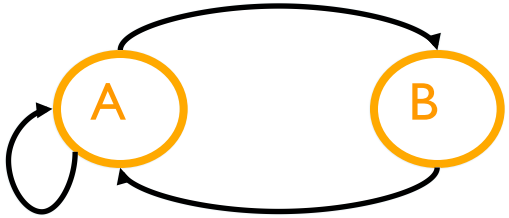
Transformation UML vers GDT



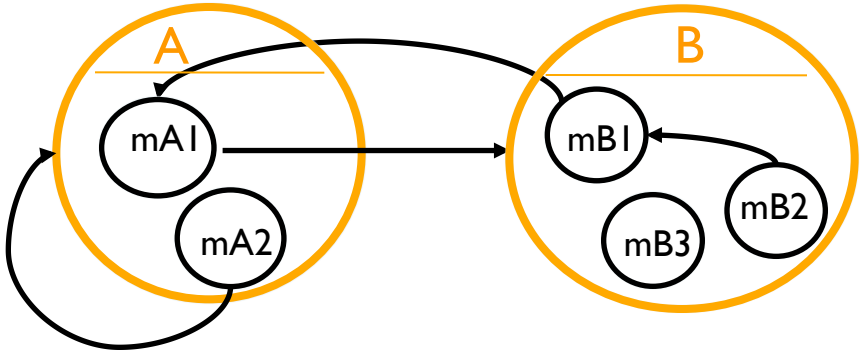
Supprimer les dépendances spécifiques à l'implantation

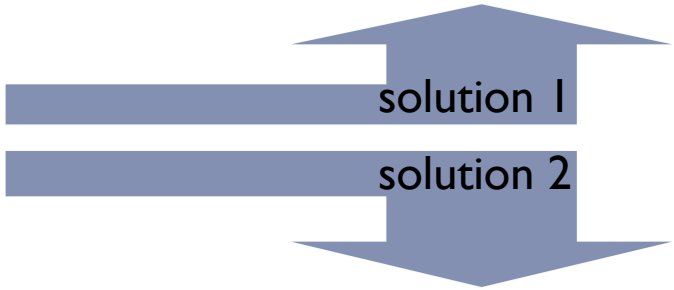
Transformation UML vers GDT

Perte d'information

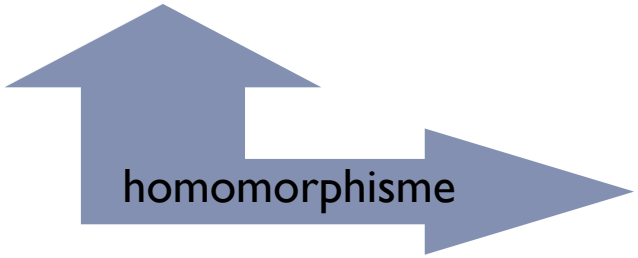


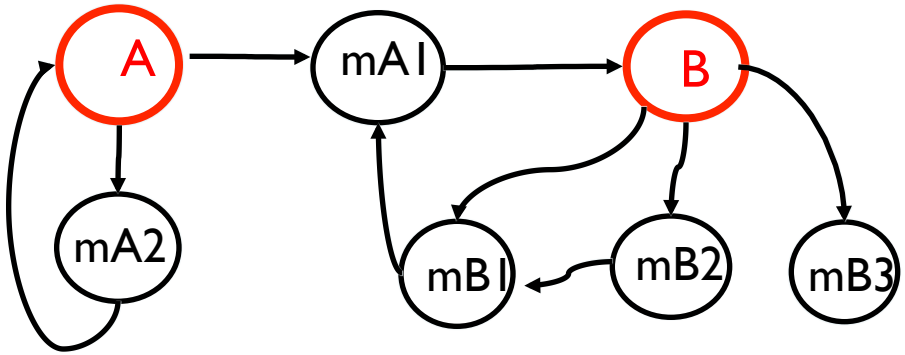
Pas un graphe classique





GDT préliminaire





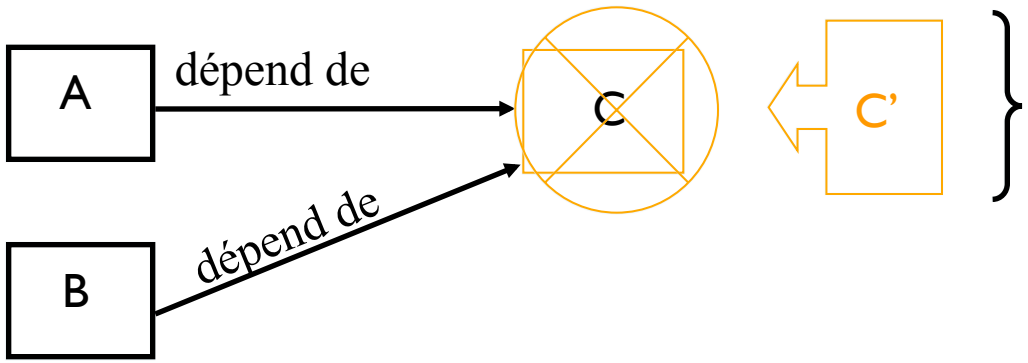
Pas de perte d'information

Une stratégie efficace pour l'ordre d'intégration

- Comment choisir un ordre d'intégration à partir du GDT?

  Minimiser le nombre de stubs à écrire

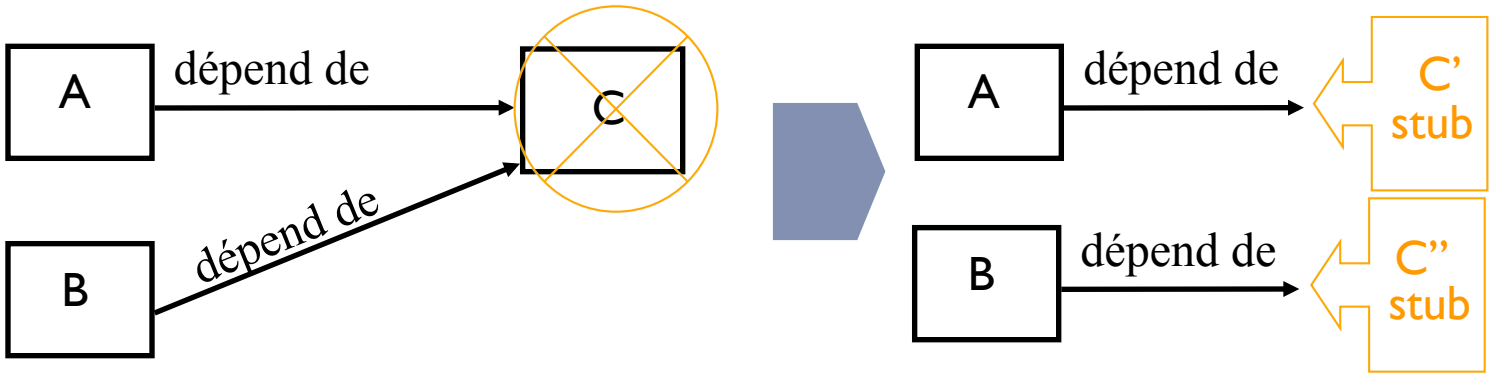
  stub réaliste => simule tous les comportements (réutiliser une ancienne version du composant)



C' simule le comportement de C vis-à-vis de A et B

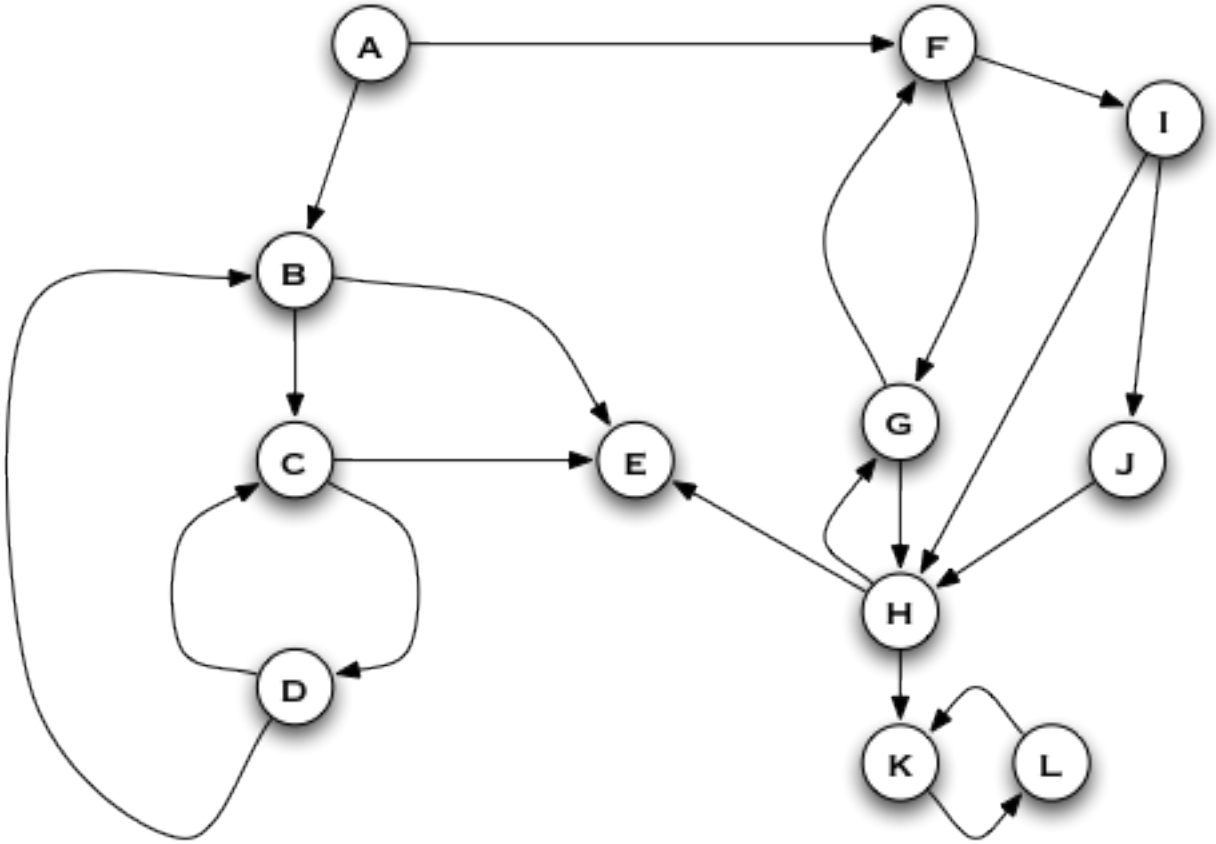
Une stratégie efficace pour l'ordre d'intégration

- Stub spécifique => ne simule que les comportements utilisés par le client



un stub spécifique de C pour A et un autre pour B

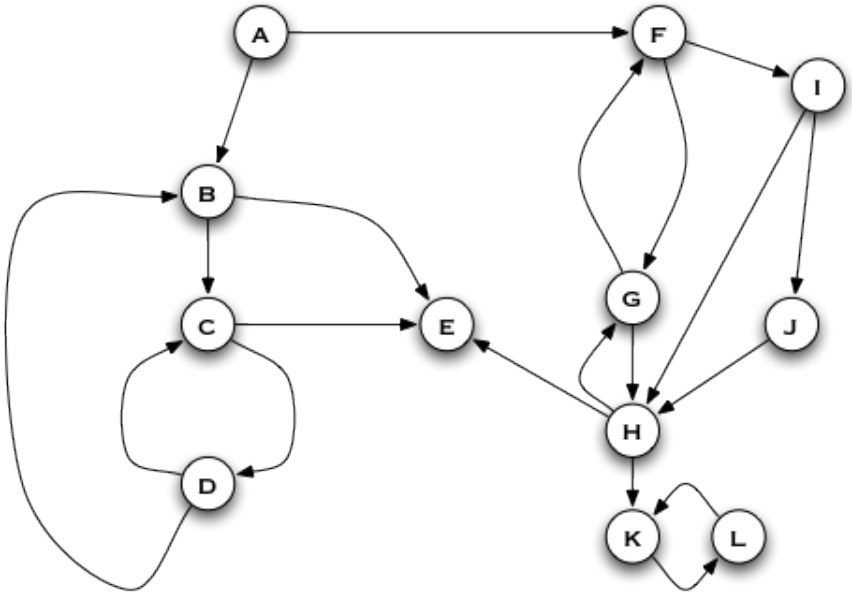
Une stratégie efficace pour l'ordre d'intégration



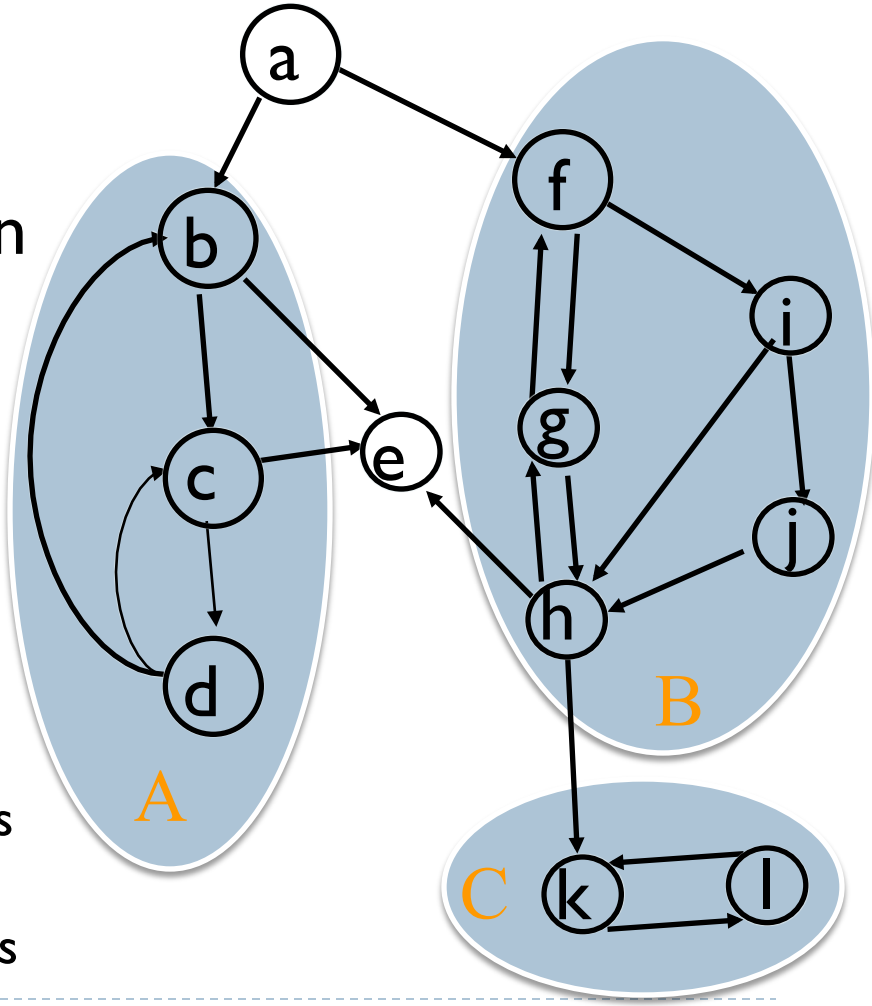
ordre optimal => NP-complet

complexité = n!

Une stratégie efficace pour l'ordre d'intégration



Tarjan

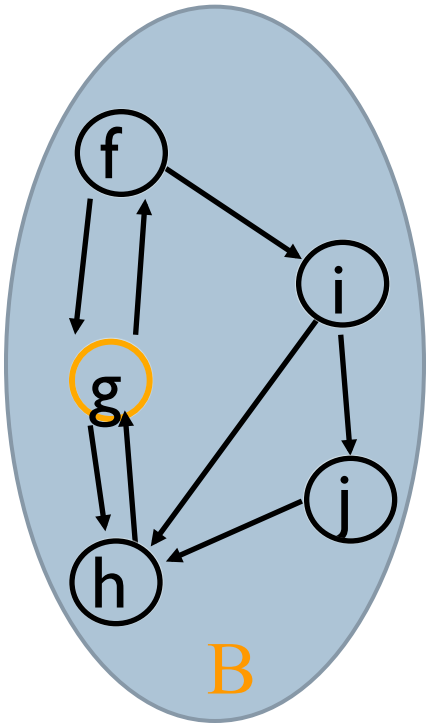


Identifie les composantes fortement connexes

ordre partiel initial [(e ou C), (A ou B), a]

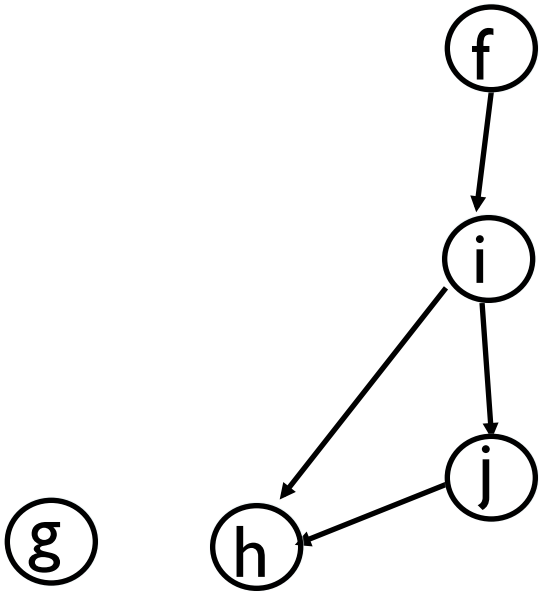
Complexité linéaire avec le nombre de noeuds

Une stratégie efficace pour l'ordre d'intégration

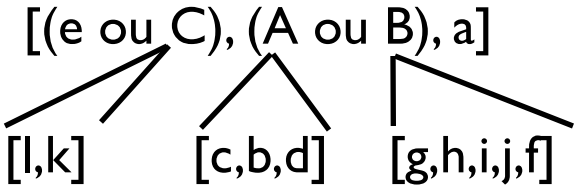


Algorithme de Bourdoncle

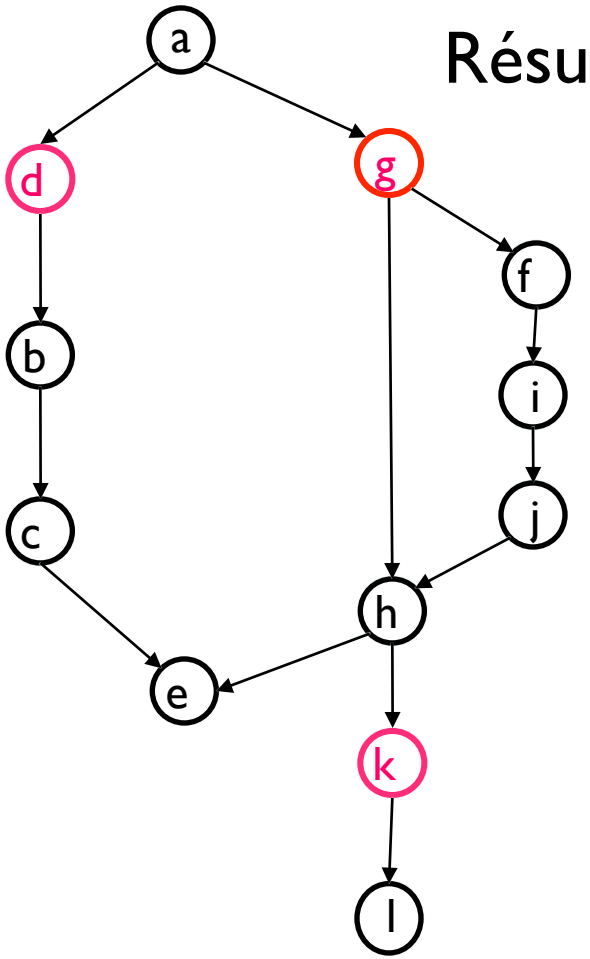
Noeud candidat = # max(fronds)



Casse les CFCs
réapplique Tarjan éventuellement



Une stratégie efficace pour l'ordre d'intégration



Résultat = un ordre partiel de toutes les stratégies possibles

Algorithme optimisé

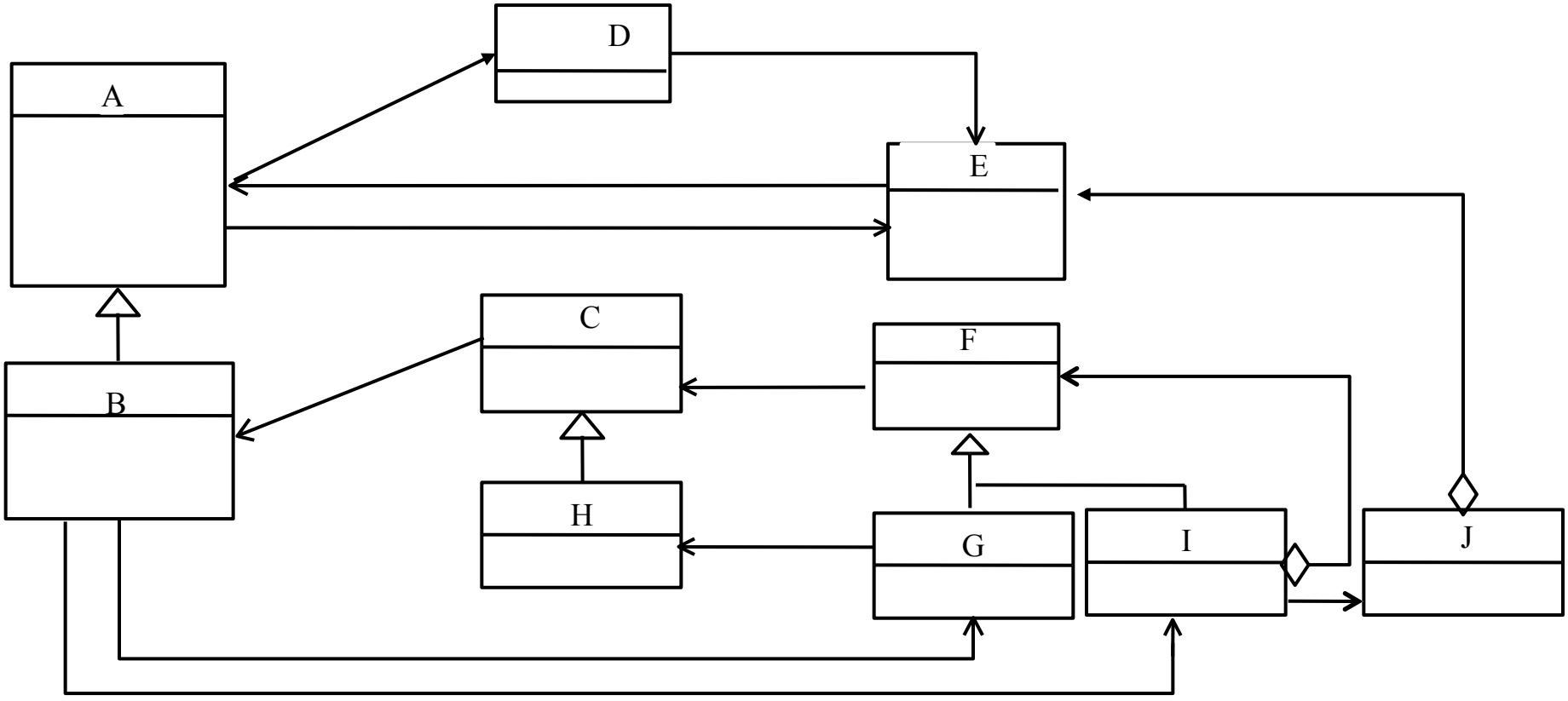
#stubs spécifiques = 4
#stubs réalistes = 3

Génération aléatoire

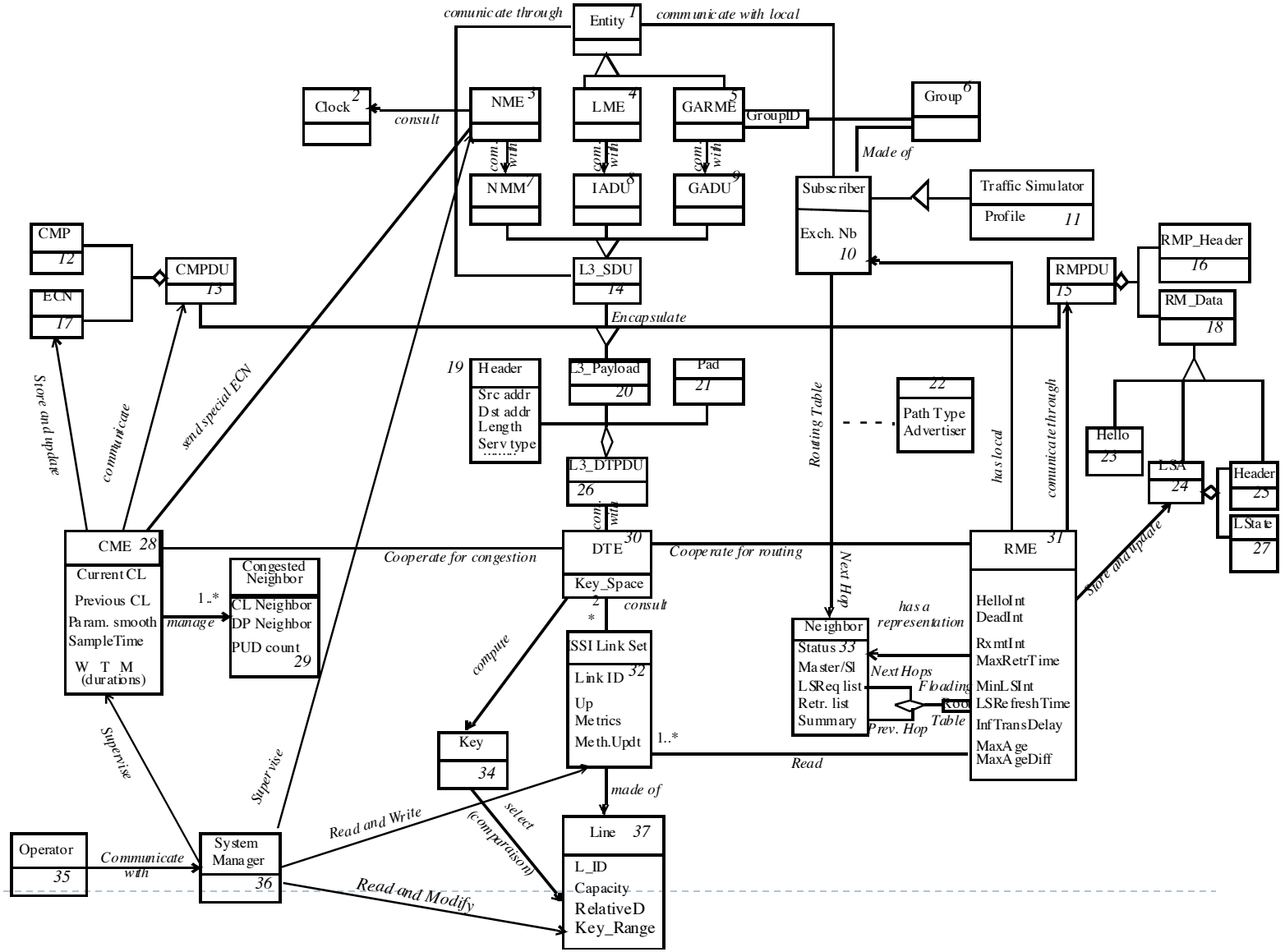
#stubs spécifiques = 9.9
#stubs réalistes = 5

Exo

- Plan de test d'intégration pour :

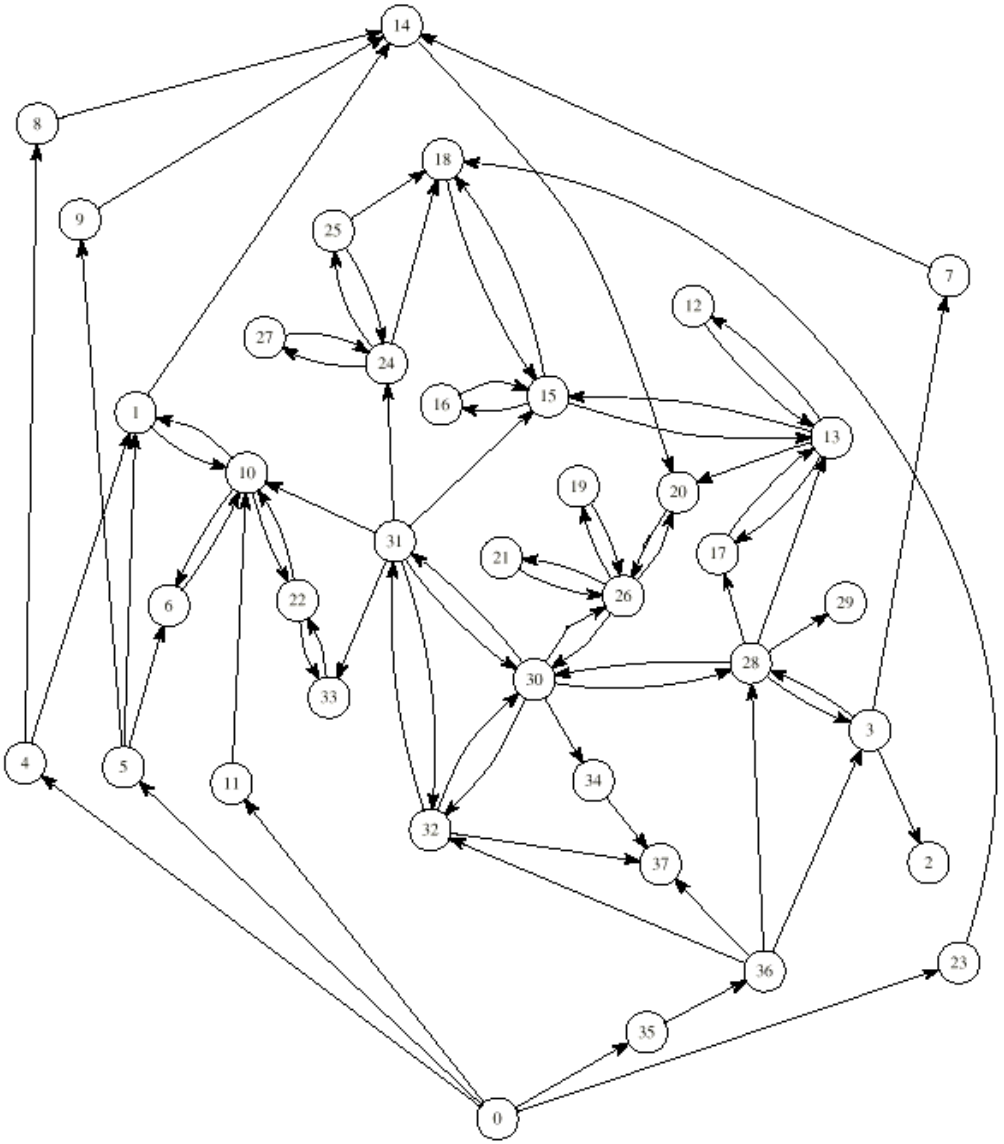


Cas encore moins simple : 37 classes



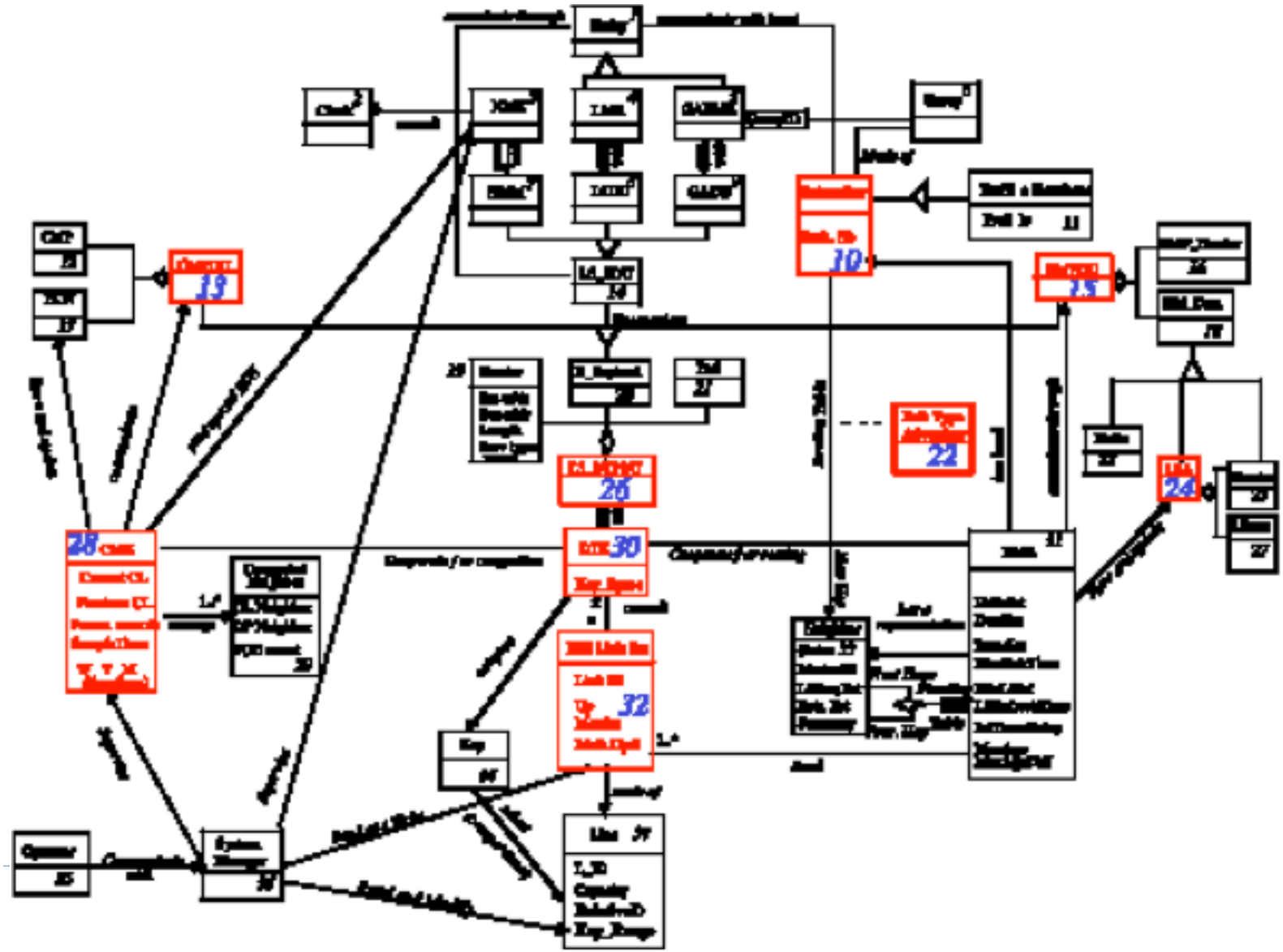
Graphe de dépendances

quelques cycles...



Bouchons de test

stubs réalistes

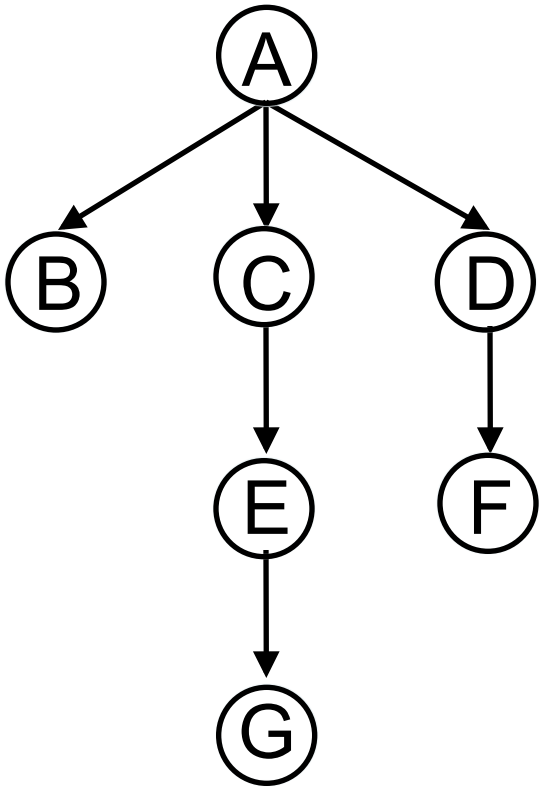


Une stratégie efficace pour l'ordre d'intégration

- Quand un ordre partiel est disponible, on peut paralléliser les tâches
  - en fonction d'un nombre fixe de testeurs
  - pour un délai minimum

Répartition des testeurs

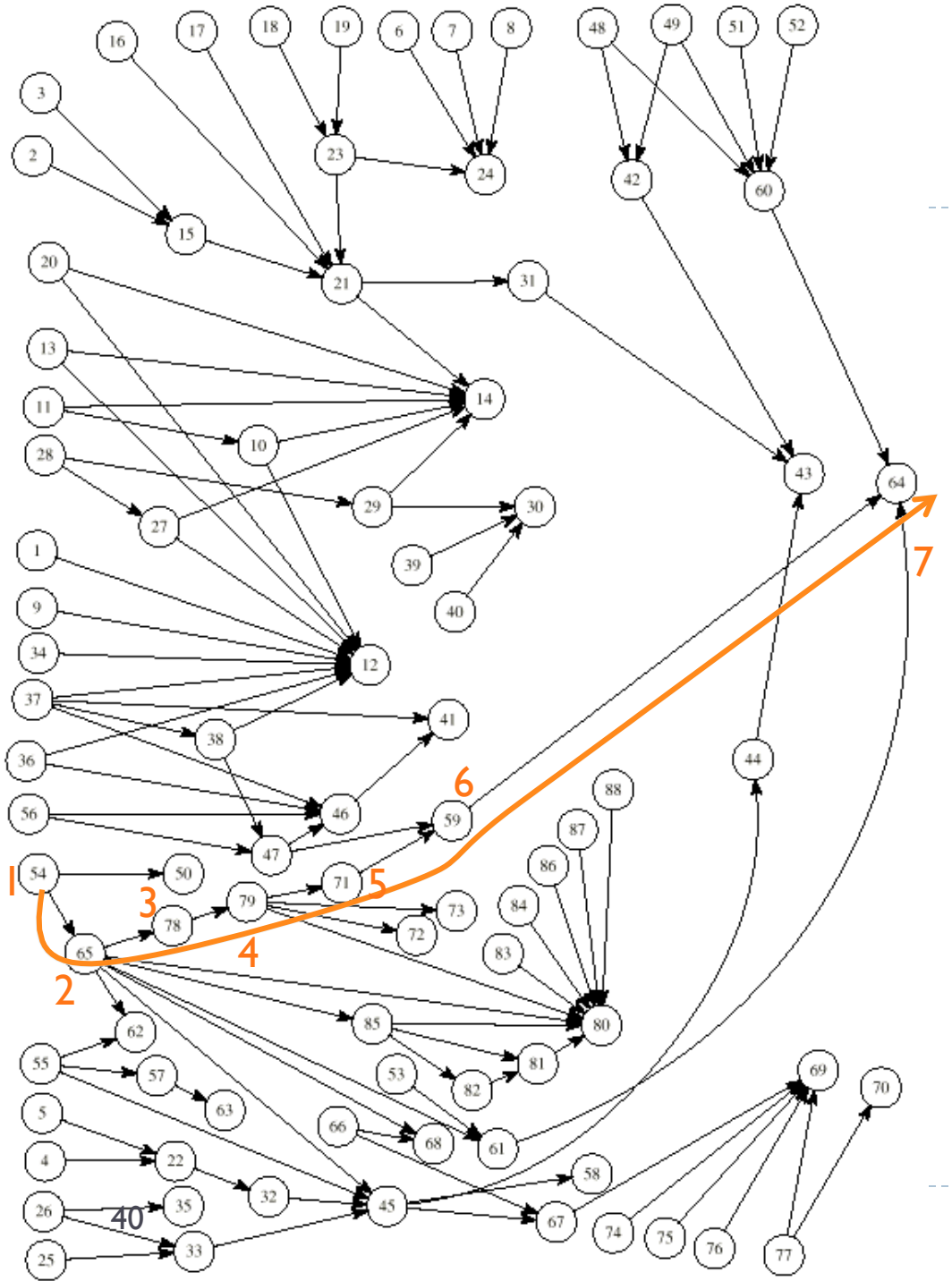
- 1 testeur : Les feuilles d'abord.
- n testeurs (n > 1)
  - Hypothèse
    - Le temps de test de toutes les unités est identique
  - Exemple n = 2
    - Un ordre possible :
      - (B, F), (D, G), E, C, A
    - Un ordre optimal :
      - (G, F), (B, E), (C, D), A
      - (B, G), (E, F), (C, D), A



Répartition des testeurs

- Propriété: min_étapes = max (A,B)
  - A = plus long chemin
  - B = (nb_noeuds / nb_testeurs) + 1
- Exemple
  - 4 testeurs et 37 noeuds; plus long chemin = 8
    - B = 10
  - min_étapes = 10

Répartition des testeurs



- Exemple de GNU-Eiffel
- Nombre de testeurs nécessaire pour intégrer en un minimum de temps
  - 88 div 7 +1 = 13 testers
- temps minimum : 7 étapes

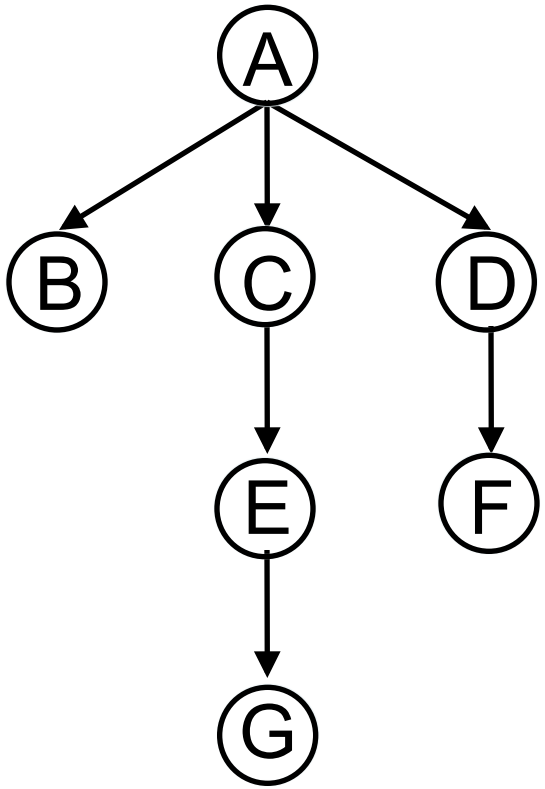
Répartition des testeurs

Component/node													Step
54	4	5	26	25	2	3	18	19	37	55	56	16	**1**
17	65	22	33	38	15	23	48	49	51	52	11	36	**2**
28	32	85	78	47	53	21	34	9	1	13	20	10	**3**
79	82	45	6	7	8	42	60	31	10	27	57	66	**4**
71	81	67	74	75	76	77	29	39	40	46	61	44	**5**
59	83	84	86	87	88	58	68	62	63	50	12	14	**6**
64	43	30	41	69	70	24	72	73	80				**7**

7 steps = Optimum delay

Répartition des testeurs

- Branches :
  - A-B
  - A-C-E-G
  - A-D-F
- Chemin critique
  - A- C-E-G
- Répartition totale
  - #testeurs = #branches
  - temps_de_test = temps_de_test_de_chemin_critique



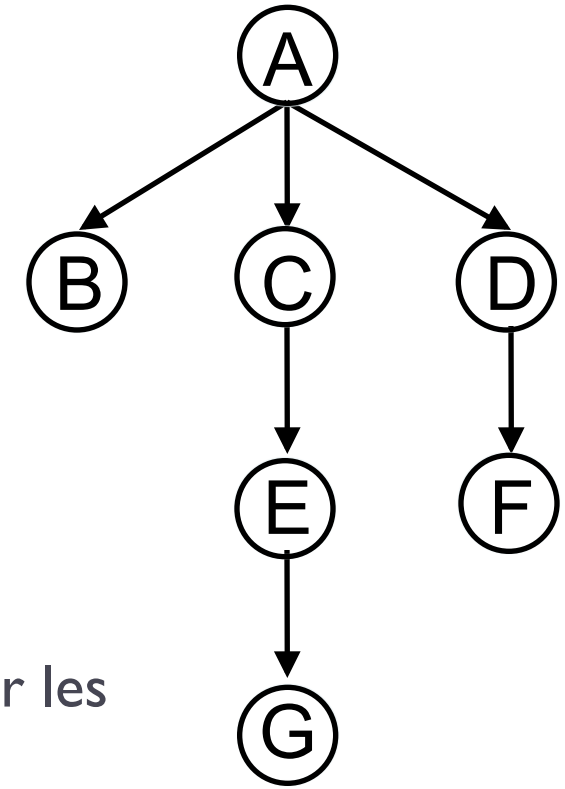
Répartition des testeurs

- Condition :
  - #testeurs < #branches
- Répartition :

  Choisir la feuille d'un chemin critique.

  Exemple avec 2 testeurs :
  - 1er : G E C A
  - 2nd : F B D
- Avantages :

  Profiter du temps libre d'un testeur sur les branches non-critiques



Répartition des testeurs

Une solution optimale pour l'exemple du SMDS

Composants				Étape	Testeur
20	33	18	6	1	4
14	22	27	16	2	4
1	25	7	17	3	4
24	12	2	3	4	4
15	13	29	37	5	4
31	10	34	28	6	4
21	19	9	32	7	4
8	30	36	5	8	4
26	35	4	11	9	4
23				10	1

37 nœuds, 4 testeurs $\Rightarrow$ (minimum = 10)

Une stratégie efficace pour l'ordre d'intégration

- Stratégie efficace

  casse les cycles de dépendances avec un minimum de stubs

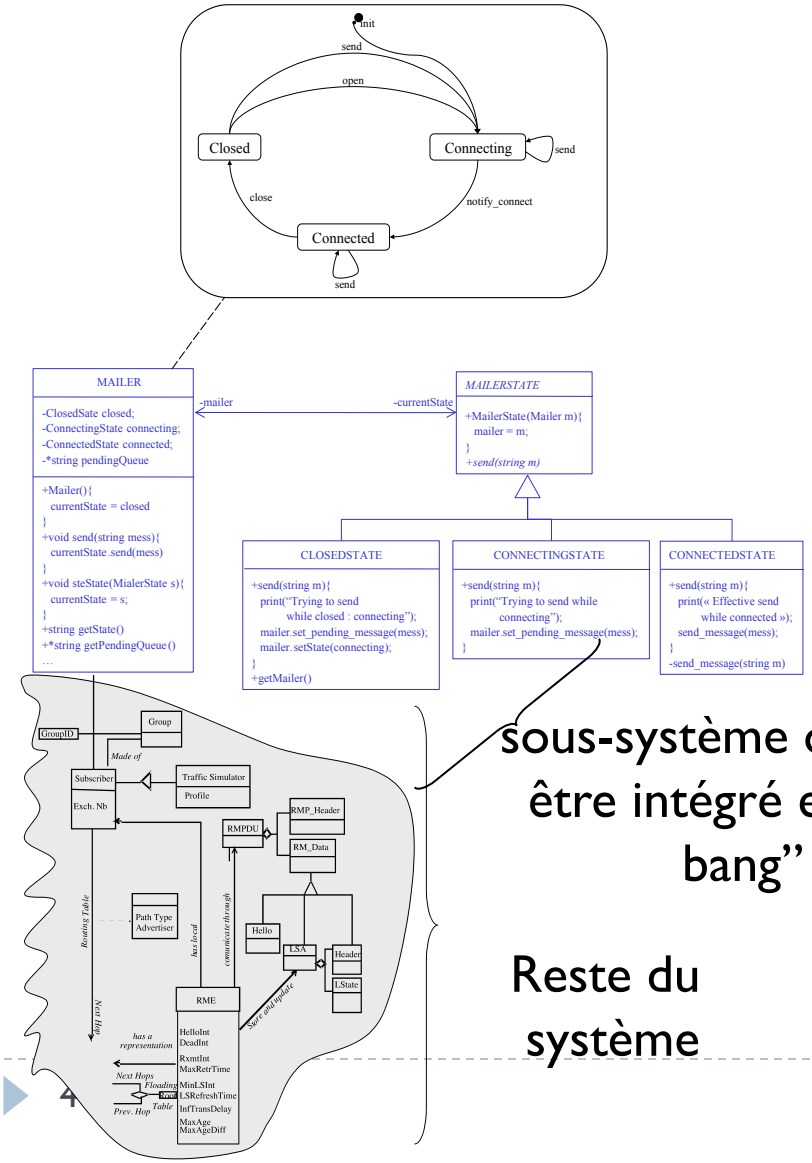
- Autre stratégie

  prend en compte les pratique de conception OO

  certains cycles sont très cohérents du point de vue fonctionnel (Ex: design patterns)

  - ça peut être intéressant d'intégrer cette interdépendance d'un coup

Stratégie mixte



sous-système qui peut être intégré en "big bang"

Reste du système

- Minimise encore le nombre de stubs
- Maintient un niveau de cohérence dans l'intégration
- Pas complètement automatisable
  - pattern matching