Des langages d'assemblages dans les modèles à composants logiciels

Christian Perez

Avalon, LIP, Lyon, France

christian.perez@inria.fr

Journée Langages @ LIP, Lyon
17 mars 2016

Contenu

- Du besoin de composants logiciels
- Des composants logiciels
  - Concepts de base via le diner des philosophes
  - Langage de description d'assemblage
  - Des assemblages plus abstraits
    - Hiérarchie, généricité, squelettes algorithmiques
  - Notion de connecteurs
- Composants logiciels et gestion de ressources
  - Exemple via un cas d'utilisation « cloud »
- Conclusion

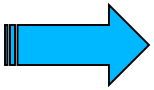
De la réalisation d'une application

Développement sur mesure

- Avantage
  - Très spécifique, très bien adapté …
- Inconvénients
  - … quand cela marche
  - Très coûteux en temps
  - Solution sous-optimale par manque de spécialistes
  - Manque d'interopérabilité
- Exemples
  - Logiciels pour les administrations ou les grands comptes

Logiciels standard

- Avantages
  - Temps de mise sur le marché plus court
  - Juste à configurer
  - Produit acquis
    - maintenance, évolution à la charge du « vendeur »
- Inconvénients
  - Réorganisation locale du « business » (adaptation)
  - Les concurrents ont aussi accès aux fonctionnalités
  - Adaptabilité aux besoins limitée
- Exemple
  - Excel

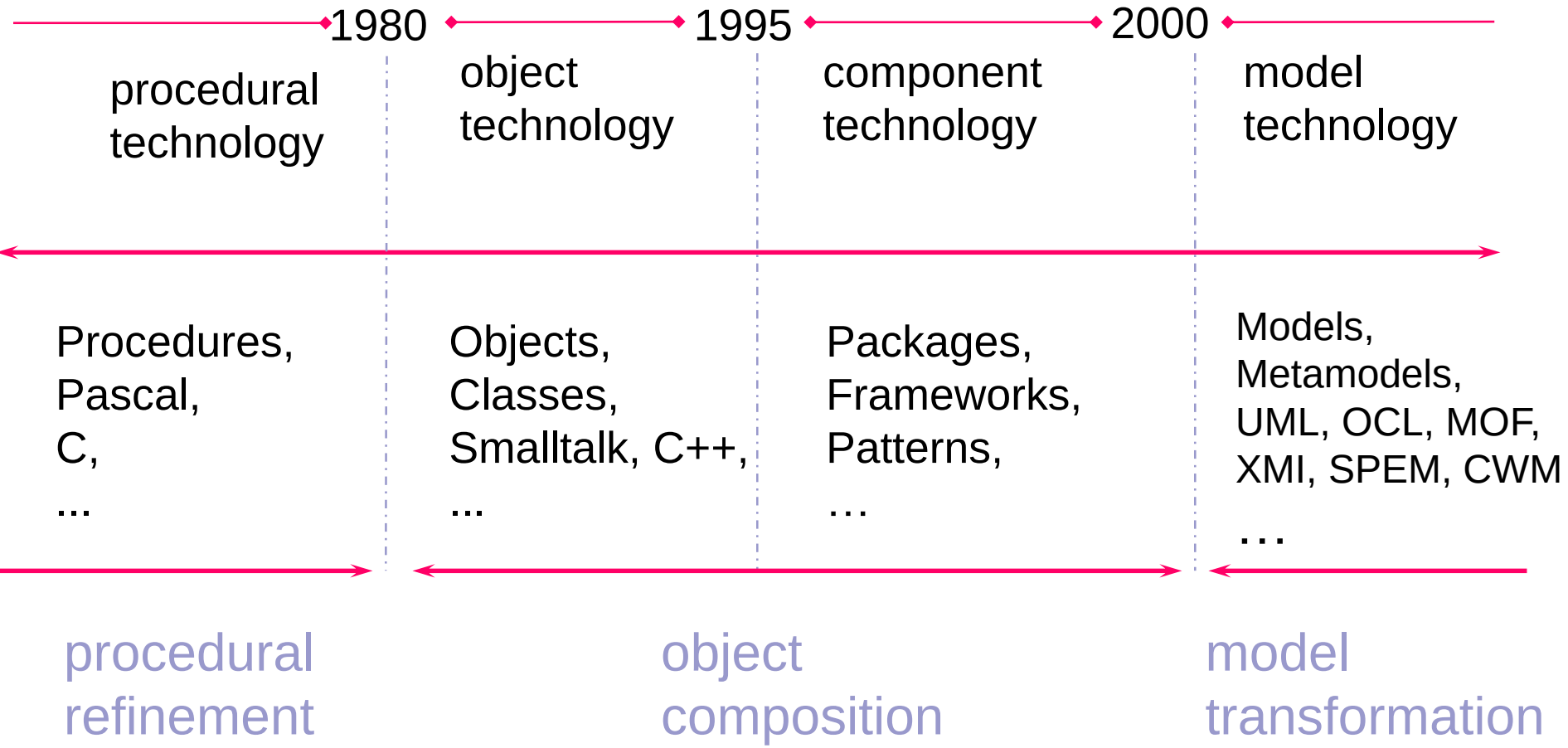


Acquérir ce qui est disponible et développer ce qui est spécifique

Le chaînon manquant ?

- La notion de composant est très largement répandue...
  - Composants électroniques
  - Composants d'une voiture (depuis Henry FORD)
  - Composants d'un meuble, d'une construction
  - On fait même de la cuisine « d'assemblage »
- Des composants partout ... sauf en informatique !
  - On reste ainsi au stade artisanal …
- Et pourtant …c'est un ancien rêve de l'informatique
  - Les premiers travaux datent de la fin des années 60

A global view of software engineering evolution

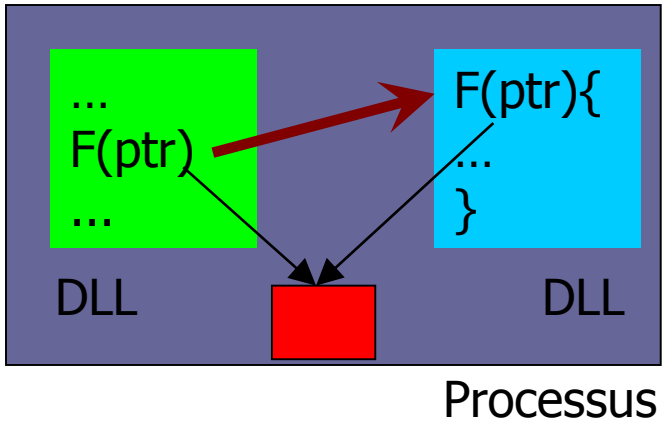


From Jean Bézivin presentation, ATLAS group (Inria & LINA), Nantes, France

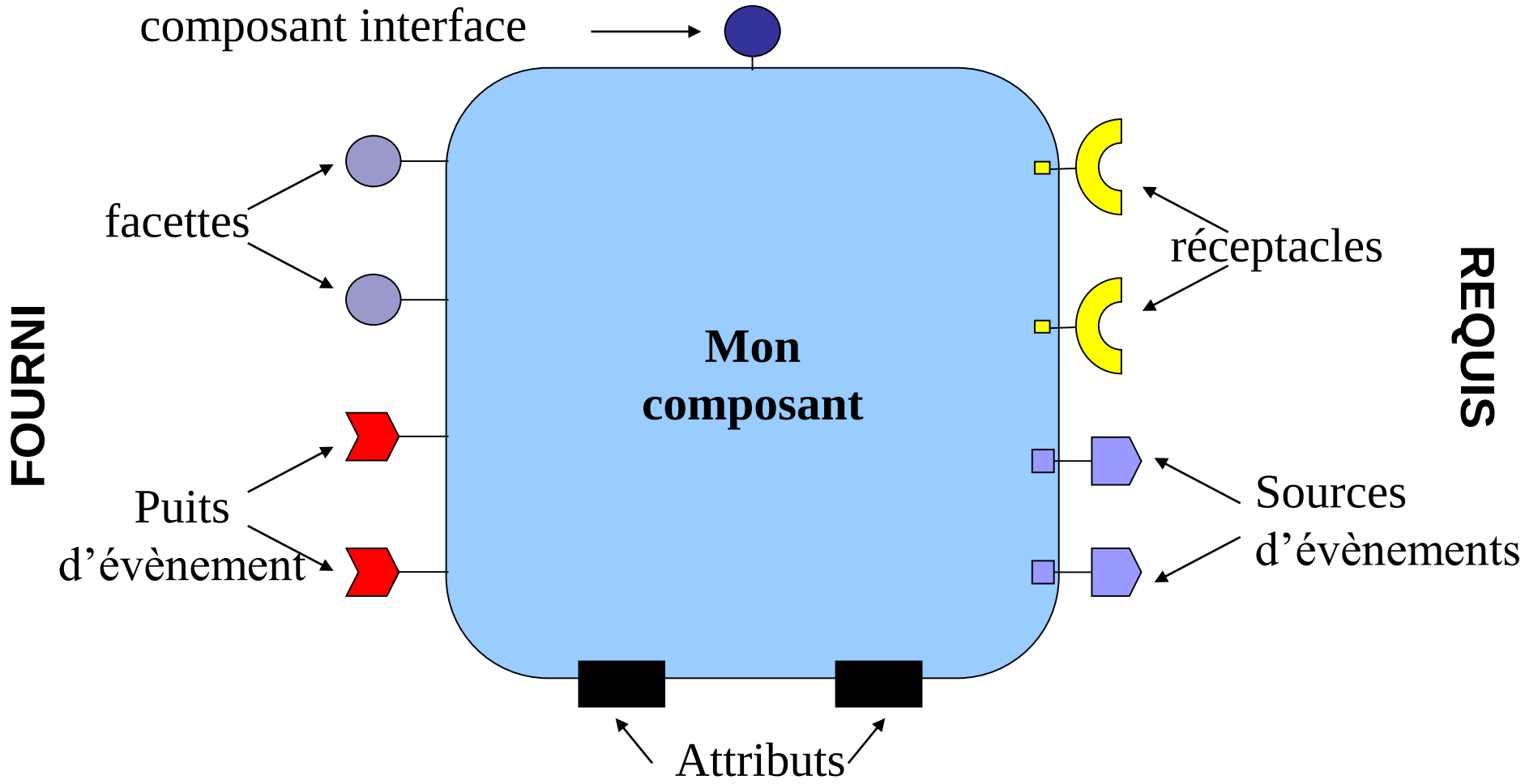
NOTION DE COMPOSANTS LOGICIELS

(Une) définition d'un composant logiciel

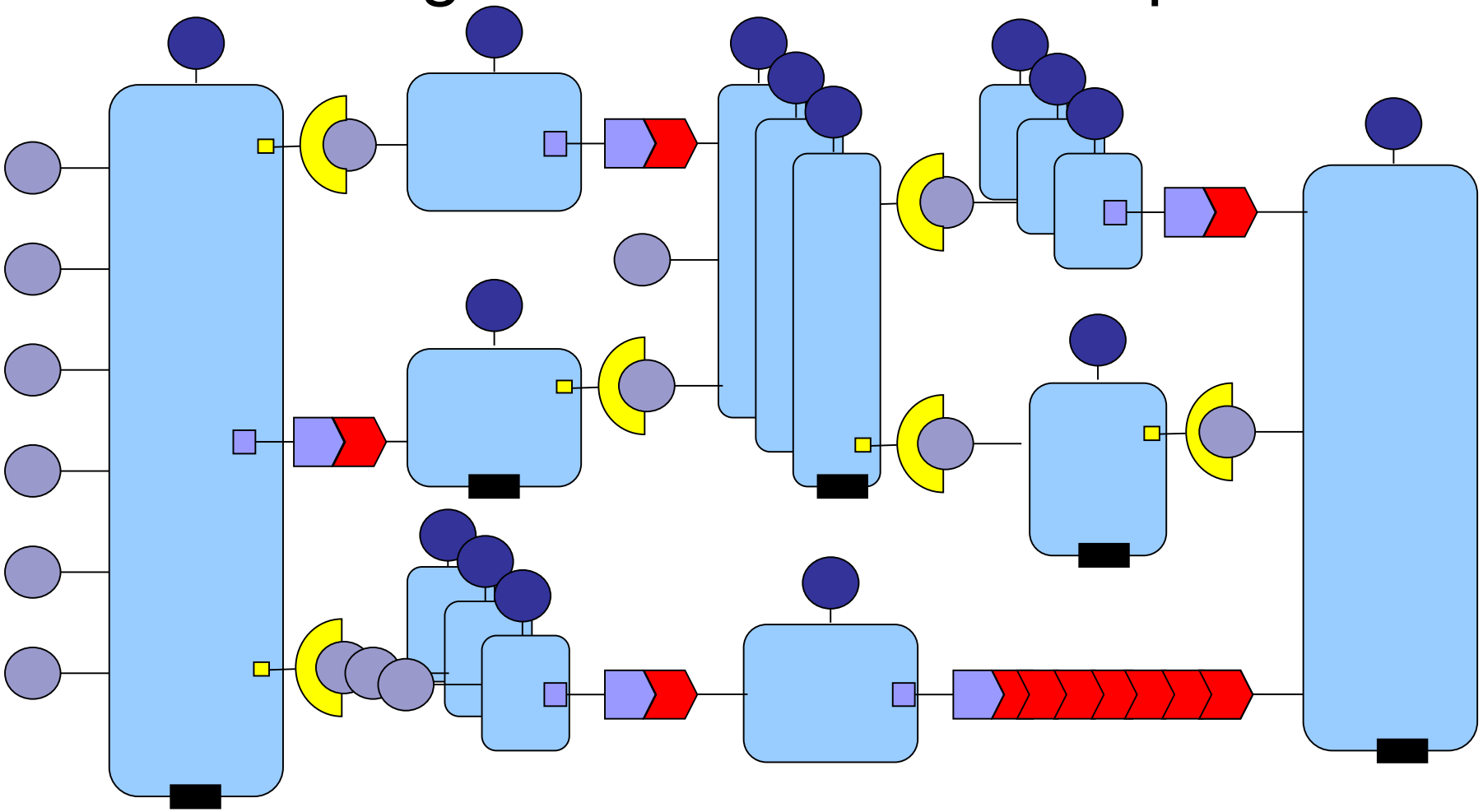
- *Un composant logiciel est une **unité de composition** qui a , par **contrat**, spécifié uniquement ses **interfaces** et ses **dépendances** explicites de contextes. Un composant logiciel peut être **déployé** indépendamment et est sujet à **composition** par des tierces entités.*
  - Définition de Szyperski et Pfister, 1997.
- Exemple
  - « Composants Logiciels »
    - fonctions dans une DLL
  - Canevas
    - un processus
  - Supports d'interaction
    - registre/mémoire
- Contre exemple: les macros en C



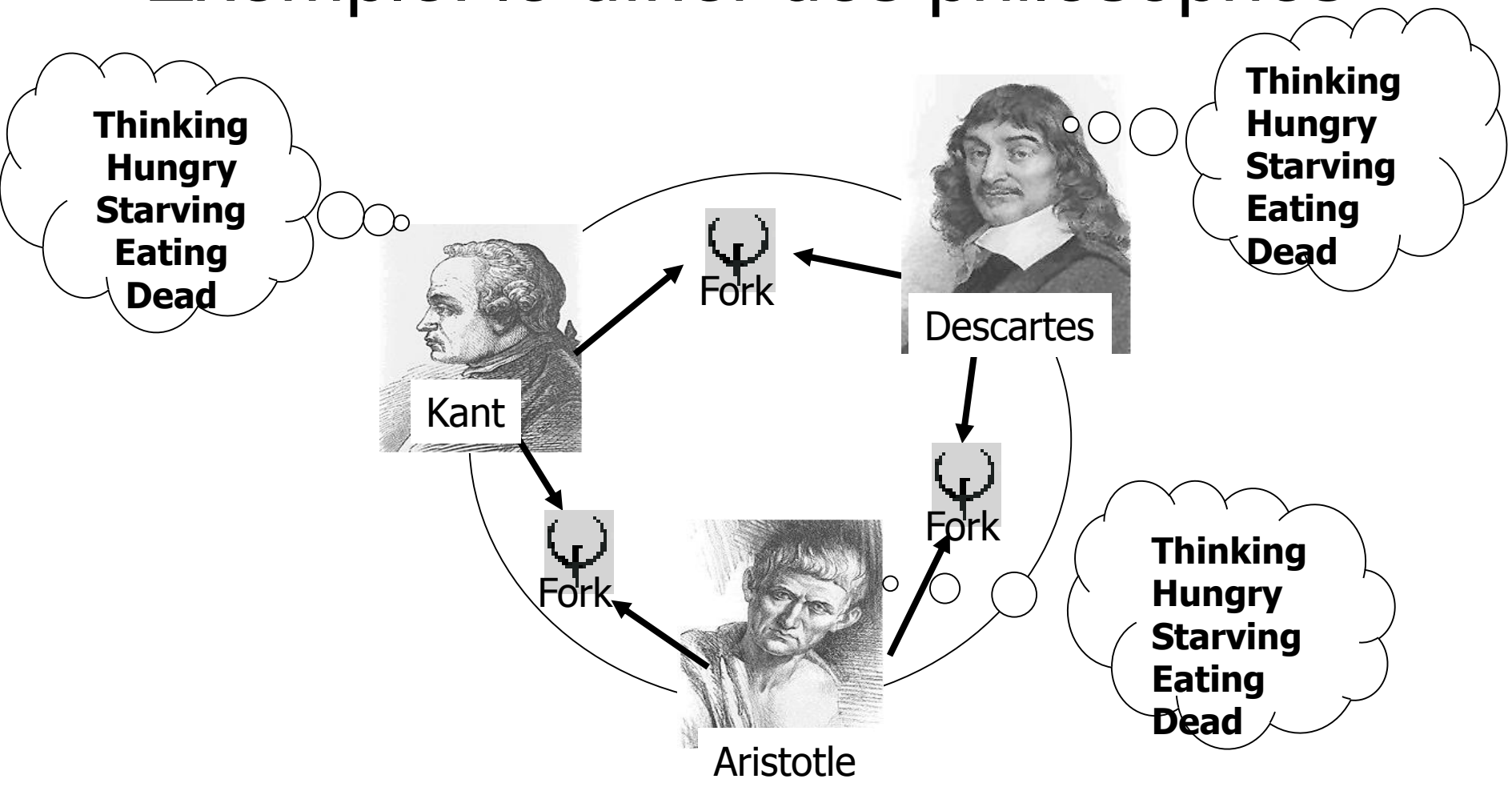
Un composant CORBA



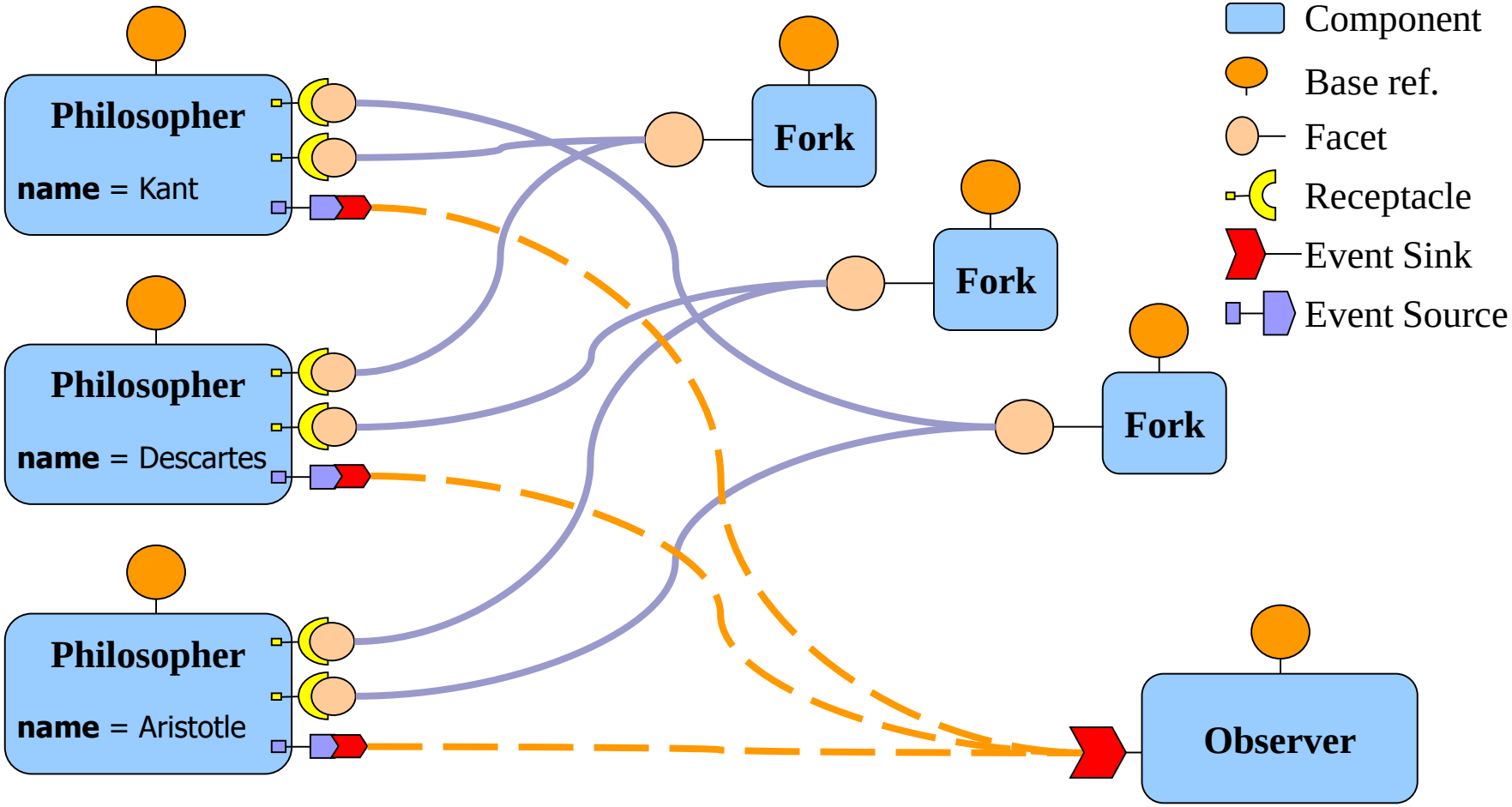
Construction d'une application CCM = Assemblage d'instances de composants



Exemple: le dîner des philosophes



Le dîner des philosophes (CCM)

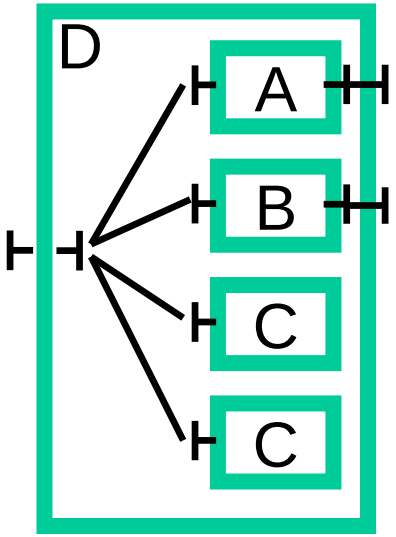


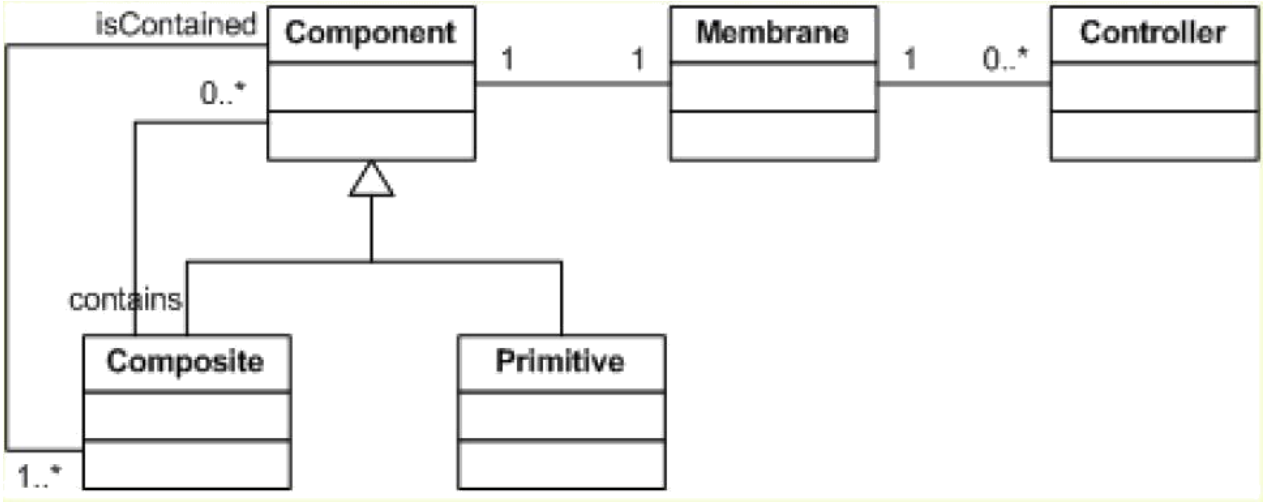
Langage de description d'assemblage (ADL)

- Les ADL permettent de décrire une telle structure
  - Les ADL ne sont pas restreints aux composants
- Description d'une sorte de graphe
  - Composant: nœud avec des points de connections nommées (les ports)
  - Connection: lien entre 2 ports

Le modèle FRACTAL

- Modèle hiérarchique
  - Composants primitives
  - Composants composites
- Modèle à membrane
  - Emplacement pour les contrôleurs (composants)





Fractal ADL

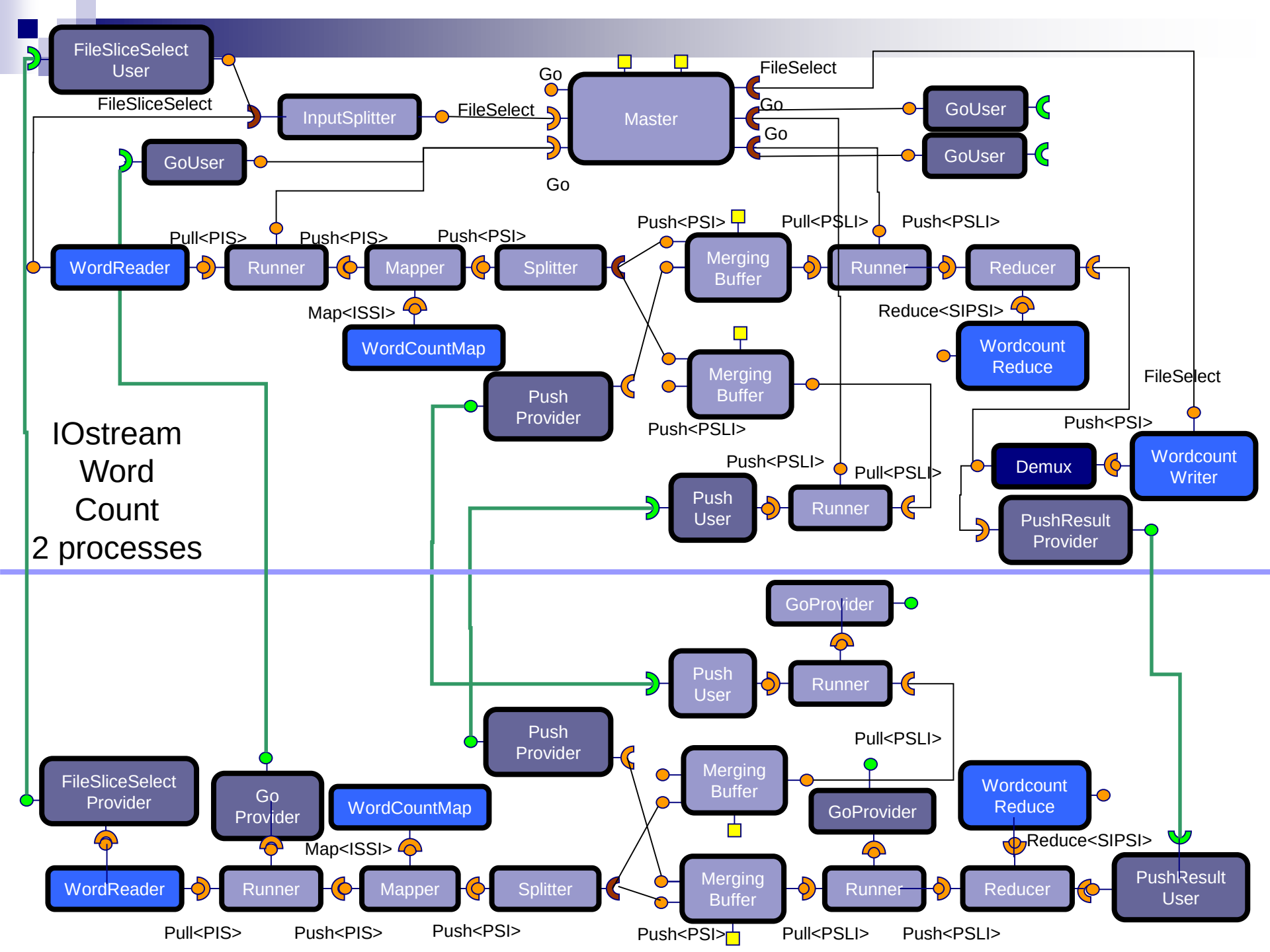
- Primitive component

```
<definition name="Client">
  <interface name="r" role="server"
             signature="java.lang.Runnable"/>
  <interface name="s" role="client" signature="Service"/>
  <content class="ClientImpl"/>
</definition>
```

Fractal ADL

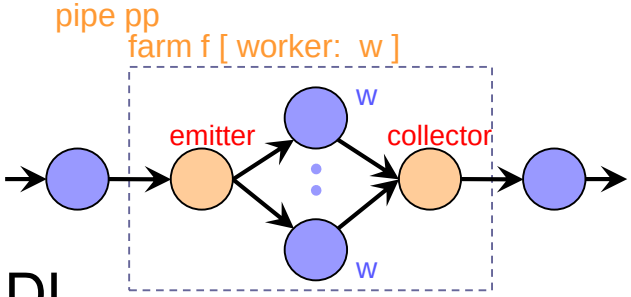
- Composite component

```
<definition name="HelloWorld">
  <interface name="r" role="server"
             signature="java.lang.Runnable"/>
  <component name="client">
    <interface name="r" role="server"
               signature="java.lang.Runnable"/>
    <interface name="s" role="client" signature="Service"/>
    <content class="ClientImpl"/>
  </component>
  <component name="server">
    <interface name="s" role="server" signature="Service"/>
    <content class="ServerImpl"/>
  </component>
  <binding client="this.r" server="client.r"/>
  <binding client="client.s" server="server.s"/>
</definition>
```


FileSliceSelect User
FileSliceSelect
InputSplitter
FileSelect
Go
Master
FileSelect
Go
Go
GoUser
GoUser
GoUser
Go
WordReader
Pull<PIS>
Runner
Push<PIS>
Mapper
Push<PSI>
Splitter
Push<PSI>
Merging Buffer
Pull<PSLI>
Runner
Push<PSLI>
Reducer
Map<ISSI>
WordCountMap
Reduce<SIPSI>
Wordcount Reduce
Push Provider
Merging Buffer
Push<PSLI>
FileSelect
Push<PSI>
Wordcount Writer
Demux
Push<PSLI>
Pull<PSLI>
Push User
Runner
PushResult Provider
IOstream
Word
Count
2 processes
GoProvider
Push User
Runner
Push Provider
Pull<PSLI>
FileSliceSelect Provider
Go Provider
WordCountMap
Merging Buffer
GoProvider
Wordcount Reduce
Map<ISSI>
Reduce<SIPSI>
WordReader
Runner
Mapper
Splitter
Merging Buffer
Runner
Reducer
PushResult User
Pull<PIS>
Push<PIS>
Push<PSI>
Push<PSI>
Pull<PSLI>
Push<PSLI>

Vers des ADL plus abstraits

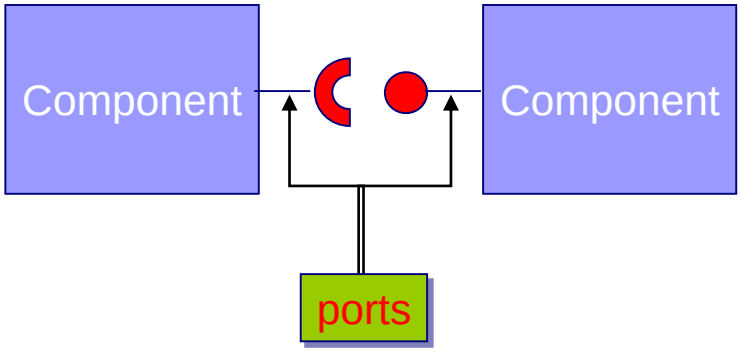
- Support de la généricité (à la C++)
  - Introduit au niveau du composite
    - aComposite<2, «toto », anInterface, aComponent>
- Squelettes algorithmiques
  - Pipe, farm, etc.
  - Introduit comme construction d'un ADL
    - Par exemple, via un méta-type Farm au lieu de meta-type Component (Spécialisation)



- Transformation en un ADL « simple » possible

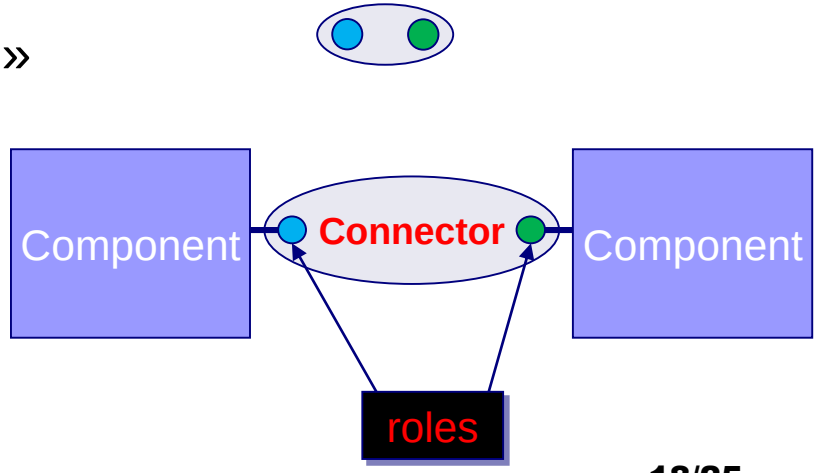
Des liens aux connecteurs

- Modèle de liens
  - Connection directe entre ports via les modèles d'interactions fournis par le modèle de composant



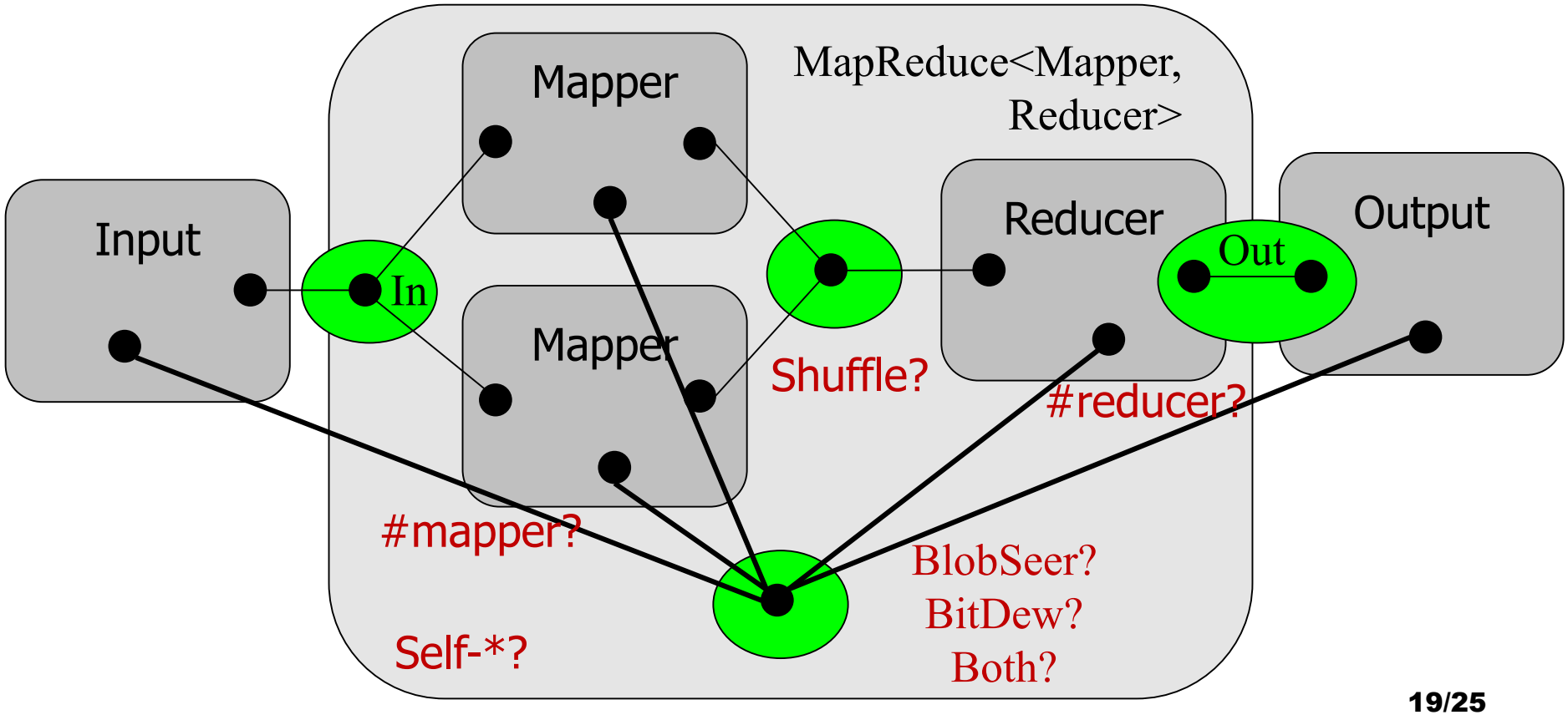
- Modèle à connecteurs
  - Réification des connections
    - Un nom
    - Un ensemble de rôles
  - Connecteur primitive/«composite »
    - Fourni par le modèle à l'exécution
    - Définissable par l'utilisateur
  - Exemple
    - Consensus, Data sharing
    - MxN communications

connector UseProvide
< role user, role provider >;



Vers un squelette MapReduce

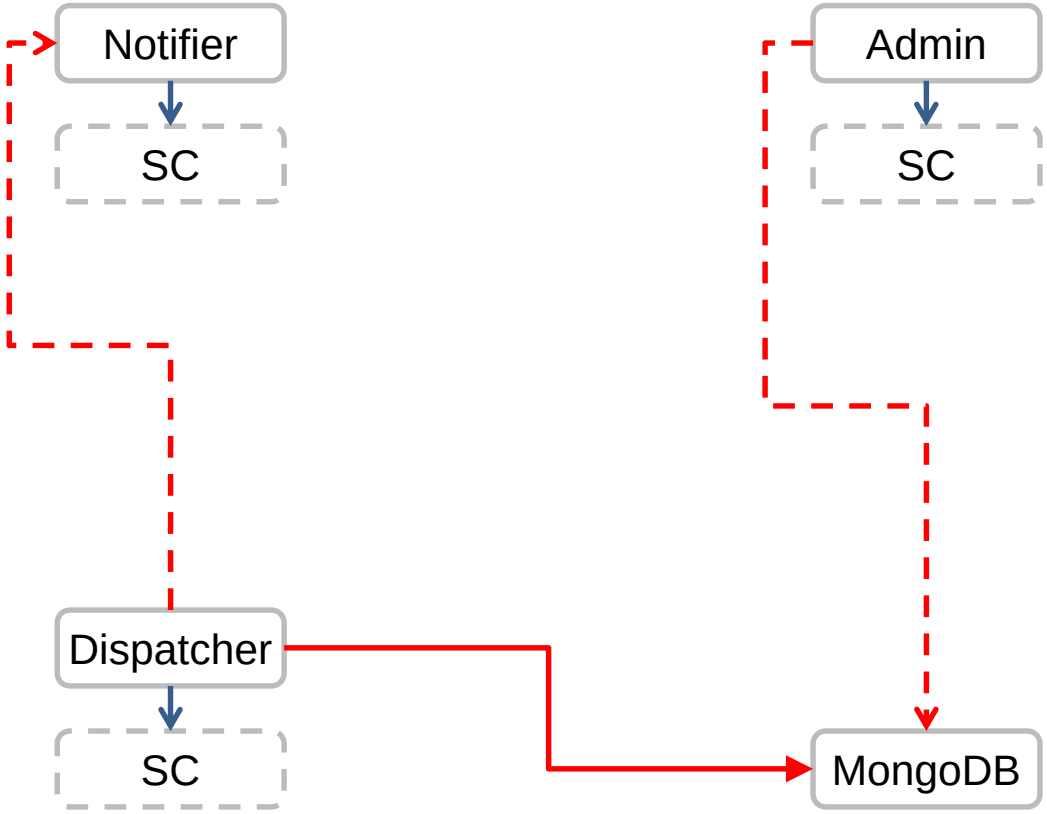
Component MapReduce<Component Map, Component Reduce>
exposes { In, Out}



DESCRIPTION D'APPLICATIONS ET DE RESSOURCES « CLOUD »

MultiCloud application deployment and provisioning
(cloudml.org)

CloudML abstract model (CPIM)

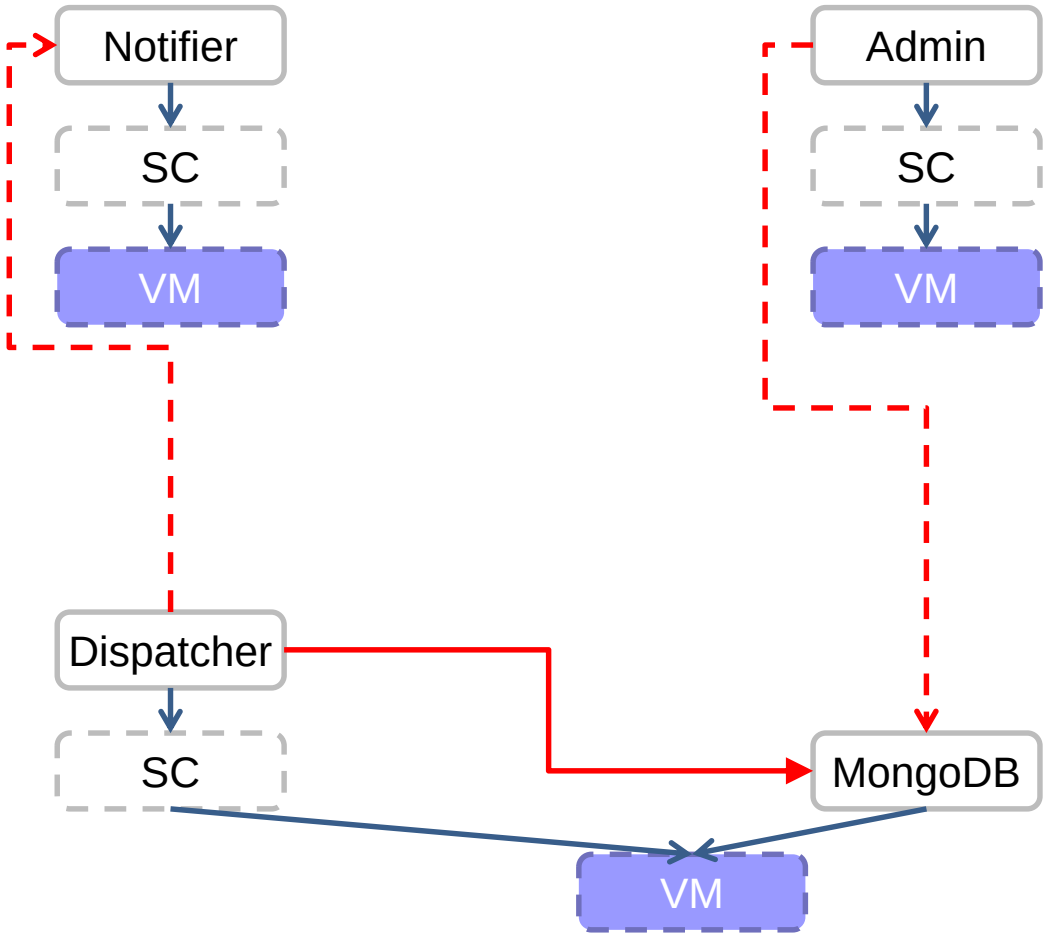


VM: Virtual machine
SC: Servlet container

MultiCloud application deployment and provisioning

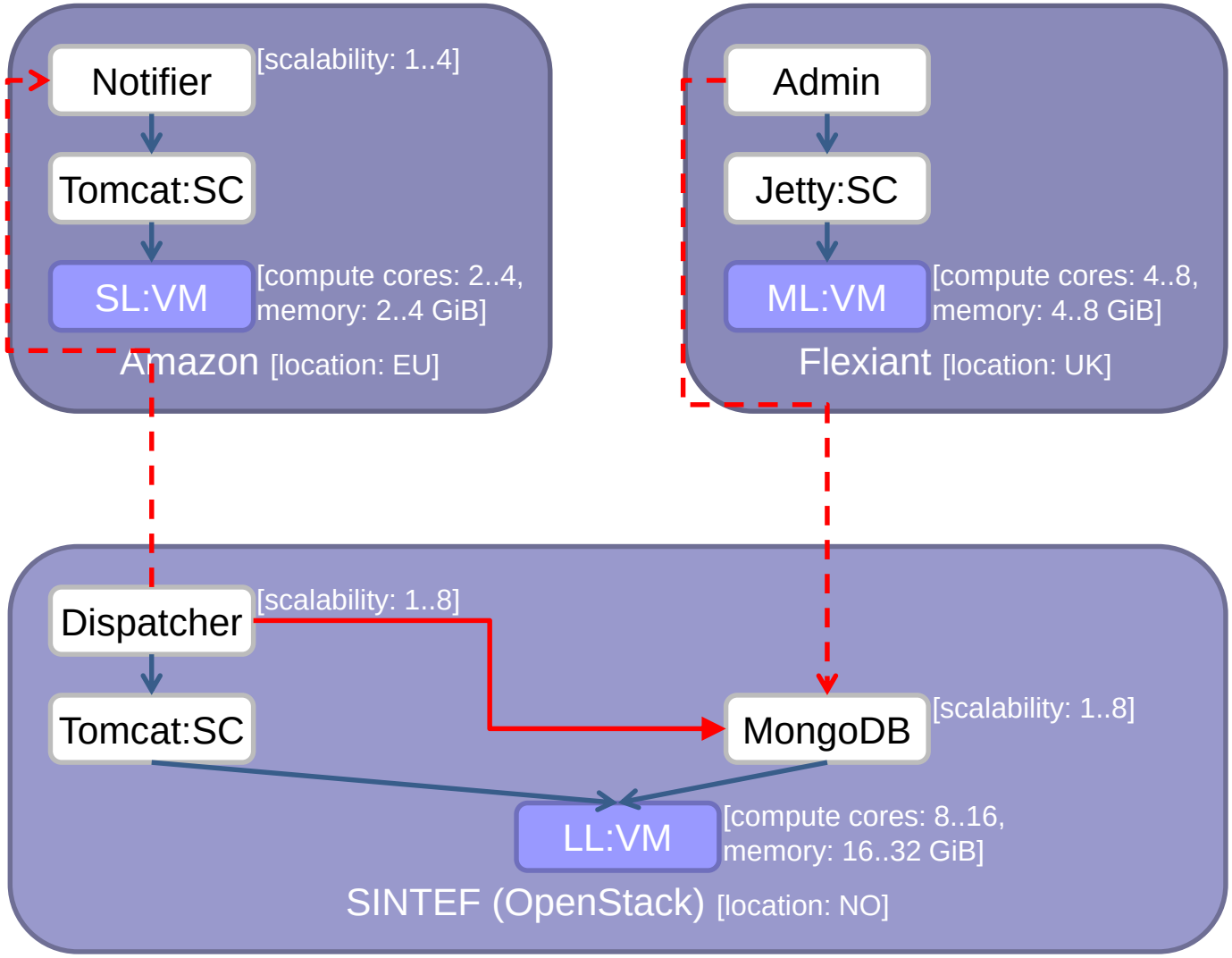
(cloudml.org)

CloudML abstract model (CPIM)



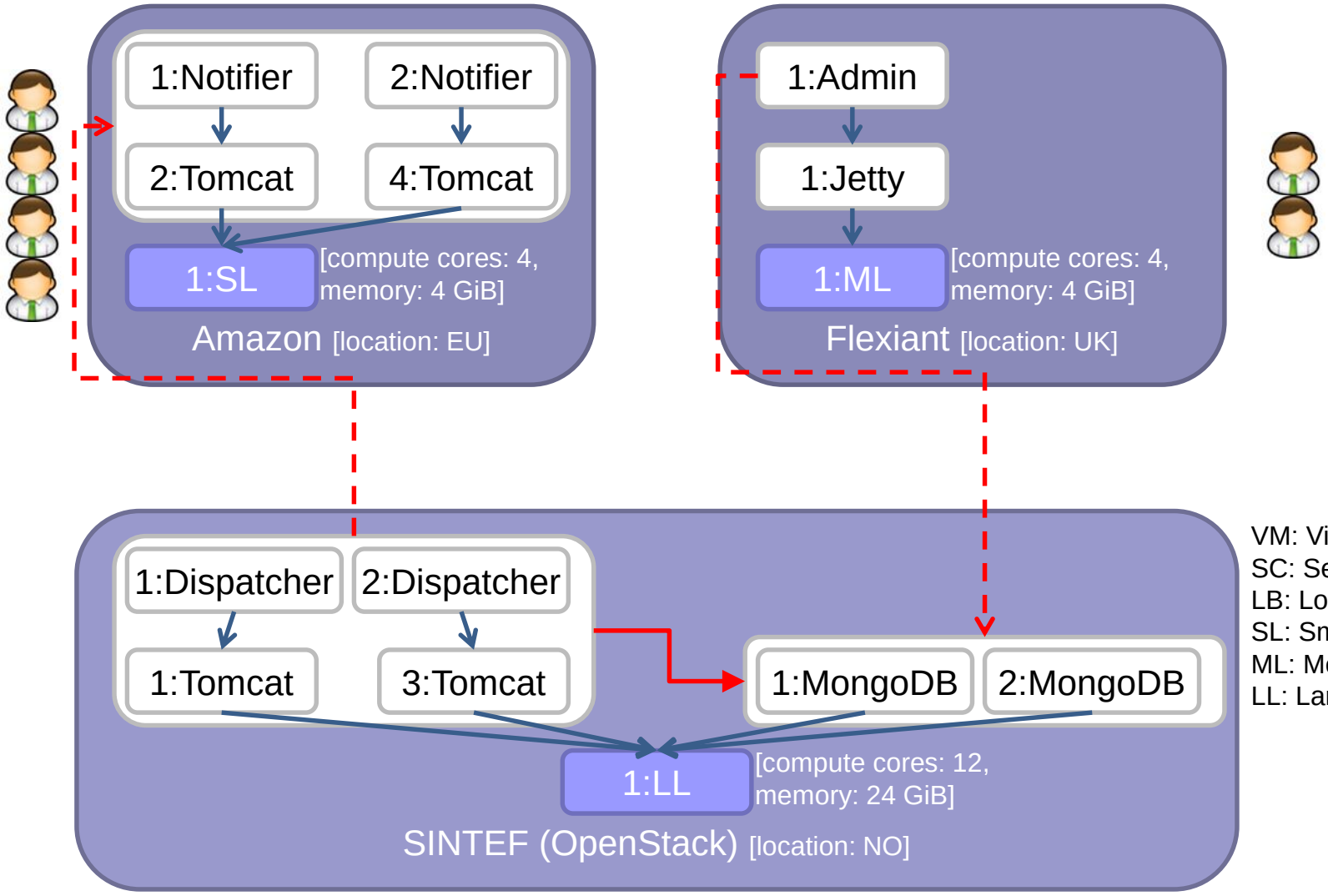
VM: Virtual machine
SC: Servlet container

Adding platform specific constraints (CPSM)



VM: Virtual machine
SC: Servlet container
LB: Load balancer
SL: Small GNU/Linux
ML: Medium GNU/Linux
LL: Large GNU/Linux

Snapshot CloudML@run-time



VM: Virtual machine
SC: Servlet container
LB: Load balancer
SL: Small GNU/Linux
ML: Medium GNU/Linux
LL: Large GNU/Linux

Conclusion

- Motivation pour les modèles à composant
  - Faciliter la ré-utilisation du code
    - Gestion des dépendances
  - Exprimer/manipuler la structure de l'application
    - Via des ADL et/ou des interfaces d'introspection
- Langage de description d'assemblage
  - De simples descriptions de graphes
  - à des modèles plus abstrait
    - Hiérarchie, généricité, squelette algorithmique, etc
- De la transformation d'ADL pour la gestion de ressources
  - Statique
    - Calculer un déploiement pour cloud
  - Dynamique / reconfiguration
- ADL et modèle de data/workflow (grain fin & gros)