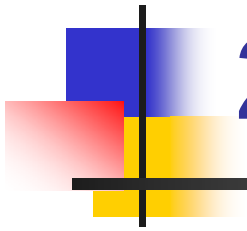


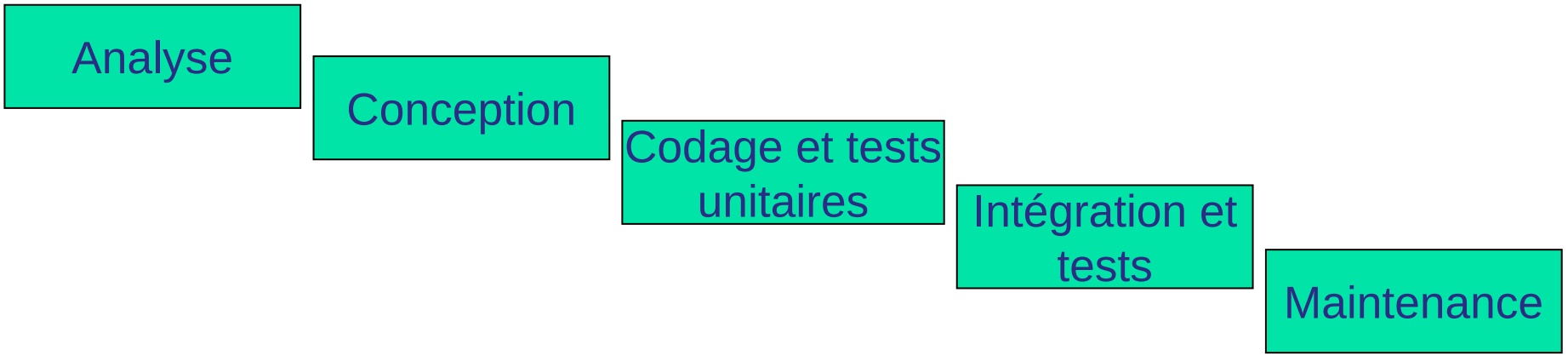
GL - 2
2.4 Architecture logicielle

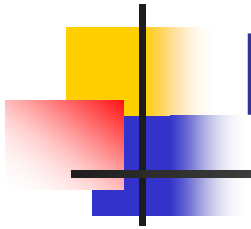
Lydie du Bousquet
Lydie.du-bousquet@imag.fr

En collaboration avec Ph. Lalanda

Activités logicielles

- Analyse : récolte des exigences
- Comment commencer la conception ?



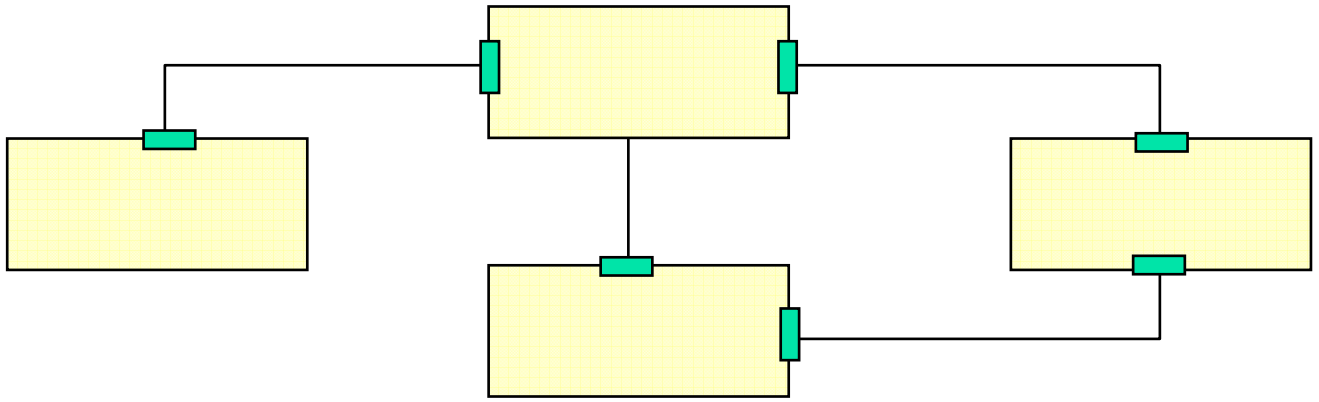


Plan

- Notion d’architecture
- Intérêt de l’architecture
- Représentation
- Conclusion

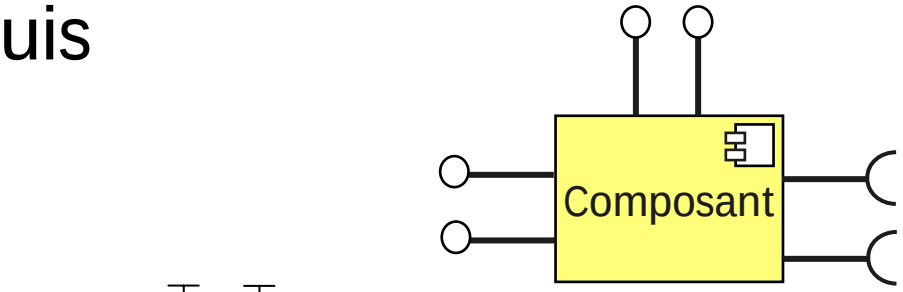
Définition de l’architecture

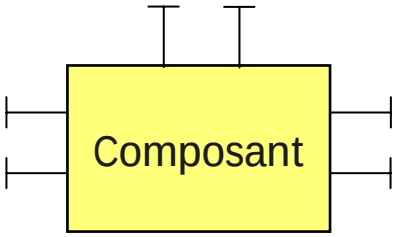
- Une architecture logicielle est une représentation abstraite d'un système exprimée essentiellement à l’aide de composants logiciels en interaction via des connecteurs



Composants logiciels

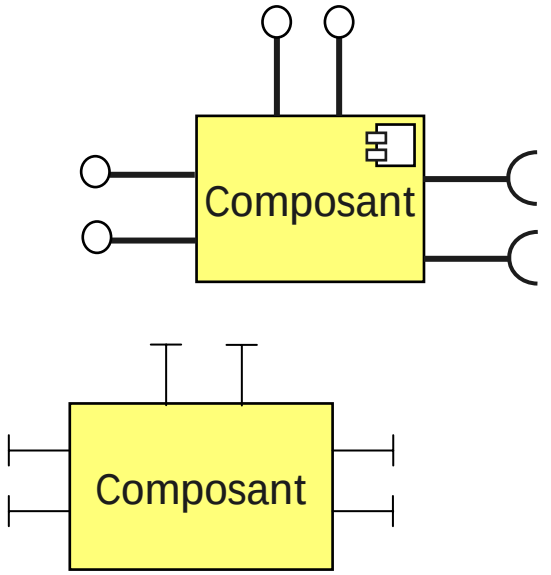
- Les composants sont des spécifications d'unités fonctionnelles
  - clairement définies
  - sémantiquement cohérentes et compréhensibles
- Développés ou acquis
- Ne pas confondre
  - spécification
  - réalisation





Description des composants

- Propriétés fonctionnelles
  - Services requis
  - Services fournis
  - Cycle de vie
- Contraintes
  - Type de communication
  - Ordonnancement
- Propriétés non fonctionnelles
  - Performance, robustesse, ...

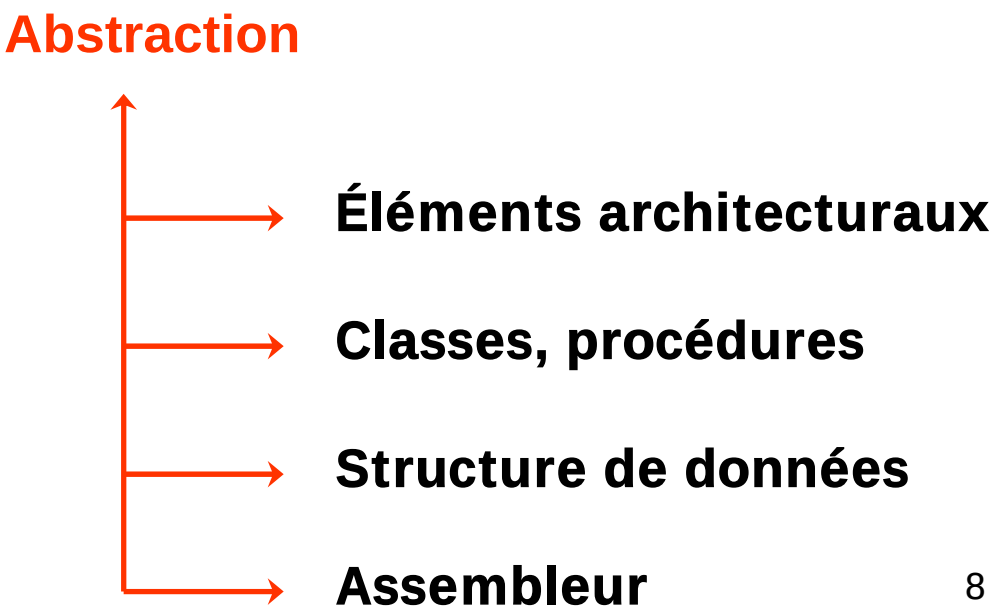


Connecteurs

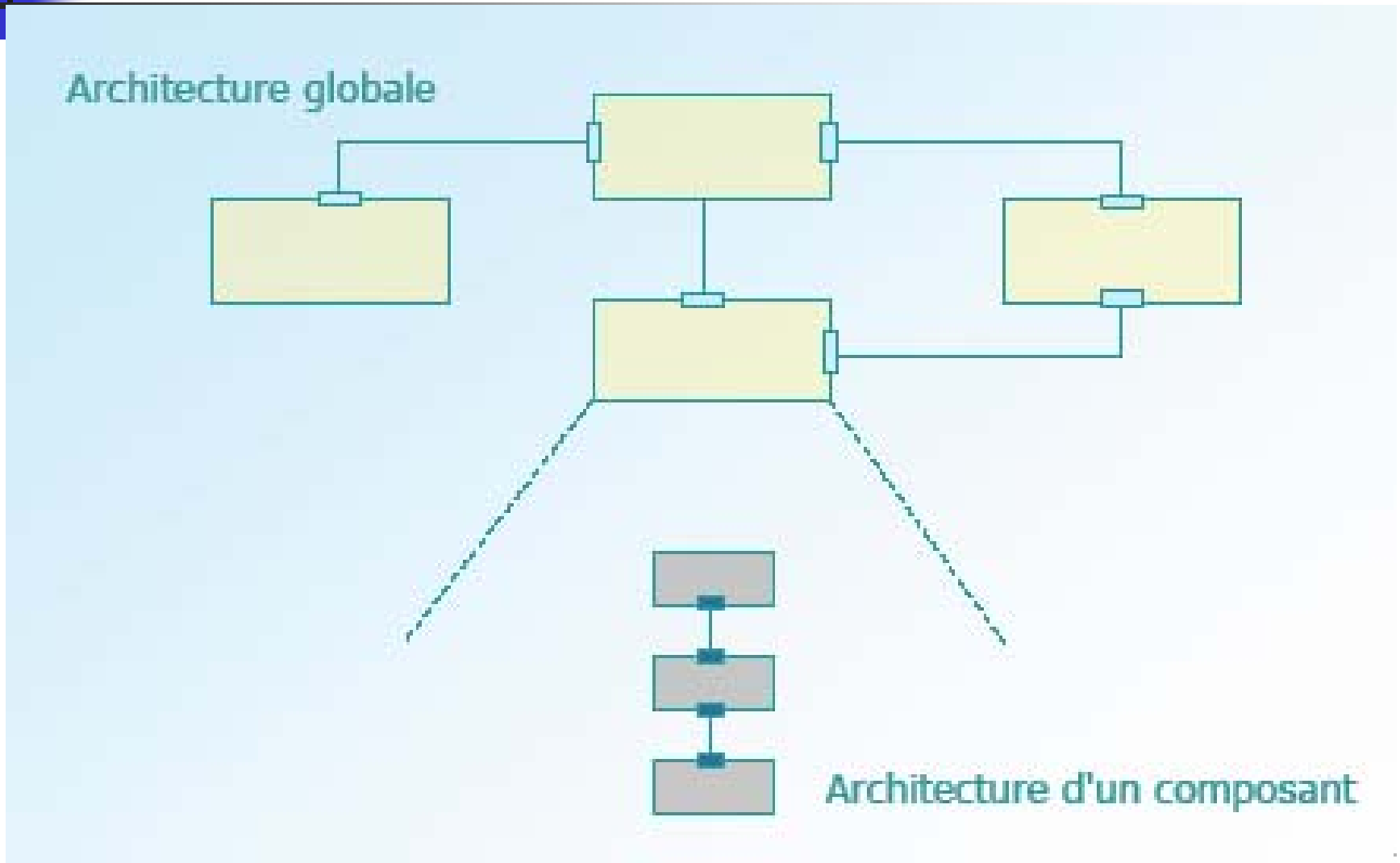
- Ce sont des objets du premier ordre
- Assurent les interactions entre composants
- Peuvent être de complexité variable
  - du simple appel de méthode à l'ordonnanceur
- Permettent la flexibilité et l'évolution
- Pas de langage de spécification de connecteurs

L’architecture est une abstraction supplémentaire

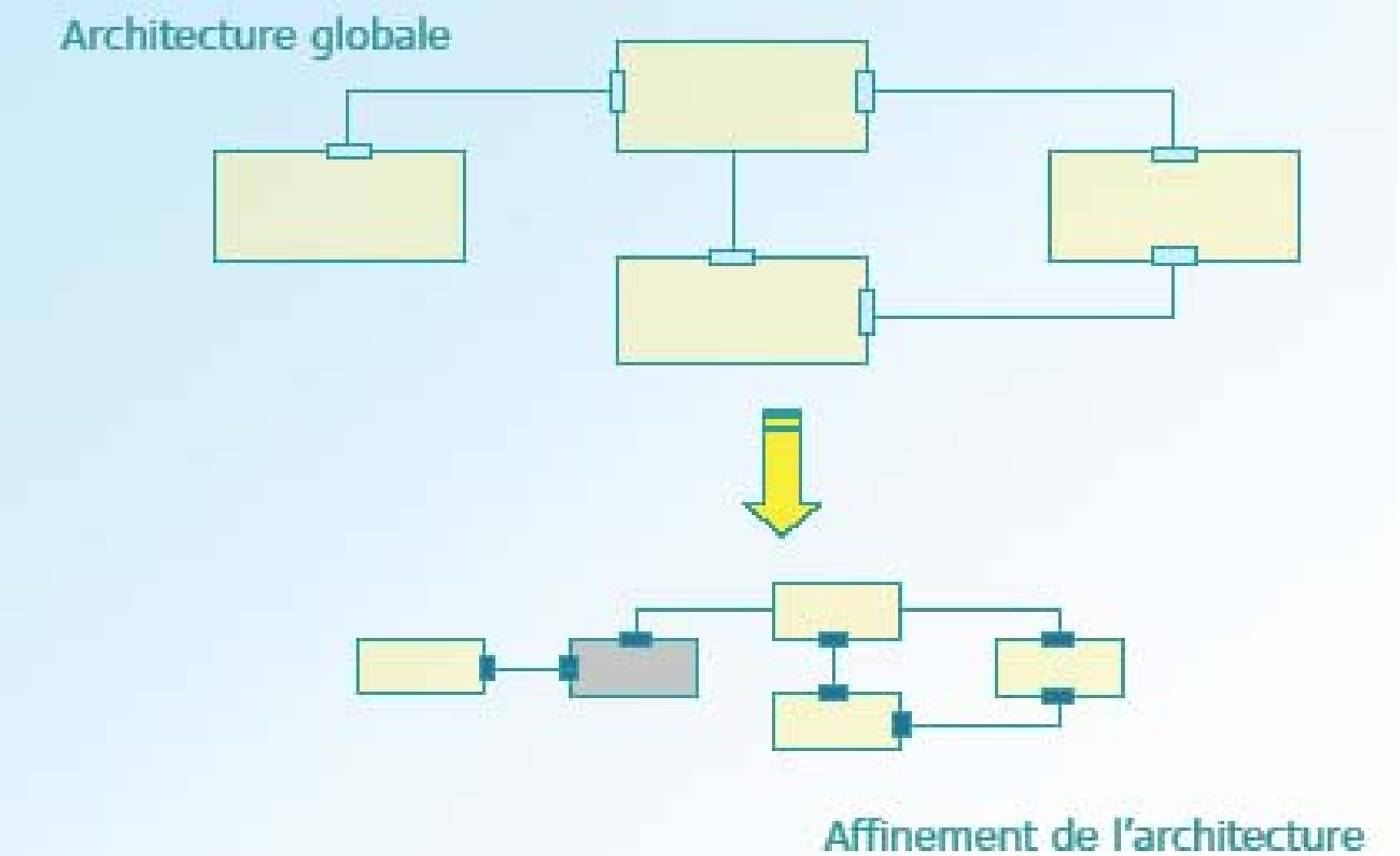
- Ne fournit que les propriétés externes des éléments structurants
- Ne se préoccupe pas des détails d'implantation



Une succession d’abstractions

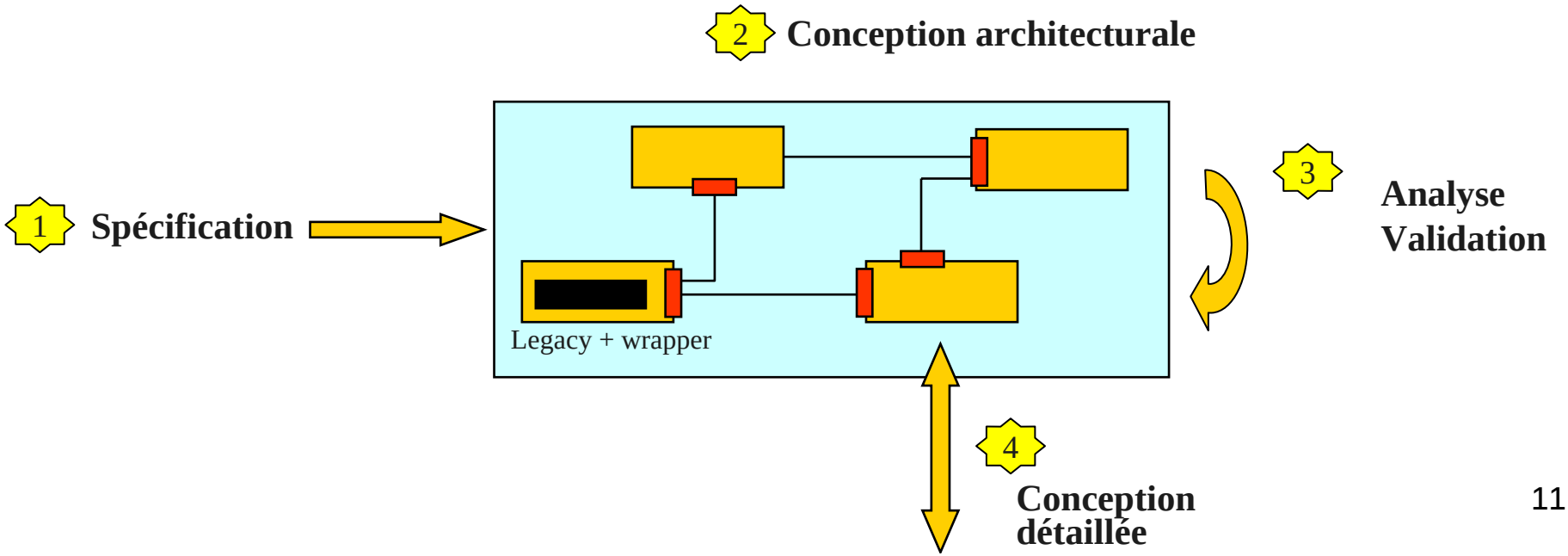


Une succession d'abstractions

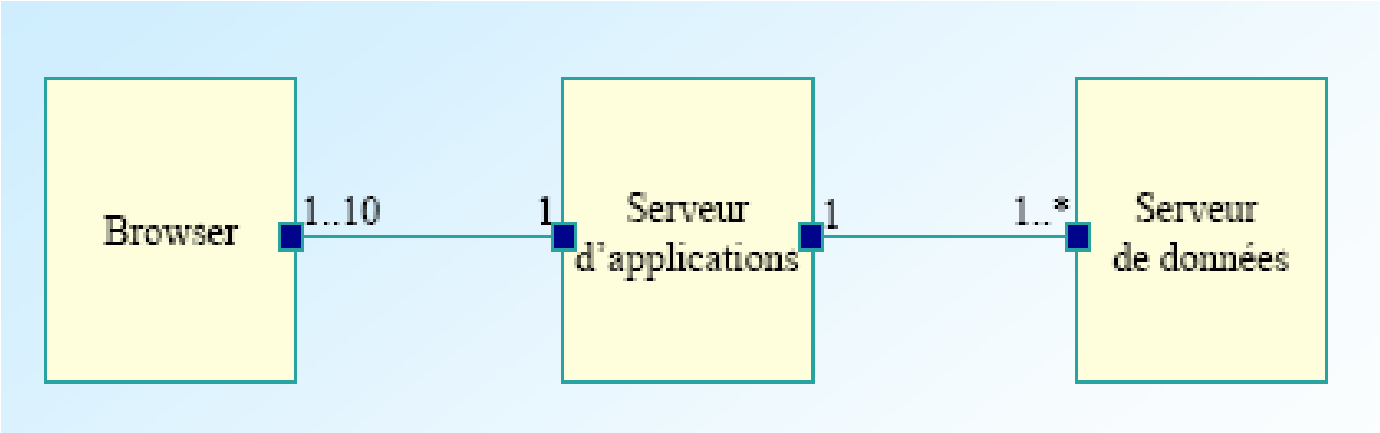


Positionnement

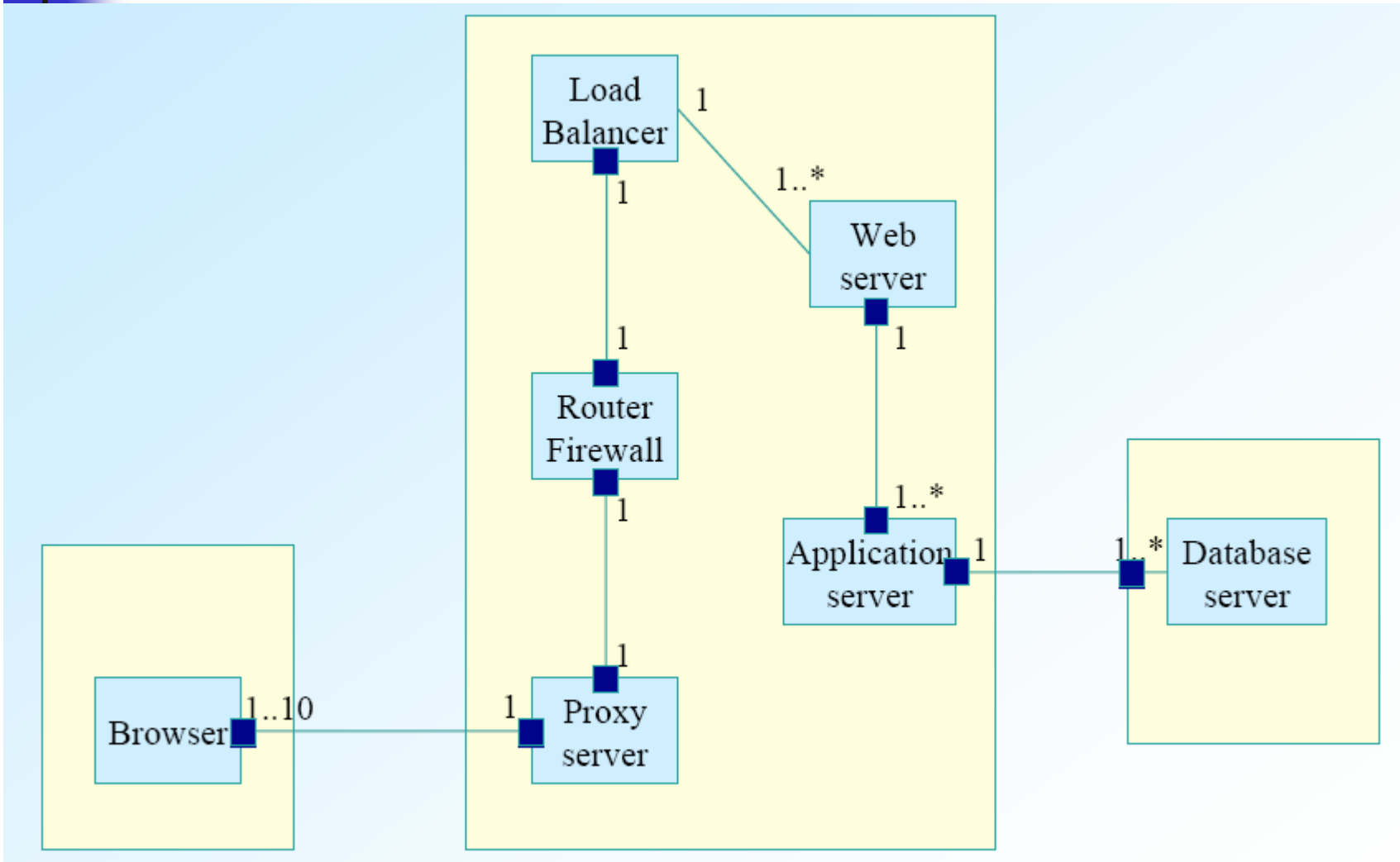
- L'architecture = première étape de conception
  - réduire la complexité du système abordé en le structurant en composants logiciels



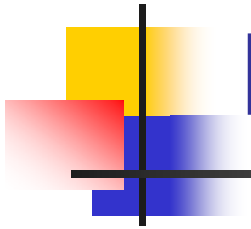
Exemple : application e-commerce



Exemple : application e-commerce



Exemple : application e-commerce



Plan

- Notion d’architecture
- **Intérêt de l’architecture**
- Représentation
- Conclusion

Enfin des réponses...

- Première étape de conception
- Permet de réfléchir et répondre aux questions:

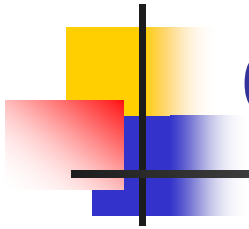
Où développer ?

Comment développer ?

Quelles équipes, quelles techno ?

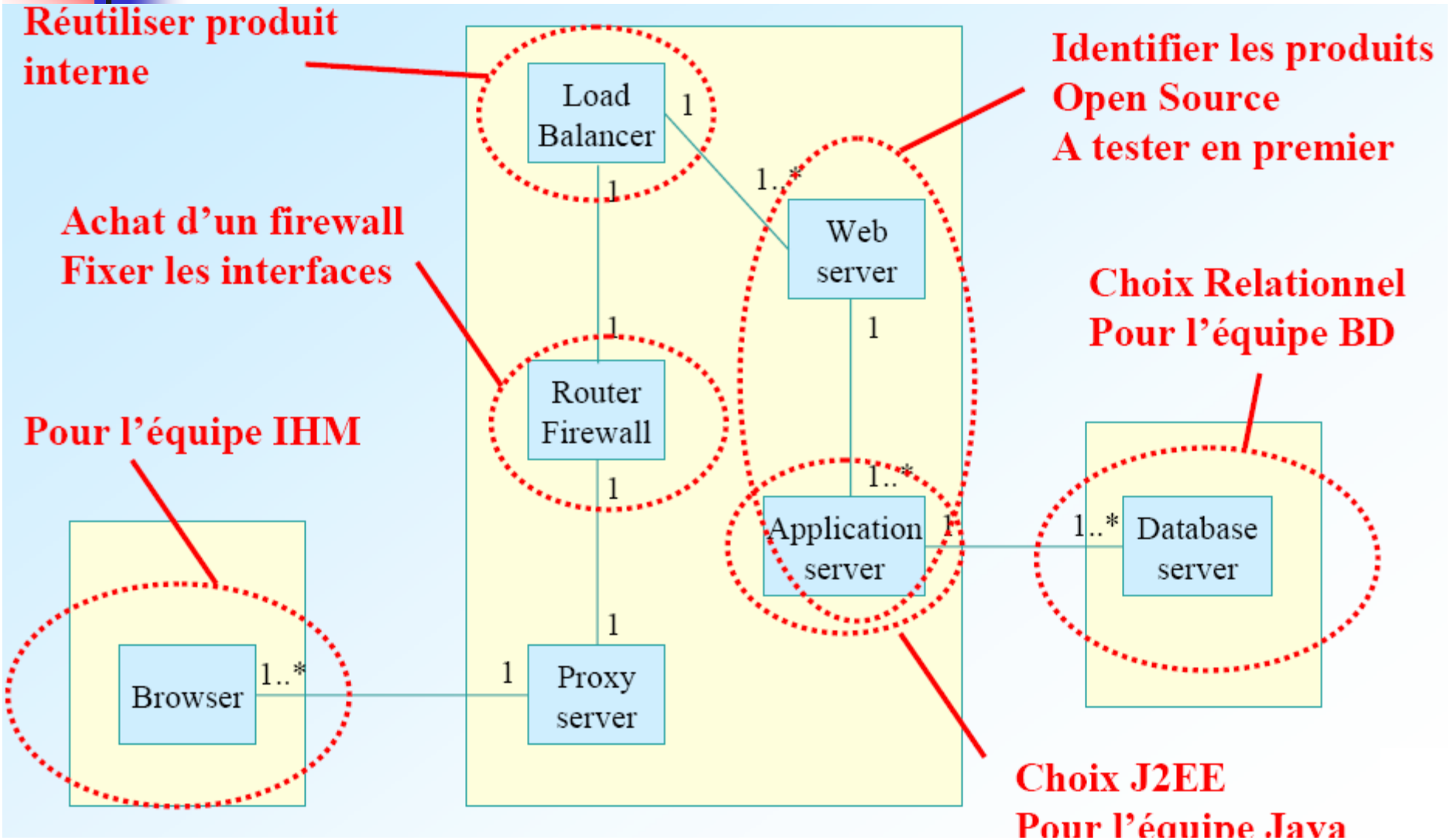
Quel coût ?

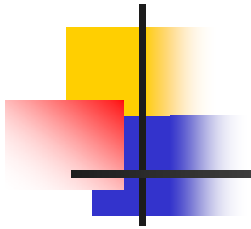
A partir de l'architecture, on peut



- Définir un plan de travail
- Répartir le travail entre les équipes
- Allouer les ressources
- Imposer des contraintes techniques
- Structurer les différentes étapes
  - le développement
  - Les tests
  - La documentation
  - La maintenance

Exemple de structuration

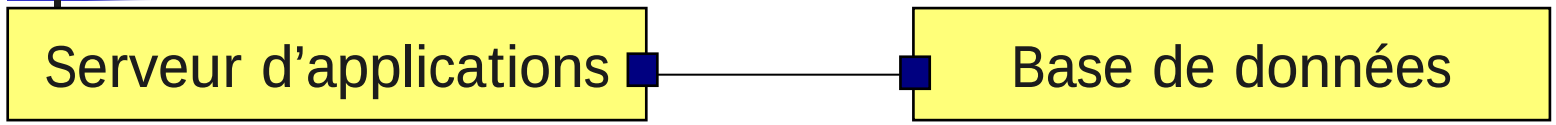


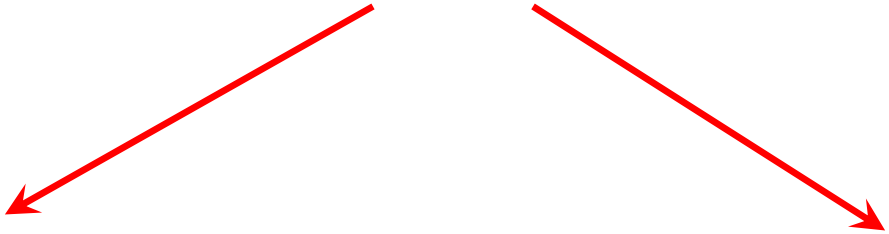


Impact sur la qualité logicielle

- L'architecture a une forte **influence** sur les propriétés finales d'un système
- La structuration architecturale **favorise ou pénalise** les propriétés non fonctionnelles telles que
  - Performance
  - Sécurité
  - Sûreté
  - Disponibilité
  - Maintenabilité
  - etc.
- Un premier niveau de **compromis** se fait au niveau architectural. Et ce, de façon quasi définitive ...

Exemple



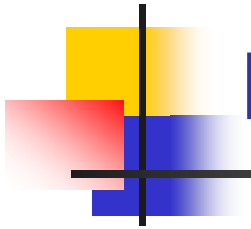


Sur la même machine

+ sécurité
+ performance (à voir)

Sur deux machines

+ disponibilité (caches, ...)
+ maintenabilité
+ sûreté (réplication possible)

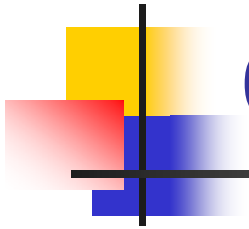


Des éléments à prendre en compte

- Peu de composants favorisera la performance (moins de communication)
- Beaucoup de composants favorisera la maintenance (au détriment de la performance)
- La redondance peut favoriser la sûreté, mais pas la compacité ou la sécurité

- Le casse tête commence !

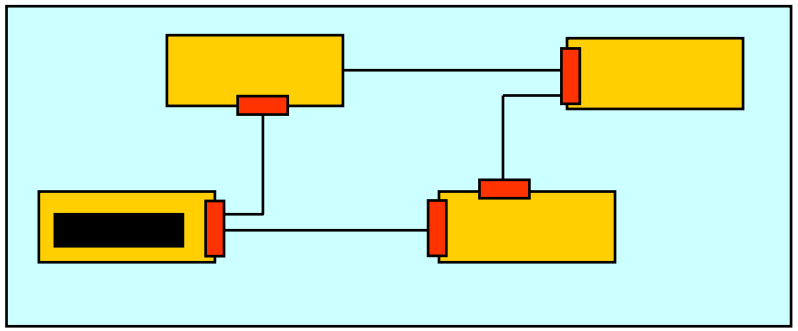
Architecture : première étape de validation



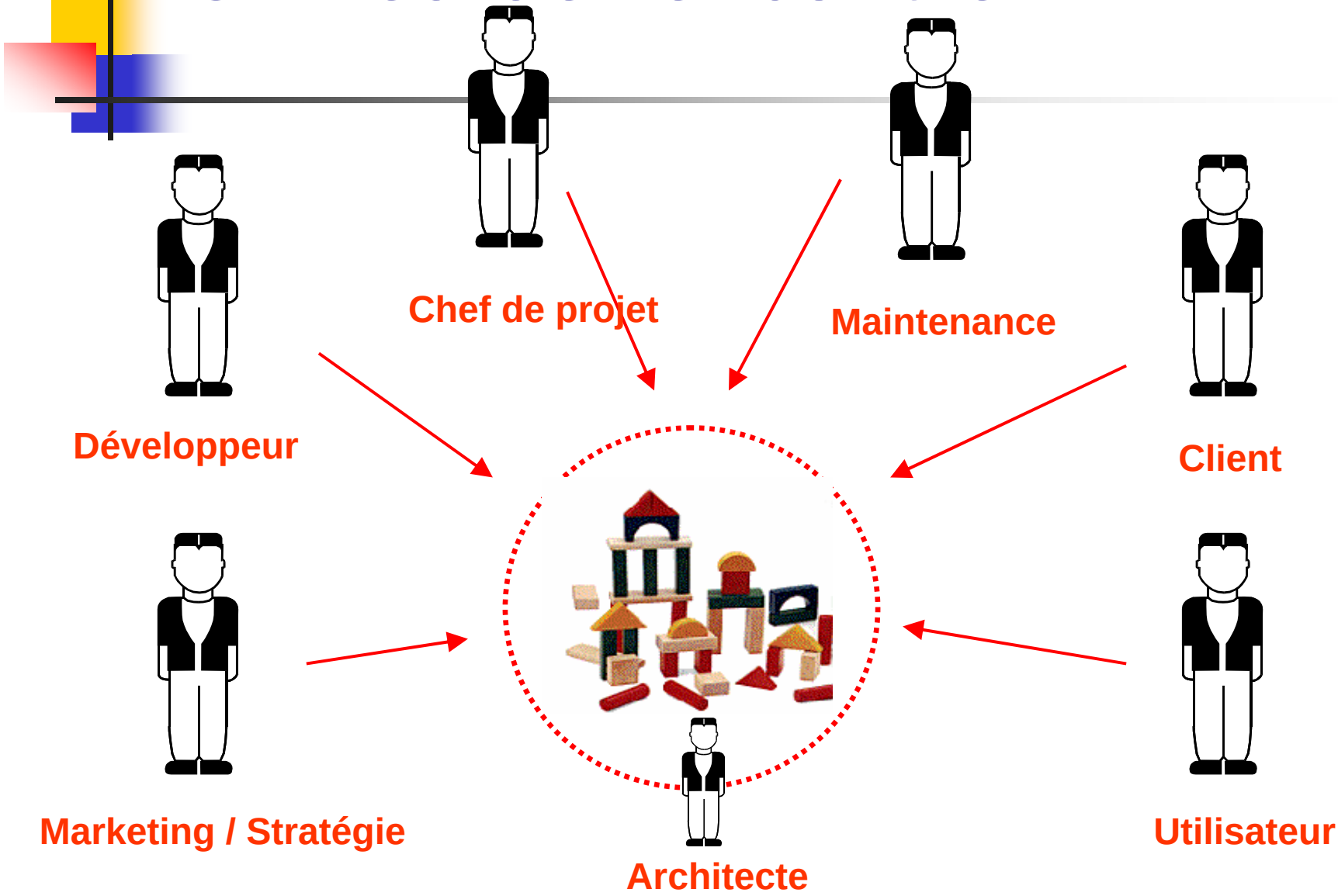
- Les décisions architecturales ont un impact important et durable
- À valider soigneusement et au plus tôt
  - revues d'évaluation
  - développement de prototypes
  - évaluation de technologies clef
  - utilisation de techniques formelles
- L'architecture influence les qualités mais ne les garantie pas

Vecteur de communication

- L'architecture fournit un canevas permettant à tous d'exprimer ses intérêts et de négocier
  - réunion de tous les intervenants autour de l'architecture
  - négociation des exigences avec les utilisateurs
  - négociation des évolutions à apporter
  - présentation régulière aux clients et au management des avancées (fonctions / coûts / échéances)
  - structuration des équipes et allocations des ressources



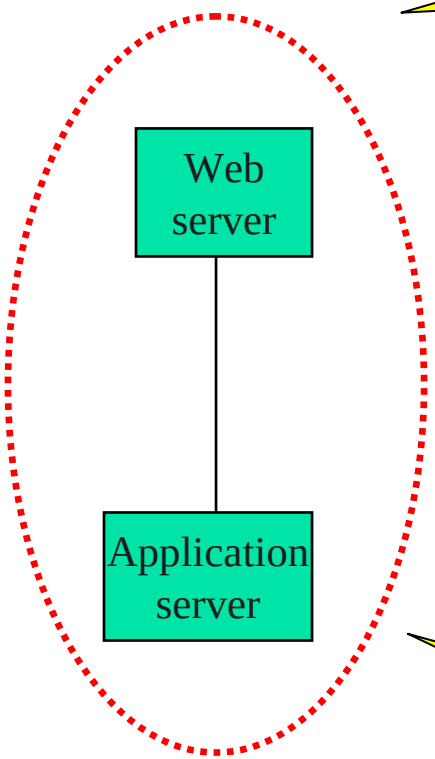
Un lieu de rencontre



Exemple

Architecte : nécessaire vu la complexité des applications

Programmeur : super cool !

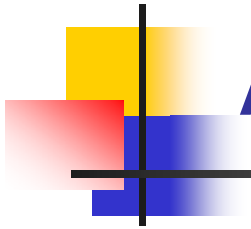


Utilisateur : est-ce que les interfaces ne vont pas changer tout le temps ?

Maintenance : non, trop instable !

Client : non, trop cher !

Manager : cool, cela va être cher !

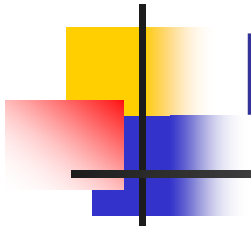


Autre exemple…

- La société YYY-SA est contactée pour une « tierce maintenance applicative » par XX-SA
- L'application est un site web commercial
  - Elle a été développée il y a 2 ans
  - Plus de programmeurs, plus de chef de projet
  - Pas de documentation
- XX-SA souhaite re-vendre l'application à un autre client
  - Dépoussiérage (customisation pour le nouveau client)
  - Ajout d'élément de sécurité

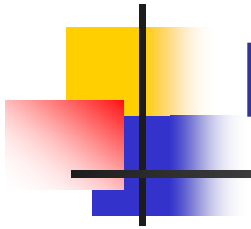
Comment allez-vous faire ?

Ceci n’est pas de la fiction...



Plan

- Notion d’architecture
- Intérêt de l’architecture
- **Représentation**
- Conclusion

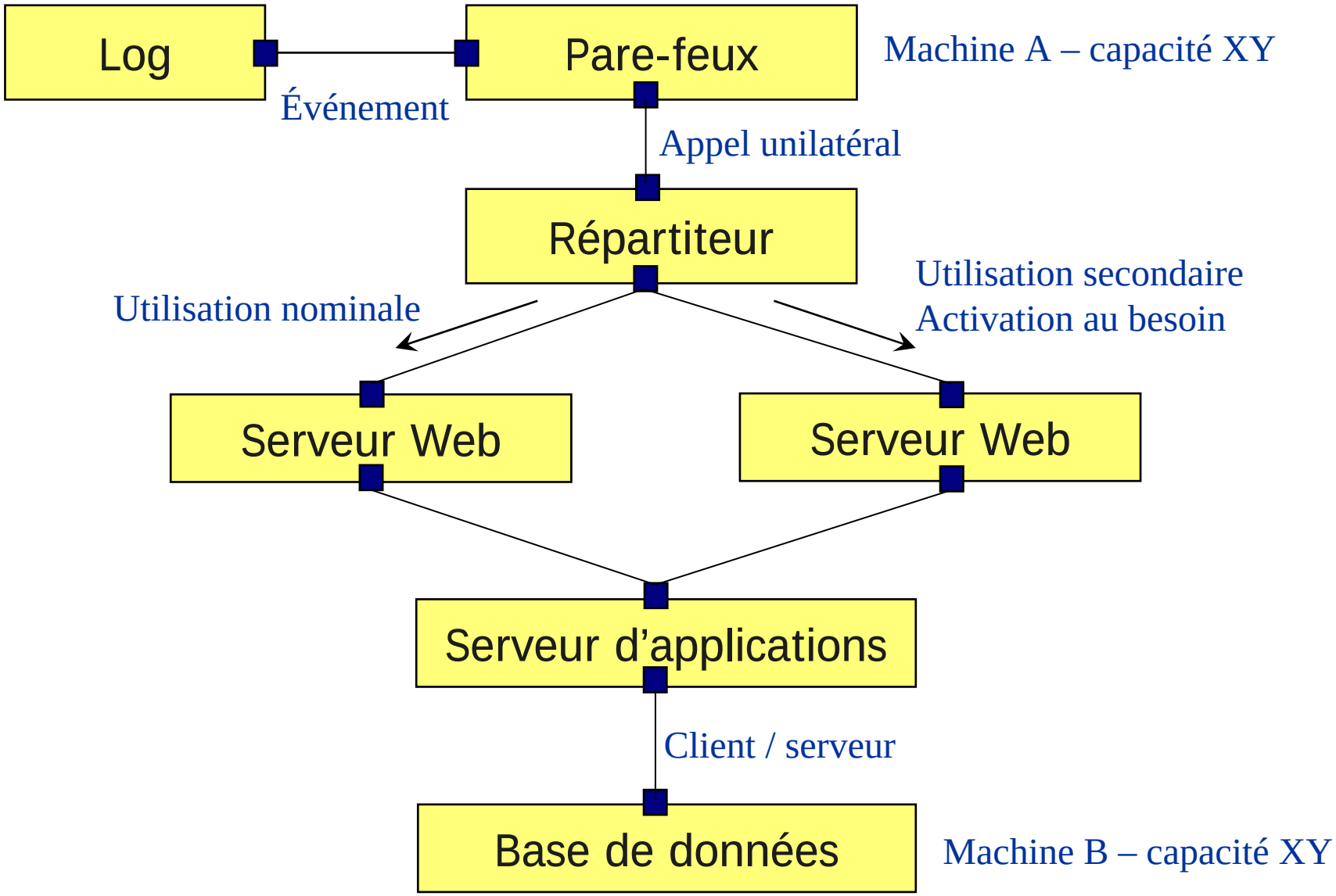


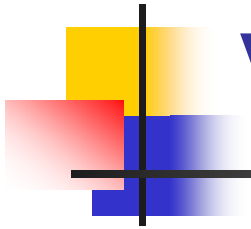
Représentation des architectures

- Le but est de représenter toutes les informations liées aux composants logiciels :
  - leur structure et leurs interfaces
  - Leurs interactions
  - leurs propriétés et les contraintes associées
  - leurs supports d’exécution, ...

- L'idéal est de tout représenter
  - sous forme graphique et sur un seul schéma
  - C'est ce qui se fait la plupart du temps

Exemple


Log
Pare-feux
Machine A – capacité XY
Événement
Appel unilatéral
Répartiteur
Utilisation secondaire
Activation au besoin
Utilisation nominale
Serveur Web
Serveur Web
Serveur d'applications
Client / serveur
Base de données
Machine B – capacité XY



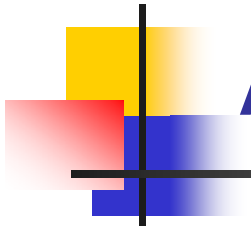
Verdict

- Hétérogène, incomplet, peu lisible, voire ambigu
  - Informations très différentes qui ne s'adressent pas aux même personnes
    - mélange des préoccupations
    - difficile à lire
  - Il manque de nombreuses informations
    - Combien de machines ?
    - Activation du premier serveur web ?
    - Protocole de communication entre le pare-feux et le répartiteur ?
    - etc.
- Besoin d'une approche structurée et répétable

Analogie

- Dans un bâtiment, on doit également manipuler des structures différentes
  - la topologie (pièces, couloirs, portes, etc.)
  - le câblage électrique
  - la plomberie
  - la ventilation, etc.

- Plans spécialisés servant de spécification
  - utilisés par des personnes différentes
    - électriciens, maçons, etc.
  - utilisés pour apporter des propriétés différentes
    - Densité des murs, diamètres des câbles, pression des tuyaux, ...

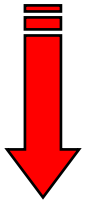


Application au logiciel

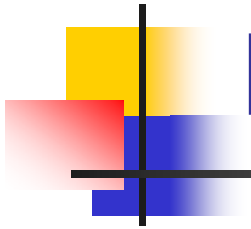
- Un logiciel est également composé de plusieurs structures (Parnas, 74)
  - Représentation indépendante des ces différentes structures au niveau des programmes
- On peut aussi appliquer cette décomposition au niveau architecture
- Notion de vues architecturales

Première conclusion

Architecture = composants + connecteurs



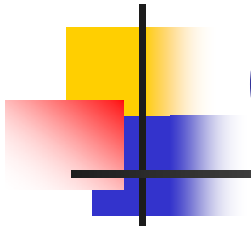
Représentation = ensemble de vues



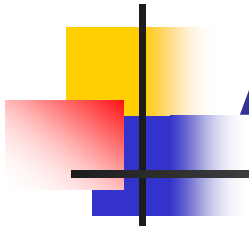
Définition des vues logicielles

- Une vue offre une perspective spécifique sur un logiciel
  - Séparation des préoccupations
- Une vue définit :
  - Les éléments logiciels représentables sur cette vue
  - Les relations représentables
  - Un formalisme
  - Éventuellement un vocabulaire
  - Éventuellement un langage de contraintes
- On utilise plusieurs types de vues complémentaires
- Elles doivent être complètes, cohérentes

Vues usuelles (généralement utilisées)

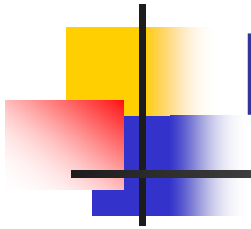


- Vue(s) logique(s)
  - Comment le logiciel est structuré en unités d'exécution (les composants)
- Vue(s) dynamique(s)
  - Comment les composants interagissent au cours du temps
- Vue(s) d'allocation(s)
  - Projection des composants vers un environnement d'exécution



Autres vues

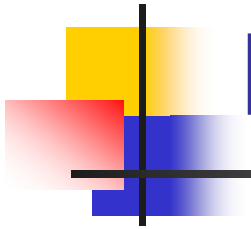
- Diagramme de contexte
  - Pour déterminer les limites du systèmes
- Diagrammes de cas d'utilisation
  - Pour déterminer les principales fonctions du systèmes



Plusieurs vues

- On peut se focaliser et travailler sur un aspect donné
- On gagne en cohérence locale
- On abaisse significativement la complexité

- L'architecture se dématérialise un peu ...
- Cohérence générale ?
- Et comment recoller les morceaux ?

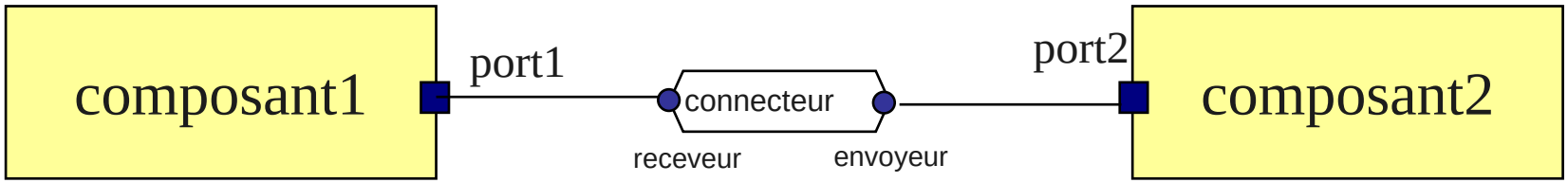


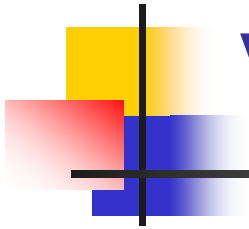
Plan

- Notion d'architecture
- Intérêt de l'architecture
- **Représentation**
  - Les vues structurelles
  - Les vues dynamiques
  - Les vues d'allocation
- Conclusion

Vue logique : définition

- Cette vue définit la structure de l'architecture
  - décomposition en éléments logiques
- Tous les composants et leurs connexions sont décrits
  - les connexions décrites sont potentielles
  - les composants peuvent avoir une architecture, les connecteurs peuvent être complexes



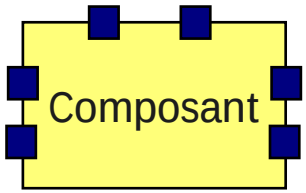


Vue logique : contenu

- Éléments à spécifier
  - les composants
  - les connecteurs
  - les contraintes et les principes
  - des commentaires éventuels

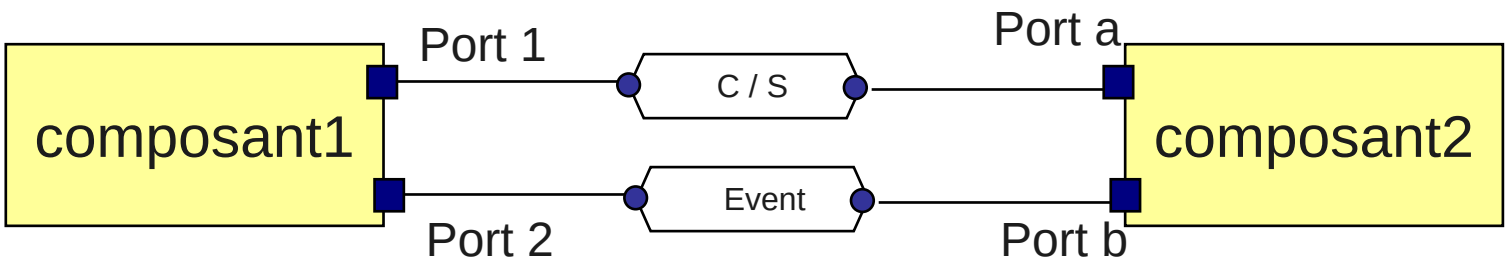
Spécification des composants

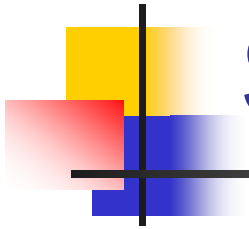
- Ports
  - fonctionnalités fournies par le composant
  - fonctionnalités requises par le composant
  - Services de gestion du cycle de vie
- Contraintes
  - Type de communication à utiliser
  - Ordonnancement entre appels, ...
- Propriétés non fonctionnelles
  - Performance, persistance, robustesse, ...



Spécification des ports

- Les ports sont les canaux d'interaction des composants
  - ils sont nommées
  - ils regroupent les messages entrants et sortants
  - ils mettent en place un protocole de communication
  - un composant peut en posséder plusieurs
  - un port regroupe souvent plusieurs interfaces



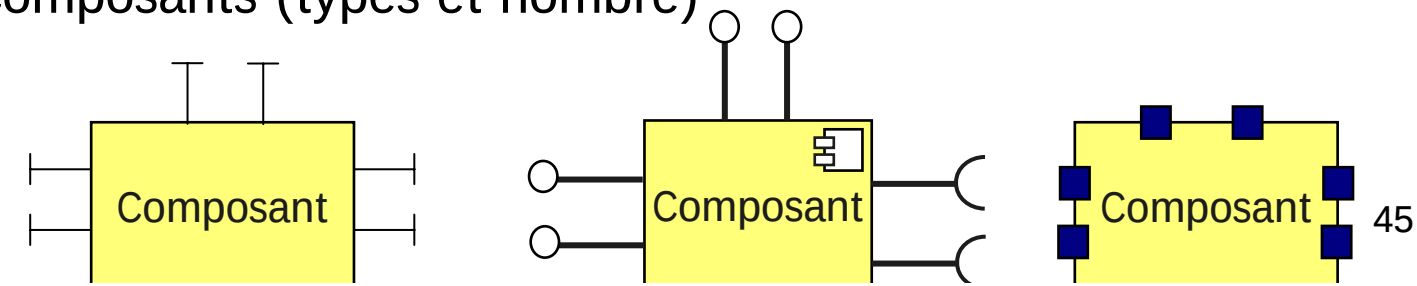


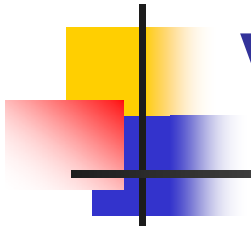
Spécification des interactions

- Rôle
  - communication
  - coordination
  - conversion
  - facilitation
- Modes de communication
  - « Procedure Call » (PC, RPC, C/S, méthodes, ...)
  - événements
  - Publication (P/S...)
  - Communication multi-parties (broadcast,...)

Note sur le formalisme

- Le formalisme en lui-même n'a pas grande importance
- Une vue logique doit être
  - Cohérente et complète
  - Explicite au niveau du vocabulaire (composants et connecteurs)
  - Suffisamment simple pour être comprise d'un coup (et par tous)
  - Focalisée et n'abordant pas les problèmes d'implantation
  - Point de départ de discussions à propos des interfaces des composants (types et nombre)

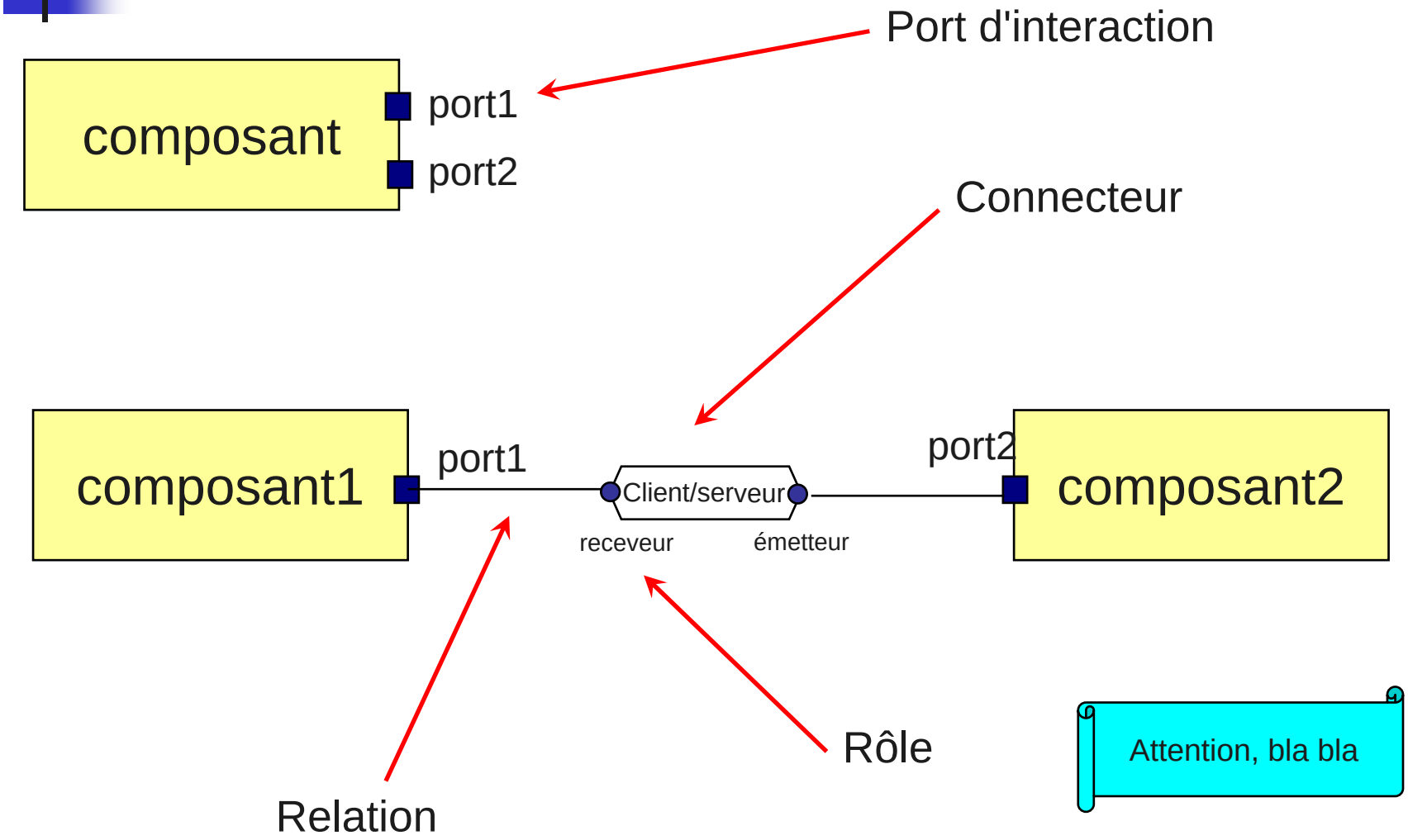




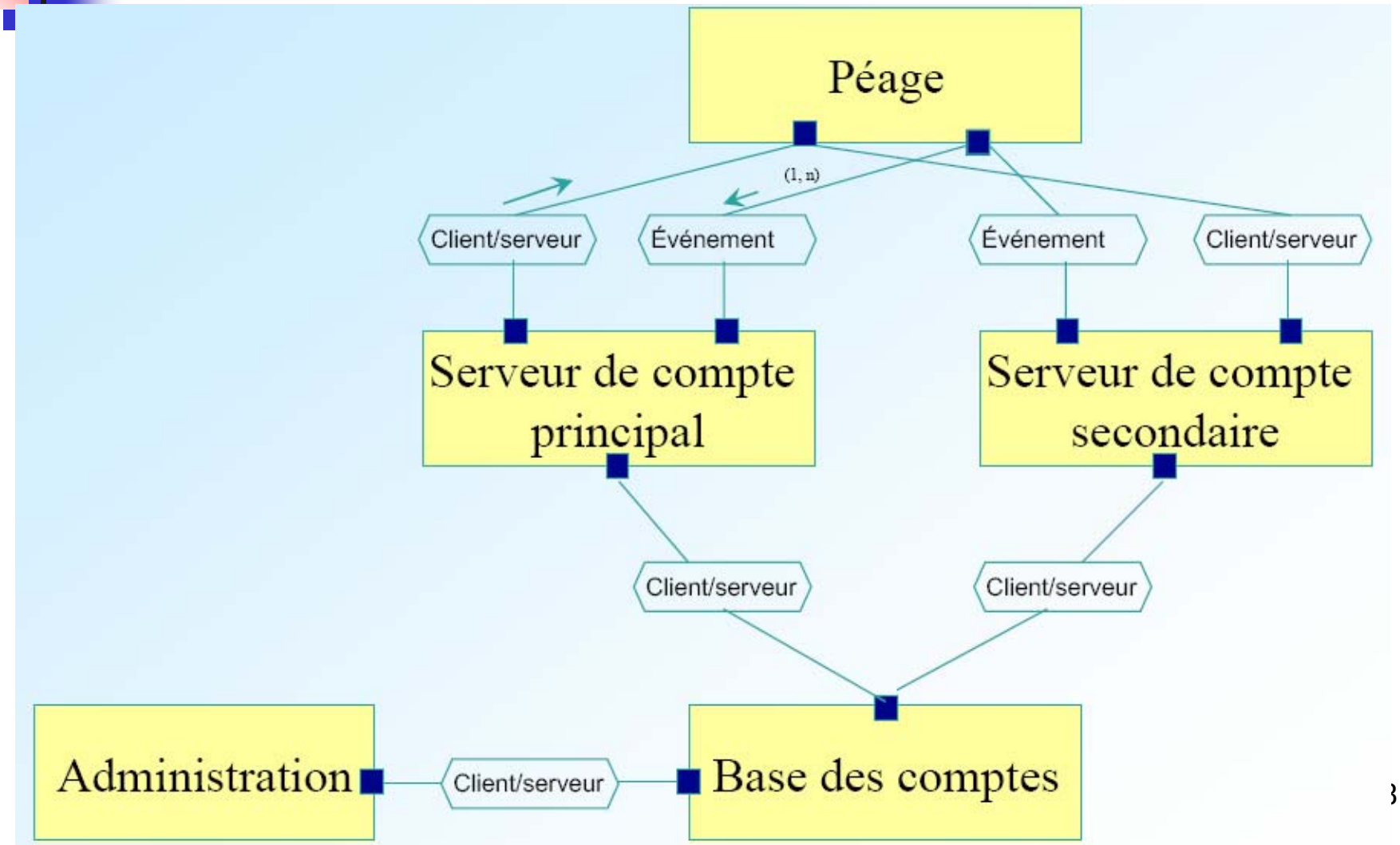
Vue logique : intérêts

- Ces vues permettent de répondre aux types de questions suivantes
  - Quels sont les principaux composants (calculs) et leurs relations ?
  - Quels sont les principaux entrepôts de données?
  - Quels sont les protocoles d'interaction utilisés et donc les besoins en infrastructure d'exécution ?
  - Quels sont les chemins critiques et sensibles ?
  - Existe-t-il des points d'étranglement à étudier ?
  - Quel est le niveau de couplage et de cohésion ?
  - Etc.

Vue logique : formalisme

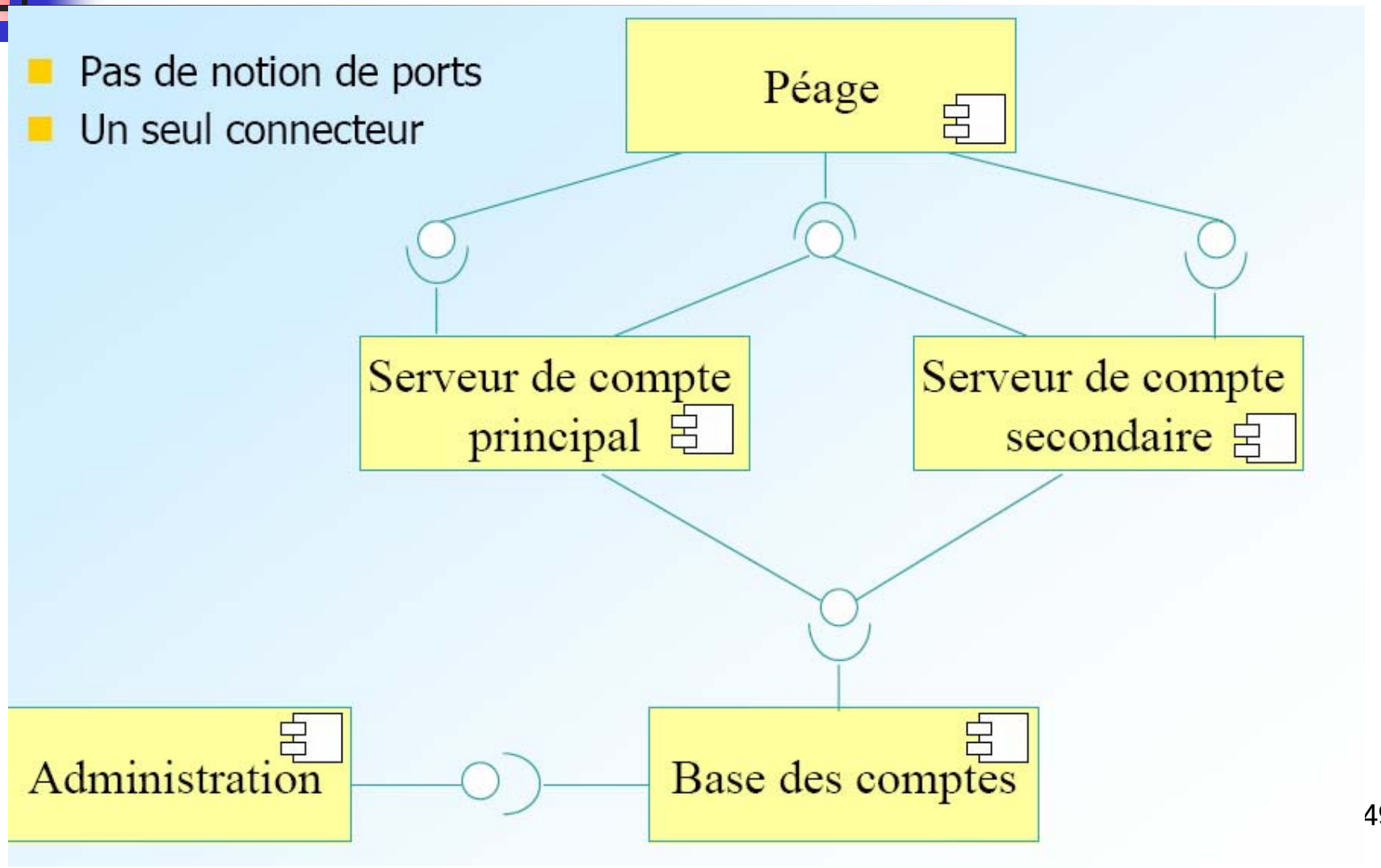


Exemple


Péage
(1, n)
Client/serveur
Événement
Événement
Client/serveur
Serveur de compte principal
Serveur de compte secondaire
Client/serveur
Client/serveur
Administration
Client/serveur
Base des comptes

Exemple représenté en UML

- Pas de notion de ports
- Un seul connecteur



Autre formalisme : ACME (formel)

Définition des types

```
Family PipeFilter = {
   Port Type OutputPort;
   Port Type InputPort;
   Role Type Source;
   Role Type Sink;

   Component Type Filter;
   Connector Type Pipe = {
            Role src : Source;
            Role snk : Sink;
            Properties {
              latency : int;
              pipeProtocol : String = ...;
            }
   }
};
```

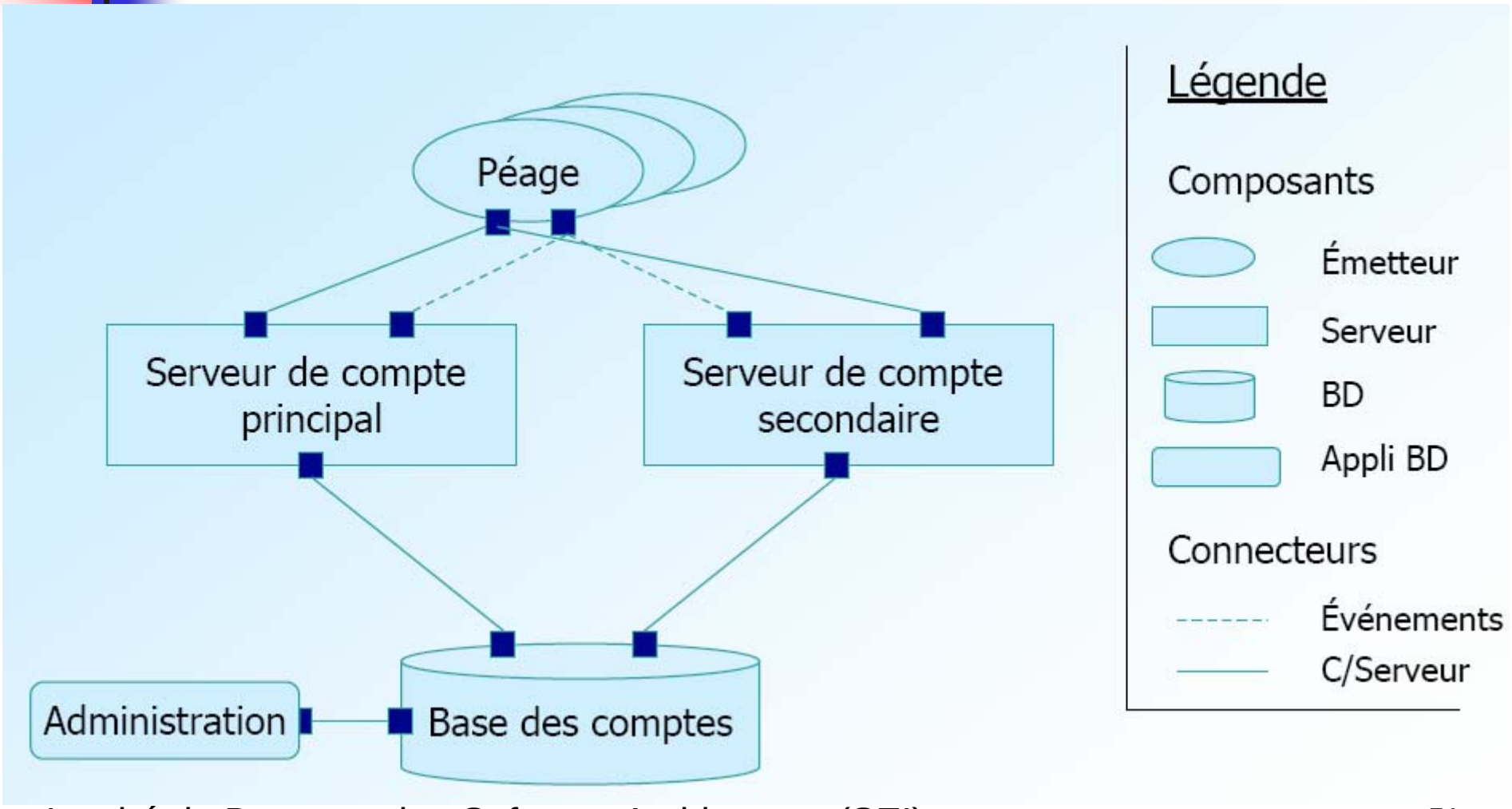
Définition du système

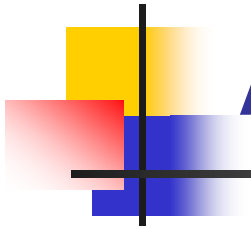
```
System simple : PipeFilter = {
// définition des composants
  Component Splitter : Filter = {
    Port pIn : InputPort = new InputPort;
    Port pOut1 : InputPort = new OutputPort
    Port pOut2 : InputPort = new OutputPort
    Properties {...}
   }
...
// définition des connecteurs
Connector SpliStream : Pipe = new Pipe;
...
// définition des attachements
Attachements {
Splitter.pOut1 to SplitStream1.src;

};
};
```

Extrait de : Documenting Software Architecture (SEI)

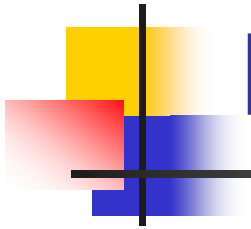
Formalisme ad-hoc





A propos du formalisme

- N’a pas une grande importance
- Vue logique doit être
  - Cohérente et complète
  - Explicite au niveau du vocabulaire
  - Simple pour être comprise par tous
    - pas trop de composants
  - Focalisée (pas de problème d’implantation)
  - Point de départ de discussions (composants, ports, ...)

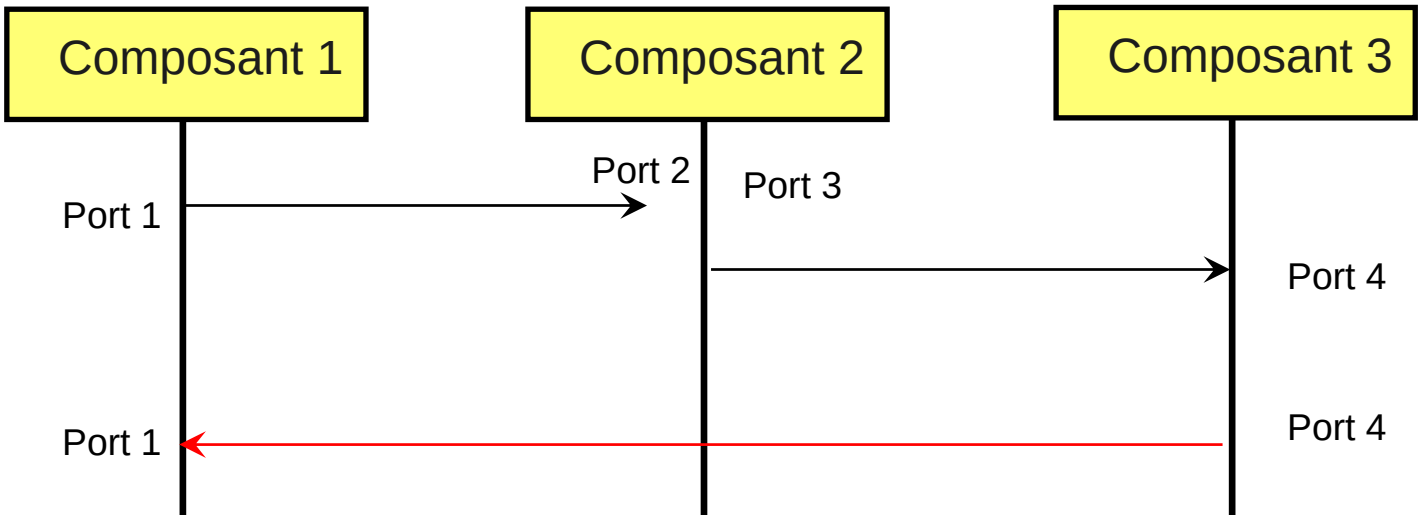


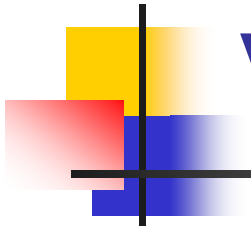
Plan

- Notion d'architecture
- Intérêt de l'architecture
- **Représentation**
  - Les vues structurelles
  - **Les vues dynamiques**
  - Les vues d'allocation
- Conclusion

Vue dynamique : définition

- Définit le comportement dynamique de l'application
- Éléments constituants
  - Composants (et leurs ports)
  - Axe temporel

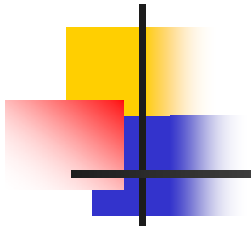




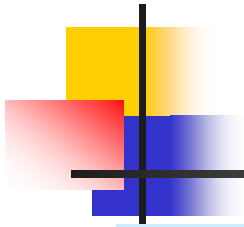
Vue dynamique

- Cette vue définit les interactions au sein de l'architecture
  - Quand et pourquoi elles ont lieu (événements déclenchants)
  - Comment elles se déroulent
- Seules les relations effectives sont présentées
- Les interactions sont souvent contextuelles
  - Événements déclenchants
  - État global du système / des composants
  - Des résultats intermédiaires de l'interaction

Vue dynamique : Éléments à modéliser



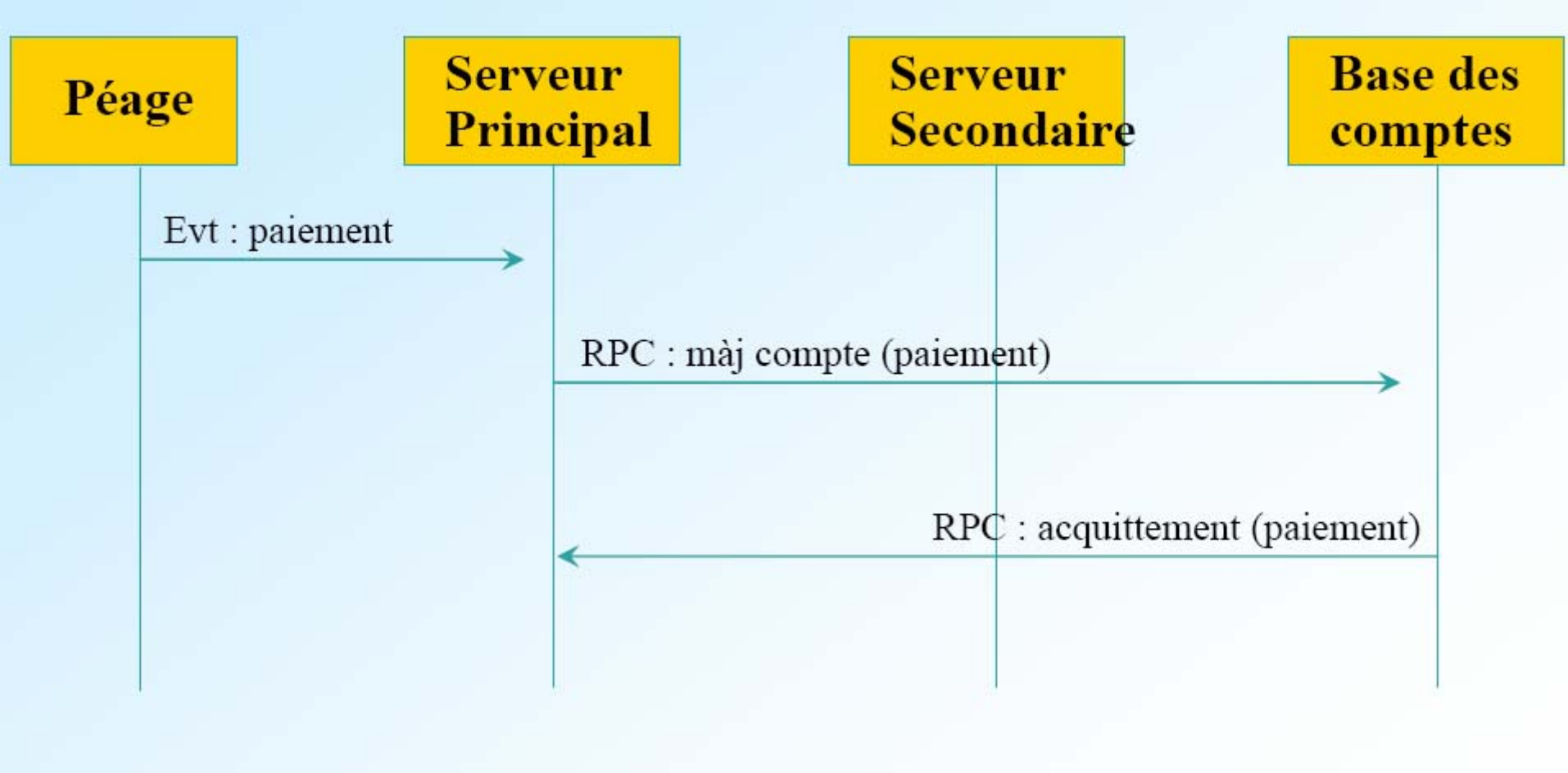
- la succession des activités
  - déclencheurs (stimuli - événement)
  - ordonnancement (avec d'éventuelles conditions)
  - périodicité éventuelle
  - durée (si pertinent)
- la nature des interactions
  - nature des communications
  - les données échangées
  - possibilité de concurrence

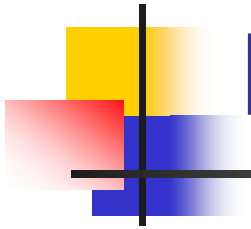


Exemple 1

- Scénario 1
  - Description : ce scénario montre ce qu'il se passe lorsqu'un péage envoie un événement de type « paiement » vers le serveur de compte principal
  - Événement déclenchant : un paiement est effectué. Un tel événement peut arriver à tous moments.
  - Contexte : fonctionnement nominal. Le serveur principal n'est pas en surcharge (par exemple, par ce qu'il n'est pas en cours de traitement d'un nombre important d'événements).

Exemple - suite

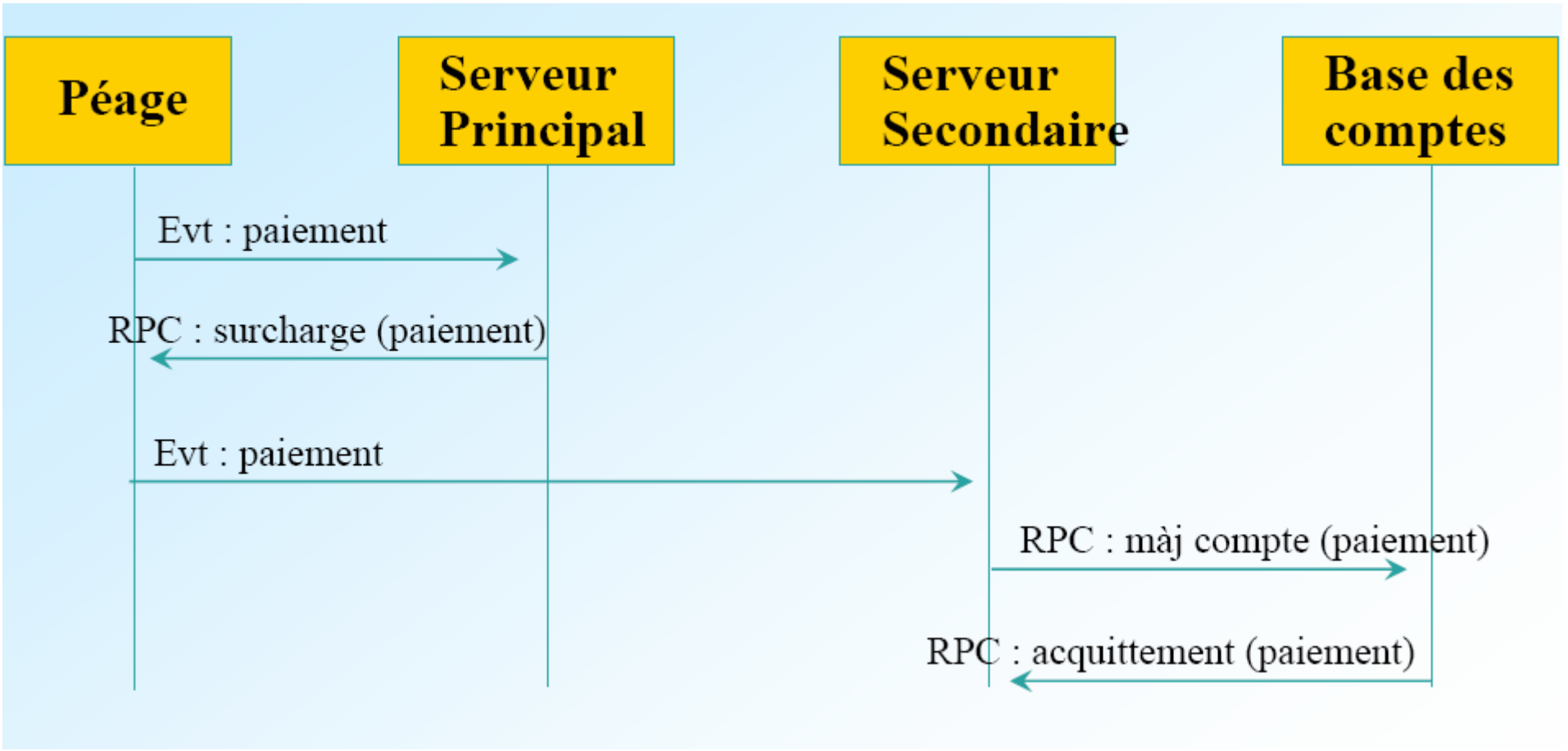


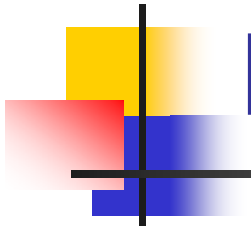


Exemple 2

- Scénario 2
  - Description : ce scénario montre ce qu'il se passe lorsqu'un péage envoie un événement de type « paiement » alors que le serveur de compte principal est en surcharge
  - Événement déclenchant : un paiement est effectué. Un tel événement peut arriver à tous moments.
  - Contexte : Le serveur principal est en surcharge (par exemple, il est en cours de traitement d'un nombre maximal d'événements).

Exemple 2 - suite

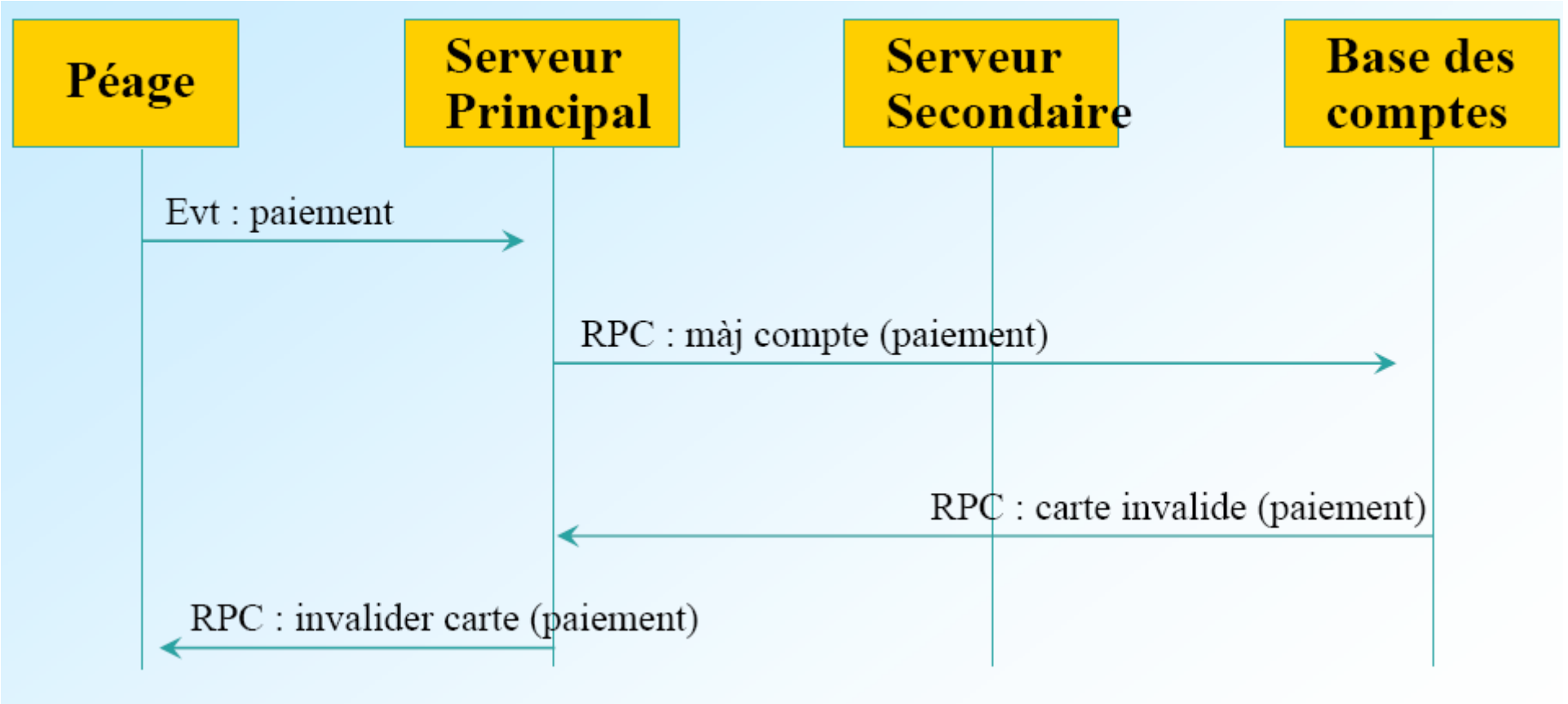




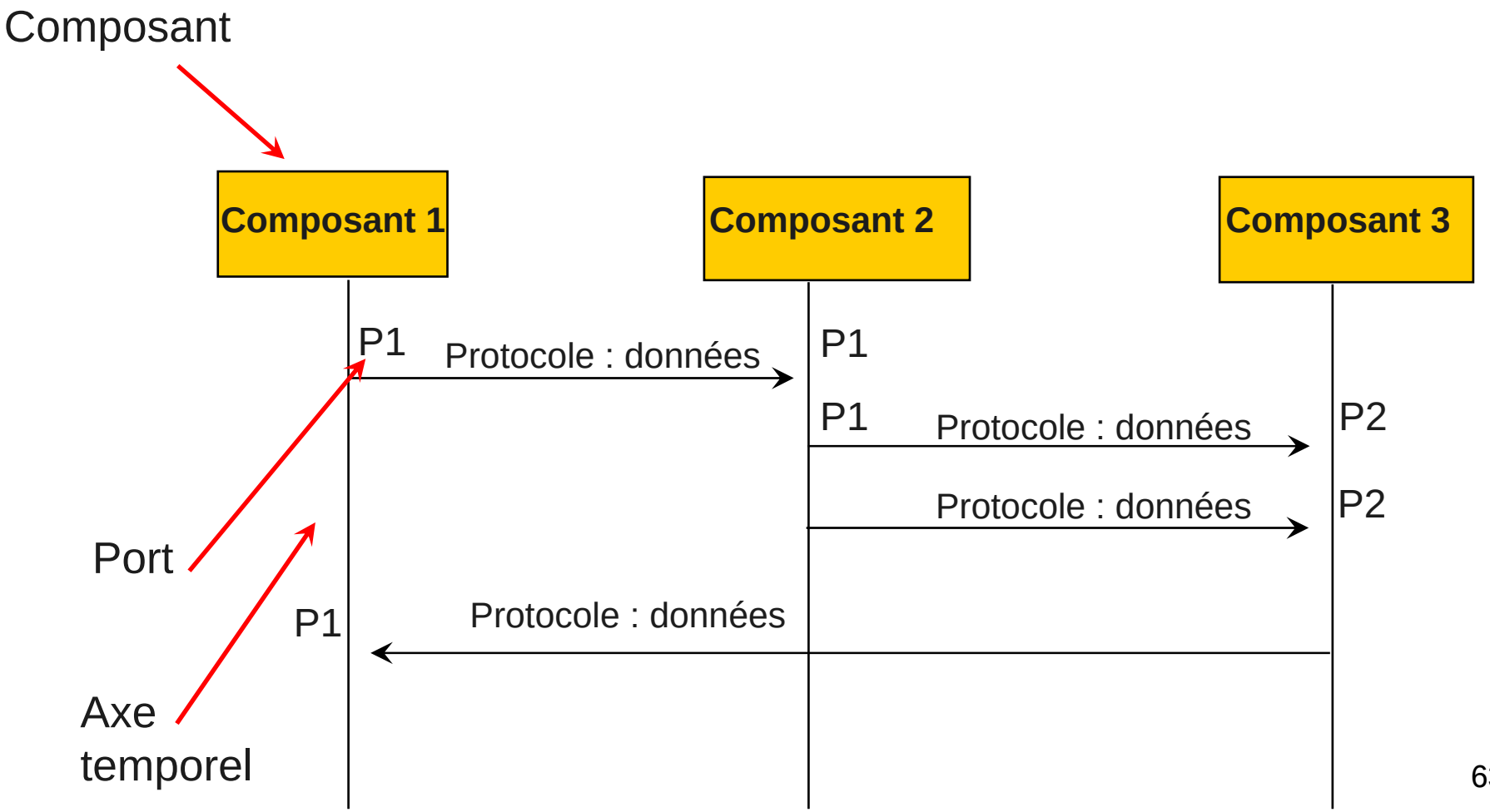
Exemple 3

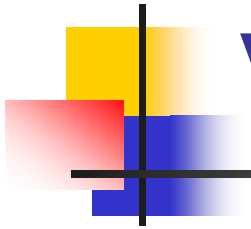
- Scénario 3
  - Description : ce scénario montre ce qu'il se passe lorsqu'un péage est payé avec une carte invalide (volée, non créditée, etc.)
  - Événement déclenchant : un paiement est effectué. Un tel événement peut arriver à tous moments.
  - Contexte : Fonctionnement nominal pour le serveur principal.
  - Résultat intermédiaire : la carte n'est pas validée par la base des comptes

Exemple 3 - suite



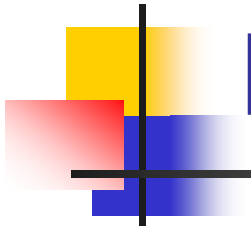
Vue dynamique : formalisme retenu





Vue dynamique : conclusion

- Cette vue décrit les aspects contrôle
  - Nécessaire pour passer à la conception détaillée et au codage
    - spécification des chemins, du parallélisme, ...
  - Important pour les validations au niveau architectural
    - vérification des enchaînements, des transactions, ...
    - vérification des performances, des synchronisations, ...
- Grande difficulté de représentation
  - Scénarios : empiriques, incomplets, complexes à identifier
  - Formel : limité à certaines propriétés (niches)

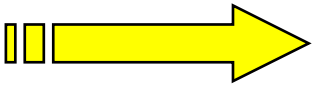


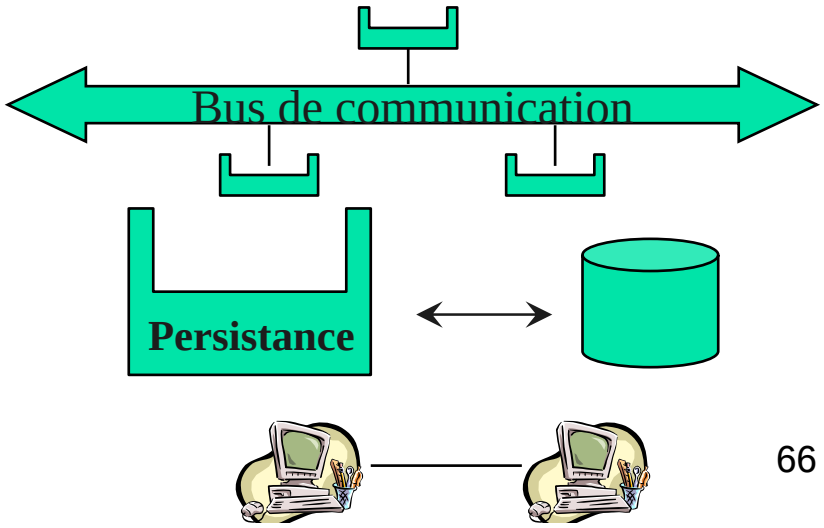
Plan

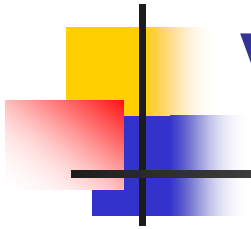
- Notion d'architecture
- Intérêt de l'architecture
- **Représentation**
  - Les vues structurelles
  - Les vues dynamiques
  - **Les vues d'allocation**
- Conclusion

Vues d'allocation

- Présentent la projection de la vue logique sur les ressources physiques ou les environnements logiciels d'exécution (« middleware »).



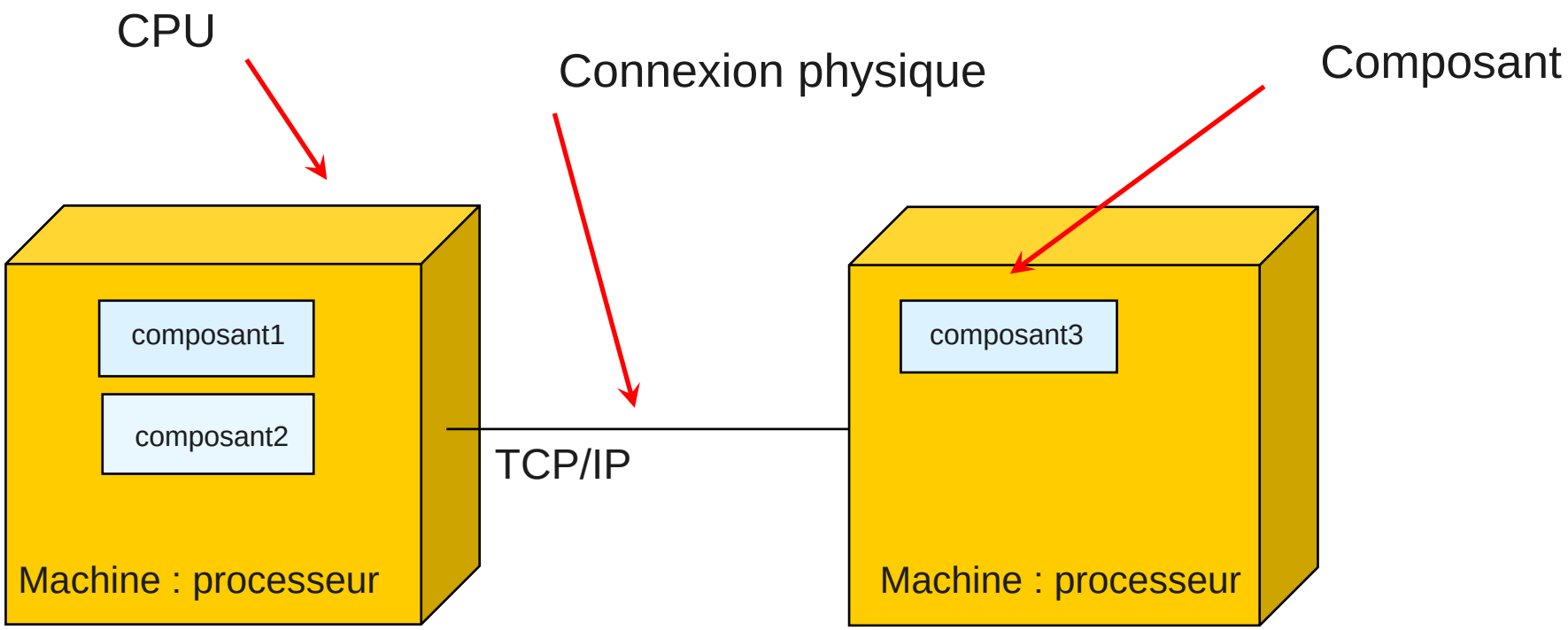




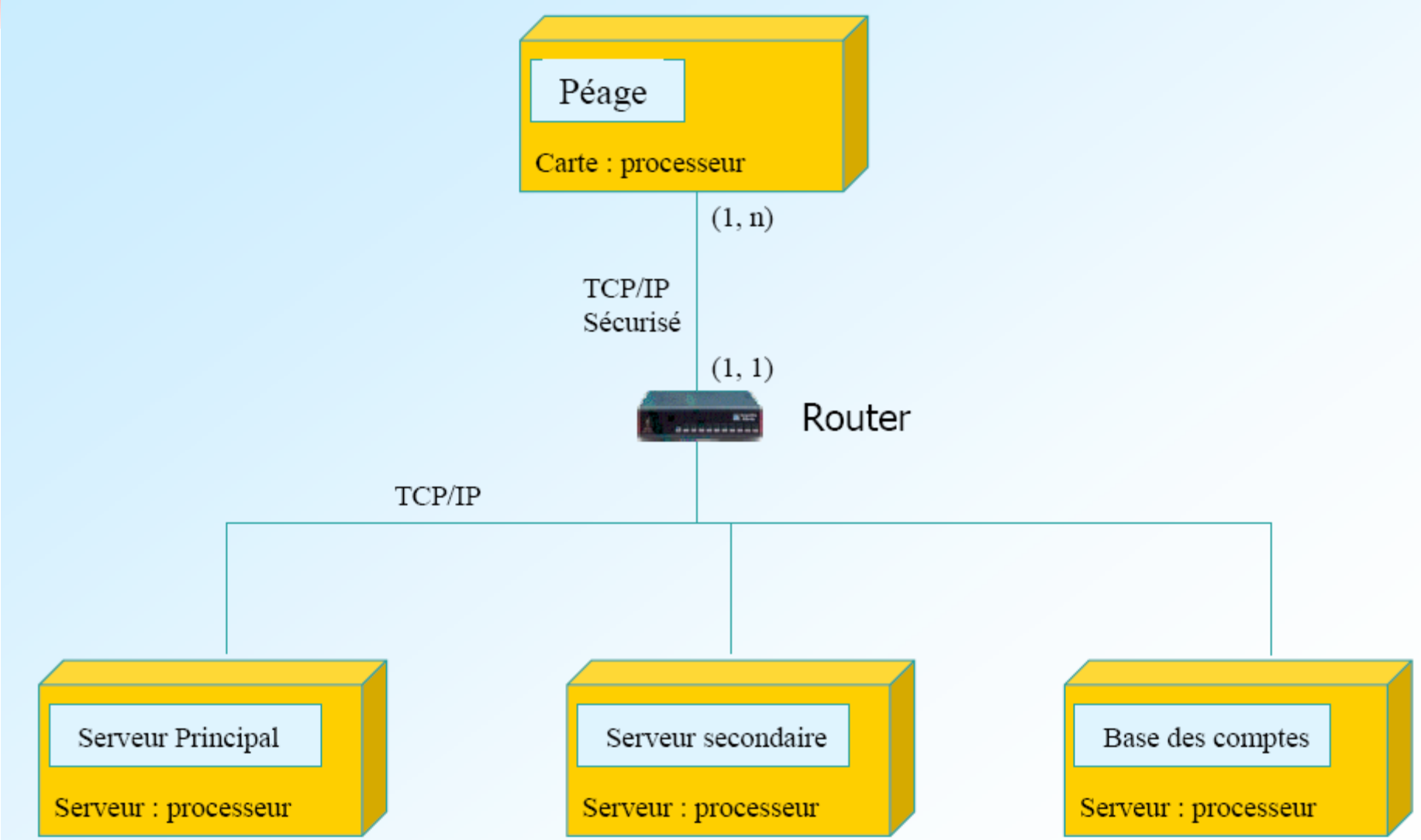
Vue physique : définition

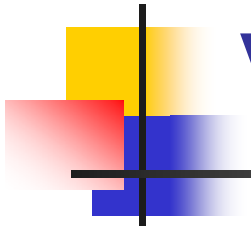
- Cette vue montre comment les éléments de la vue (composants) sont alloués à des plate-formes d'exécution. On détaille :
  - Les contraintes des éléments logiciels
  - Les caractéristiques des éléments hardware
- Propriétés liées aux éléments hardware
  - CPU, mémoire, disques, ...
- Propriétés liées aux éléments logiciels
  - Ressources nécessaires
  - Criticité : par exemple, un élément doit être toujours actif
- Propriétés liées à l'allocation

Vue physique : représentation



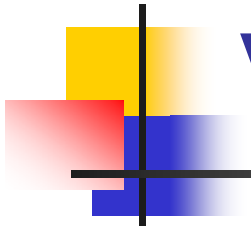
Vue physique : exemple





Vue physique : intérêts

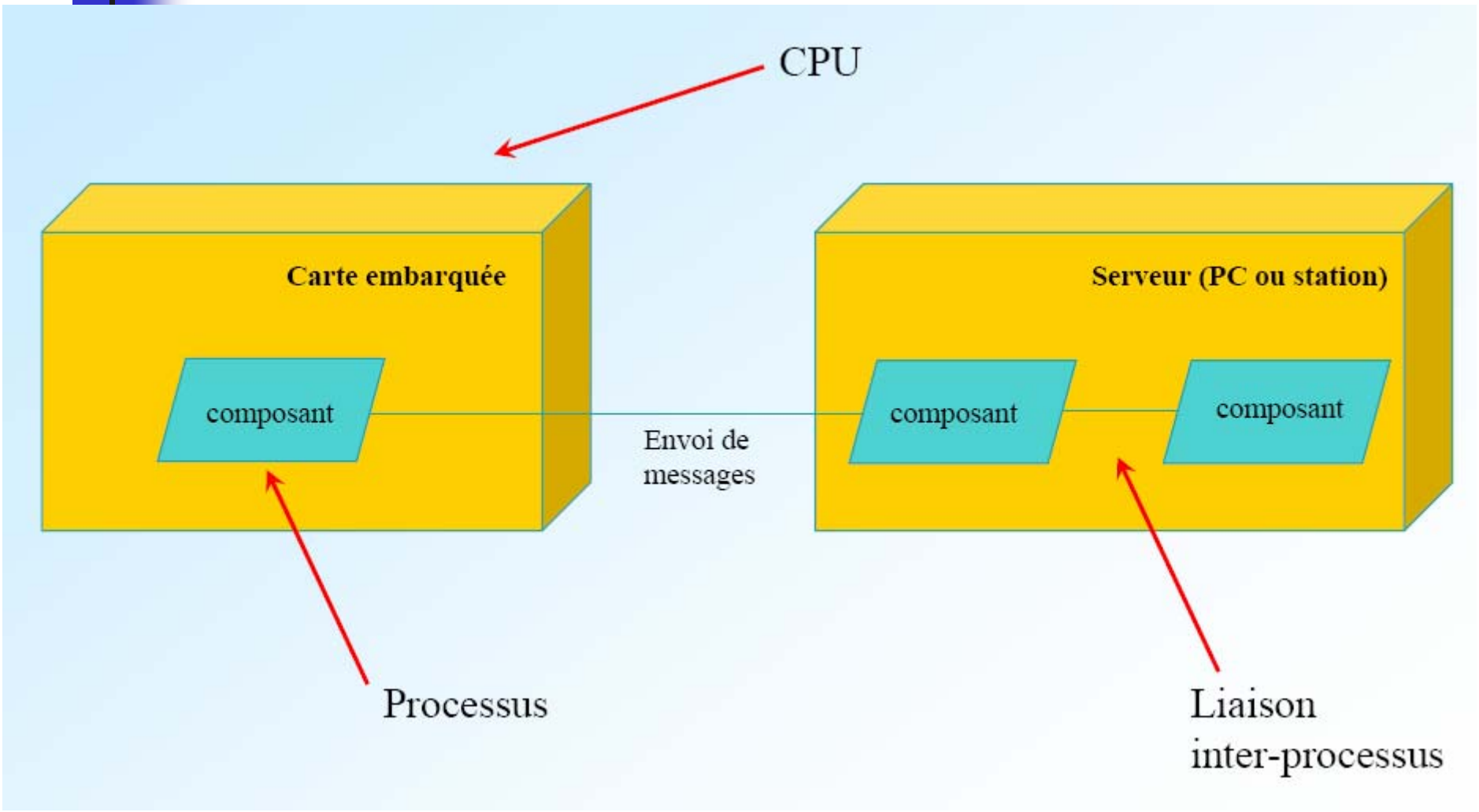
- Analyse de performance
  - élimination des « bottlenecks »
  - co-localisation des composants ayant de fortes communications
  - → le volume et la fréquence des communications est un des points les plus étudiés lors de la construction de la vue de déploiement
- Analyse de la fiabilité
  - duplication de composants critiques ou
  - règles de migration (si on peut voir venir les problèmes)
- Sécurité
  - dispositifs de sécurité, éléments de sécurisation, chemins de communication admis, etc.

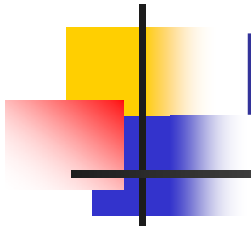


Vue processus

- Se focalise sur les processus exécutés par le système et leur modes de communication
  - Très important pour les systèmes temps-réel (répartition, concurrence, communication)
- Décrit la projection des processus sur les ressources physiques
- Décrit (si possible) la projection des composants sur les processus

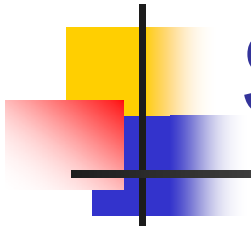
Vue processus - représentation





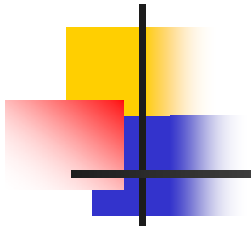
Plan

- Vue logique
- Vue dynamique
- Vues d’allocation
- Conclusion



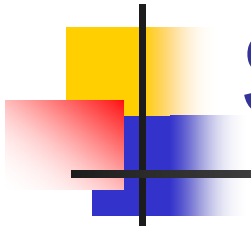
Synthèse

- L'architecture est une abstraction d'un système
  - elle est basée sur les notions de composants logiciels et de connecteurs
  - elle est incontournable pour faire face à la complexité des systèmes actuels
- C’est le premier niveau de conception
  - L’architecture doit apparaître explicitement et être utilisée
  - Elle doit rester en synchronisation avec les développements



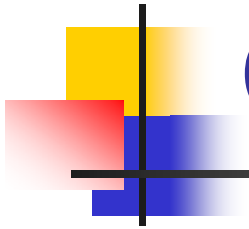
Importance de l’architecture

- Aujourd'hui reconnu comme une étape fondamentale du développement logiciel
  - elle apparaît dans tous les projets sérieux
  - les entreprises y attachent une grande importance
    - création de la fonction d'architecte
    - création d'équipes d'architectes (pour capitaliser et pérenniser les connaissances architecturales métier)
  - Bill Gates est « Chief Software Architect » !



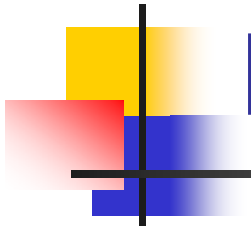
Synthèse

- Une architecture est représentée par plusieurs vues complémentaires
  - Séparation des préoccupations
  - Difficile d'assurer la cohérence et la complétude
- Vues importantes
  - Vue logique
  - Vue dynamique
  - Vues d'allocation (ou de déploiement)
- Pas de formalisme standard
  - Utiliser un formalisme connu quand c'est possible (UML par exemple)
  - ajouter les informations que l'on juge importantes de la meilleure manière possible (notes ou notation personnelle)



Quelques soucis ...

- L’architecture est un domaine récent
  - il n'existe pas de représentation standard
  - il existe peu d'aide pour la conception architecturale
  - les méthodes de vérification/validation sont immatures



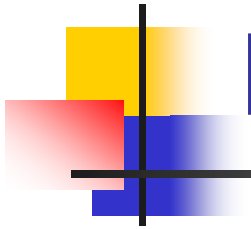
Références

- Software architecture in practice - second edition
  Len Bass, Paul Clements, Rick Kazman
  Addison Wesley, 2003

- Pattern-oriented software architecture
  Buschmann, Meunier, Rohnert, Sommerlad, Stal
  Wiley, 1996

- Applied software architecture
  Hofmeister, Nord, Soni
  Addison Wesley, 2000

- Design and use of software architectures
  Jan Bosch
  Addison Wesley, 2000



Référence

- Documenting software architectures
  Paul Clements et al
  Addison Wesley
  2002