Gestion de projets informatiques complexes

une brève histoire du management des systèmes complexes en environnement distribué, racontée aux petits enfants.

«Tout problème simple a une solution complexe…
… qui ne fonctionne pas».
O.Lockert

Antoine CELIER

Projets informatiques

ARCHITECTURE MULTI TIERS
Rien n'est simple
Référence : CARAPACE-164c Version 11/28/2018

Diviser pour mieux régner

Les modèles d'architecture
- Le modèle conventionnel
- Le modèle des 4+1 vues

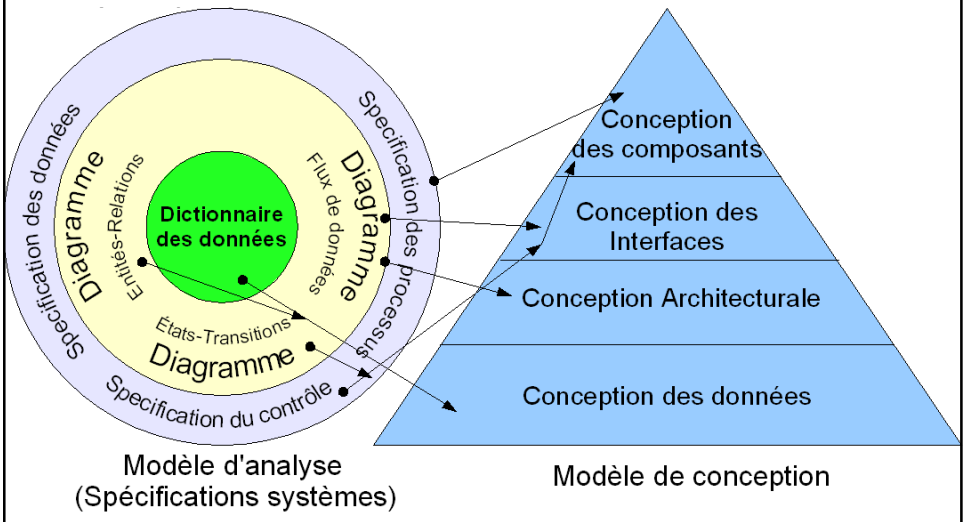
Les styles architecturaux
Les architectures N-tiers

1 Les modèles d'architecture

- **La description d'un système complexe**
  - Tel un logiciel informatique, peut être faite selon plusieurs points de vue différents mais chacun obéit à la formule de Perry et Wolf:

    architecture = éléments + formes + motivations
  - Selon le niveau de granularité, les éléments peuvent varier en tailles (lignes de code, procédures ou fonctions, modules ou classes, paquetages ou couches, applications ou systèmes informatiques), ils peuvent varier en raffinement (ébauche, solution à améliorer ou solution finale) et en abstraction (idées ou concepts, classes ou objets, composants logiciels).
  - Les éléments peuvent également posséder une *temporalité* (une existence limitée dans le temps) et une *localisation* (une existence limitée dans l'espace).

1.1 Le modèle conventionnel



Modèle d'analyse
(Spécifications systèmes)

Modèle de conception

1.1 Le modèle conventionnel

- **Le modèle conventionnel**
  - Ce diagramme décrit, à gauche, les spécifications systèmes qui sont également représentées par des diagrammes (Entités-Relations, Flux de données, États-Transitions).
  - Et à droite, nous avons les différentes activités de conception prenant comme intrants les livrables de la phase d'analyse.
  - Nous voyons que l'architecture logicielle traditionnelle nécessiterait de produire au moins quatre vues distinctes : une architecture des données (conception des données), une architecture fonctionnelle et/ou modulaire (conception architecturale), une autre architecture fonctionnelle et/ou modulaire pour les interfaces utilisateurs (conception des interfaces) et une architecture détaillée (ordinogrammes, états-transitions) des différents modules (conception des composants).

1.1 Le modèle conventionnel

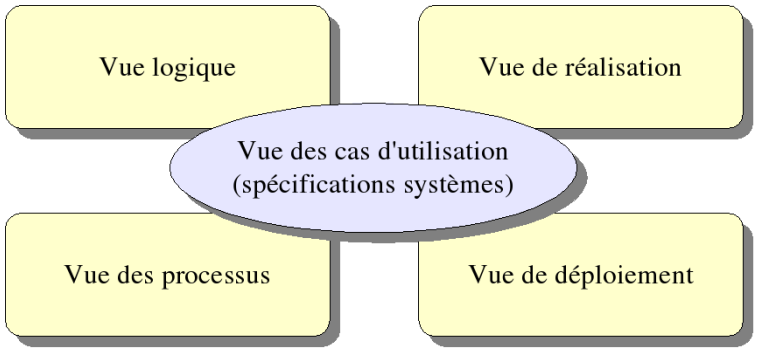
- **La pyramide**
  - Exprime que chaque couche est bâtie sur la précédente.
  - En effet, les composants réalisant les fonctionnalités du logiciel doivent manipuler des éléments de données qui doivent donc être préalablement décrits.
  - De même, les composants réalisant les interfaces utilisateurs doivent utiliser les fonctionnalités du logiciel préalablement décrites.
  - Et finalement, la création de l'architecture détaillée de chacun des composants du logiciel nécessite, évidemment, que ceux-ci soient préalablement inventés.
- **Ce modèle d'architecture impose une séparation claire entre les données, les traitements et la présentation.**

1.1 Le modèle conventionnel

- **Modèle d'analyse ou modèle d'architecture ?**
  - Puisque l'analyse produit également des diagrammes, il est naturel de se questionner
  - En effet, quand se termine l'analyse et quand commence l'architecture ?
  - La réponse à cette question est fort simple
    - Les éléments des diagrammes d'analyse correspondent à des éléments visibles et compréhensibles par les utilisateurs du système
    - Alors que les éléments des diagrammes d'architectures ne correspondent à aucune réalité tangible pour ceux-ci.

1.2 Le modèle des 4 + 1 vues

- **Le modèle de Kruchten dit modèle des 4 + 1 vues est celui adopté dans l'Unified Process** (*)
  - Ici encore, le modèle d'analyse, baptisé vue des cas d'utilisation, constitue le lien et motive la création de tous les diagrammes d'architecture.



(*) PU vient compléter la systémique des modèles UML

1.2 Le modèle des 4 + 1 vues

- **La vue des cas d'utilisation**
  - La vue des cas d'utilisation est un modèle d'analyse formalisé par Ivar Jacobson.
  - Un cas d'utilisation est défini comme un ensemble de scénarios d'utilisation, chaque scénario représentant une séquence d'interaction des utilisateurs (acteurs) avec le système.
  - L'intérêt des cas d'utilisation est de piloter l'analyse par les exigences des utilisateurs. Ceux-ci se sentent concernés car ils peuvent facilement comprendre les cas d'utilisation qui les concernent.

1.2 Le modèle des 4 + 1 vues

- **La vue des cas d'utilisation**
  - Cette méthode permet donc d'aider à formaliser les véritables besoins et attentes des utilisateurs; leurs critiques et commentaires étant les briques de la spécification du système.
  - L'ensemble des cas d'utilisation du logiciel en cours de spécification est représenté par un diagramme de cas d'utilisation, chacun des scénarios de celui-ci étant décrit par un ou plusieurs diagrammes dynamiques : diagrammes d'activités, de séquence, diagrammes de communication ou d'états-transitions.

1.2 Le modèle des 4 + 1 vues

- **La vue logique**
  - La vue logique constitue la principale description architecturale d'un système informatique et beaucoup de petits projets se contentent de cette seule vue.
  - Cette vue décrit, de façon statique et dynamique, le système en termes d'objets et de classes.
  - La vue logique permet d'identifier les différents éléments et mécanismes du système à réaliser.
  - Elle permet de décomposer le système en abstractions et constitue le coeur de la réutilisation. En effet, l'architecte informatique récupérera un maximum de composants des différentes bibliothèques et cadriciels (framework) à sa disposition. Une recherche active de composants libres et/ou commerciaux pourra également être envisagée.

1.2 Le modèle des 4 + 1 vues

- **La vue logique**
  - La vue logique est représentée, principalement, par des diagrammes statiques de classes et d'objets enrichis de descriptions dynamiques : diagrammes d'activités, de séquence, diagrammes de communication ou d'états-transitions.

1.2 Le modèle des 4 + 1 vues

- **La vue des processus**
  - La vue des processus décrit les interactions entre les différents processus, threads (fils d'exécution) ou tâches, elle permet également d'exprimer la synchronisation et l'allocation des objets.
  - Cette vue permet avant tout de vérifier le respect des contraintes de fiabilité, d'efficacité et de performances des systèmes multitâches.
  - Les diagrammes utilisés dans la vue des processus sont exclusivement dynamiques : diagrammes d'activités, de séquence, diagrammes de communication ou d'états-transitions.

1.2 Le modèle des 4 + 1 vues

- **La vue de réalisation**
  - La vue de réalisation permet de visualiser l'organisation des composants (bibliothèque dynamique et statique, code source...) dans l'environnement de développement.
  - Elle permet aux développeurs de se retrouver dans le capharnaüm que peut être un projet de développement informatique. Cette vue permet également de gérer la configuration (auteurs, versions...).
  - Les seuls diagrammes de cette vue sont les diagrammes de composants.

1.2 Le modèle des 4 + 1 vues

- **La vue de déploiement**
  - La vue de déploiement représente le système dans son environnement d'exécution.
  - Elle traite des contraintes géographiques (distribution des processeurs dans l'espace), des contraintes de bandes passantes, du temps de réponse et des performances du système ainsi que de la tolérance aux fautes et aux pannes.
  - Cette vue est fort utile pour l'installation et la maintenance régulière du système.
  - Les diagrammes de cette vue sont les diagrammes de composants et les diagrammes de déploiement.

2 Les styles architecturaux

- **Les styles architecturaux**
  - L'architecture logicielle, tout comme l'architecture traditionnelle, peut se catégoriser en styles
  - En effet, malgré les millions de systèmes informatiques construits de par le monde au cours des cinquante dernières années, tous se classent parmi un nombre extrêmement restreint de styles architecturaux
  - De plus, un système informatique peut utiliser plusieurs styles selon le niveau de granularité ou l'aspect du système décrit.
  - Nous ferons remarquer que, comme en architecture traditionnelle, c'est souvent par le mélange d'anciens styles que les nouveaux apparaissent.

2.1 Architecture en appels et retours

- **Architecture en appels et retours**
  - L'architecture en appels et retours est basée sur le raffinement graduel proposé par Niklaus Wirth.
  - Cette approche, également appelée décomposition fonctionnelle, consiste à découper une fonctionnalité en sous-fonctionnalités qui sont également divisées en sous-sous-fonctionnalités et ainsi de suite.
  - La devise diviser pour régner est souvent utilisée pour décrire cette démarche.

2.1 Architecture en appels et retours

- **Architecture en appels et retours**
  - Si à l'origine cette architecture était fondée sur l'utilisation de fonctions, le passage à une méthode modulaire ou objet est toute naturelle
  - La fonctionnalité d'un module ou d'un objet est réalisée par des sous-modules ou des sous-objets baptisés travailleurs (worker).
  - Le terme hiérarchie de contrôle est alors utilisé pour décrire l'extension de cette architecture au paradigme modulaire ou objet.
  - Une forme dérivée de cette architecture est l'architecture distribuée où les fonctions, modules ou classes se retrouvent répartis sur un réseau.

2.2 Architecture en couches

- **Architecture en couches**
  - La conception de logiciels nécessite de recourir à des bibliothèques.
  - Une bibliothèque très spécialisée utilise des bibliothèques moins spécialisées qui elles-mêmes utilisent des bibliothèques génériques.
  - De plus, comme nous l'avons déjà mentionné, le développement efficace de composants réutilisables nécessite de créer une bibliothèque logicielle.

2.2 Architecture en couches

- **Architecture en couches**
  - L'architecture en couches est la conséquence inéluctable d'une telle approche.
  - En effet, les nouveaux composants utilisent les anciens et ainsi de suite, la bibliothèque tend donc à devenir une sorte d'empilement de composants.
  - La division en couches consiste alors à regrouper les composants possédant une grande cohésion (sémantiques semblables) de manière à créer un empilement de paquetages de composants
  - Tous les composants des couches supérieures dépendants fonctionnellement des composants des couches inférieures.

2.3 Architecture centrée sur les données

- **Architecture centrée sur les données**
  - Dans l'architecture centrée sur les données, un composant central (SGBD, Datawarehouse, Blackboard) est responsable de la gestion des données (conservation, ajout, retrait, mise à jour, synchronisation, ...) .
  - Les composants périphériques, baptisés clients, utilisent le composant central, baptisé serveur de données, qui se comporte, en général, de façon passive (SGBD, Datawarehouse).
  - Un serveur passif ne fait qu'obéir aveuglément aux ordres alors qu'un serveur actif (Blackboard) peut notifier un client si un changement aux données qui le concerne se produit.

2.3 Architecture centrée sur les données

- **Architecture centrée sur les données**
  - Cette architecture sépare clairement les données (serveurs) des traitements et de la présentation (clients) et permet ainsi une très grande intégrabilité, en effet, des clients peuvent être ajoutés sans affecter les autres clients.
  - Par contre, tous les clients sont dépendants de l'architecture des données qui doit rester stable et qui est donc peu extensible.
  - Ce style nécessite donc un investissement très important dans l'architecture des données.
  - Les datawarehouses et les bases de données fédérées sont des extensions de cette architecture.

2.4 Architecture en flot de données

- **Architecture en flot de données**
  - L'architecture en flot de données est composée de plusieurs composants logiciels reliés entre eux par des flux de données.
  - L'information circule dans le réseau et est transformée par les différents composants qu'elle traverse.
  - Lorsque les composants se distribuent sur une seule ligne et qu'ils ne font que passer l'information transformée à leur voisin, on parle alors d'architecture par lot (batch).
  - Si les composants sont répartis sur un réseau informatique et qu'ils réalisent des transformations et des synthèses intelligentes de l'information, on parle alors d'architecture de médiation. Les architectures orientées évènements font également partie de cette catégorie.

2.5 Architecture orientée objets

- **Architecture orientée objets**
  - Les composants du système (objets) intègrent des données et les opérations de traitement de ces données.
  - La communication et la coordination entre les objets sont réalisées par un mécanisme de passage de messages.
  - L'architecture orientée objets est souvent décrite par les trois piliers : encapsulation, héritage et polymorphisme.
  - L'encapsulation concerne l'architecture détaillée de chaque objet, les données étant protégées d'accès direct par une couche d'interface.
  - De plus, les sous-fonctions, inutiles pour utiliser l'objet, sont masquées à l'utilisateur de l'objet.

2.5 Architecture orientée objets

- **Architecture orientée objets**
  - L'héritage permet d'éviter la redondance de code et facilite l'extensibilité du logiciel, les fonctionnalités communes à plusieurs classes d'objets étant regroupées dans un ancêtre commun.
  - Le polymorphisme permet d'utiliser des objets différents (possédant des comportements distincts) de manière identique, cette possibilité est réalisée par la définition d'interfaces à implémenter (classes abstraites).

2.6 Architecture orientée agents

- **Architecture orientée agents**
  - L'architecture orientée agents correspond à un paradigme où l'objet, de composant passif, devient un composant projectif :
  - En effet, dans la conception objet, l'objet est essentiellement un composant passif, offrant des services, et utilisant d'autres objets pour réaliser ses fonctionnalités
  - L'architecture objet n'est donc qu'une extension de l'architecture en appels et retours, le programme peut être écrit de manière à demeurer déterministe et prédictible.

2.6 Architecture orientée agents

- **Architecture orientée agents**
  - L'agent logiciel, par contre, utilise de manière relativement autonome, avec une capacité d'exécution propre, les autres agents pour réaliser ses objectifs
  - Il établit des dialogues avec les autres agents, il négocie et échange de l'information, décide à chaque instant avec quels agents communiquer en fonction de ses besoins immédiats et des disponibilités des autres agents.

3 Les architectures N-tiers

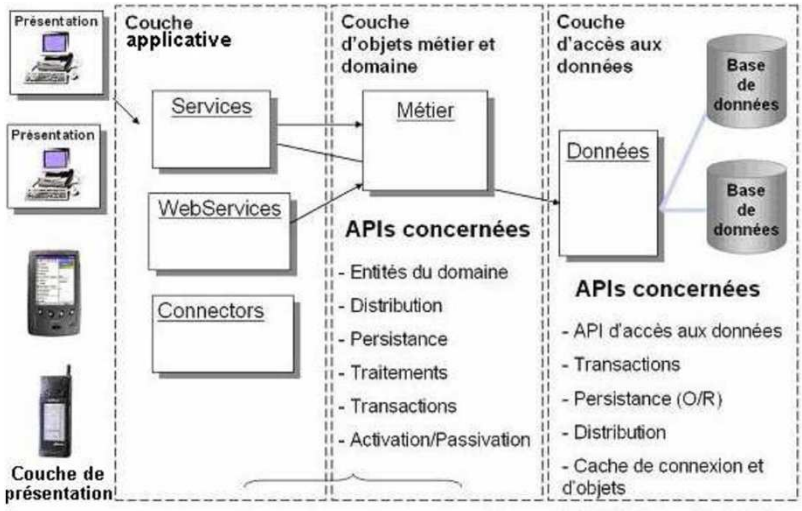
- **L'architecture N-tier (anglais tier : étage, niveau)**
  - Encore appelée multi-tier
  - C'est une architecture client-serveur dans laquelle une application est exécutée par plusieurs composants logiciels distincts.
- **Exemple d'architecture 3-tier :**
  - Tier de présentation : interfaces utilisateurs sur un PC poste de travail, qui s'adressent à des applications serveur
  - Tier des règles de gestion : applications serveur qui contiennent la logique de gestion et accèdent aux données stockées dans des bases de données
  - Tier de base de données : serveurs de bases de données

3 Les architectures N-tiers

- **Avantages des architectures N-tier :**
  - Le lien entre les niveaux est défini et limité à des interfaces
  - Les interfaces assurent la modularité et l'indépendance technologique et topologique de chaque niveau

3 Les architectures N-tiers

- **Les diifférentes couches d'une architecture 4-tier :**



3 Les architectures N-tiers

- **Les différentes couches d'une architecture 4-tier :**
  - La **couche de présentation** contient les différents types de clients, léger (ASP, JSP) ou lourd (Applet)
  - La **couche applicative** contient les traitements représentant les règles métier (créer un compte de facturation, calculer un amortissement ...)
  - La **couche d'objets métier** est représentée par les objets du domaine, c'est à dire l'ensemble des entités persistantes de l'application (Facture, Client ...)
  - La **couche d'accès aux données** contient les usines d'objets métier, c'est à dire les classes chargées de créer des objets métier de manière totalement transparente, indépendamment de leur mode de stockage (SGBDR, Objet, Fichiers, ...)

3 Les architectures N-tiers

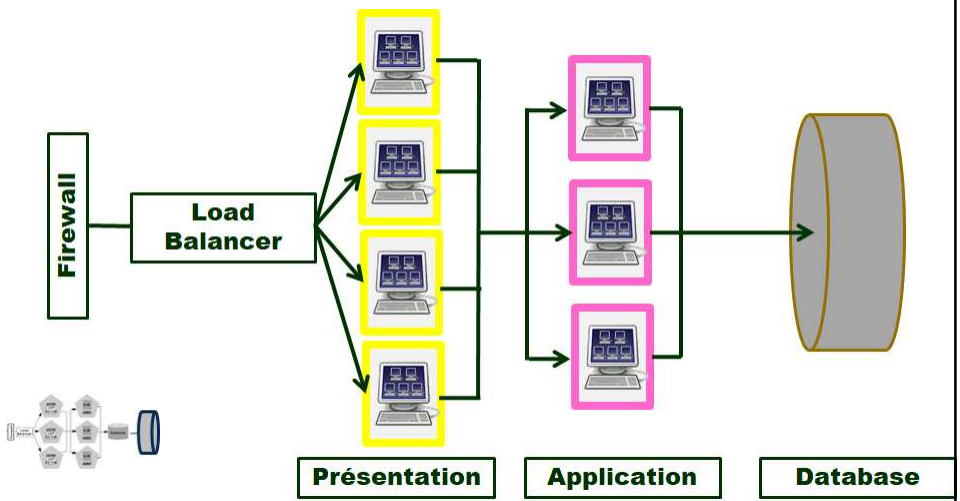
- **La valeur ajoutée des architectures n-tier :**
  - Cette séparation par couches de responsabilités sert à découpler au maximum une couche de l'autre afin d'éviter l'impact d'évolutions futures de l'application.
  - Par exemple :
    - Si l'on est amené à devoir changer de base de données relationnelle
    - Seule la couche d'accès aux données sera impactée
    - La couche de service et la couche de présentation ne seront pas concernées car elles auront été découplées des autres.

3 Les architectures N-tiers

- **Une architecture N-tiers favorise:**
  - Une architecture multi-tenants.
  - Une haute disponibilité
  - La gestion de la sécurité
  - La réduction du temps d'exécution
    - Du redéploiement d'une machine virtuelle
    - Du backup et du restore
  - Mais également…

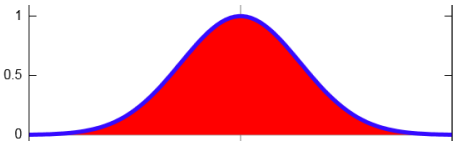
3 Les architectures N-tiers

- **La qualité de l'architecture, par la prise en compte native du design pattern MVC**



3 Les architectures N-tiers

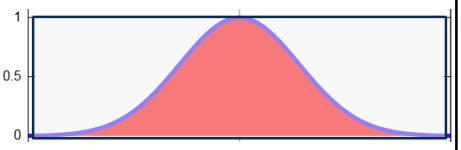
- Une réduction du coût de déploiement



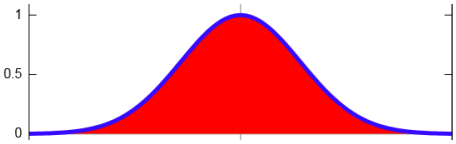
Users connectés au service = f(t)

3 Les architectures N-tiers

- Une réduction du coût de déploiement



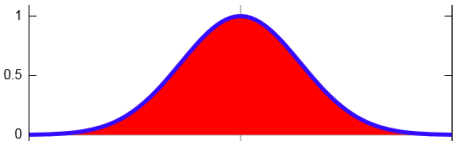
1 grande machine - Coût = 100



Users connectés au service = f(t)

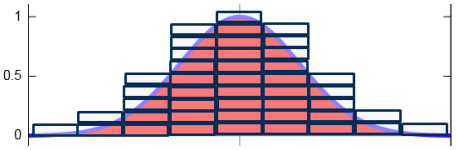
3 Les architectures N-tiers

- Une réduction du coût de déploiement



Users connectés au service = f(t)

1 grande machine - Coût = 100



Plein de petites machines – Coût = 50

3 Les architectures N-tiers

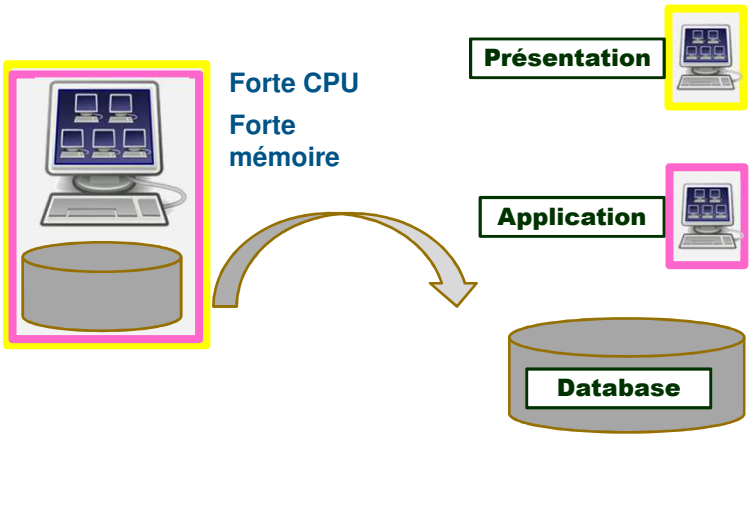
- Adéquation machine virtuelle - Tier

Forte CPU

Forte mémoire

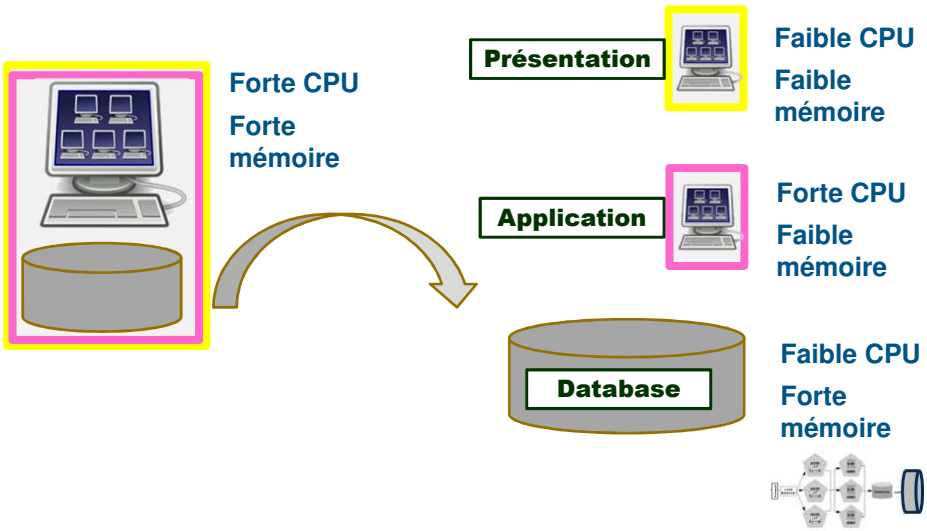
3 Les architectures N-tiers

- Adéquation machine virtuelle - Tier



3 Les architectures N-tiers

- Adéquation machine virtuelle - Tier



3 Les architectures N-tiers

- **Adéquation machine virtuelle - Tier**

Forte CPU

Forte mémoire

Présentation

Faible CPU

Faible mémoire

Application

Forte CPU

Faible mémoire

Database

Faible CPU

Forte mémoire

« le premier € de bénéfice est l'€ économisé »

Aphorisme de Charles Edelstenne's