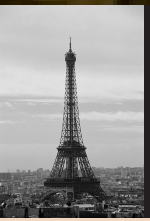


SUNGARD CADEXTAN

L'informatique qui réinvente la finance

Arnaud Nauwynck

Modélisation Objet de Librairie de Pricing

Partie 4 : Informatique
Conception Orientée-Objet d'Instruments, Langage de description d'Instruments

SUNGARD

Plan Général : Partie 1, 2, 3, **4**

- Partie 1 : Mathématique
  - Définition Instrument et Valorisation
  - Equation différentielle de Pricing
- Partie 2: Algorithmique
  - Méthode Numériques, Différences Finies
  - Algorithme de Décomposition d'Options
- Partie 3: Informatique
  - Conception des couches de librairie de Pricing
  - Plan d'exécution, SPI : Slice, API: Greeks/RA
- **Partie 4: Informatique**
  - **...**

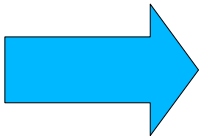
Plan

- Rappel
- Payoff, classes d'Expressions Mathématiques
- Design Pattern Visitor
- Hierarchie de classe (AST) d'Instruments
- Comparaison Langage LexifiML

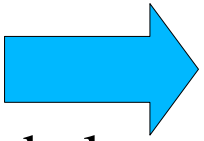
SUNGARD

Rappel : Instruments et Pricing

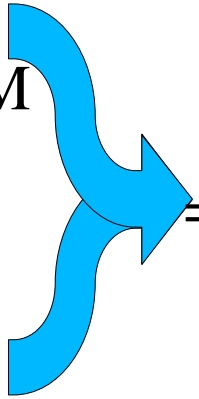
Choix
Modèle de Diffusion
ex: Black-Scholes



Calcul de Ito
EDP de Pricing
ex: BS => eq de la chaleur

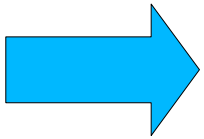


Choix
Méthode Numérique
ex: DF, FEM



=V

Définition d'un Instrument
(combinaisions de briques de bases = sum, pay, option, trigger..)

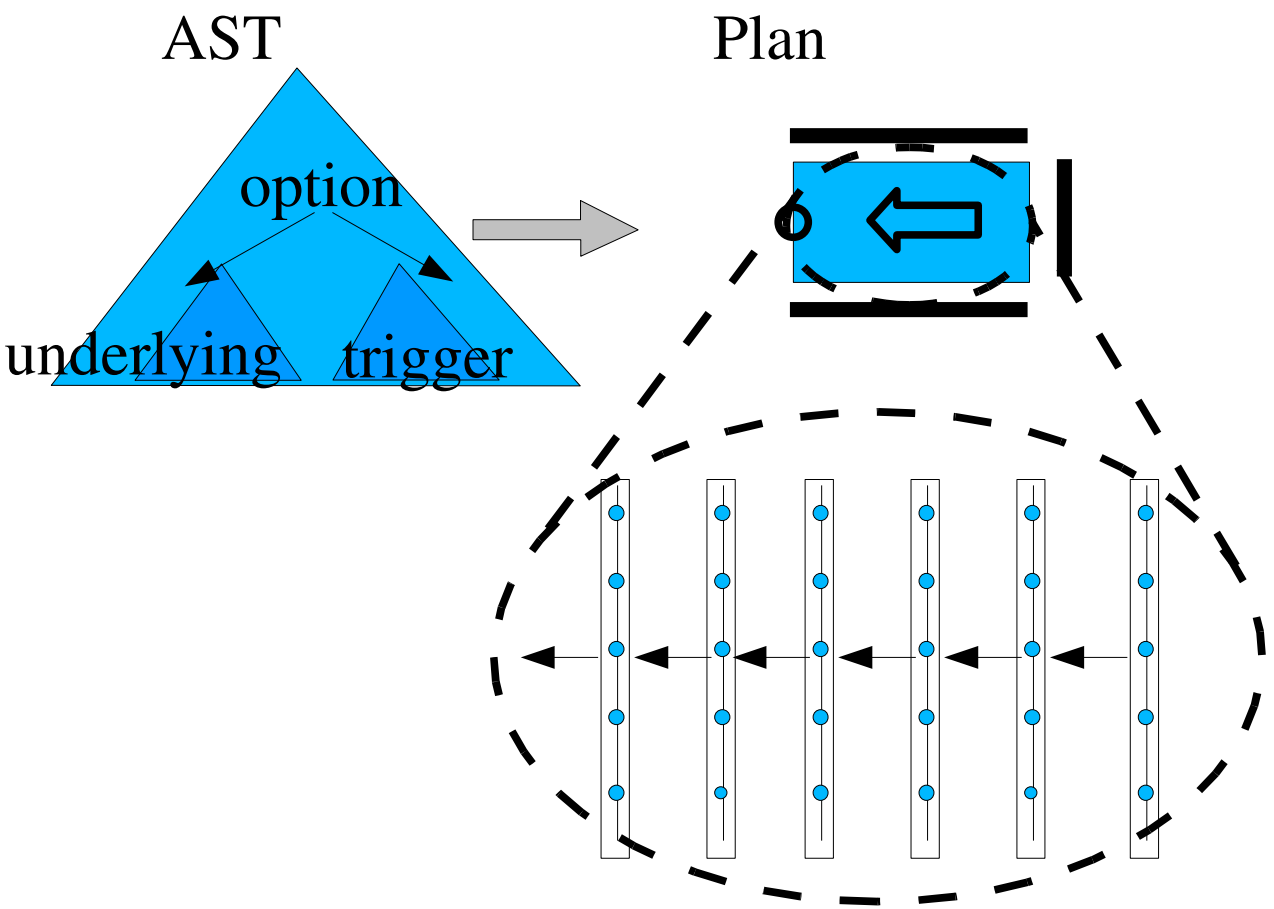


Plan Algébrique de Pricing
path dep => Monte-Carlo
backward =>
Domaine Géométrique

Rappel : Plan d'Exécution, Vue Microscopique

- Discrétisation de l'EDP de diffusion en pas de temps
  - cf. Méthodes numériques

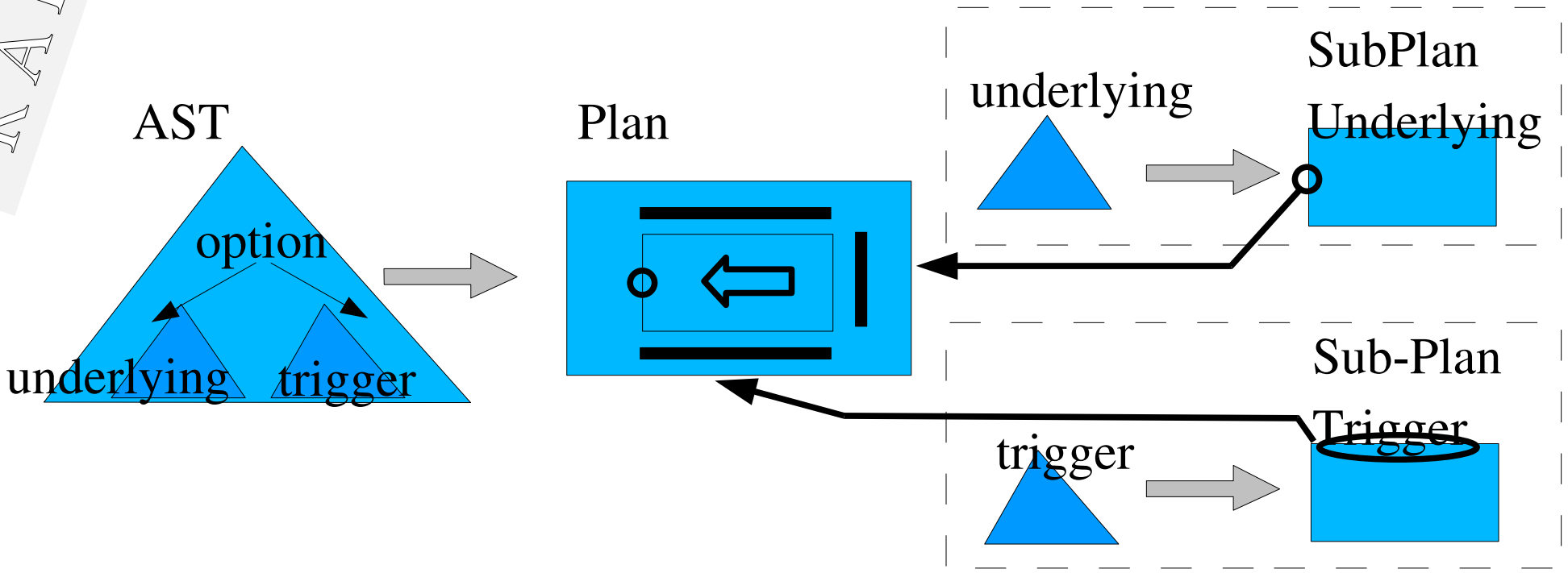
RAPPEL



Rappel : Plan d'Exécution, Vue Macroscopique

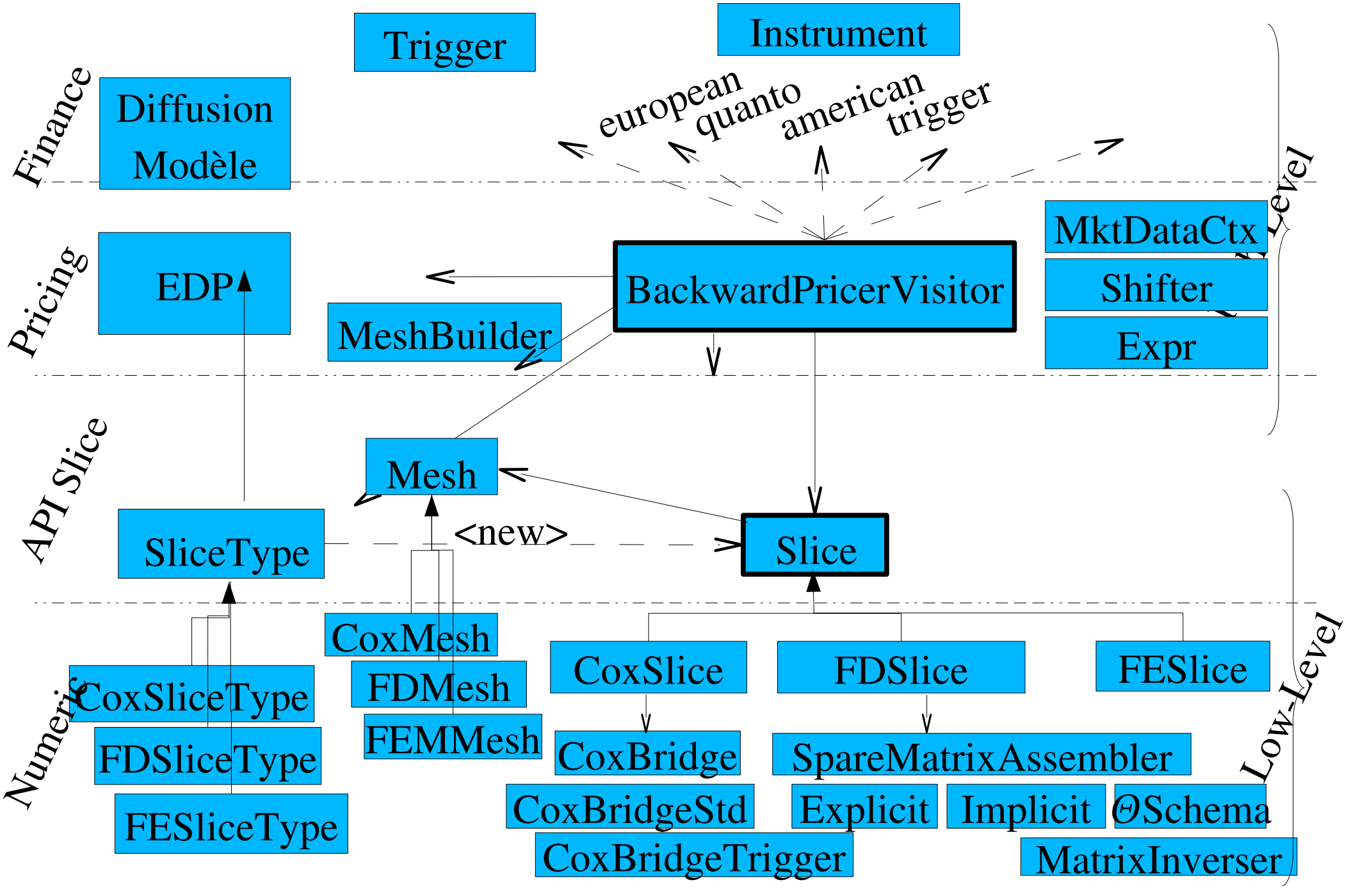
- La décomposition est un calcul récursif
  - pour les conditions initiales = les sous-jacents
  - pour les conditions limites = les triggers, rebates
  - pour les transitions = activations d'événements

RAPPEL



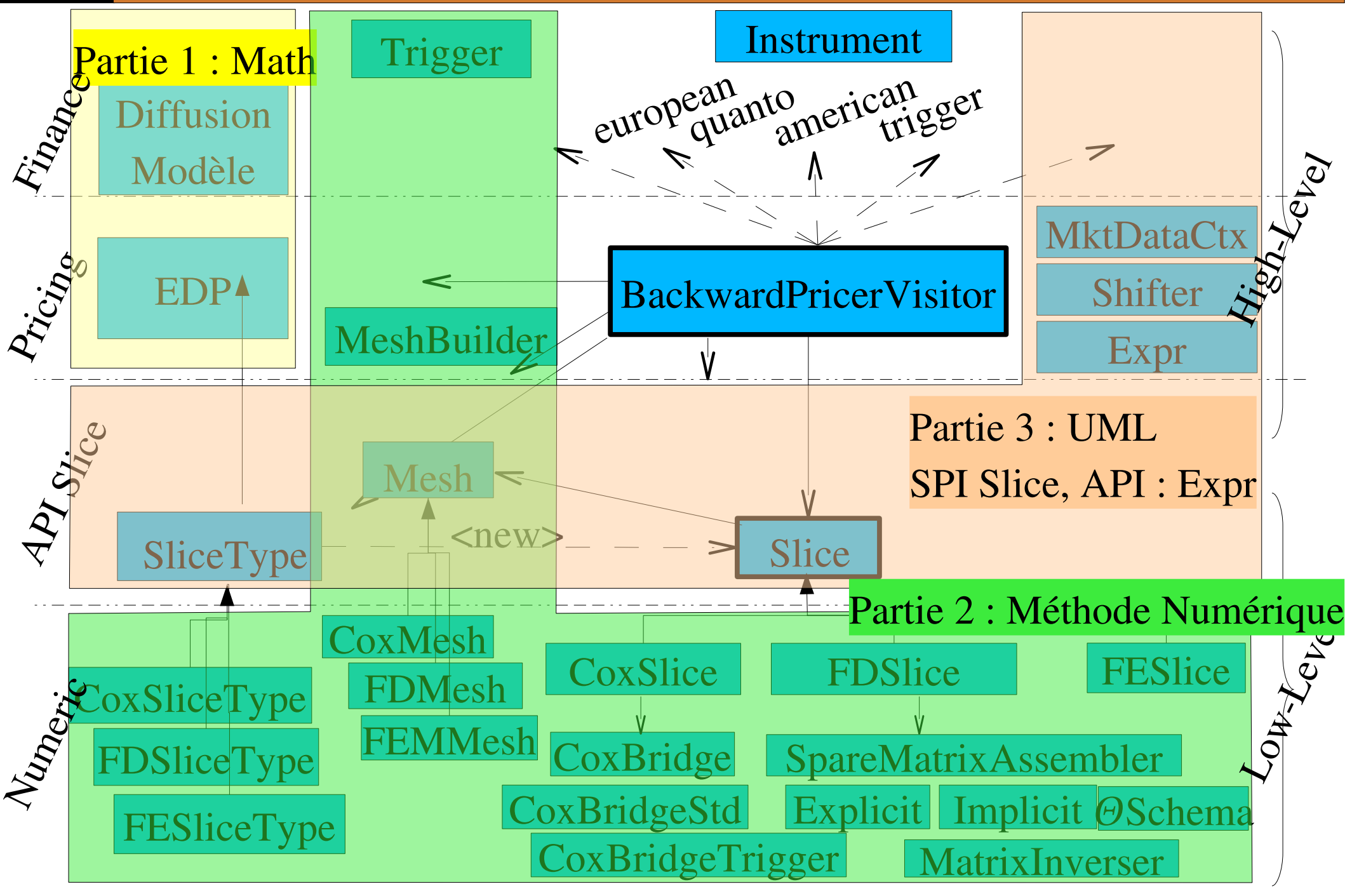
SUNGARD

Rappel : Diagrame UML



SUNGARD

Rappel : Séparation des parties 1,2,3,4



Plan

- Rappel
- Payoff, classes d'Expressions Mathématiques
- Design Pattern Visitor
- Hierarchie de classe (AST) d'Instruments
- Comparaison Langage LexifiML

SUNGARD

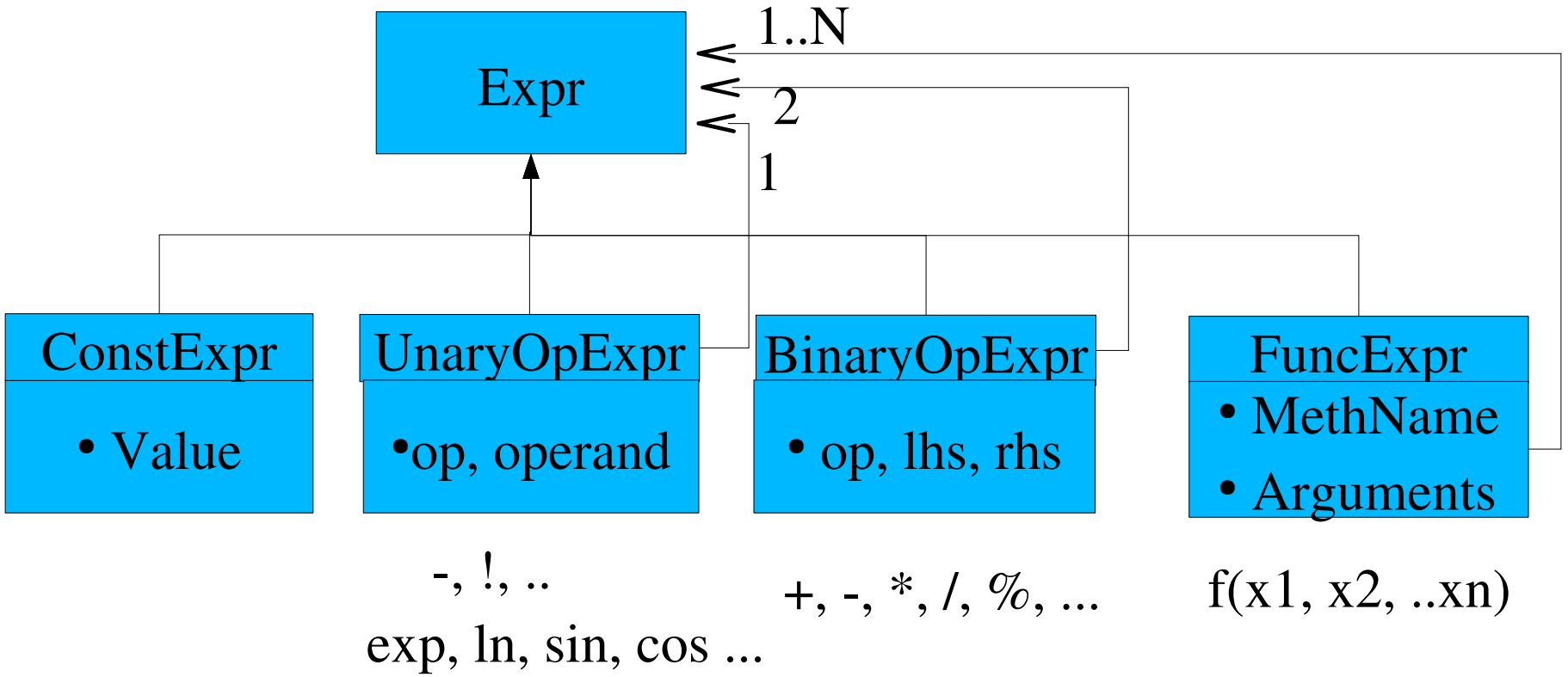
Payoff – Introduction aux Expressions

- Payoff classiques =
  - mono sous-jacent: call, put, bin...

  - combinaisions = fonction affine par morceaux
- Plus Généralement :
  - quanto, multi-sous jacent... $(|S_1-K|+.S_2)$
  - profil = fonction quelconque...
- Besoin de notations Mathématiques / d'Expressions Algébriques

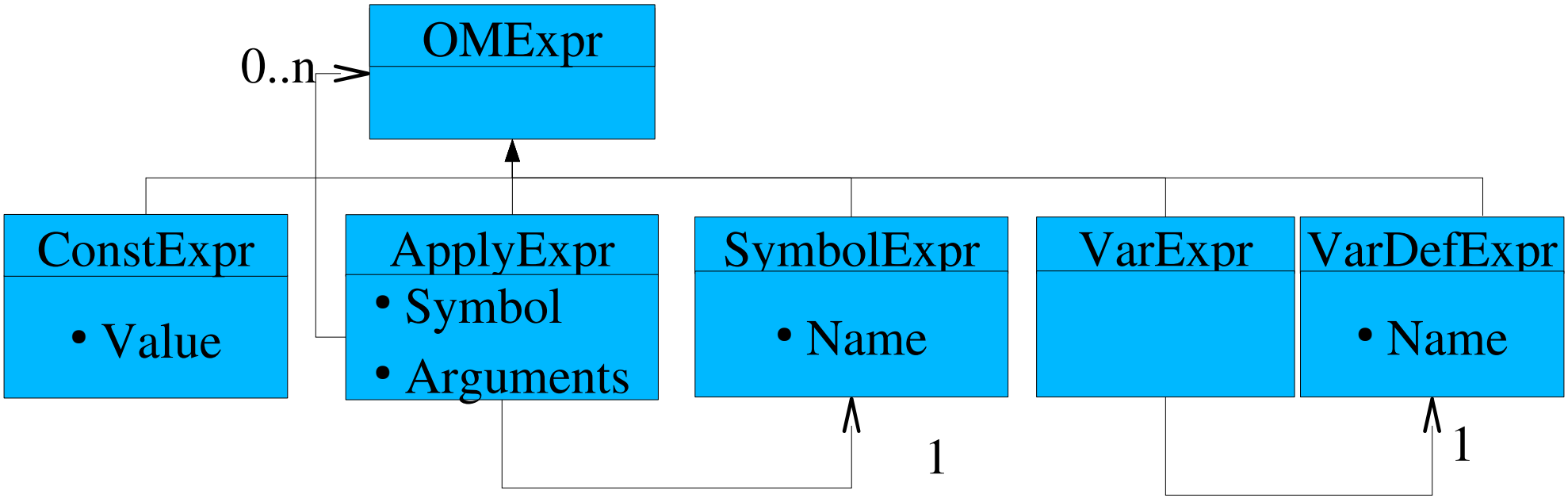
Classes d'Expressions Mathématiques

- 1ere approche... UnaryOpExpr, BinaryOpExpr, FuncExpr



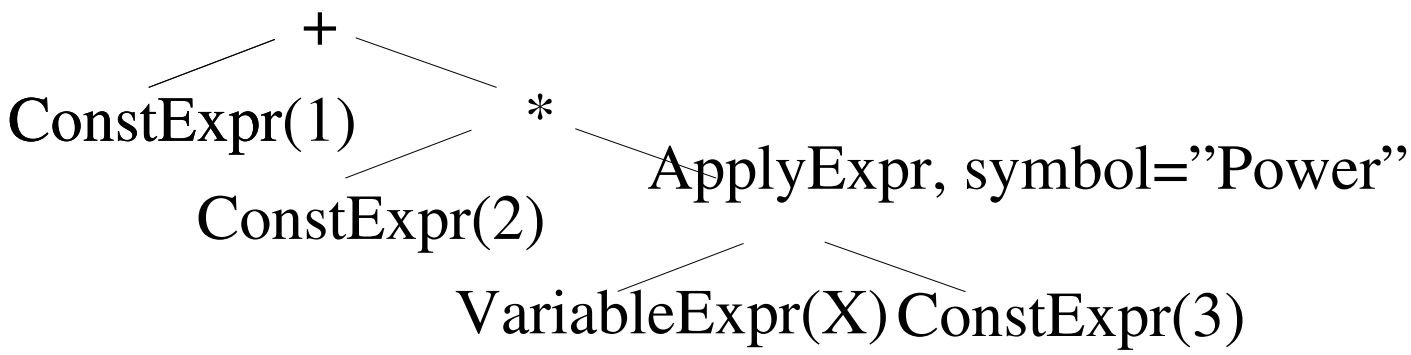
CAS (Computer Algebra System)

- Extension:
  - Apply instead of (Unary/Binary/NAry)
  - Rajout de Variables (Use / Def)
  - Rajout de Symboles (cf dictionary)
- ... = Norme OpenMath
  - (format d'echange Mathematica-Mapple-...)



Hiérarchie de Classe : AST

- AST = **Abstract Syntaxic Tree**
- Lien avec la théorie des languages:
  - classe composite = noeud non terminal
    = **règle** de réduction dans la **grammaire**
  - classe simple = feuille terminale
    = **token** dans le language
  - CST = Concrete Syntaxic Tree (ex. “( )”, “;”, ...)
- Exemple : “ 1 + 2 * X³ “



Plan

- Rappel
- Payoff, classes d'Expressions Mathématiques
- Design Pattern Visitor
- Hierarchie de classe (AST) d'Instruments
- Comparaison Langage LexifiML

Design Pattern = Visitor

- cf. Livre “GoF” Erich Gamma
- Pattern Fondamental
- Le moins intuitif ? Le plus beau ?
- Visitor = “switch” orienté-objet
  - permet d'organiser des traitements par classes
  - permet de vérifier à la compilation les cas
- Utilisé partout pour des traitements récursifs, sur des hiérarchies d'objets
  - exemples : AST, compilateurs, frameworks...

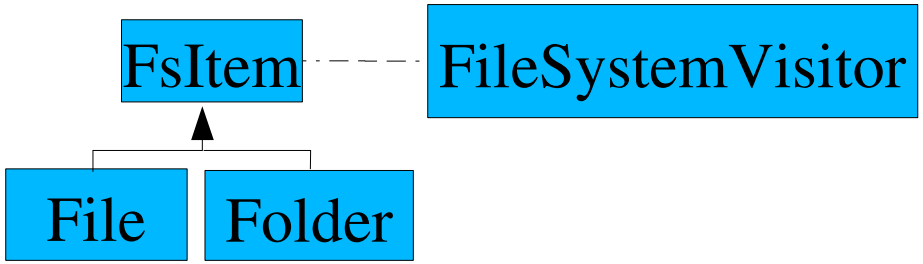
Principe du Pattern Visitor

- Appel :
  Visitor v = new VisitorXX();
  x.accept(v);
- est équivalent à :
  if (x instanceof A) { v.caseA((A)x); }
  else if (x instanceof B) { v.caseB((B)x); }
  else ...
- Principe: le switch utilise la “virtual table” :

```
class A extends Visitable {
   public void accept(Visitor v) { v.caseA(this); }
}
```

Principe du Pattern Visitor (2)

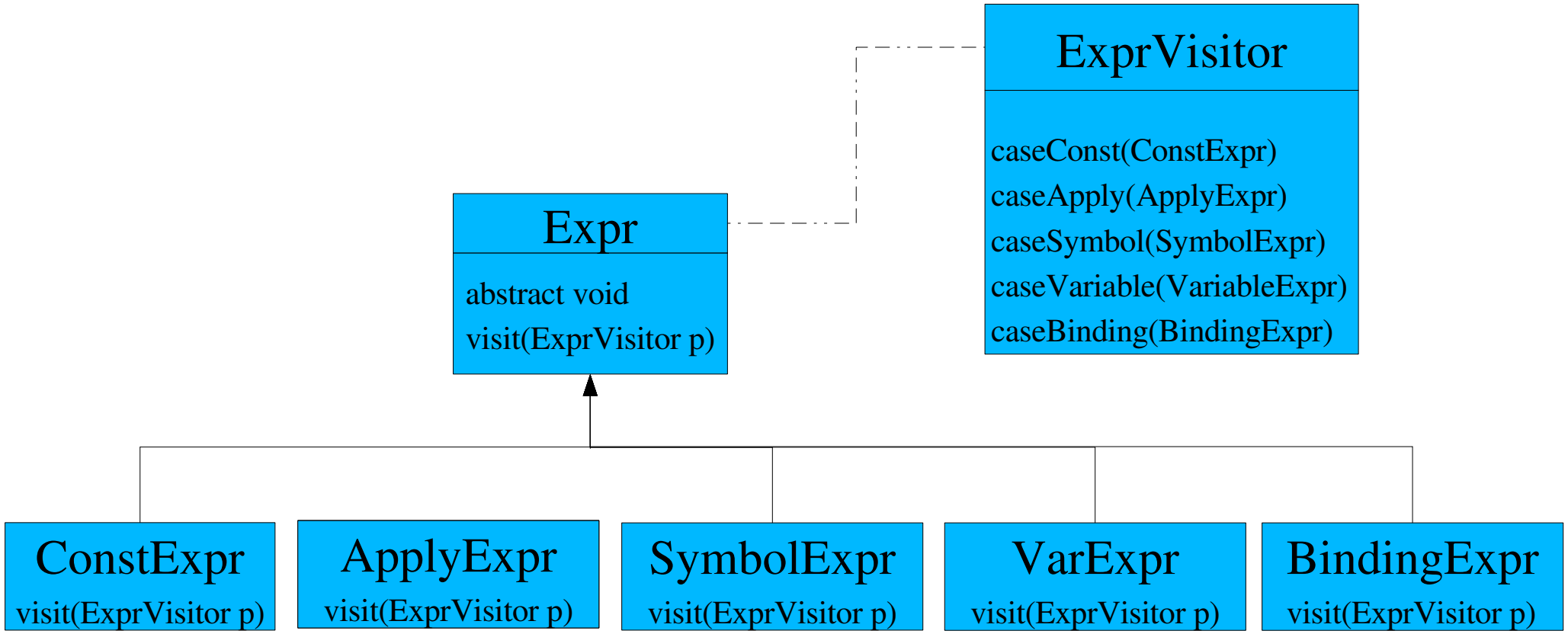
- En général, les traitements sont récursifs...
  - d'où son nom Visitor (parfois de Switcher, Scanner / TraversalIterator, Handler ...)
- Autre exemple: parcours récursif de File / Folder



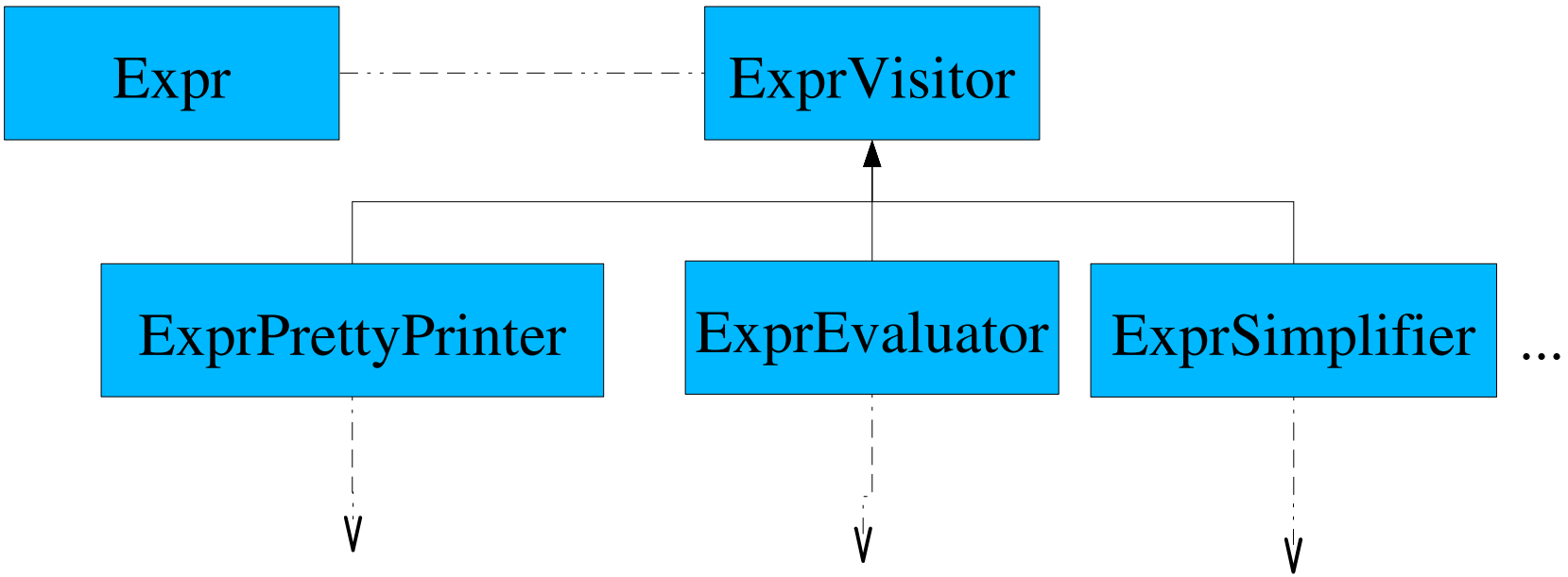

```
final List resultFiles = new ArrayList();
folder.accept(new FileSystemVisitor() {
    public void caseFile(File f) { resultFiles.add(f); }
    public void caseFolder(Folder f) { ... recurse ... ; }
});
```

SUNGARD

Visiteur pour Classes d'Expression



Exemples de Visiteurs



formattage
+ indentation
+ calcul de précédence
pour parenthèses

env. runtime
+ calcul

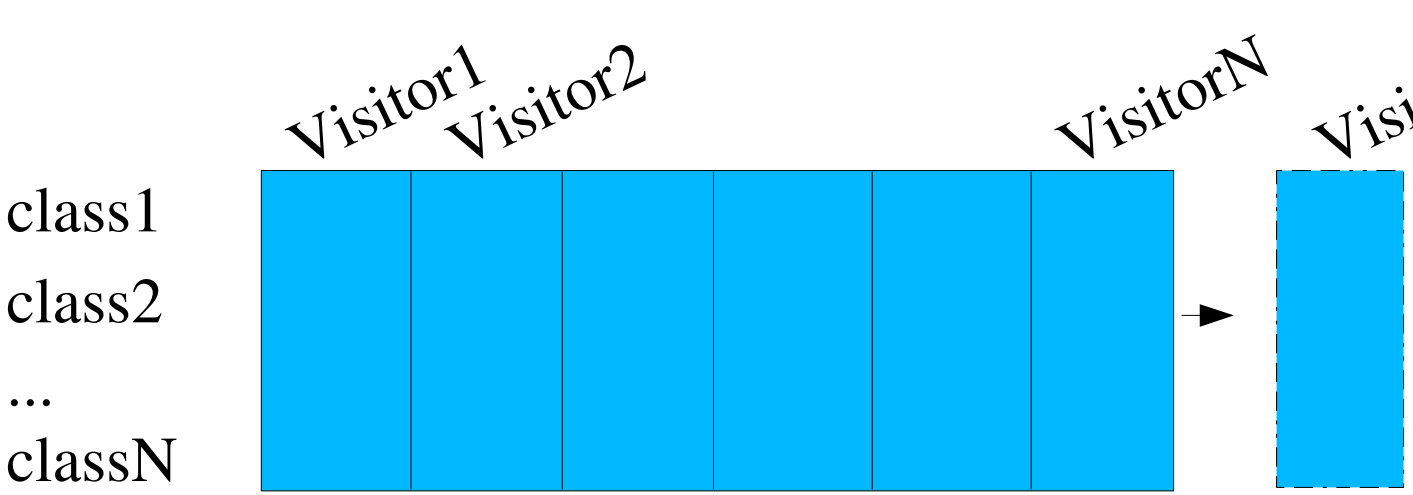
logique de “constant folding”
+ associativité
+ permutation

Exemple de Visiteur (2)

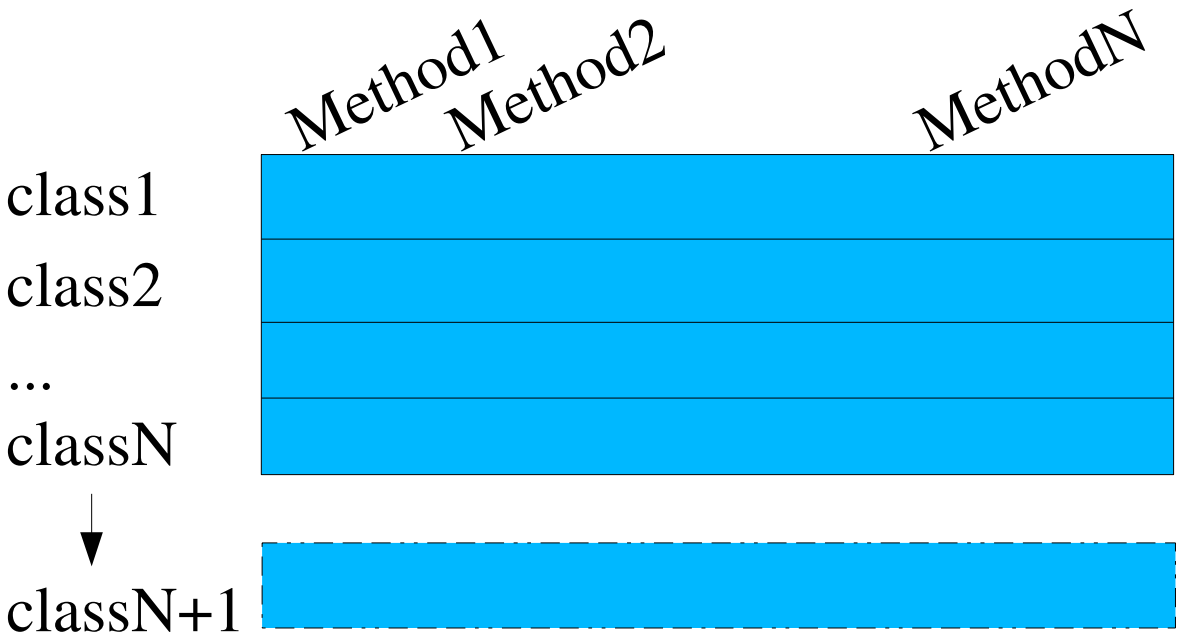
```
public class PrettyPrinter implements Visitor {
    PrintWriter output;
    public void caseConst(ConstExpr p) {
        output.print(p.getValue());
    }
    public void caseApplyExpr(ApplyExpr p) {
        output.print(p.getSymbol() + " (");
        for(Expr elt : p.getArguments()) {
            elt.accept(this);  // visit child recursively
            if (next()) output.print(", ")
        }
        output.print(")");
    }  .... }
```

SUNGARD

Visiteurs versus Méthodes Virtuelles



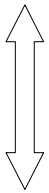
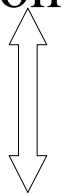
OK :
ajout de
nouvelles
fonctionalités
sans modification
de l'existant



Risqué / impossible sans
modification de l'existant

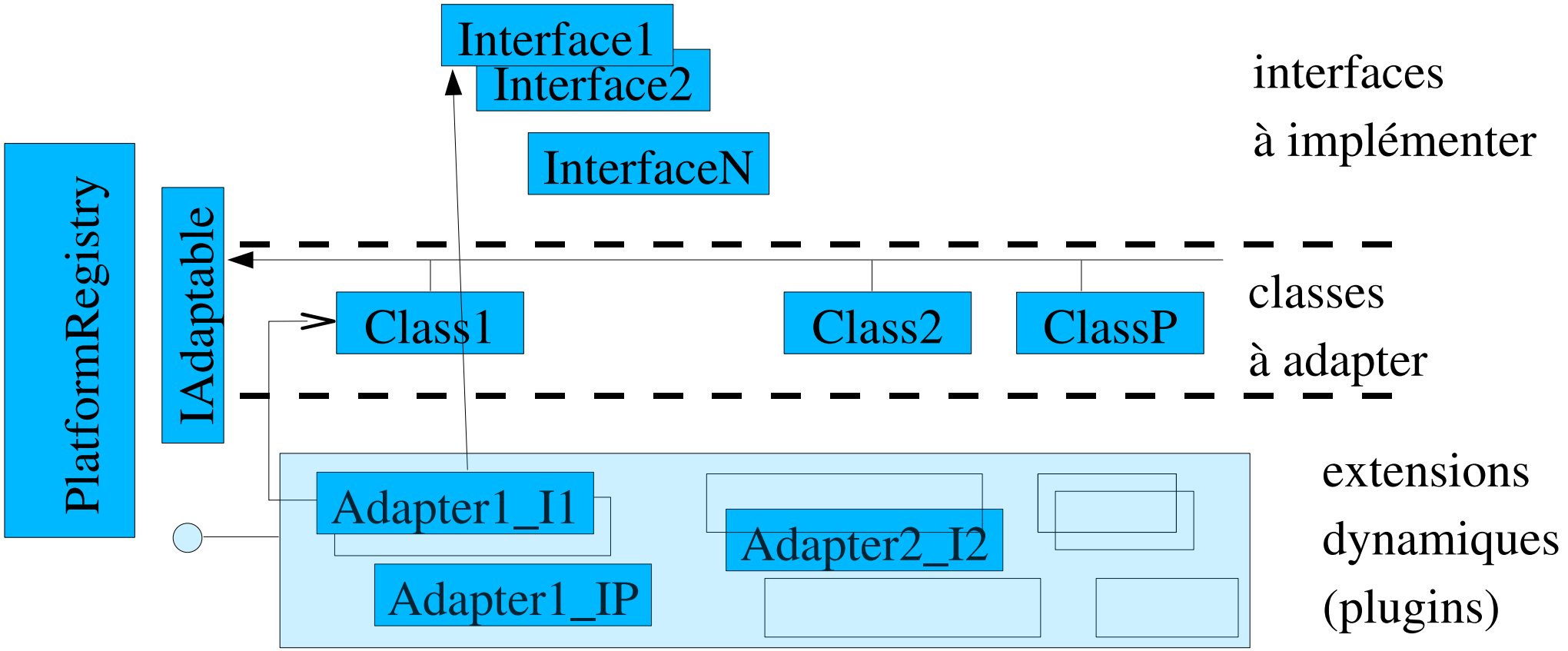
AST Visitor - Langage Fonctionnel - Math

- Les liens entre AST, Théorie des grammaires, Langages Fonctionnels et Mathématiques sont très forts...

AST	Grammaires	Langage Fonctionnels	Théorie des Constructeurs	Algèbre
				
Visitor	**transformation, grammaire attribuée** (ex: XSLT apply-match)	Ex: Caml f(x) : case x when a+b => f(a) + f(b) when c.a => c . f(a)	Déf de fonction par **induction** (~ récurrence)	fonction linéarire... qq soit a,b in E/ f(a+b)=f(a)+f(b)

Autres Approches Objet: IAdaptable

- Eclipse framework/patterns : IAdaptable
  - but : étendre des classes déjà compilées
  - moyen : enregistrer dynamiquement un Adapter
  - Registry=Map<{Class,Class}, AdapterFactory>



Détail sur IAdaptable

```
Class1 obj = ...

ISupportA supp = (ISupportA)
   PlatformRegistry.getAdapter(ISupportA.class, obj);

if (suppA != null) {

    suppA.foo();

} // else not supported


// ... instead of test hard-coded interfaces

if (obj instanceof ISupportA) {
   ISupportA suppA = (ISupportA)obj;
   suppA.foo();

} // else not hard-coded
```

Plan

- Rappel
- Payoff, classes d'Expressions Mathématiques
- Design Pattern Visitor
- Hierarchie de classe (AST) d'Instruments
- Comparaison Langage LexifiML

Règle de Conception pour Pricing

- Par analogie avec Expressions+Visitor, il faut :

  1) analyser les comportements/classes de base

  2) définir une hiérarchie de classes, SANS méthodes, mais définissant la structure de donnée

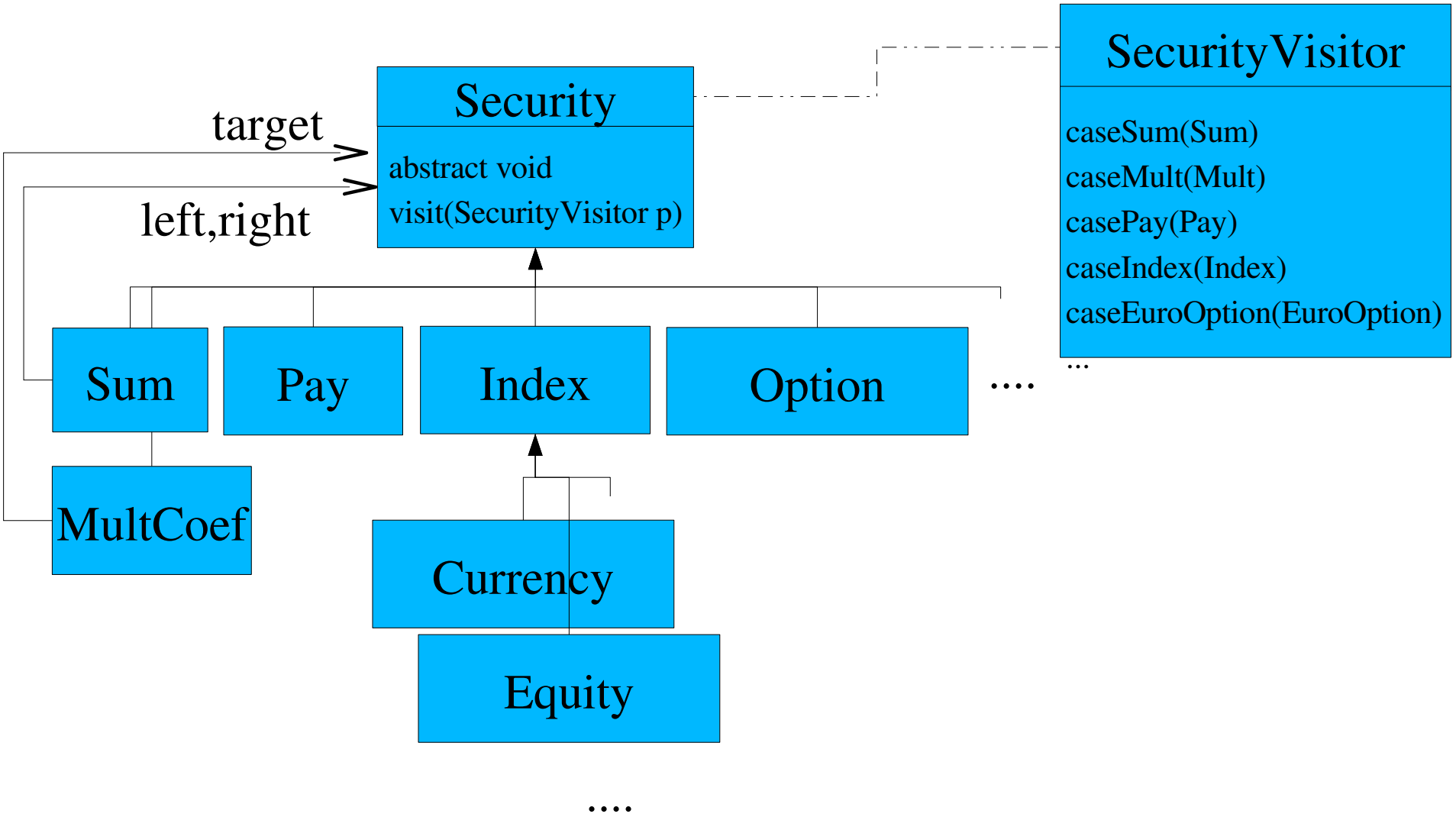
  3) Rajouter des classes techniques (Composites, Proxy, etc..)

  4) définir une interface Visiteur

  5) en dernier: implémenter les traitements

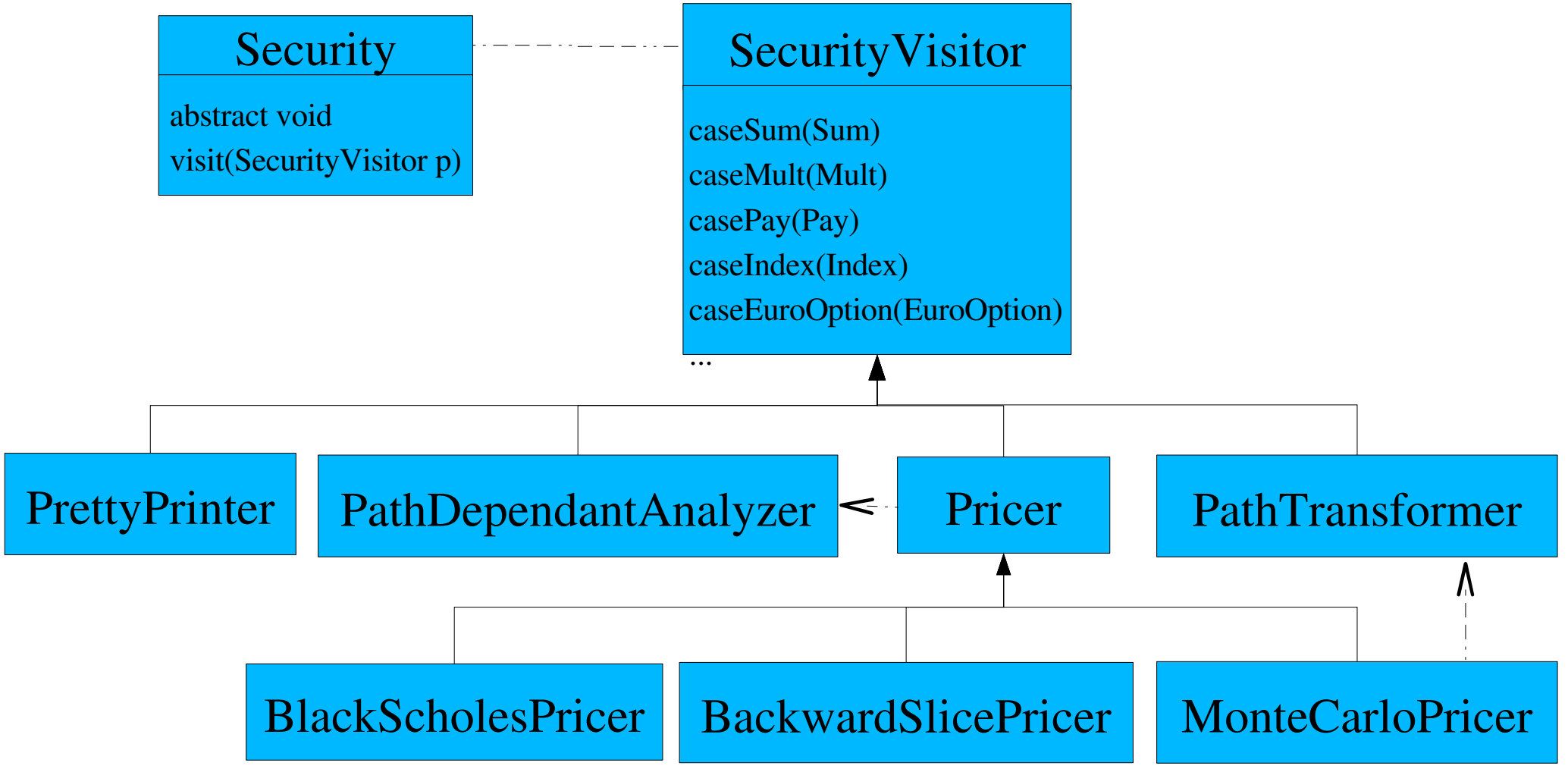
SUNGARD

Instruments et Visitor



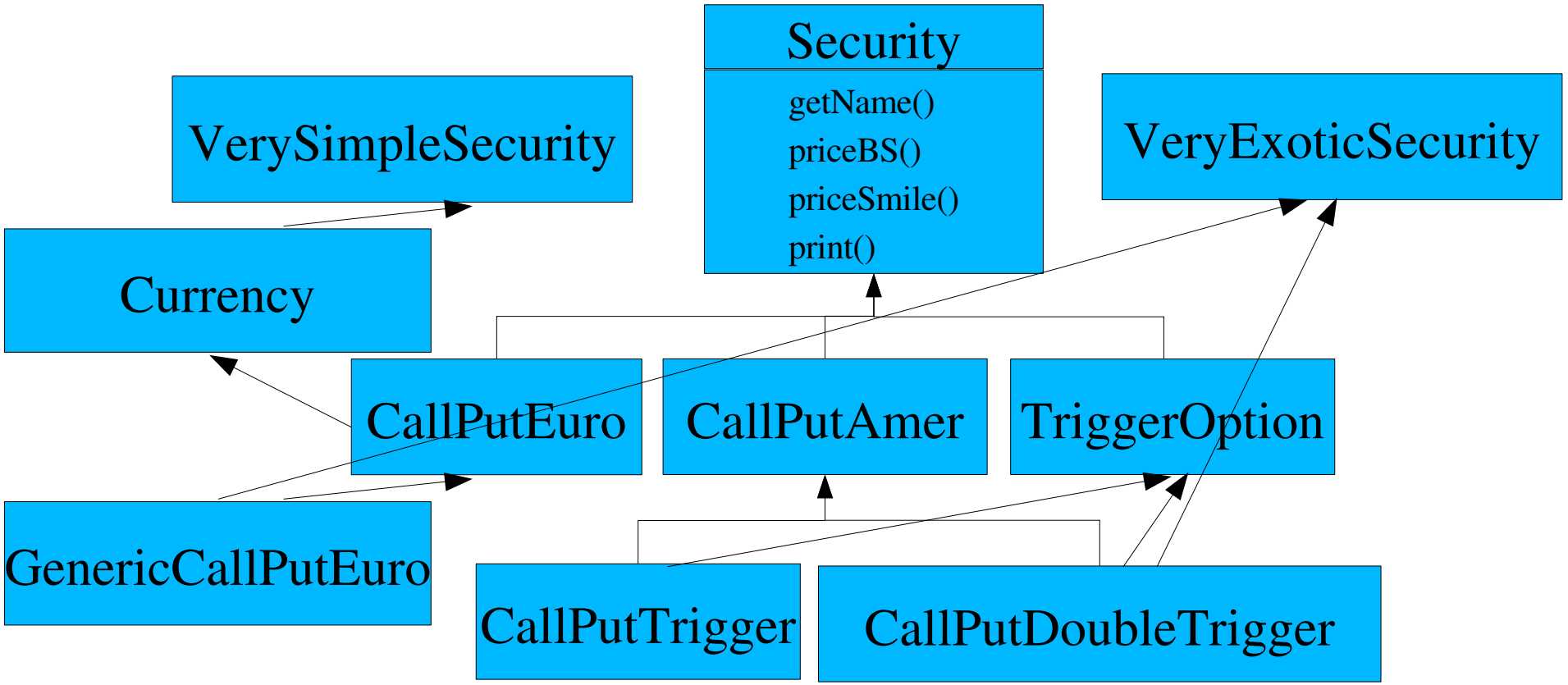
Instruments et Visitor

- Exemples de Visitor



Erreurs à ne pas faire

- Héritage de code à interdire impérativement
- Classes pas assez générales
- Classes non primitives, non nécessaires



Mauvaises Dérivations

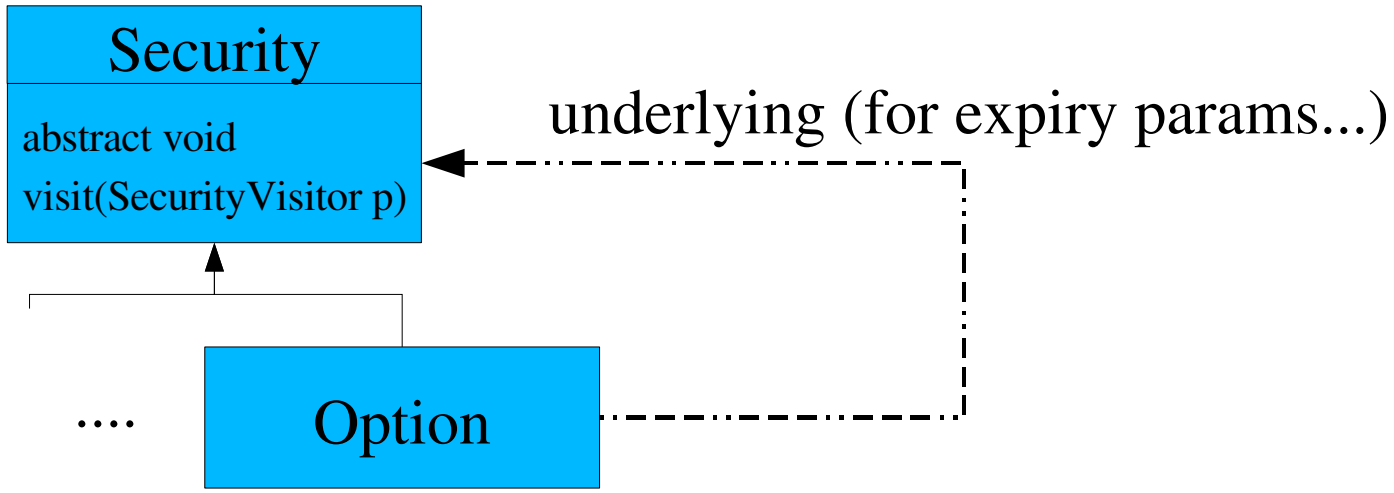
- Syndrome des débutants en Orienté-Objet
  - dérivation de classe / surcharge de méthode NON
  - dérivation d'interface + délégation = OUI
- Programmation au jour le jour, sans vision
- Pourquoi dériver des classes spaghetti ???
  - souvent : pour répondre aux question “getSecurityName()” / “getProductType()”
  - pas pour aider à pricer en tout cas !!
  - pour se brancher sur l'existant...

Visiteur vs Anti-Modularité

- Pourquoi Hard-coder dans une classe très générale des méthodes spécifiques ?
  - exemple : pricer en modèle Black-Scholes
  - pricer en modèle mono sous-jacent
- Souvent: par faignantise d'utiliser des visiteurs
- Par peur de la complexité du nombre de classes
  - FAUX, nb classes != nb de relations use/override/implements

Détail sur l'Option Européenne

- Besoin de définir toute sorte de payoffs
  - Pas seulement Call,Put,Binaire / Sous-Jacent!!
  - Ex : payoff = quanto, option sur option, ...



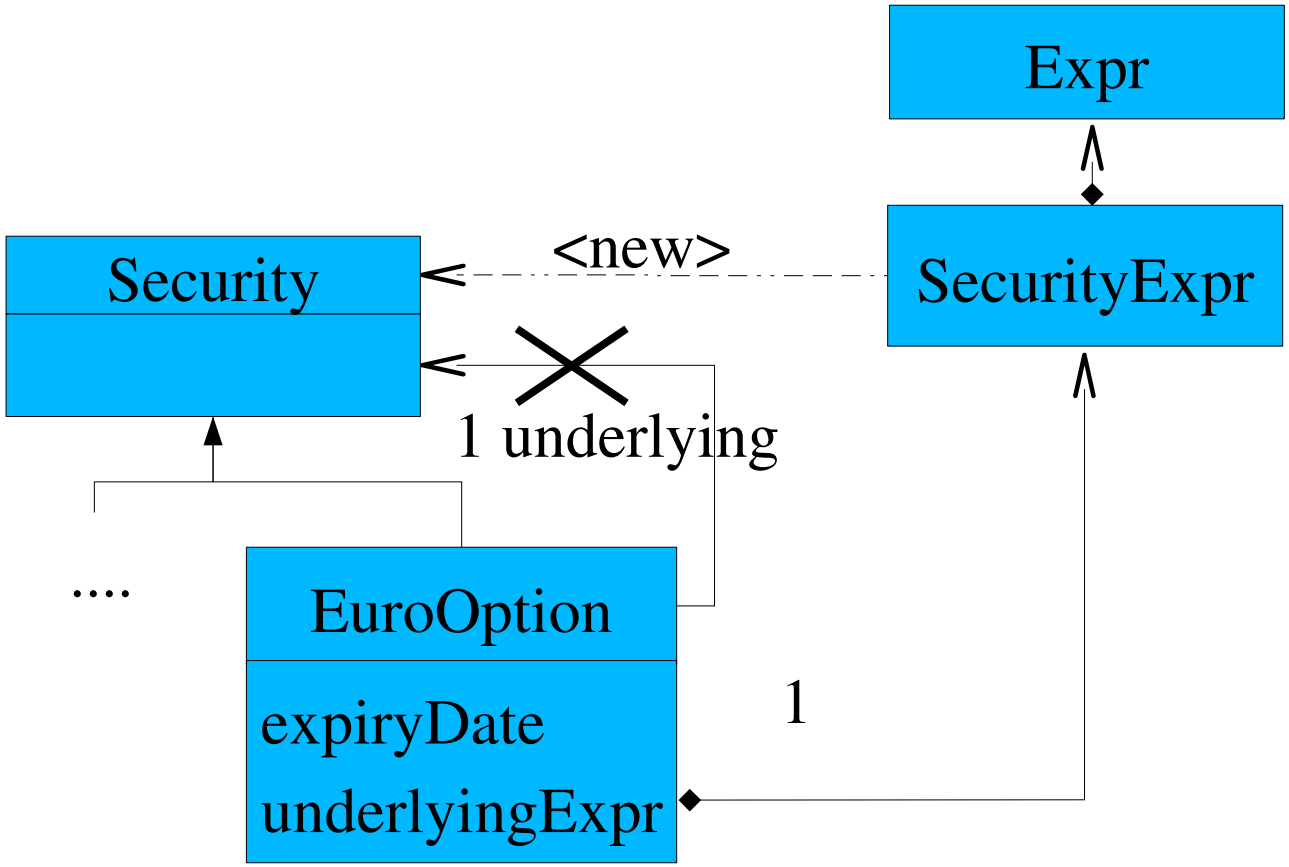
Détail sur l'Option Européenne

- Le Payoff ”doit donc” être une Expression, de type Instrument
  - flexibilité des quantités
    (fonction numérique de payoff)
  - flexibilité sur l'instrument généré
- L'option peut se transformer
  - en n'importe quel sous-jacent
  - mais à une date fixée...
    et dépendant juste des fixings à cette date

    sous-jacent = f(fixingParams(expiryDate))

SUNGARD

Option Européenne
= Option à Evénement Discret

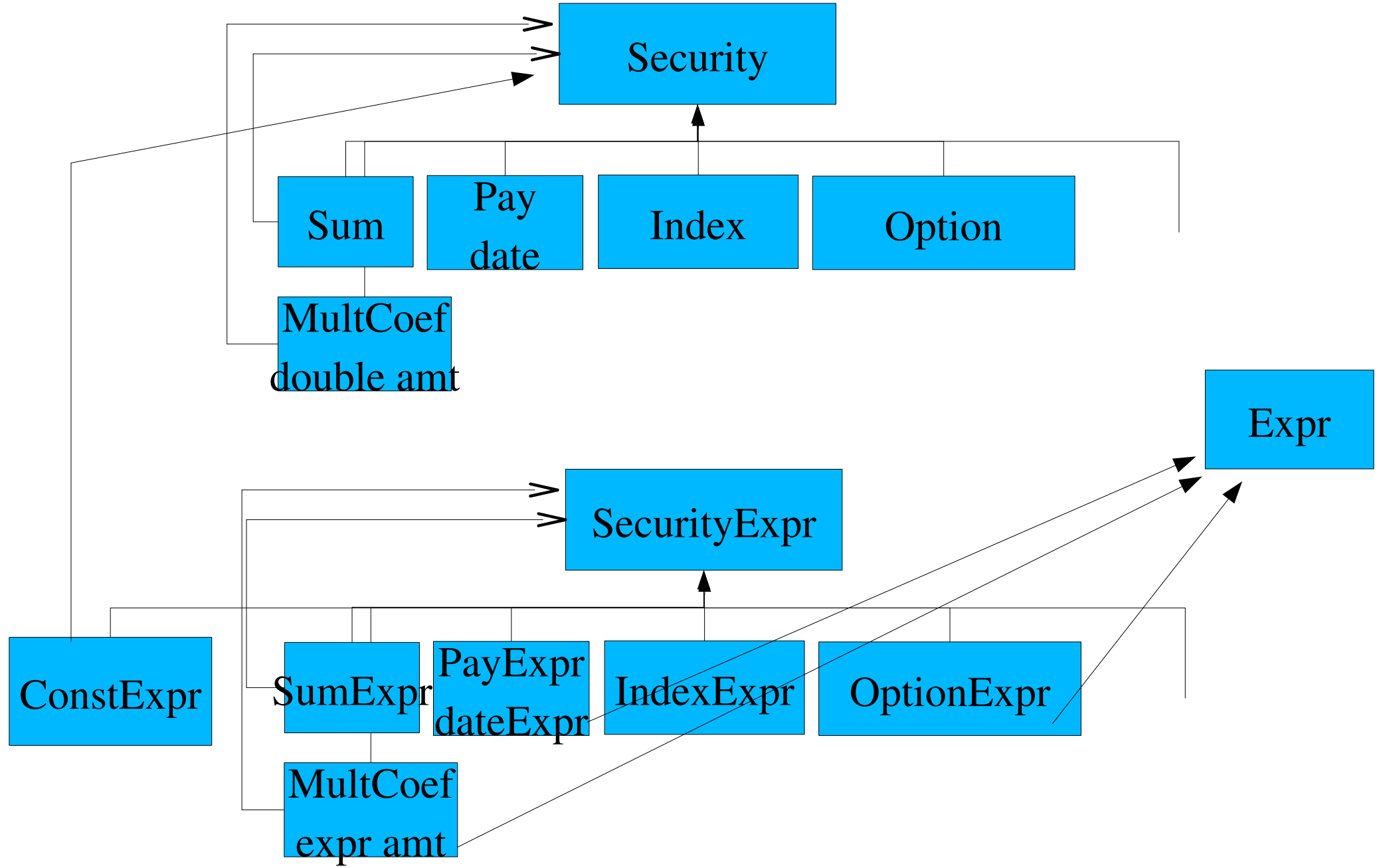


Expression d'Instrument ?

- Expression numérique... ok
- Expression d'Instrument = ? .... idem ?
- Copy&paste AST + replace value->expr !
- Copy&Paste ???
  - Pas de solution élégante en programmation classique, même orientée-objet !!
  - Avec Approche Formelle (mathematica) ... ok?
  - Avec Approche Functionnelle (camel) ... ok?

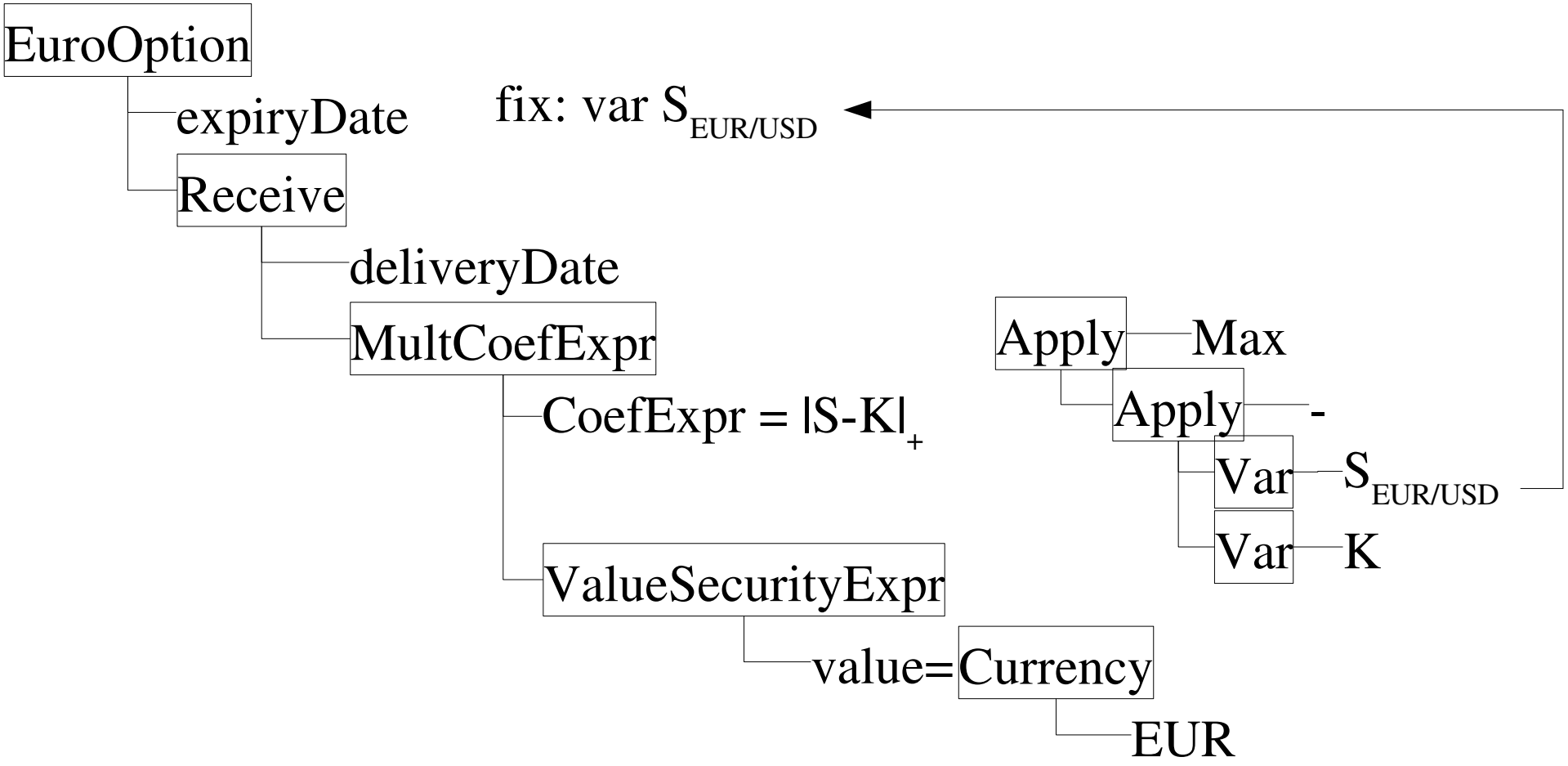
SUNGARD

Instrument / InstrumentExpr



SUNGARD

Exemple : Call Européen



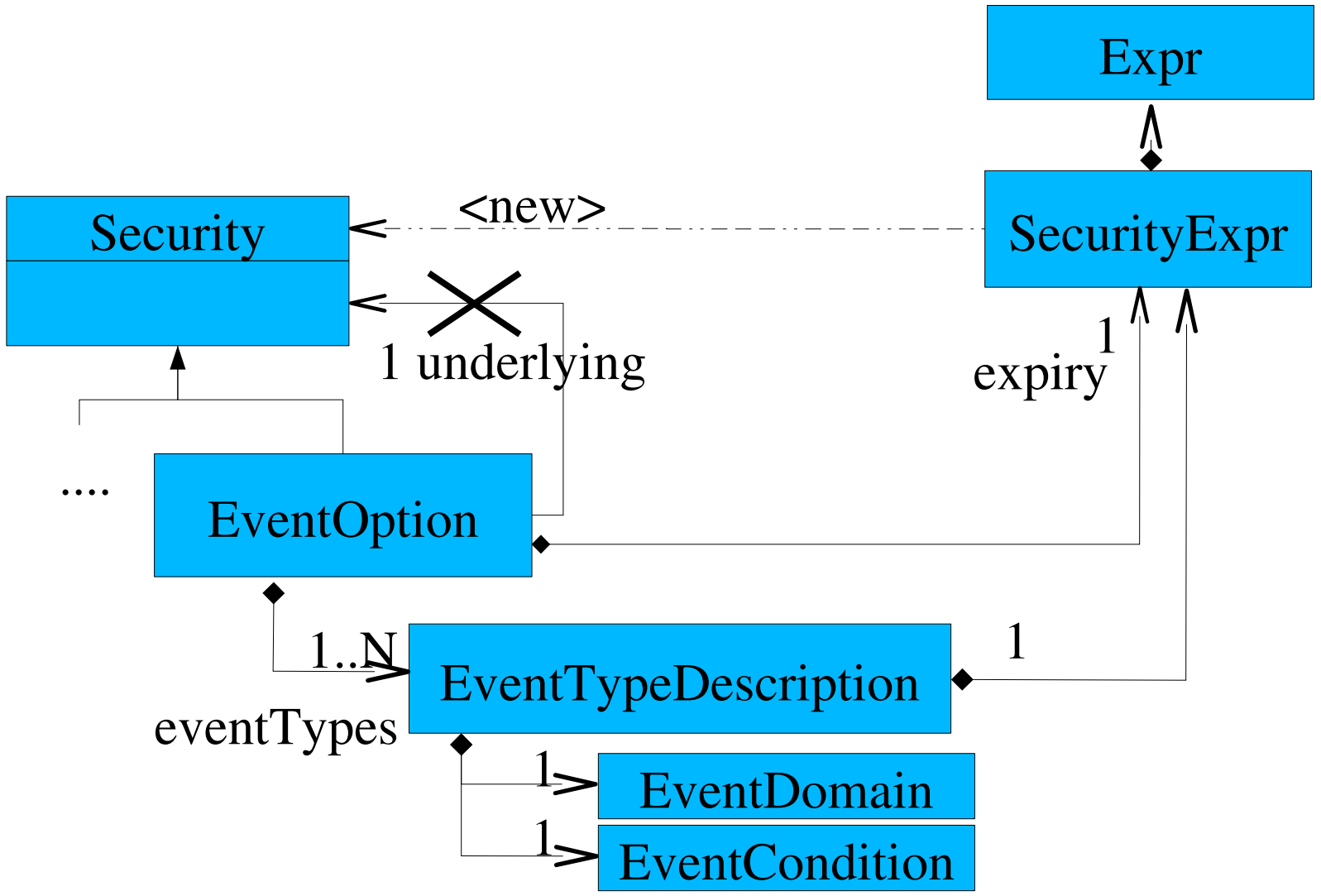
Option Américaine

- Par opposition à l'option européenne, l'option “Américaine” accepte un ou plusieurs types d'événements continus dans le temps
- Sur activation d'un événement,
  - l'option se transforme en un autre Instrument, dépendant des paramètres de l'événement,
- Sinon,
  - elle se transforme à l'échéance
- Cela généralise les Options Simples, Triggers, etc..

SUNGARD

Option Américaine

= Option à Evénements Continus



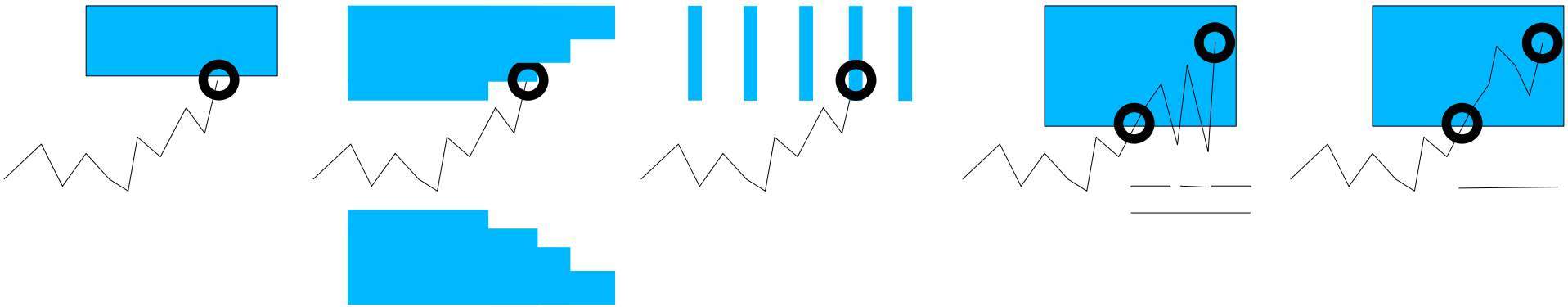
Critère d'Activation d'Evénements, Triggers

- Domaine Géométrique... insuffisant
- Indicatrice d'exerçabilité:
  - fonction : trajectoire $(X_t)_{0<t<T}$ --> boolean
- Choix d'exercice
  - max = condition type américaine
  - min = idem, pour la contrepartie
  - forcé = par le marché (ex: trigger)
- Paramètres/Transformation si activation:
  - fonction : trajectoire $(X_t)_{0<t<T}$ --> ...

SUNGARD

Exemple de Triggers Complexes

- Standard: Americaine, à Window, Telescoop
- Bermudéenne = européenne.. : sur fixings
- Parisienne, autres Path-dependant...
  - le spot reste au moins 10mn d'affilé > L
  - le spot reste au moins 10mn cumulée > L sur 20mn
  - Floue : le sous-jacent restant est proportionel au niveau atteint au dela d'un niveau L / L+E



Généricité => Expression et Macro

- Pour être générique, il faut des Expressions
  - pour les payoffs et sous-jacents
  - pour les triggers / evénements
  - pour les domaines géométriques / indicatrices
- Pour écrire des “macros”, simplifiant la modularité des produits
  - Exemple: Strip, Calendar => somme de N “Legs”
  - Définir des produits haut-niveaux (structurés) à partir de produits bas-niveaux

Avantages & Inconvénients des Expressions

- Très génériques
- Mais Difficile à manipuler
- Permet de faire des manipulations algébriques
- Description simple et homogène, même pour des instruments qui paraissent complexes ex: Quanto, Option sur Option, Multi Sous-Jacents...

Echelle de Complexité

- Besoin réel ? ... Engrenage de complexité
- Ou place-t-on ses limites ?

1) Payoff et produits hard-codés

<

1b) Payoff Générique (pointeur de fct / template)

<

1c) Instrument + AST Expr (produits à événements simples, hard-codés)

<

2) AST Instrument + Expr (comportement génériques)

<

3) AST Instrument + Expr + Macro functions

<

4) langage fonctionel + compilateur pour AST Instrument + Expr + Macro functions

Analogie complexité:

1) Boulier < 2) Calculatrice < 3) Programme < 4) Matlab < 5) Mathematica

Simplifications ?

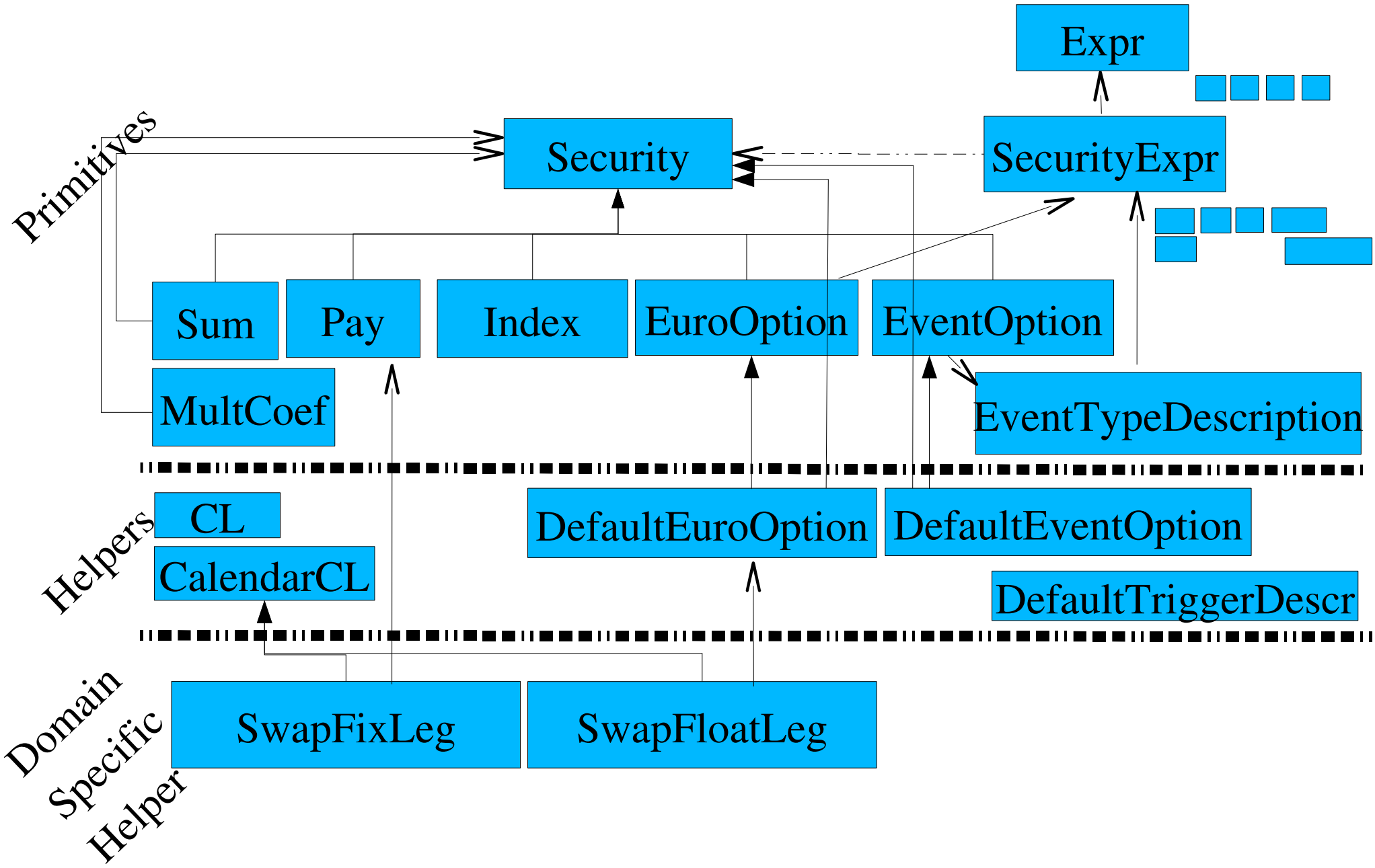
- Généric et homogène...
- Mais description verbeuse (trop détaillée) pour les 95% de cas “simples” !
  - Call : sous-jacent = payoff 1D + instrument
  - CapFloor : Sum Caplets / Strip
    = copy&paste N CALLs...
- Possibilités “dangereuses” de décrire
  - des produits incohérents (date futur/passé)
  - des produits trop complexes : path-dependant, multi-sous-jacents...

Compromis pour Simplifier

- Compromis Fonction pré-définie / expression
- Compromis sous-classes / expression
  - => utilisation des 2
  - simple “non-primitive” classes pour les cas simples
  - idem “macro” / helper
  - transparent pour le pricing : “inliné” en classes primitives
- Librairie au niveau de produits composés

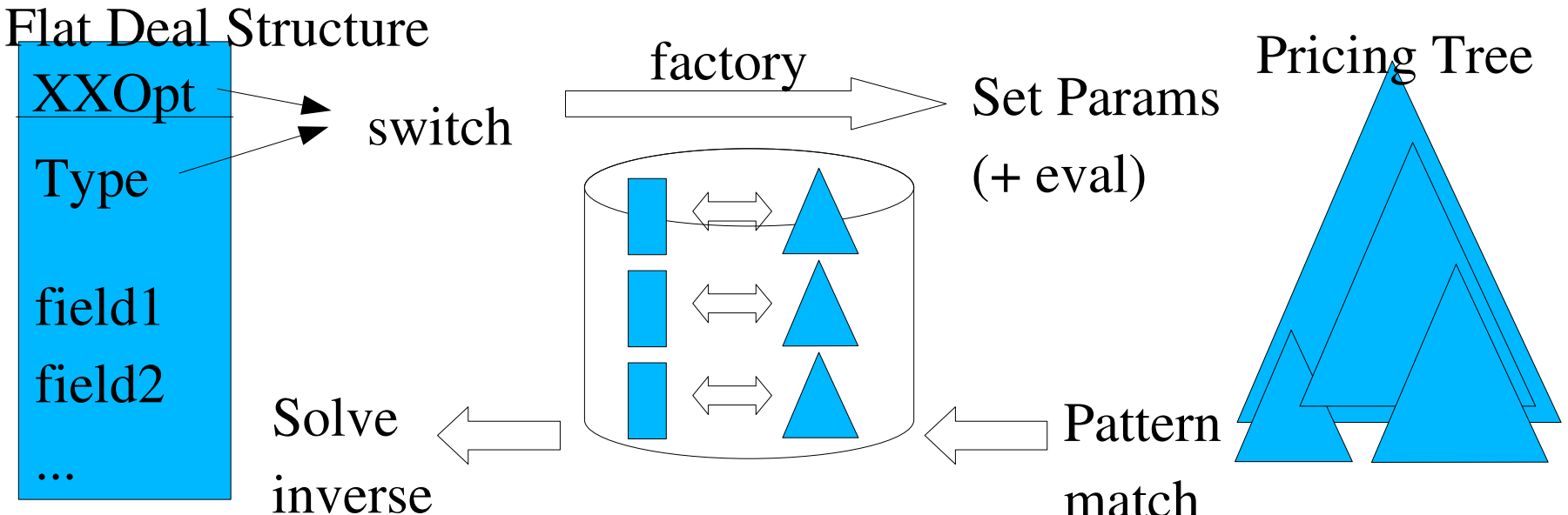
SUNGARD

Classes : Primitives / Non-Primitives



Remarque : Persistence Objet-Relationel

- Structure en Arbre avec N niveaux...
- Mapping Base-de-Données:
  - Serialization XML + Blob... OK ?
  - Problème des Querys => rajout de colonnes + indexes redundants
  - Sinon, Mapping complexe: Pricing <-> FlatDeal



Plan

- Rappel
- Payoff, classes d'Expressions Mathématiques
- Design Pattern Visitor
- Hierarchie de classe (AST) d'Instruments
- Comparaison Langage LexifiML

Macro, Grammaire et Langage Financier

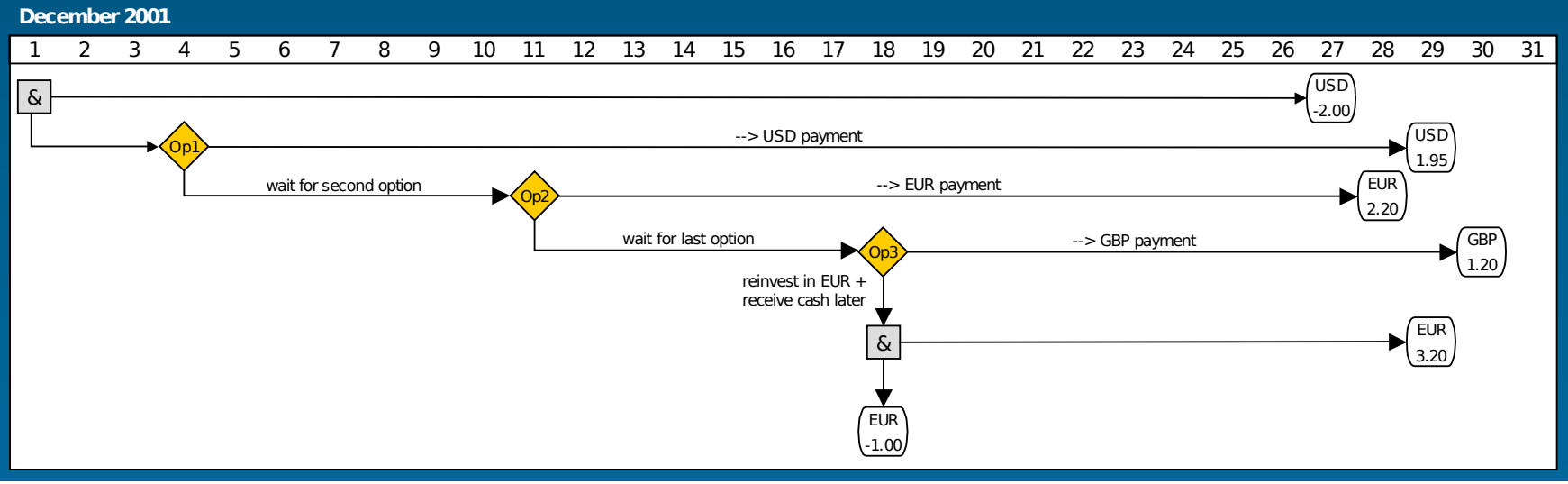
- Rappel :
  - se donner un AST est équivalent à une grammaire d'un langage
- Plutôt que de fournir un SDK, on peut fournir un parser (cacher les classes)
- Certains logiciels ont cette approche
  - Reech, Lexifi, etc..
- Exemple de description d'un produit en xml ou pseudo-langage...
  - ....

Exemple de Lexifi (depuis le tutorial)

The Financial Contract

Against the promise to pay USD 2.00 on December 27 (the price of the option), the holder has the right, on December 4, to choose between

- receiving USD 1.95 on December 29, or
- having the right, on December 11, to choose between
  - receiving EUR 2.20 on December 28, or
  - having the right, on December 18, to choose between
    - receiving GBP 1.20 on December 30, or
    - paying immediately one more EUR and receiving EUR 3.20 on December 29.



Exemple de Lexifi (depuis le tutorial)

MLFi Contract Description

Managing and monitoring this "custom build" structure with MLFi is simple

```
let option1 =

  let strike = cashflow(USD:2.00, 2001-12-27) in

  let option2 =

    let option3 =
      let t = 2001-12-18T15:00 in
      either
        ("--> GBP payment", cashflow(GBP:1.20, 2001-12-30))
        ("reinvest in EUR + receive cash later",
         (give(cashflow(EUR:1.00, t))) 'and' cashflow(EUR:3.20, 2001-12-29))
        t in

    either
      ("--> EUR payment", cashflow(EUR:2.20, 2001-12-28))
      ("wait for last option", option3)
      2001-12-11T15:00 in

  (either
     ("--> USD payment", cashflow(USD:1.95, 2001-12-29))
     ("wait for second option", option2)
     2001-12-04T15:00) 'and' (give (strike))

let cal = calendar option1
let _ = vcalfile "test.vcs" cal
```

Contract Description

Events Scheduler

Summary So Far

```
give     : contract -> contract
or       : (contract * contract) -> contract
and      : (contract * contract) -> contract
zero     : contract
acquire  : (region * contract) -> contract
anytime  : (region * contract * contract) -> contract
truncate : (date * contract) -> contract
scale    : (observable * contract) -> contract
…and some more besides…
```

- *Everything* is built from the combinators!
- We need an *absolutely precise* specification of what they mean

Comparaison et Remarques Sur Lexifi

- Langage fonctionnel
  - semblable à des langages de programmation type Caml, Lisp, Scheme
  - See http://www.lexifi.com/faq.html : “**MLFi is implemented as an extension of Caml**” !!
  - “ML” n'est pas le suffix pour “XML” !
- Langage/Librairie extensible par des nouvelles “fonctions de fonctions”
  - Très peu de fonctions built-in
  - Définition des produits combinables par librairie utilisateur

SUNGARD

Comparaison et Remarques Sur Lexifi

- Exemple de comparaison FpML-Payoff-Lexifi
  - http://www.lexifi.com/faq.html#witdbf,p-opl,am
  - Fpml: “Because it is not compositional, FpML focuses on interbank contracts and does not support complex, client-driven transactions.”
- Forces du produit =
  - Sémantique Riche = spec + impl
  - “***Describe financial contracts precisely and exhaustively with a limited set of core constructs*** “
  - Compilateur pour Pricing (Front Office)
  - Générateur d'événement pour Middle/Back-Office, etc.

Questions Ouvertes

- Comparaison des “core-AST” entre librairies et logiciels existants
  - Lexifi ML, Reech ADEP, NumeriX
  - Société Générale : Optit, Fx Option Pricer
  - Formalisme équivalent ? Le plus simple / Le plus générique ?
- Comparaison avec d'autres logiciels moins génériques
  - Infinity, Summit, Calypso, Jrisk ...

Questions Ouvertes

- Disponibilités de librairies objets sur Internet ?
  - open-sources ? Java ?
- Disponibilité de langage de description de produits ?
- Evolution des “Standards” de description de produits existants ?
  - FPML, FinML, … ?

Conclusion

- La description des Instruments financiers est un domaine simple et compliqué à la fois
  - Produits listés simples mais nombreux
  - Grande diversité de produits exotiques (OTC), et de façon de les combiner
  - Modélisation intrinsèquement liée à l'implémentations du Pricing (optimization numériques...)
- Beaucoup de logiciels sur mesure, et d'implémentations de modèles secrets!
- Beaucoup de progiciels, peu d'Open-Source

Questions ?

arnaud.nauwynck @gmail.com / @sgam.com