Programmes Adaptables par Spécialisation de Programmes

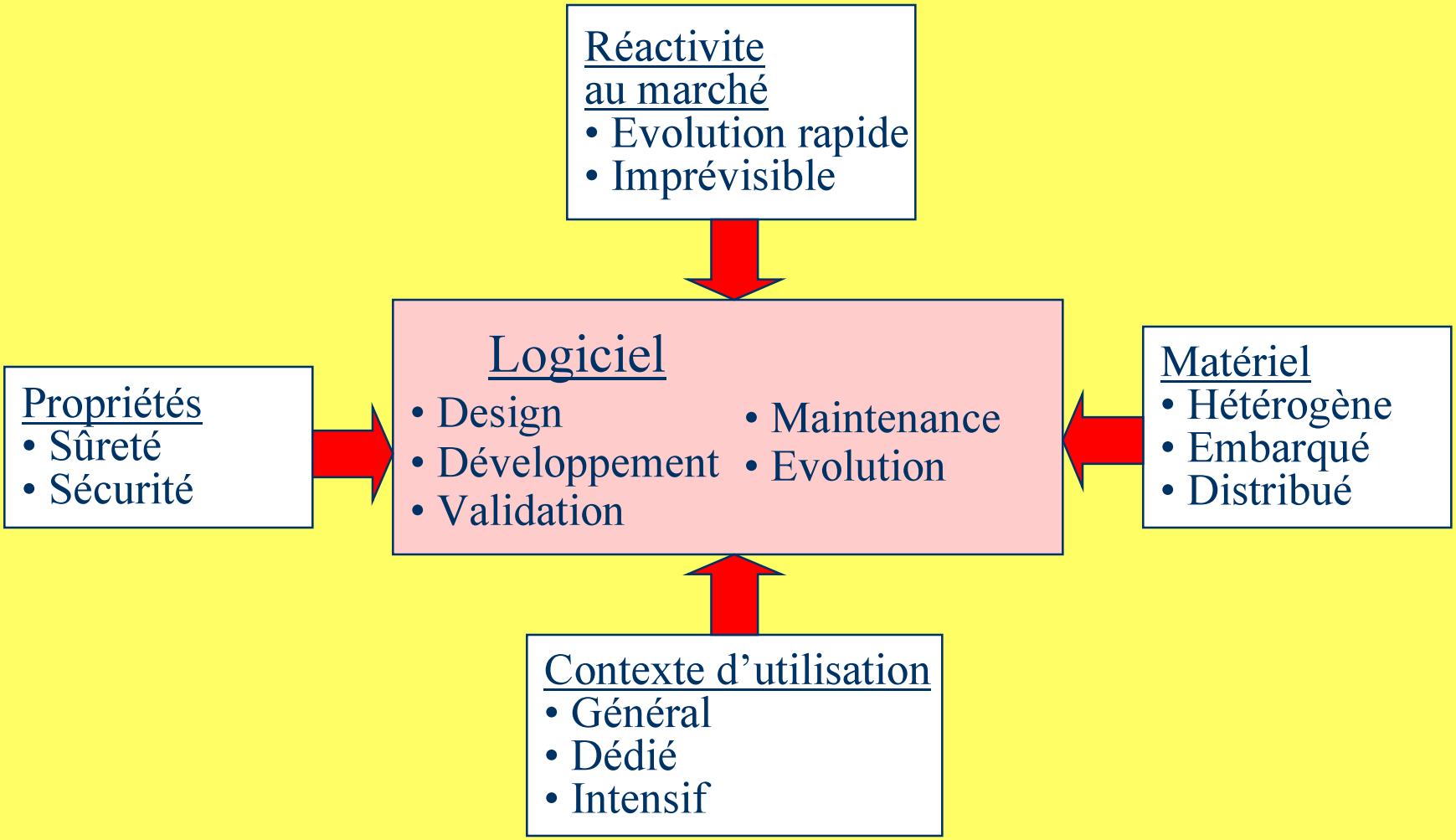
Charles Consel

Groupe Compose

IRISA/Université de Rennes I/INRIA

Novembre 1998

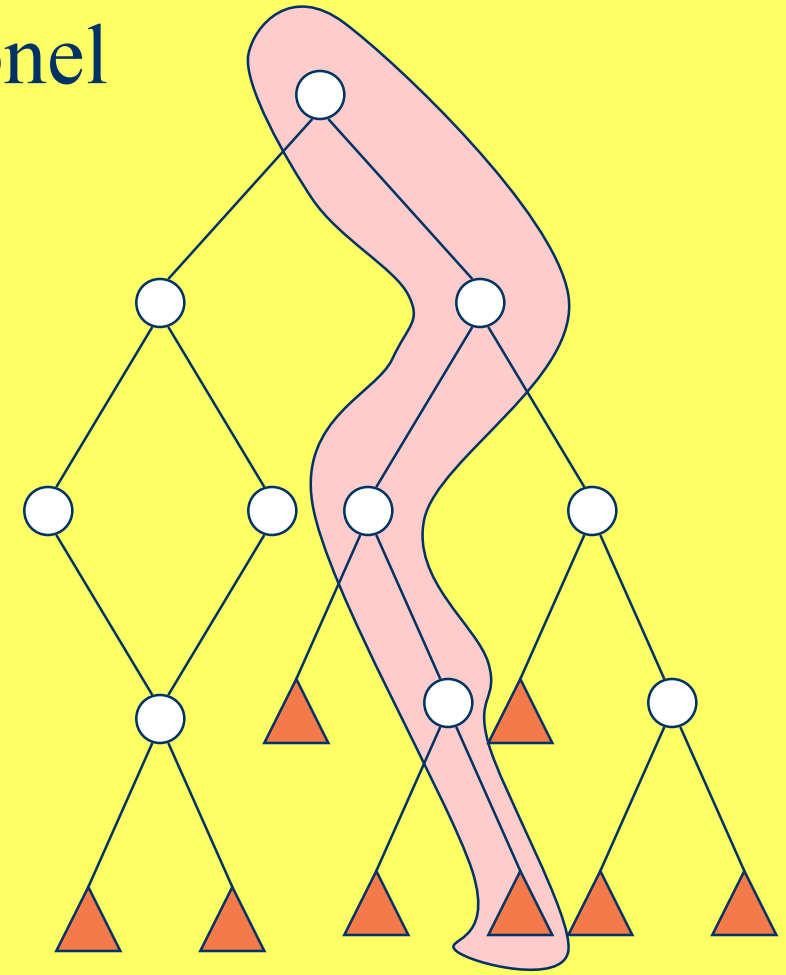
Besoins d'adaptation



Adaptation ad hoc

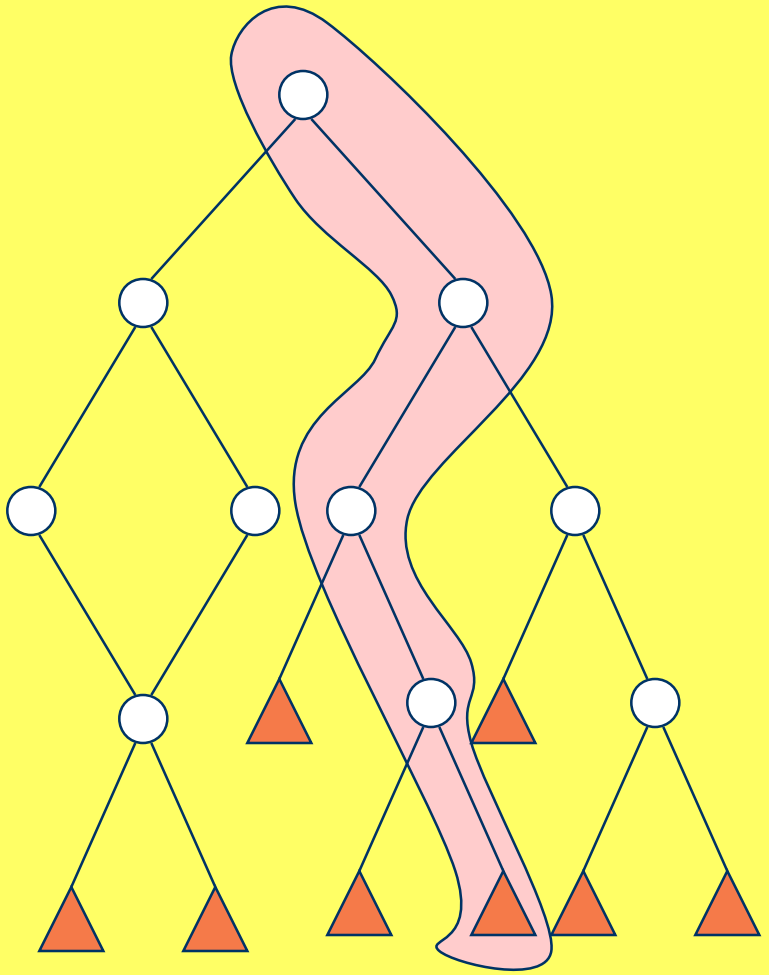
Programme = graphe décisionel

- **Données**
  - » Paramètres.
- **Contrôle**
  - » Conditionelles.
  - » Switches.
  - » Pointeurs de fonctions.
  - » Héritage.

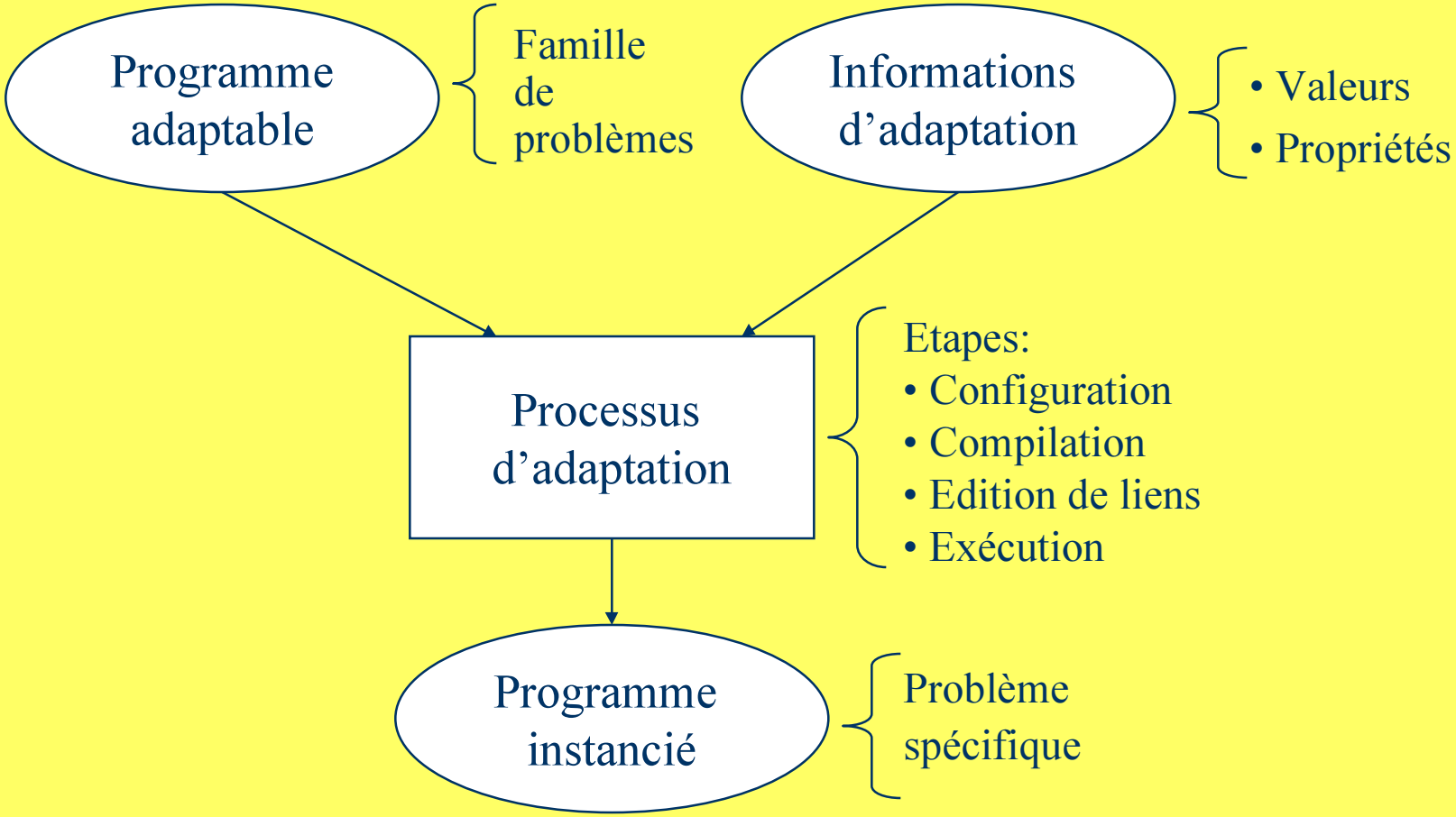


Adaptation ad hoc

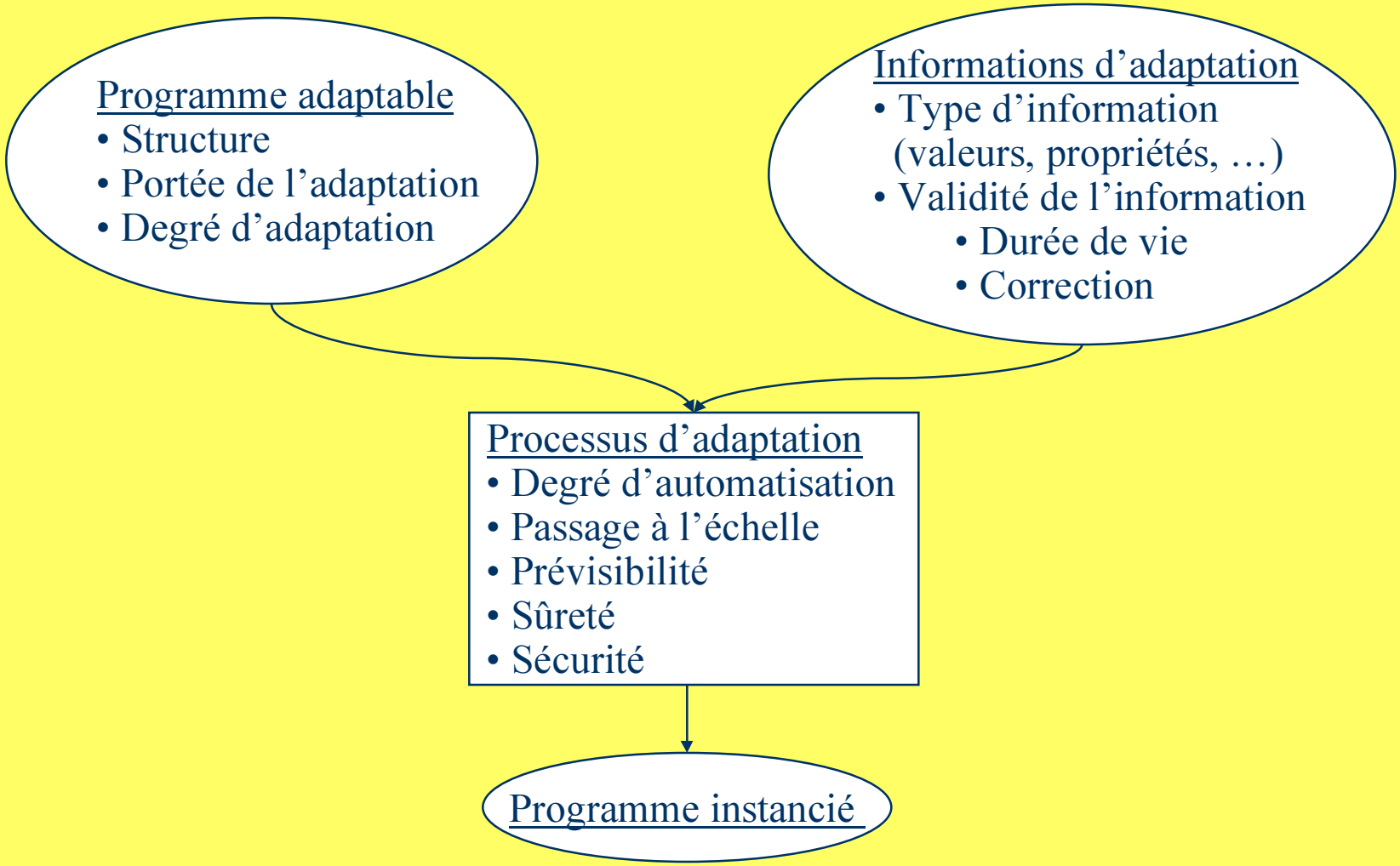
- **Coûteux en temps:**
  - » Interprétation.
- **Coûteux en taille:**
  - » Gestion de la généricité.
- **Coûteux en maintenance:**
  - » Code important et complexe.
- **Coûteux en validation:**
  - » Difficulté d'analyse.



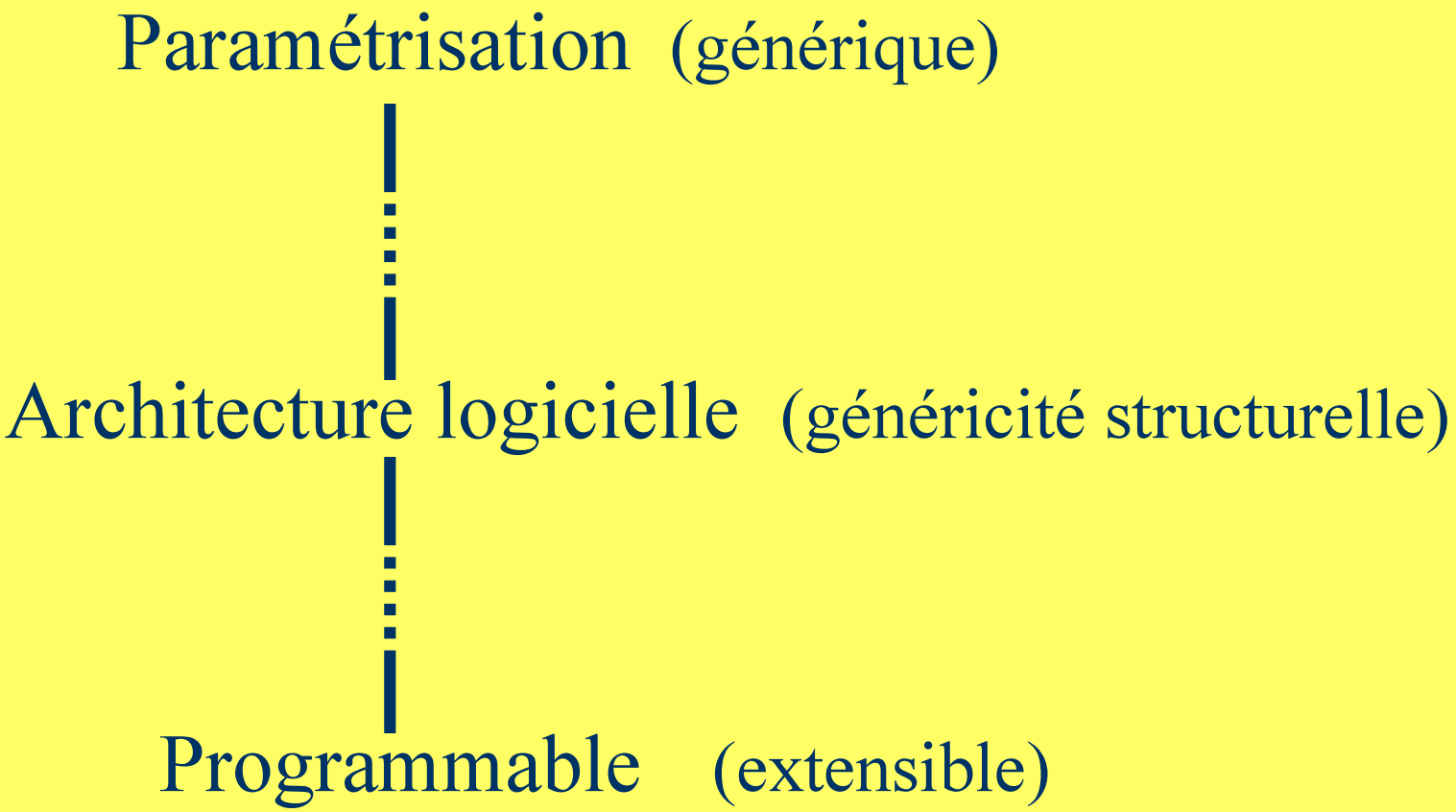
Conception de programmes adaptables



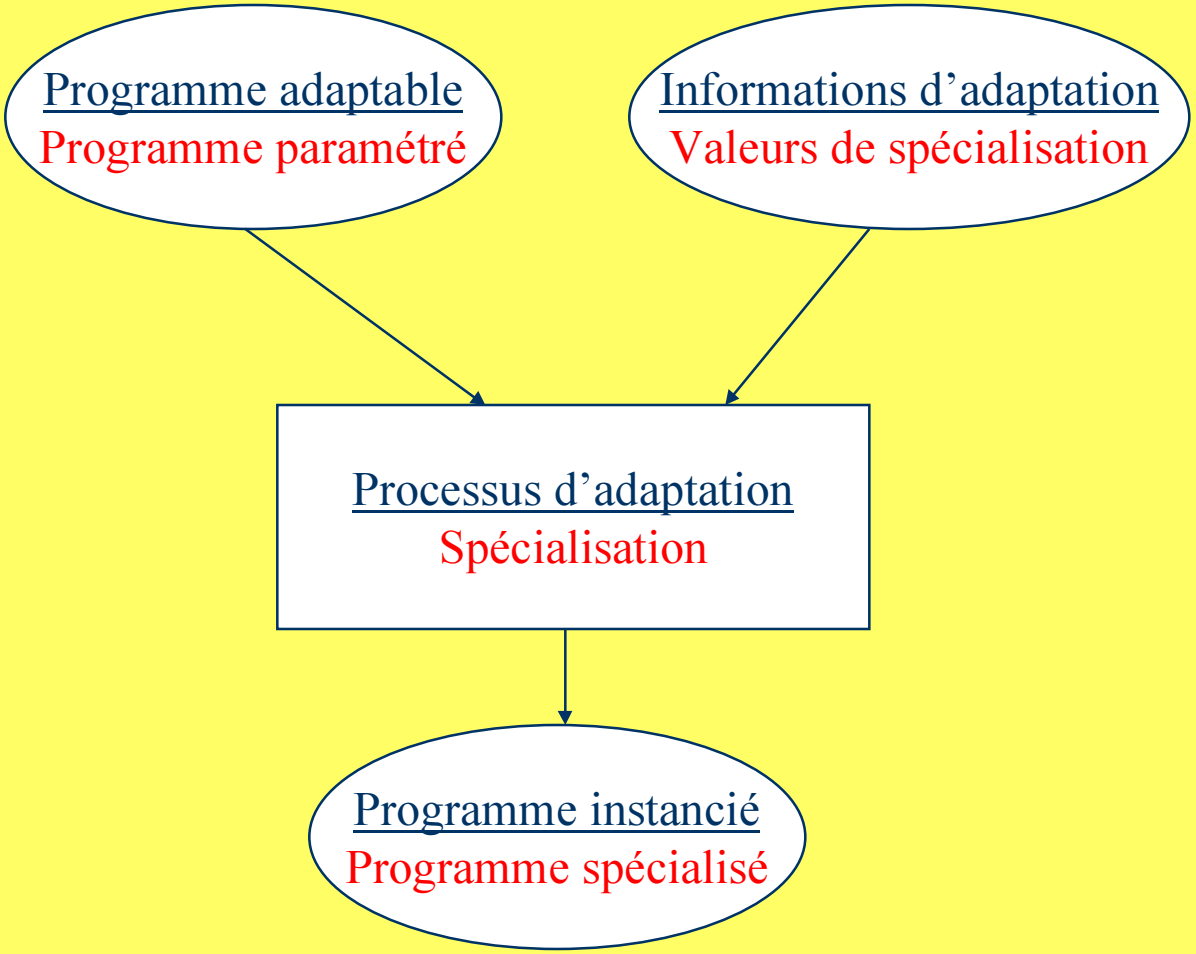
Les différentes dimensions de l'adaptation



Stratégies d'adaptation



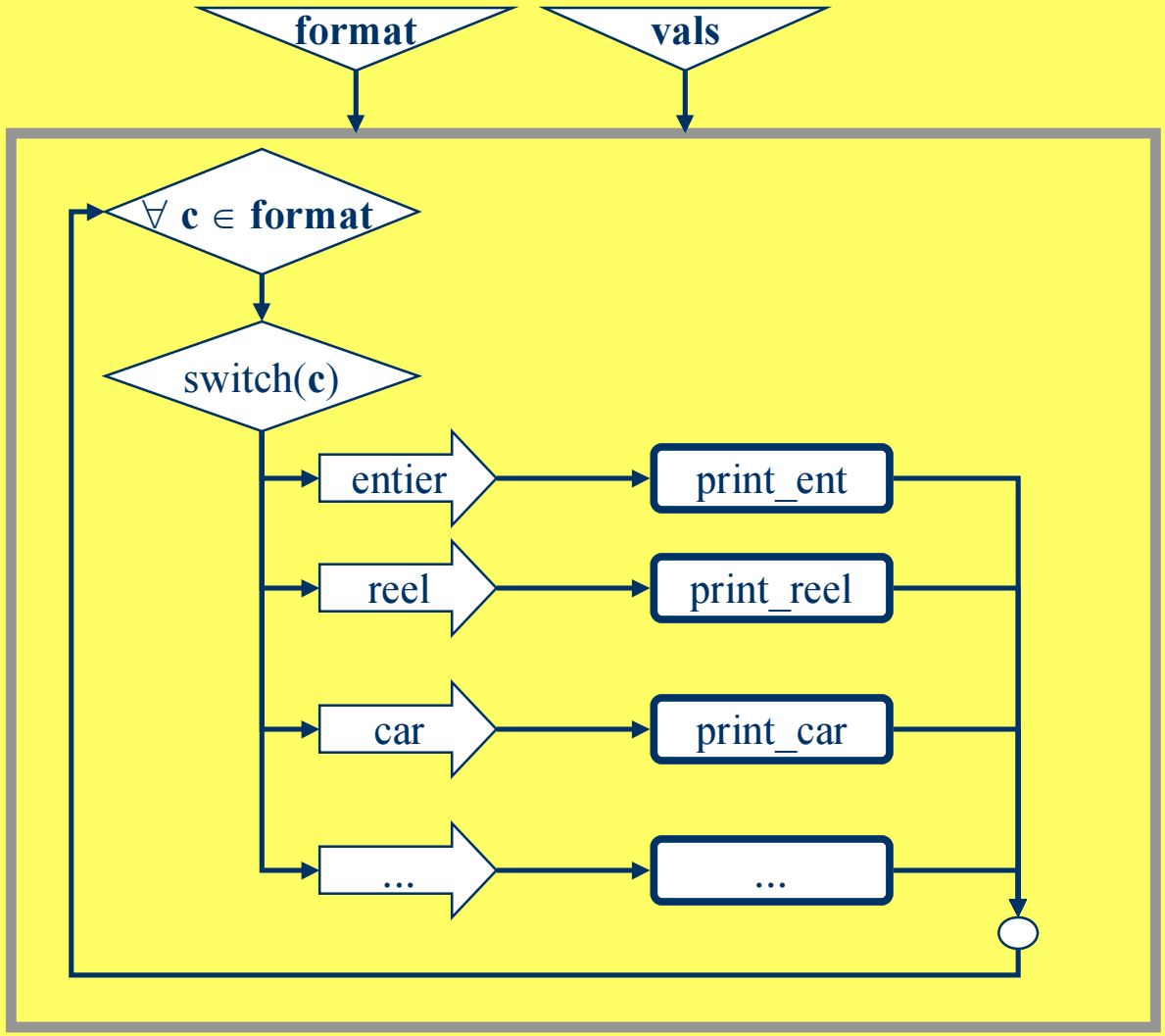
Adaptation par paramétrisation



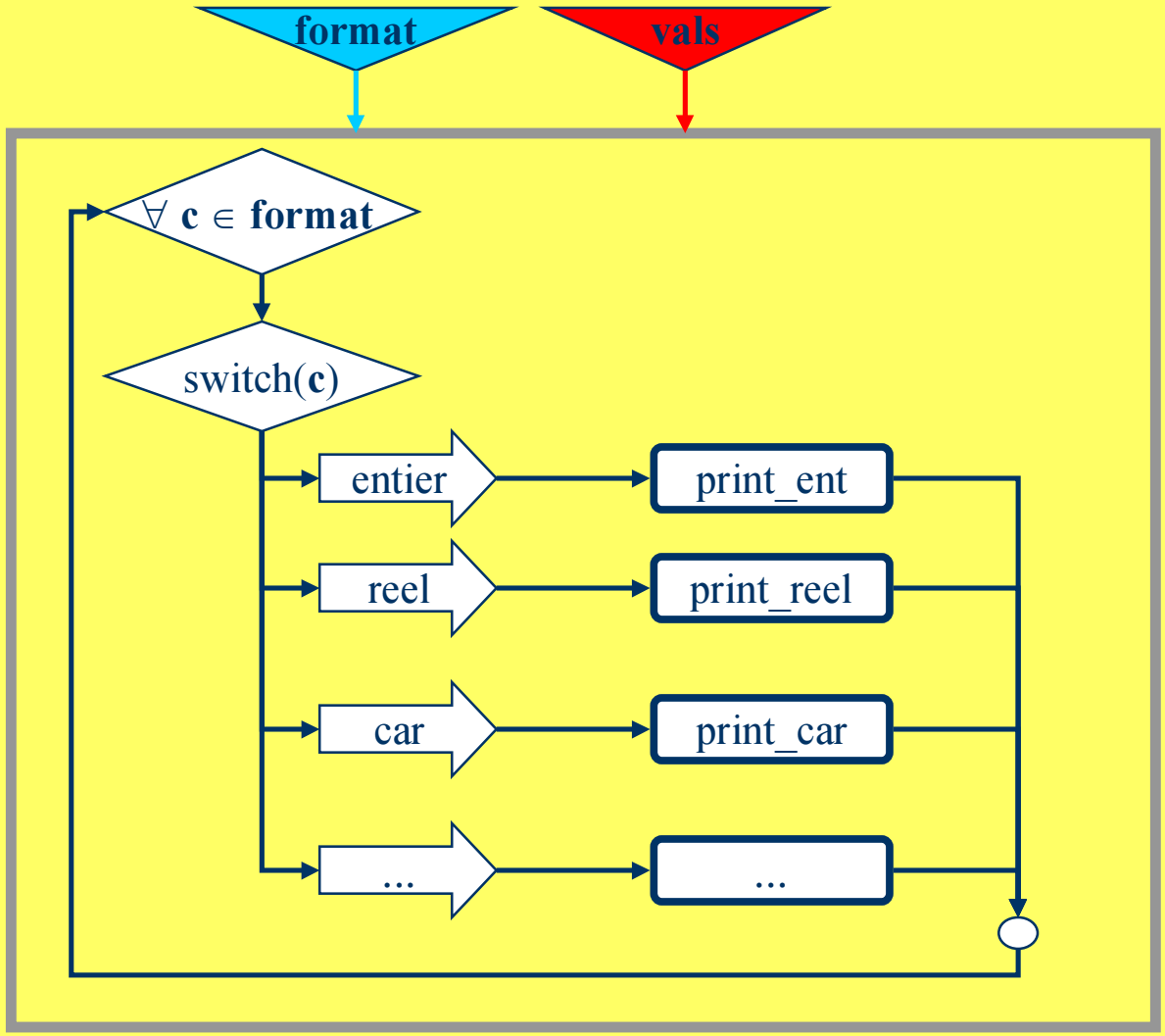
Spécialisation de programmes

```
for(i=0; i < max; i++)
{
   ...
   printf( "v1 = %d / v2 = %d",
            v1[i], v2[i]);
   ...
}
```

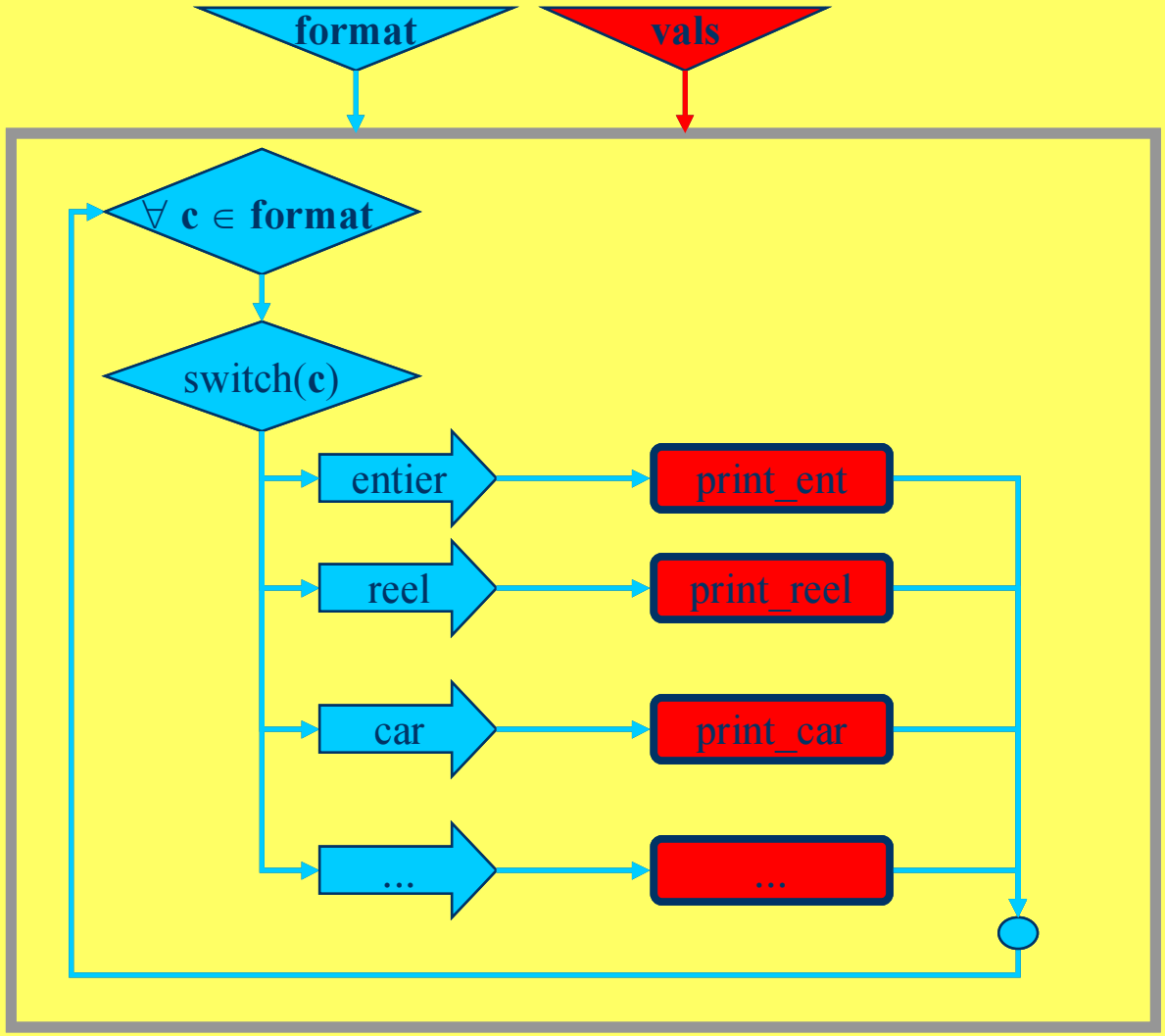
Spécialisation de programmes



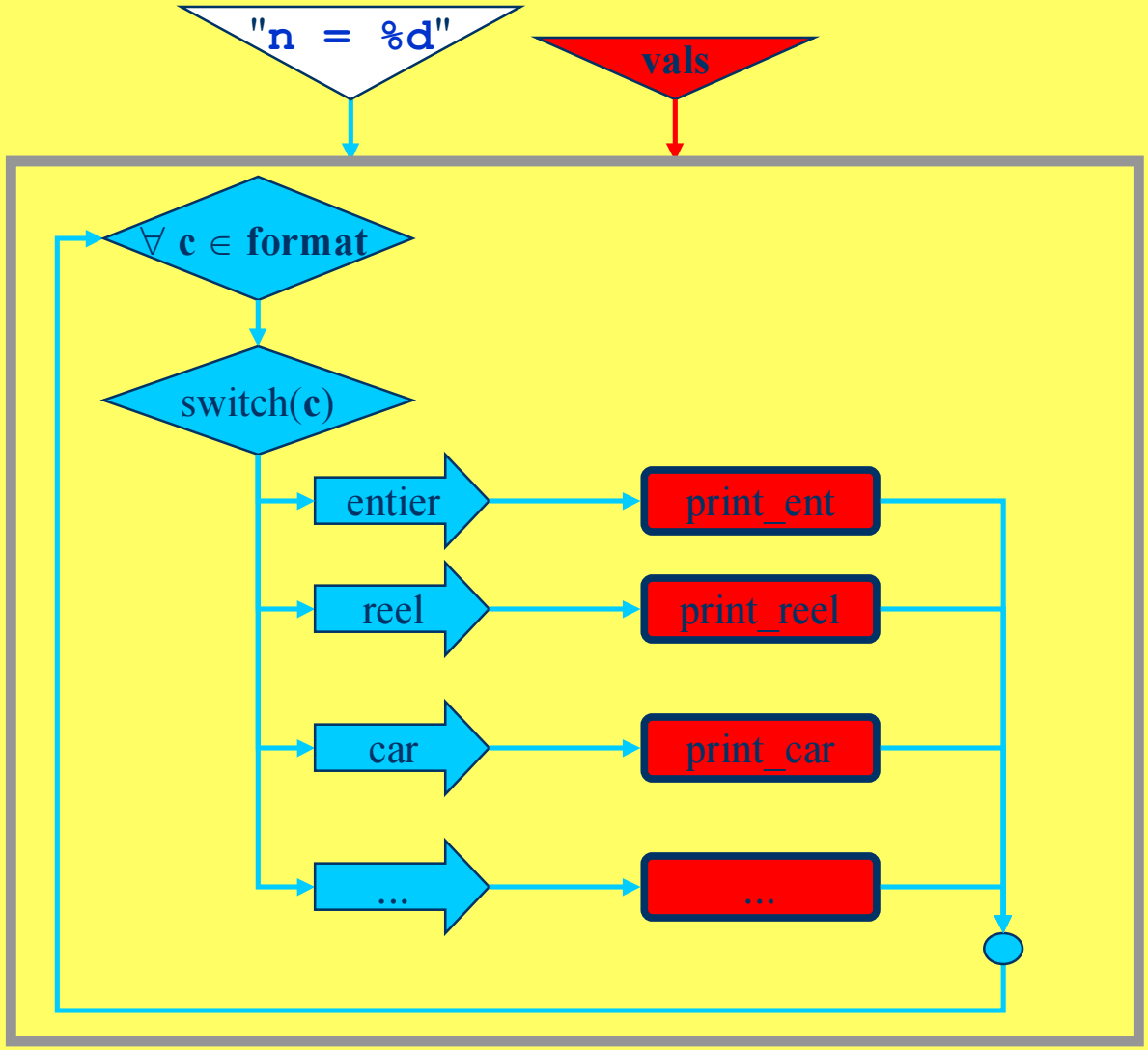
Spécialisation de programmes



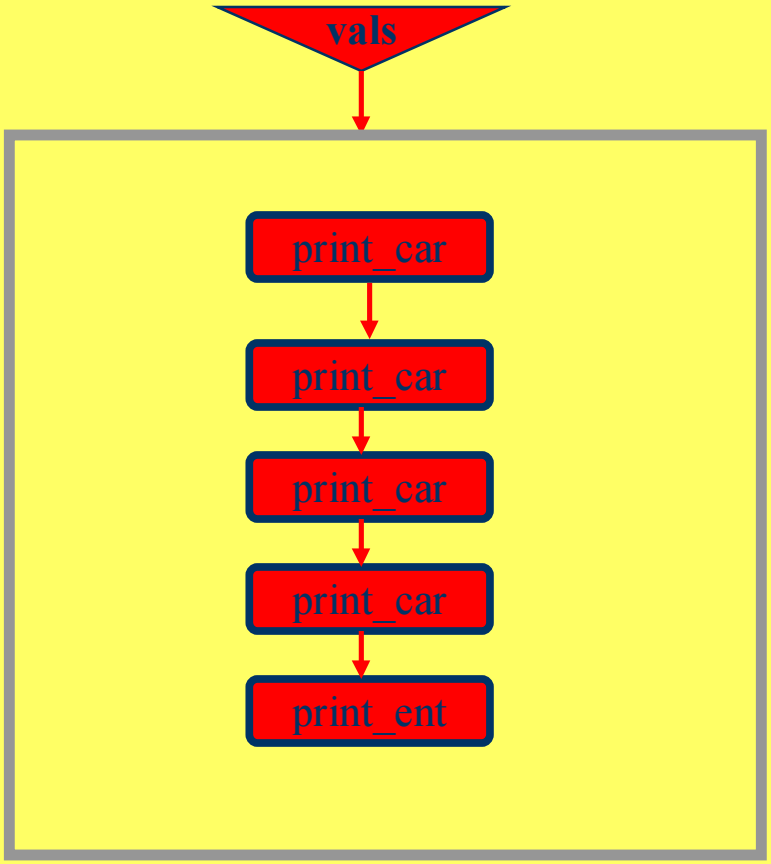
Spécialisation de programmes



Spécialisation de programmes



Spécialisation de programmes



Exemple: `mini_printf`

```
void mini_printf(char* fmt, int *values)
{
  int i;

  for (i=0 ; *fmt != '\0' ; fmt++)
    if(*fmt != '%')
      putchar(*fmt);
    else
      switch(*++fmt)
      {
        case 'd': putint(values[i++]); break;
        case '%': putchar('%'); break;
        default: error(); break;
      }
}
```

Exemple: `mini_printf`

Spécialisation avec la chaîne de contrôle

"n = %d"

```
void mini_printf_fmt(int* values)
{
  putchar('n');
  putchar(' ');
  putchar('=');
  putchar(' ');
  putint(values[0]);
}
```

Notre spécialiseur : Tempo

- Langage C.
- Spécialisation à la compilation et à l'exécution.
- Spécialisation modulaire.
- Spécialiseur *back-end* pour
  - C++
  - Fortran
  - Java
- Applications de taille réelle.

Applications de Tempo

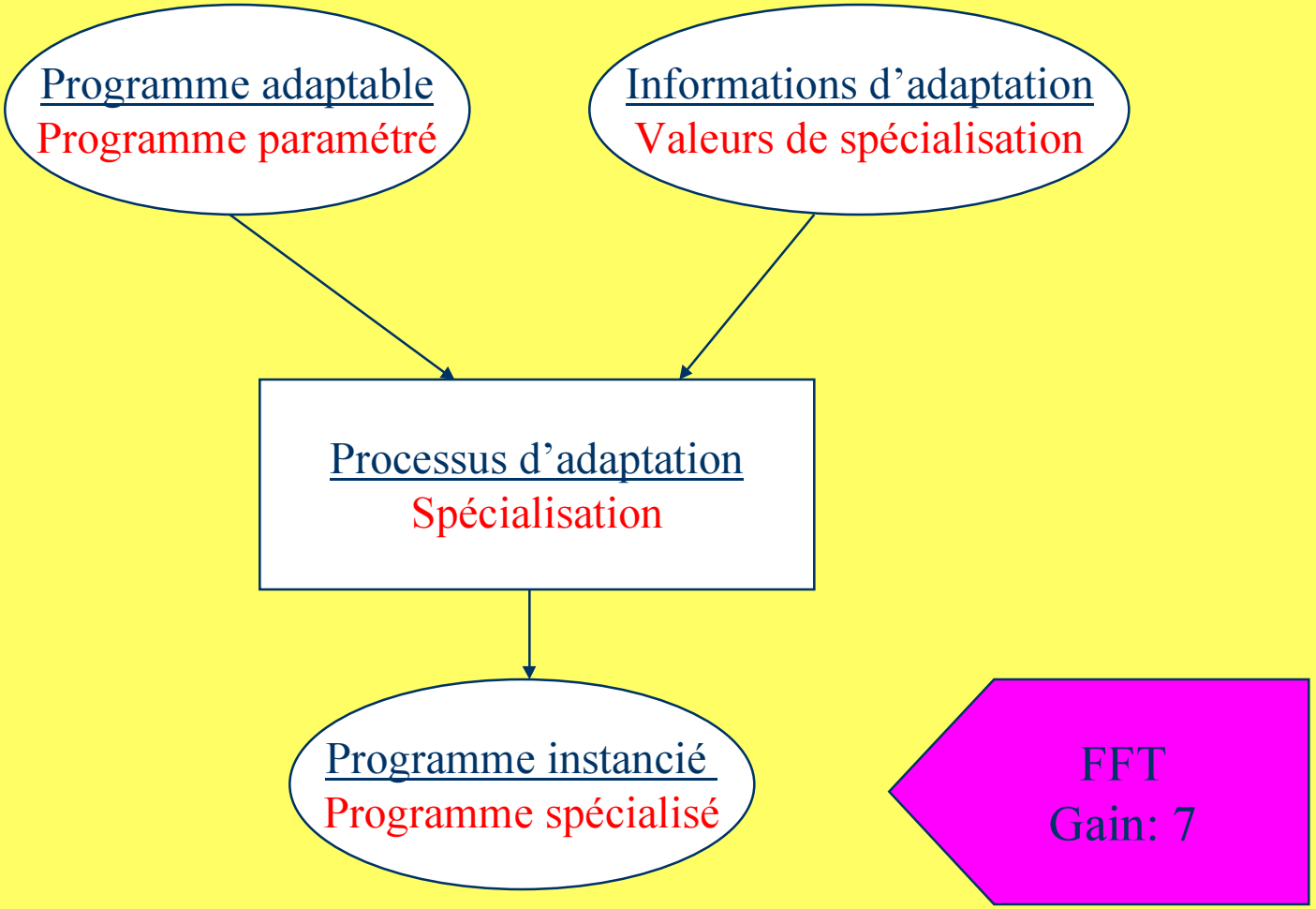
- Systèmes d'exploitation:

  (Sun RPC, Chorus IPC, …).
- Architectures logicielles

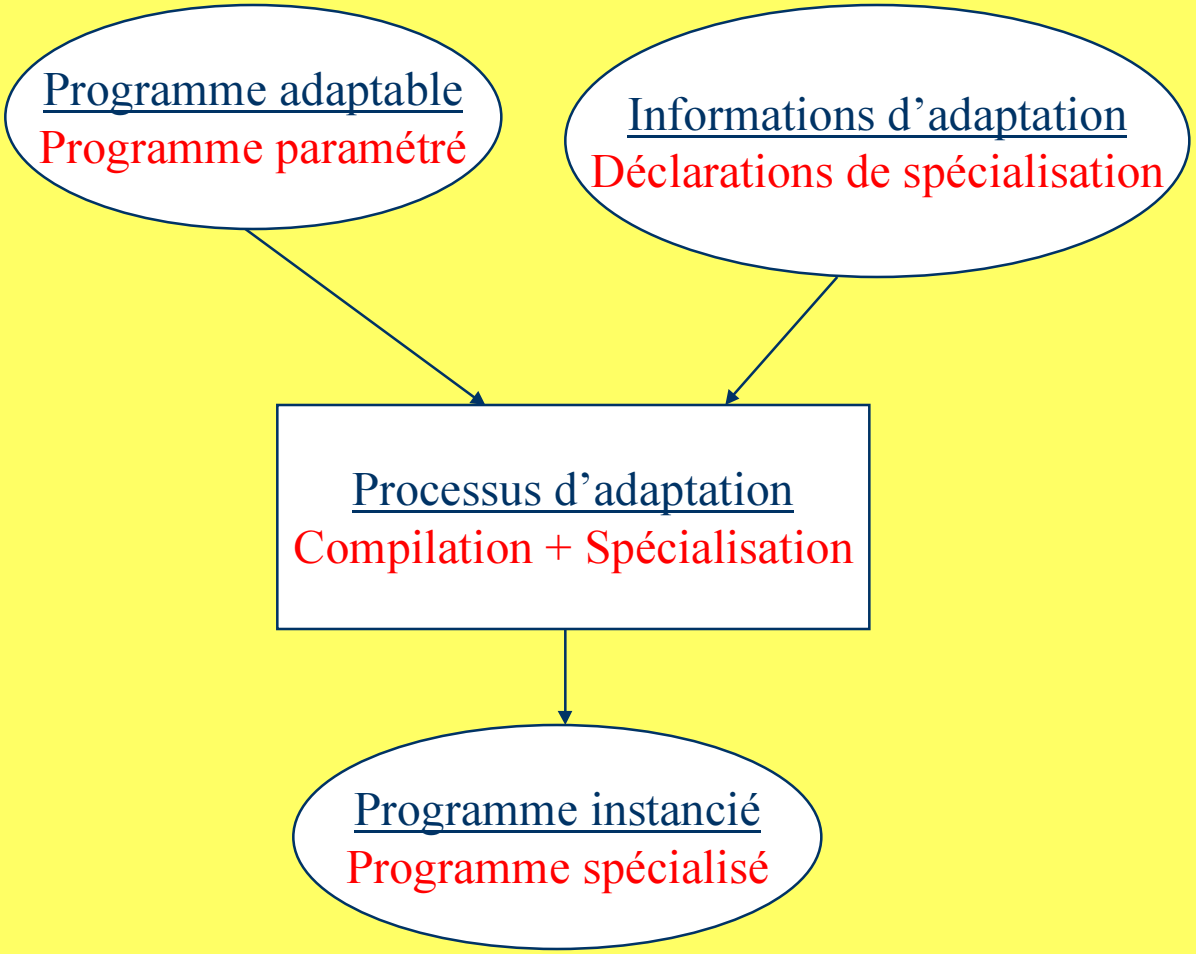
  (couches, messages, ...).
- Applications scientifiques

  (FFT, librairies, …).

Adaptation par paramétrisation



Adaptation par paramétrisation déclarée



Déclaration de spécialisation: Les classes de spécialisation

```
public class C {
   int mode;
   ...
   public void process()
   {
     make(mode, val);
   }
   ... }
```

Déclaration de spécialisation: Les classes de spécialisation

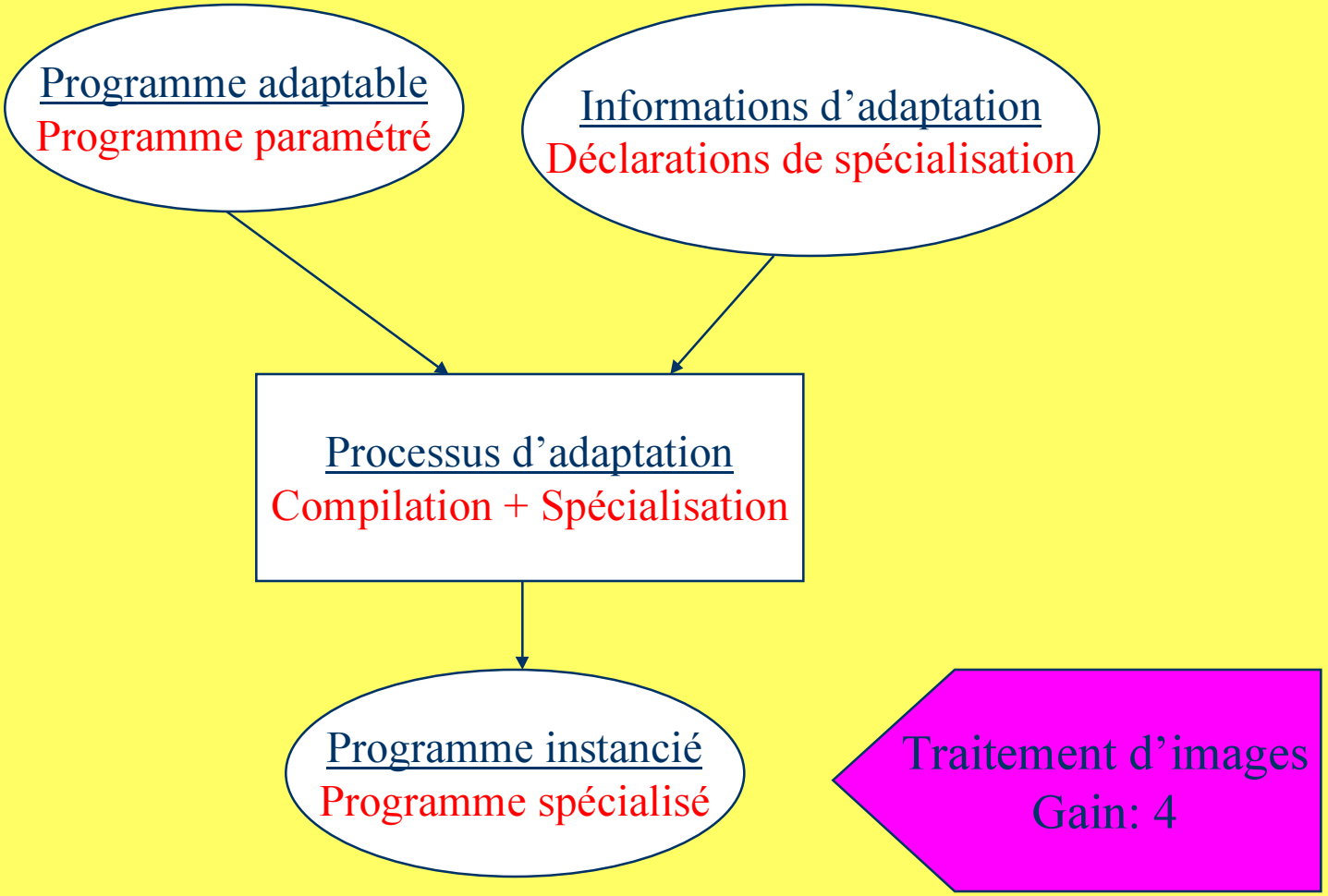
```
public class C {
   int mode;
   ...
   public void process()
   {
     make(mode, val);
   }
   ... }
```

```
specclass ReadC
    specializes class C {
   mode == READ;
   process();
}
```

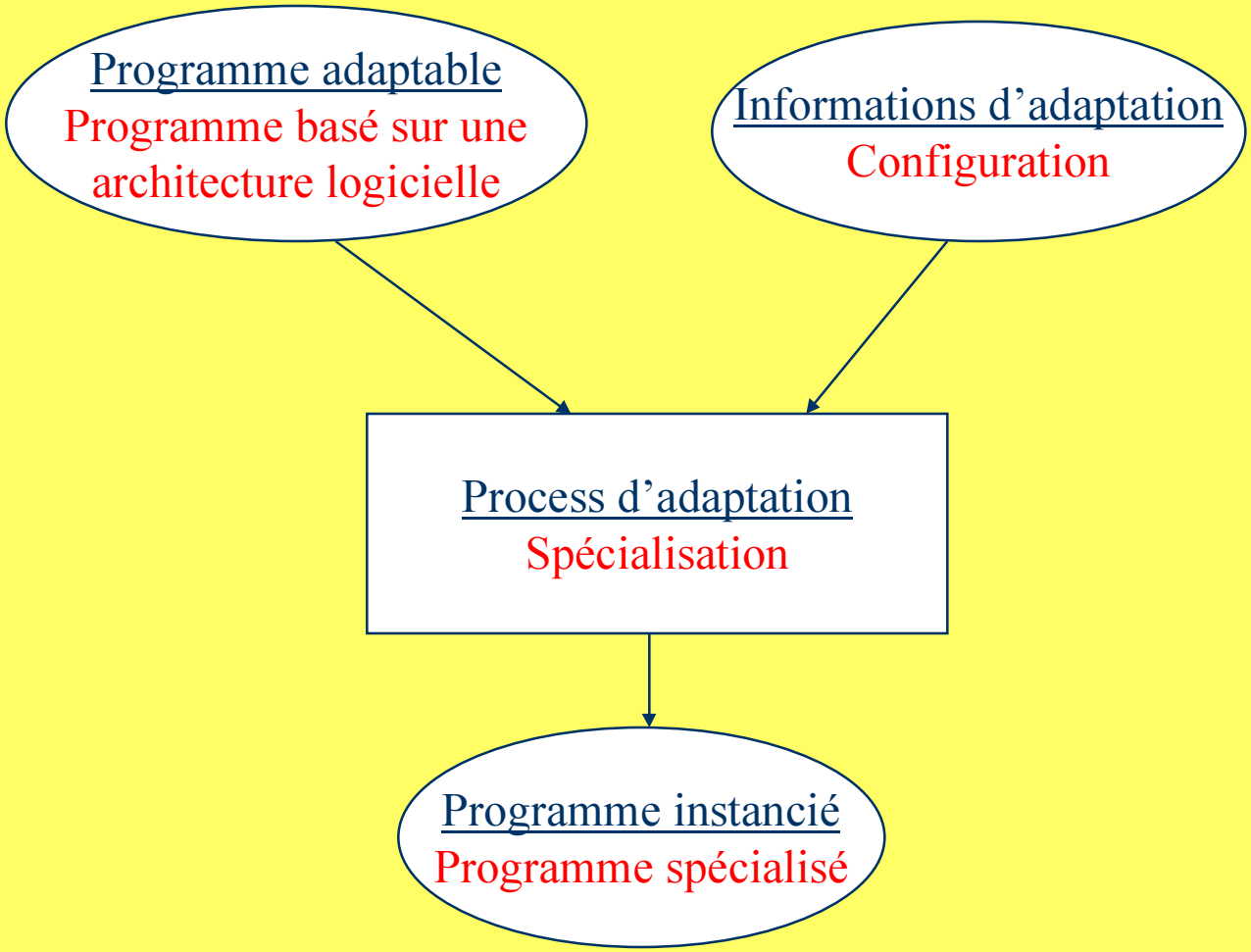
Génération du support d'exécution:

- Génération de code spécialisé.
- Gardes.
- Aiguillage: générique/spécialisé.

Adaptation par paramétrisation déclarée

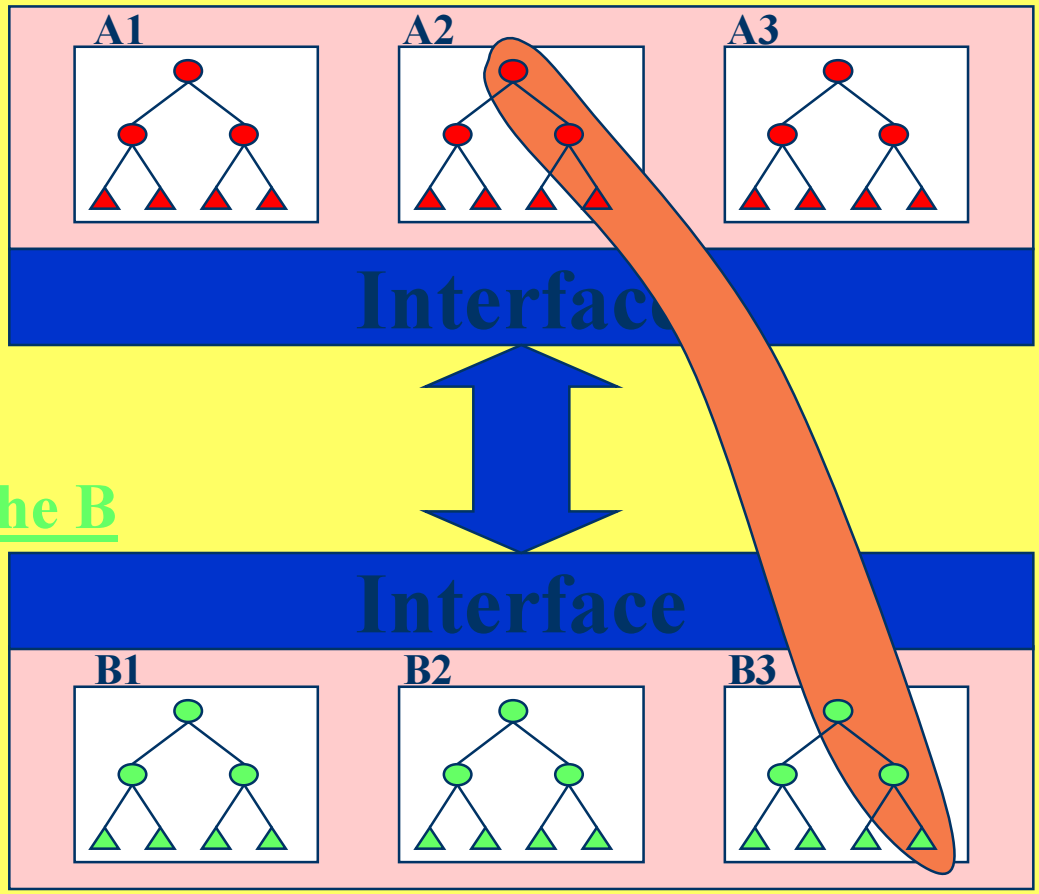


Adaptation par architectures logicielles



Adaptation via architectures logicielles

Couche A

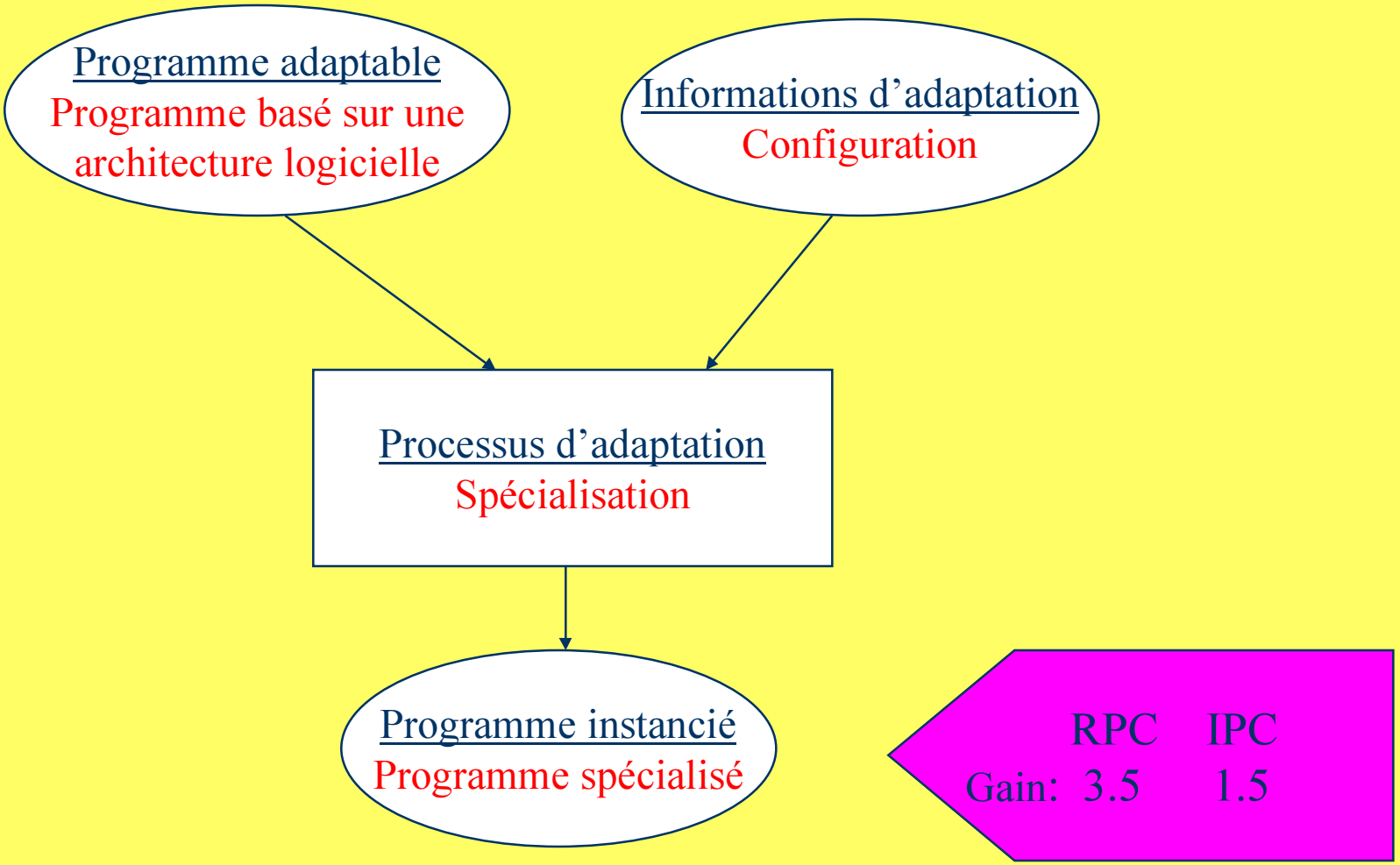


1- Données d'invocation.
2- Définir les options.
3- Encodage des données.

Couche B

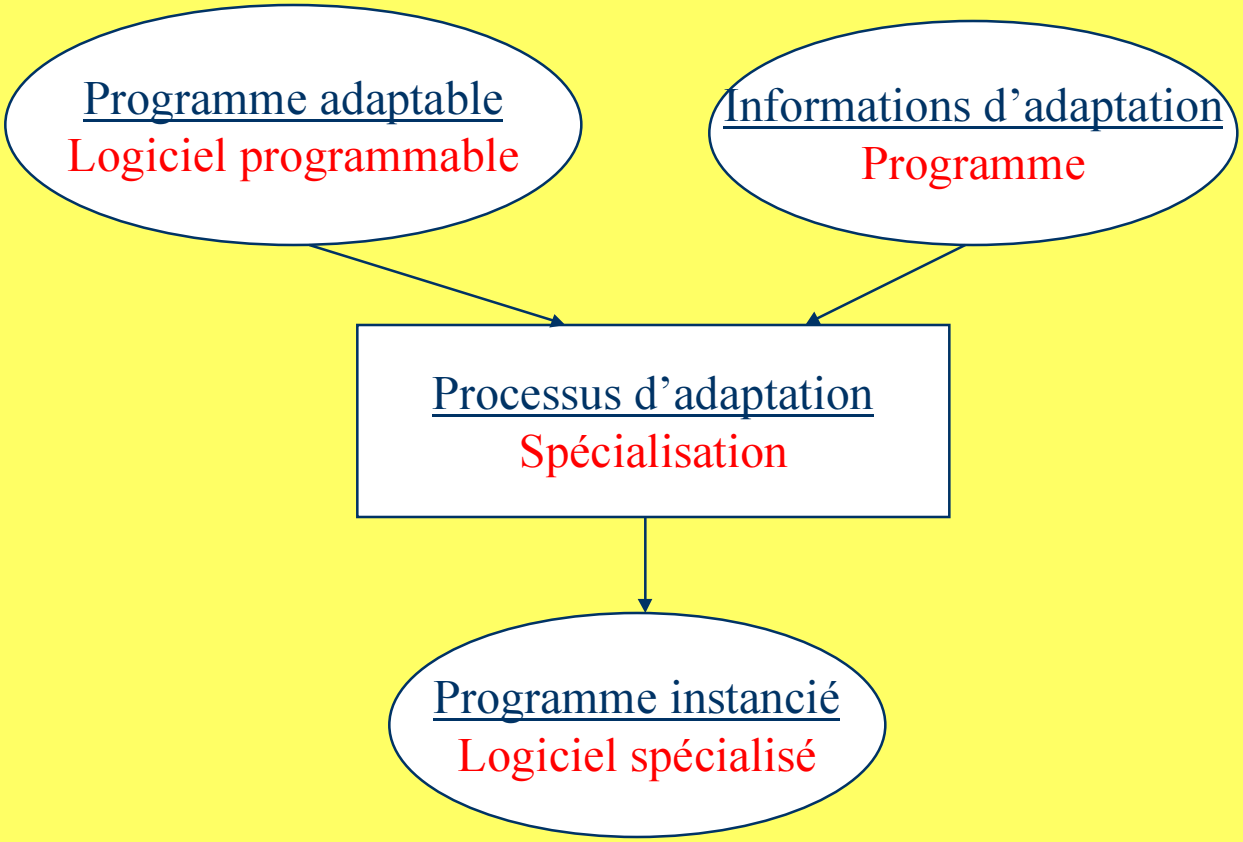
1- Décodage des données.
2- Traitement des options.
3- Vérification des assertions.
4- Invocation.

Adaptation par architectures logicielles



Adaptation par langages dédiés

Domain-specific languages (DSL)



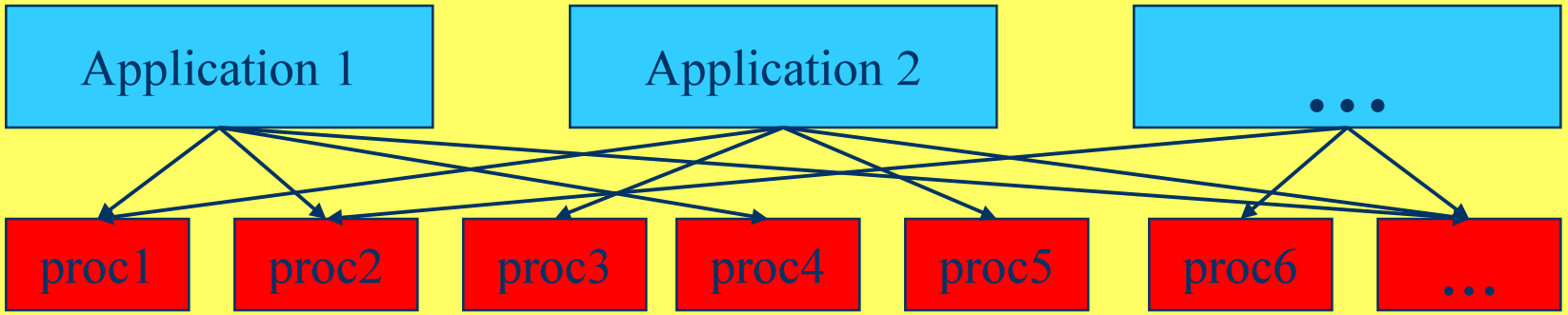
Adaptation: librairies

- Paramétrisation excessive.
- Librairies de grande taille.

➡ Difficulté d'utilisation.

➡ Expertise requise.

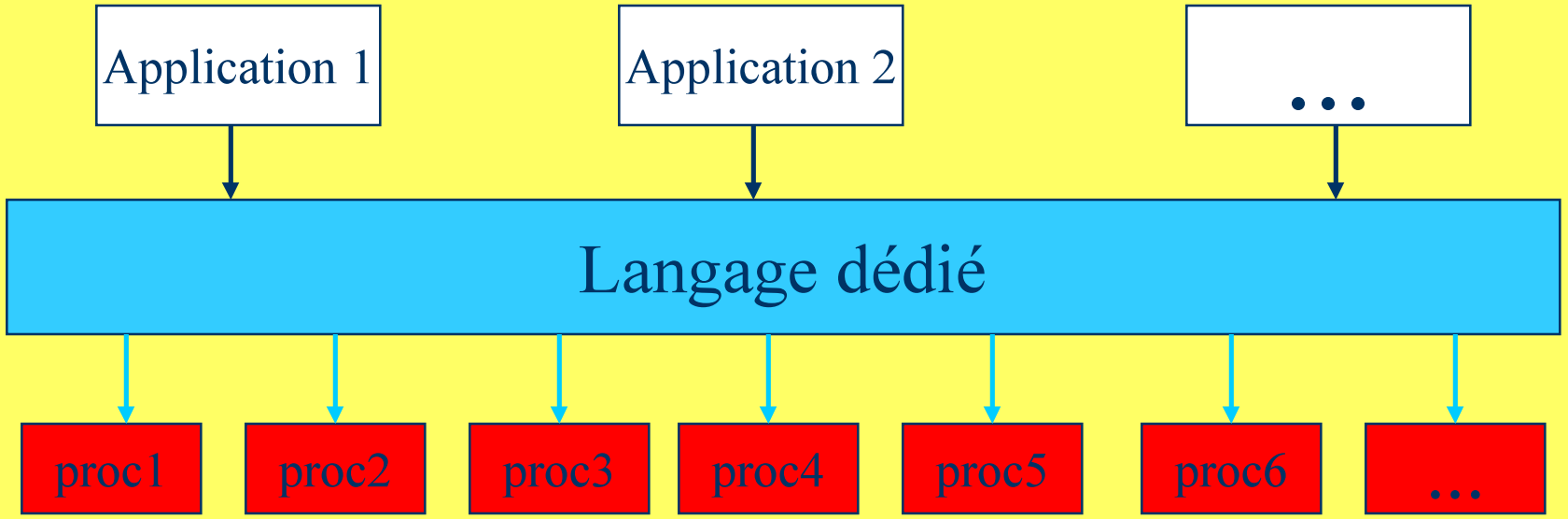
➡ Performance médiocre.



Langages dédiés

- Abstractions.
- Notations.

- Concision.
- Ré-utilisation.
- *Facilité d'analyse…*



Langages dédiés

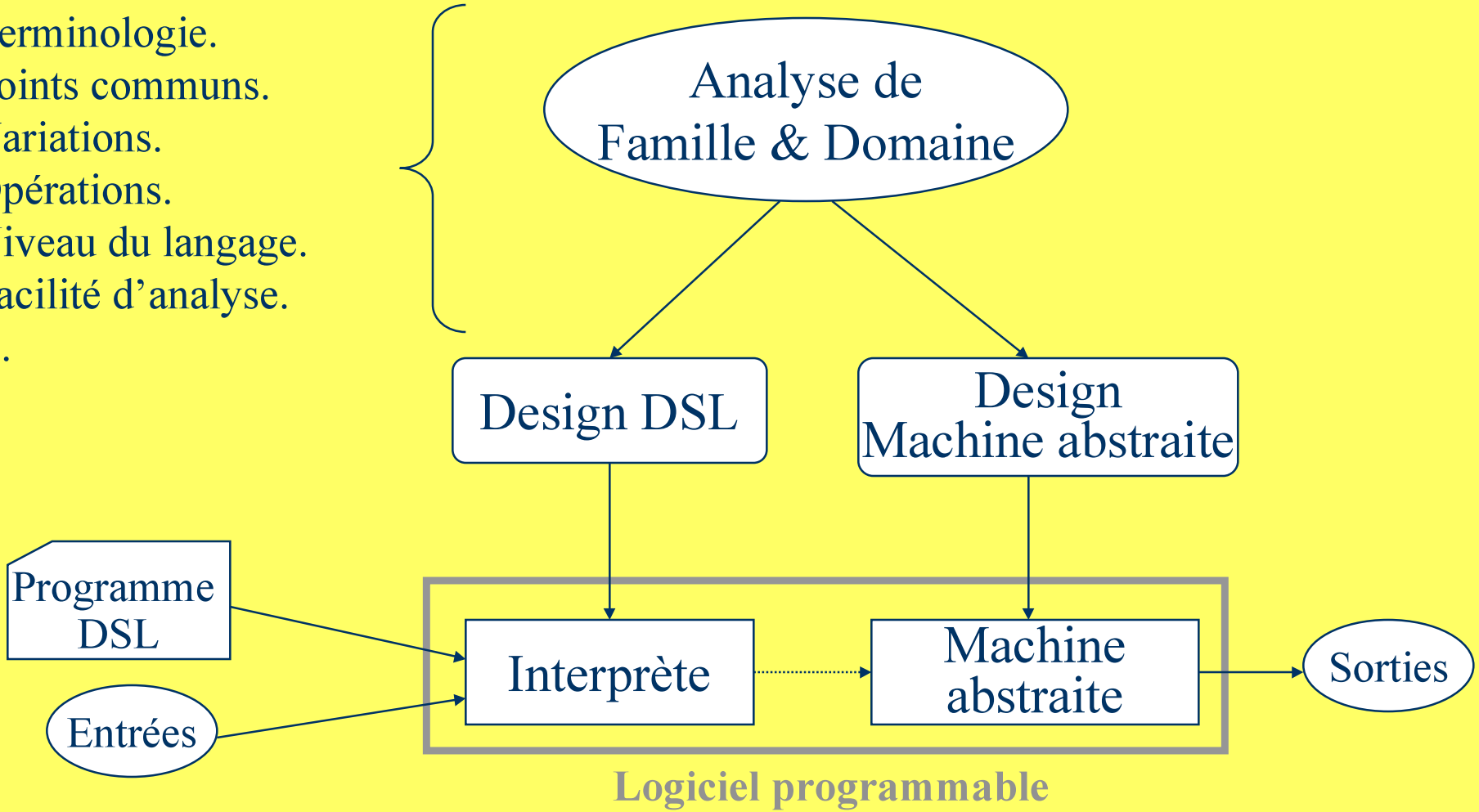
- Langages dédiés dans l'industrie
  - Produits financiers.
  - Télécommunications.
  - Pages web…
- Nos langages dédiés
  - GAL: spécification de pilotes de périphériques.
  - PLAN-P: protocoles d'applications pour réseaux.

Langages dédies: avantages

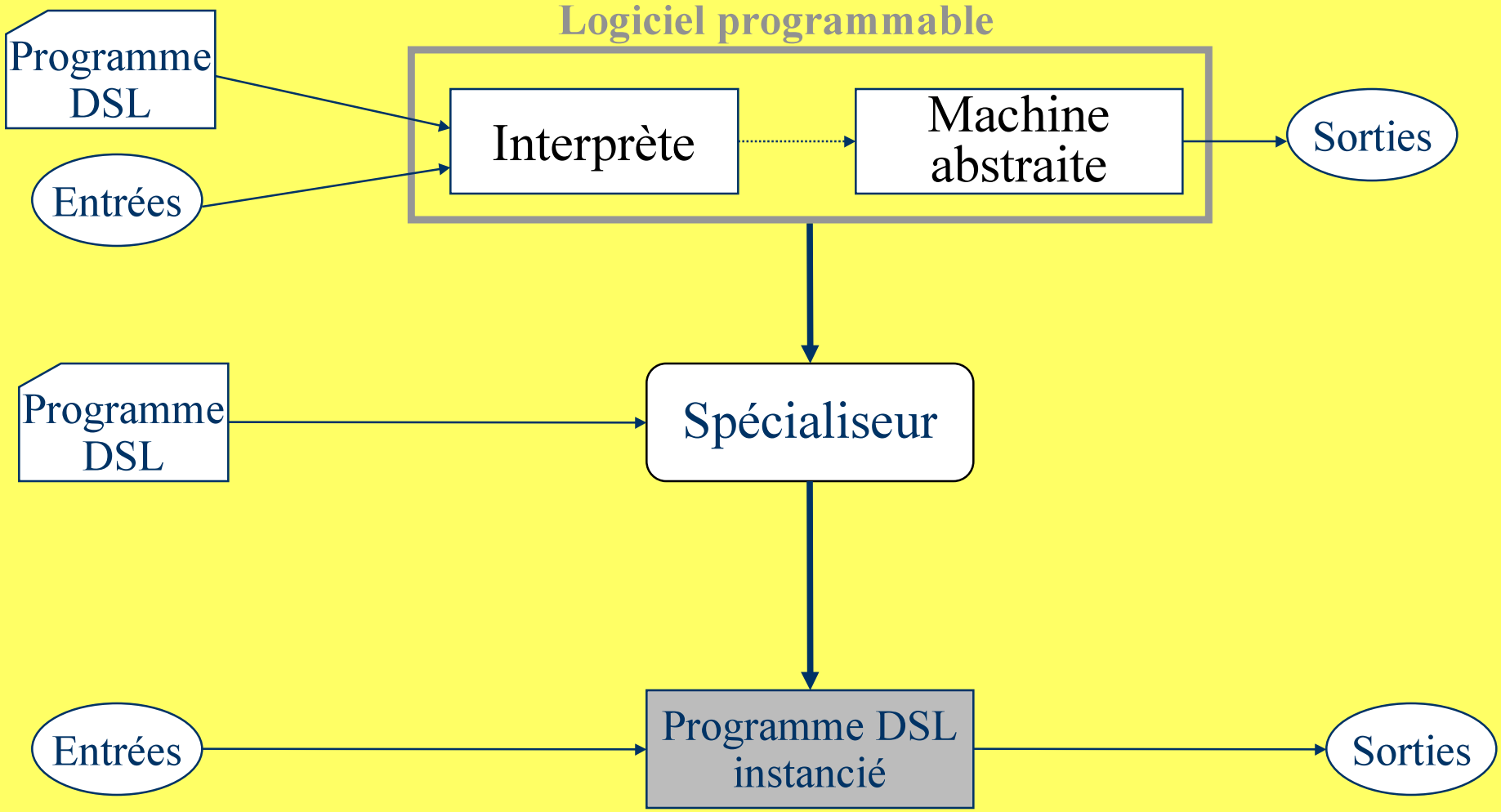
- Concision
  - Spécifications de pilotes 9 fois plus petites.
- Facilité de programmation.
- Facilité d'analyse
  - Sûreté et sécurité des programmes de routage.
- Ré-utilisation
  - Expertise.
  - Blocs de base.

Adaptation par Logiciels programmables

- Terminologie.
- Points communs.
- Variations.
- Opérations.
- Niveau du langage.
- Facilité d'analyse.
- …



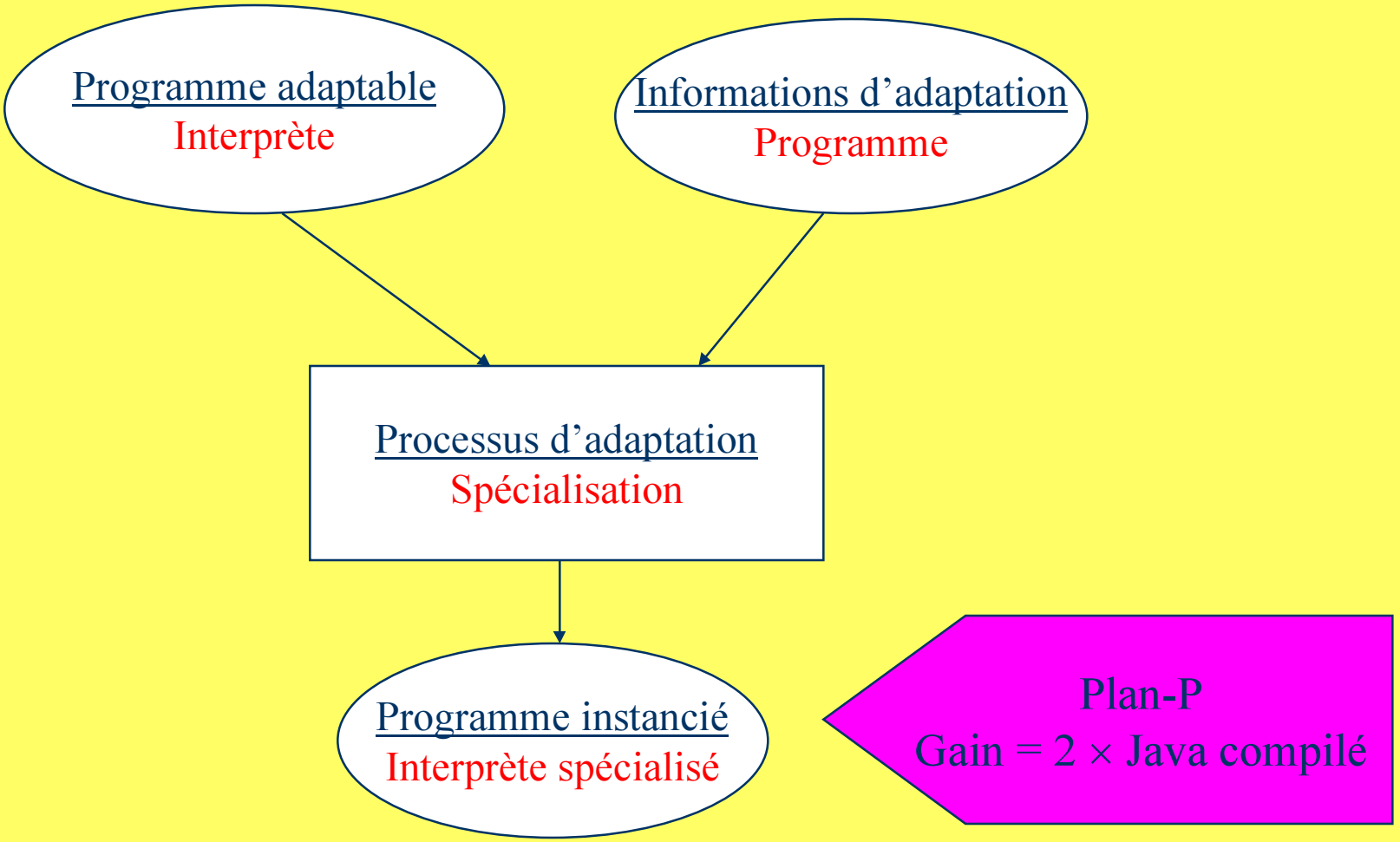
Processus d'adaptation



Evaluation de notre approche

- Coût d'implémentation:
  - Interprète *vs* compilateur.
- Interprète → compilation offline et *JIT*.
- Performances
  - GAL: comparable à C.
  - Plan-P: 2 × Java -- (offline *vs JIT*).

Adaptation par Logiciels programmables



Programmation adaptative

- Evolutions des besoins ⇒ Adaptabilité.
- Spécialisation
  - Différentes formes d'adaptabilité:

    Logiciel paramétré → Logiciel programmable.
- Avantages:
  - Réactivité.
  - Productivité.
  - Sûreté.
  - Efficacité.