

# Les langages dédiés: une approche sûre et efficace à la conception de systèmes d'exploitation

---

Gilles Muller

*Projet Compose*

*<http://www.irisa.fr/compose>*

INRIA/Irisa, Rennes (France)

# Conception de systèmes: enjeux et problèmes

---

- ◆ Réutilisation
  - temps de développement
- ◆ Sûreté/qualité du logiciel
  - confiance
  - maintenance
- ◆ Efficacité
  - temps d'exécution/taille mémoire

# Conception de systèmes: solutions actuelles

---

## ◆ Performance

- langage C
- optimisations manuelles

accès mémoire non anticipés, problèmes de réutilisation et de maintenance

## ◆ Sûreté/qualité du logiciel

- langages fortement typés (Java, ML, Modula-3)

performance ?

# Conception de systèmes : solutions actuelles

---

## ◆ Réutilisation

- micro-noyaux

coût de l'isolation, granularité de la réutilisation

- systèmes extensibles

complexité de la réutilisation des  
bibliothèques/machines abstraites internes au  
noyau

# Langages Dédiés : définition

---

- ◆ Langage restreint qui offre des abstractions spécifiques à un domaine
  - programmation concise
  - vérification de propriétés
- ◆ Interface à une bibliothèque/machine abstraite
  - langage «à colle»
  - intégration d'une expertise dans le langage
  - force l'utilisation correcte de la bibliothèque

# LD et systèmes : intérêt

---

- ◆ Simplification de la conception des noyaux
  - capitalisation de l'expertise sur les MA
  - concentration sur le quoi/comment
- ◆ Plus grande sûreté
  - restriction des opérations permises
  - vérification de propriétés
- ◆ Meilleure concision
  - maintenabilité et réutilisation accrues

# LD et systèmes : premières expérimentations

---

- ◆ GAL (spécification de pilotes graphiques)
  - langage proche des manuels constructeurs
  - concision d'un facteur 10/C
  - propriétés associées aux registres/horloges
- ◆ PLAN-P (protocoles pour les réseaux actifs)
  - concision d'un facteur 3/C
  - propriétés de sûreté et de sécurité
  - apprentissage/développement rapide

# LD et systèmes : problèmes soulevés

---

## Utilisation de nombreux langages

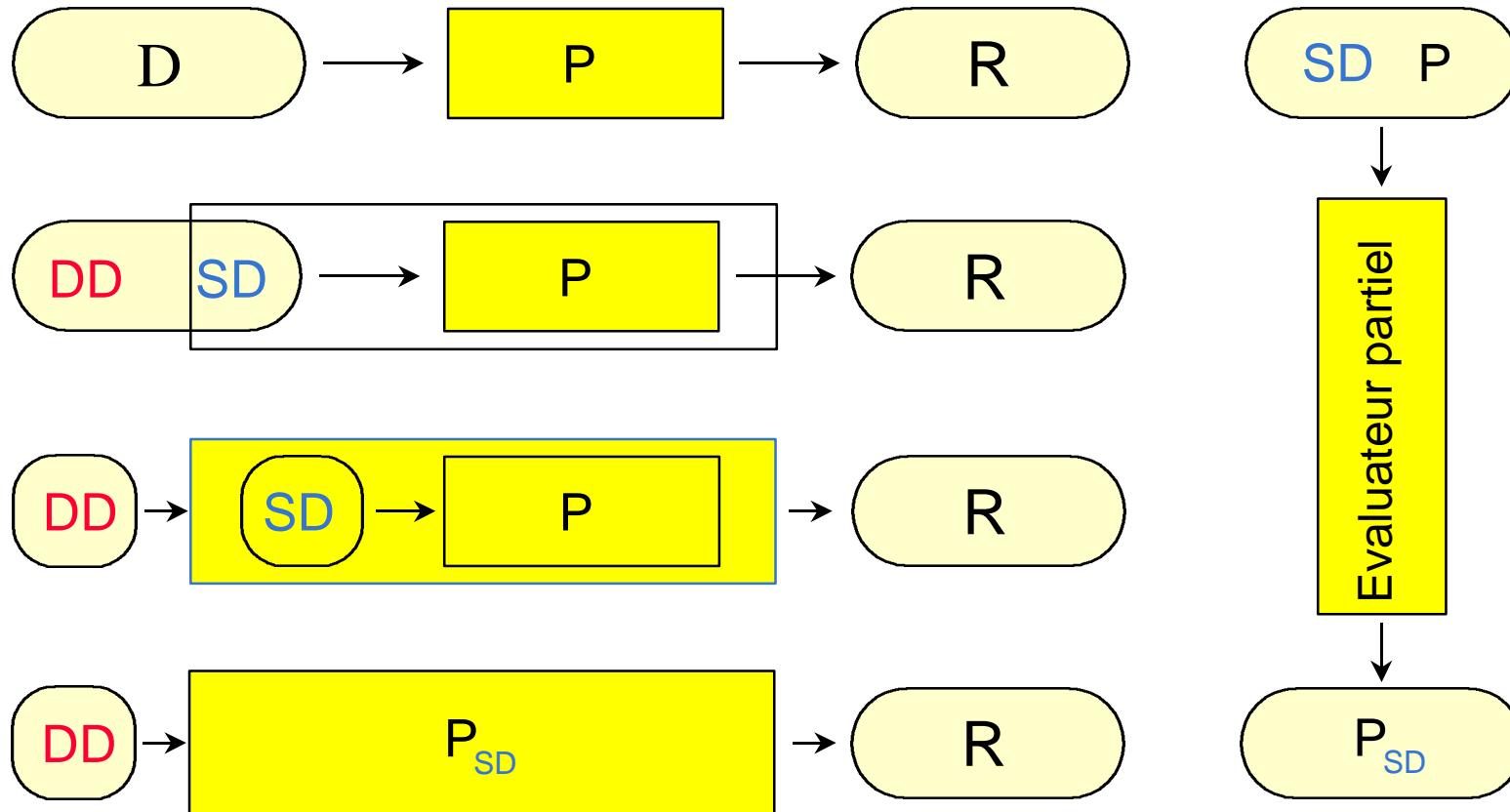
- ◆ Syndrome de la tour de Babel
- ◆ Outils d'aide à la conception des LD
- ◆ Développement de compilateurs efficaces

# LD et systèmes : compilation efficace

---

- ◆ Compilateurs optimisants
    - complexes à développer
    - spécifiques aux processeurs
    - difficulté d'évolution et de maintenance
  - ◆ Interprètes
    - simplicité de développement
    - lenteur d'exécution
- é Dilemme performance/maintenabilité

# Spécialisation (évaluation partielle)



# Spécialisation d'un LD :

## mini-printf

---

```
mini_printf(char fmt[], int
val[])
{
    int i = 0;
    for(; *fmt; fmt++) {
        switch(*fmt) {
            case '%' :
                switch(*++fmt) {
                    case '%' :
                        putchar('%'); break;
                    case 'd' :
                        putint(val[i++]); break;
                    default :
                        abort();
                } break;
            default :
                putchar(*fmt);
        }
    }
}
```

mini\_printf spécialisé avec  
fmt="n = %d"

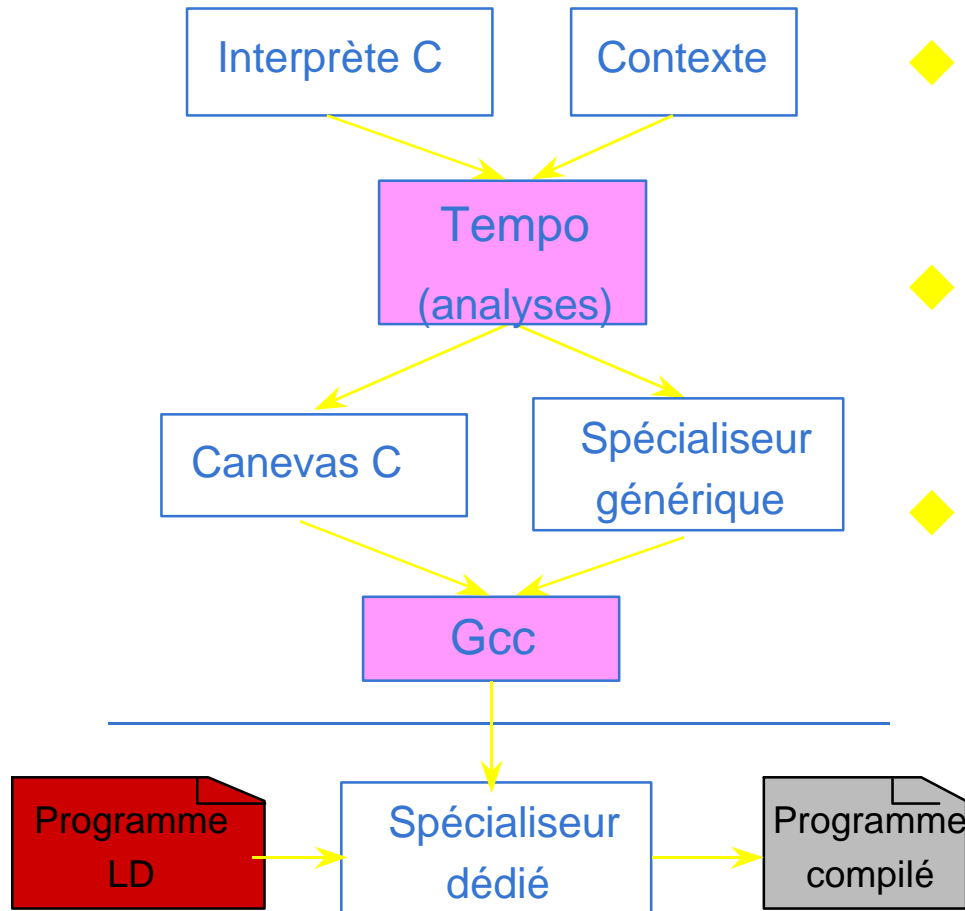
```
mini_printf_fmt(int val[])
{
    putchar('n');
    putchar(' ');
    putchar('=');
    putchar(' ');
    putint(val[0]);
}
```

# Tempo

---

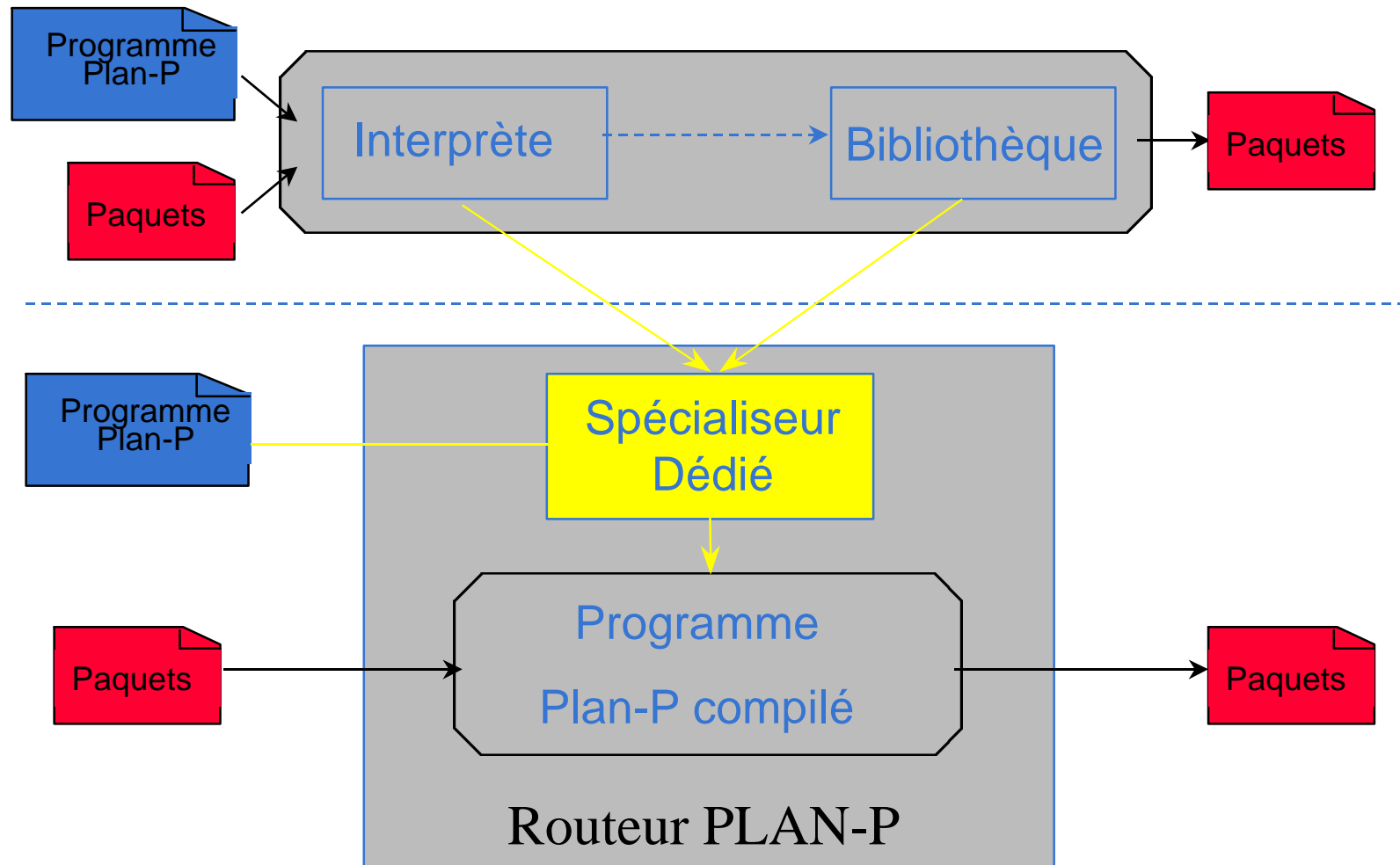
- ◆ Spécialisation de programmes C
  - code système, calcul scientifique
- ◆ Spécialisation à la compilation
  - interprète C -> compilateur traditionnel
- ◆ Spécialisation à l'exécution
  - interprète C -> compilateur à la volée (JIT)

# Génération d'un compilateur à la volée



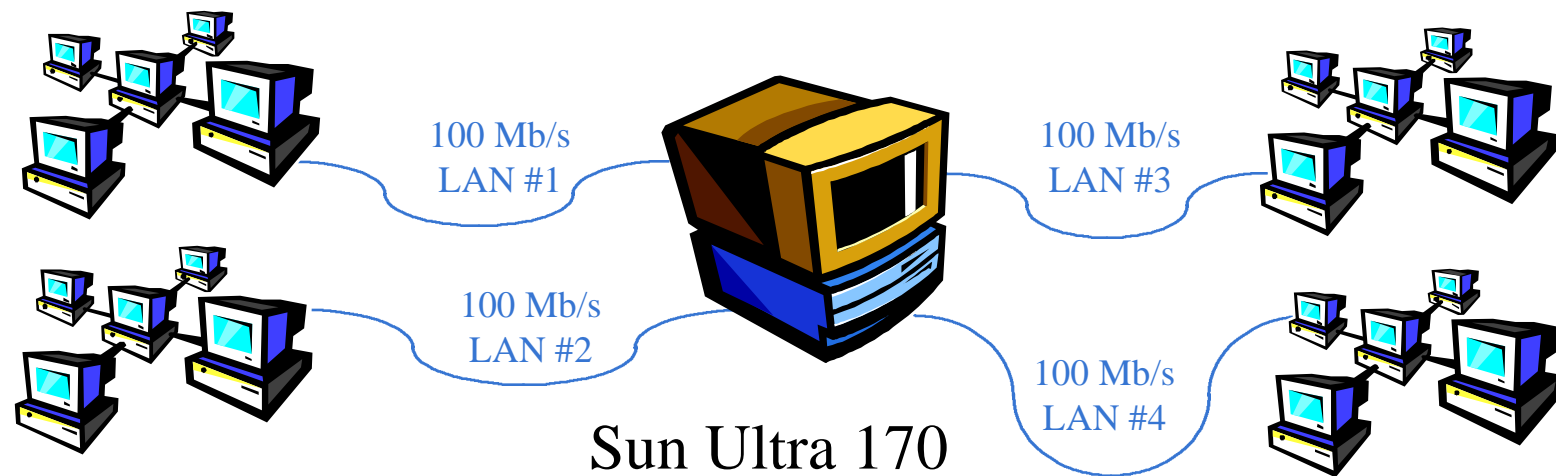
- ◆ Canevas binaires assemblés à l'exécution
- ◆ Préservation de la portabilité
- ◆ Cibles:
  - Pentium, Sparc

# LD et systèmes extensibles: PLAN-P



# Une passerelle Ethernet intelligente

- ◆ Alexander-al-sigcomm'97 (UPenn)
- ◆ Apprentissage de la localisation des machines



# Comparaison avec Alexander-97

<i>TTCP</i> 8 kbytes packets	<i>Platform</i> <i>Language</i>	<i>Direct</i> <i>Throughput</i> A	<i>C-Repeater</i> <i>Throughput</i> B	<i>Active</i> <i>Bridging</i> C
<b>Alexander-97</b> (TCP)	Pentium 166 Linux <b>Caml</b>	76 Mb/s	36.6 Mb/s 44 % of A	16 Mb/s 21% of A <b>43% of B</b>
<b>Plan-P</b> (TCP)	Sun Ultra 170 <b>Plan-P Rtspec</b>	73.7 Mb/s	42.9 Mb/s 58 % of A	40 Mb/s 54% of A <b>93% of B</b>
<b>PLAN-P</b> (UDP)	Sun Ultra 170 <b>Plan-P Rtspec</b>		88.1 Mb/s	88.1 Mb/s <b>100% of B</b>

# Les prochaines étapes

---

- ◆ Hop : généralisation de GAL
- ◆ LD pour les caches Web
- ◆ LD pour l'ordonnancement de processus
- ◆ LD pour la gestion de fichiers

# Conclusion

---

Il est réaliste d'utiliser des LD pour concevoir des systèmes d'exploitation

- ◆ systèmes extensibles
- ◆ systèmes critiques
- ◆ systèmes embarqués grand public
- ◆ systèmes de longue durée de vie

# Questions ?

---

PLAN-P, GAL et Tempo sont disponibles via  
le web

<http://www.irisa.fr/compose/>

```
protocol bridge is
```

```
.....
```

```
channel system(ps : unit, cs : (interface*time) hash_table,  
              p : ethernet*blob)  
  initstate mkTable(100) is  
  let val dst : etherhost = etherDst(#1 p)  
  in  
    if isBroadcast(dst) or isMulticast(dst) then  
      (simpleRepeat(p); (ps,cs))  
    else  
      let val t : time = getTime()  
      in  
        (insert(cs,hashKey(etherSrc(#1 p)),  
              (thisInterface(),t));  
         try  
           let val entry : interface*time =  
               find(cs,hashKey(dst))  
           in  
             if (t-(#2 entry))<timeLimit then  
               (OnNeighbor(system,#1 entry,p); (ps,cs))  
             else  
               (simpleRepeat(p); (ps,cs))  
             end  
             handle NoEntry => (simpleRepeat(p); (ps,cs)))  
           end  
        end  
      end  
  end
```

# Temps de compilation

---

	Passerelle Ethernet	Diffusion Audio Router	Diffusion Audio Player	Serveur Web	MPEG Moniteur	MPEG Player
Lignes	40	68	28	91	161	33
Temps de génération (ms)	15	11.6	6.2	15.3	53.9	6.1

# Bande passante sous TCP

