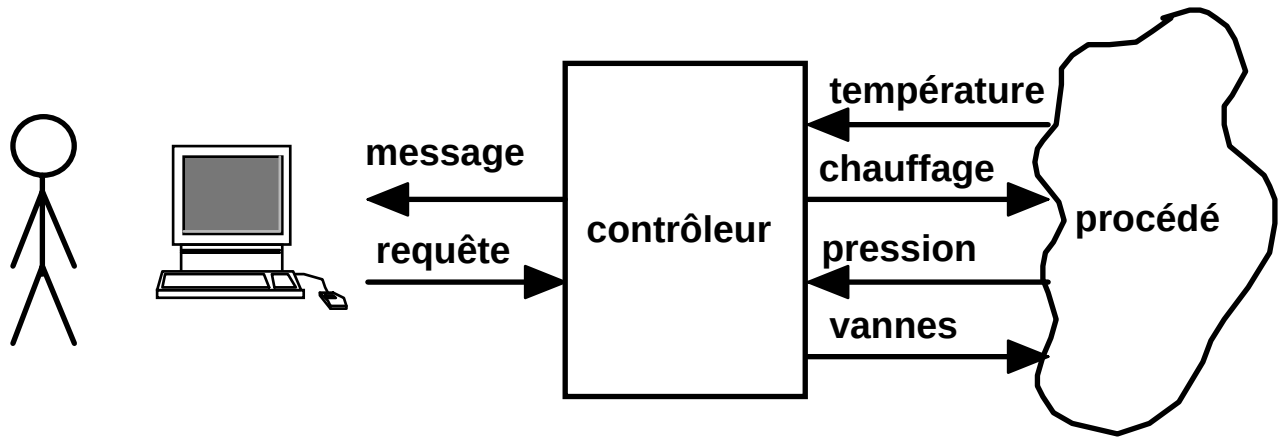
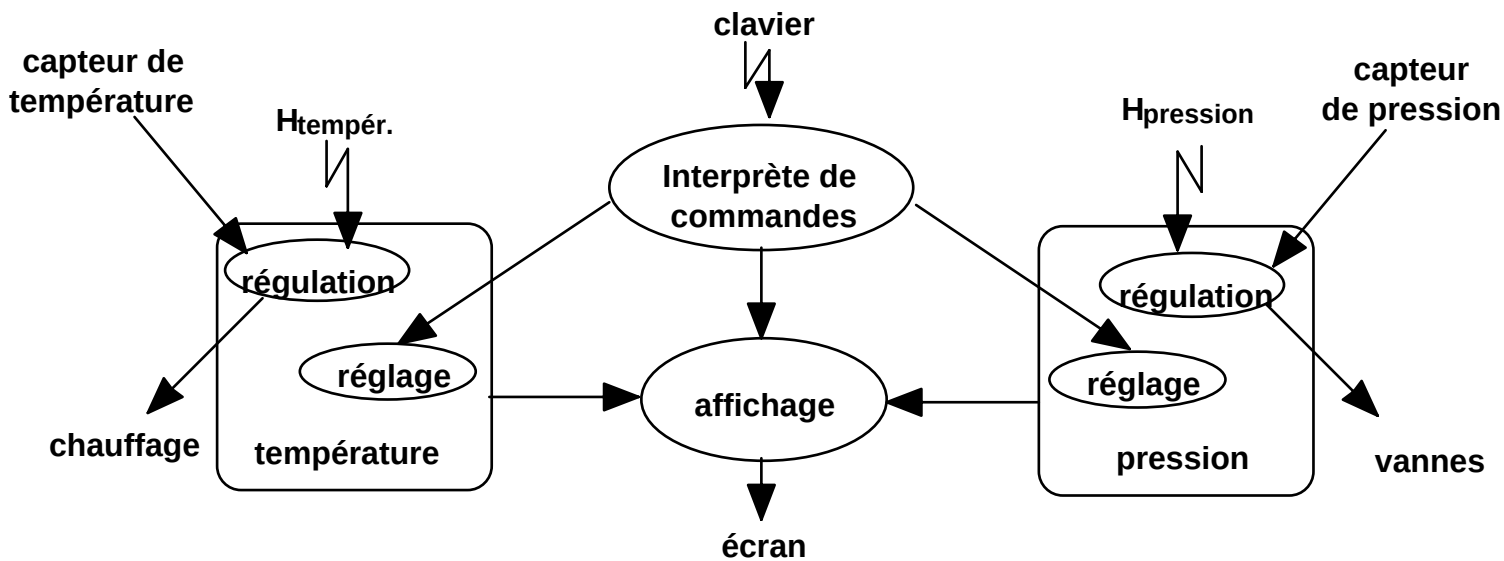
Exécutifs Temps Réel

SCEPTRE

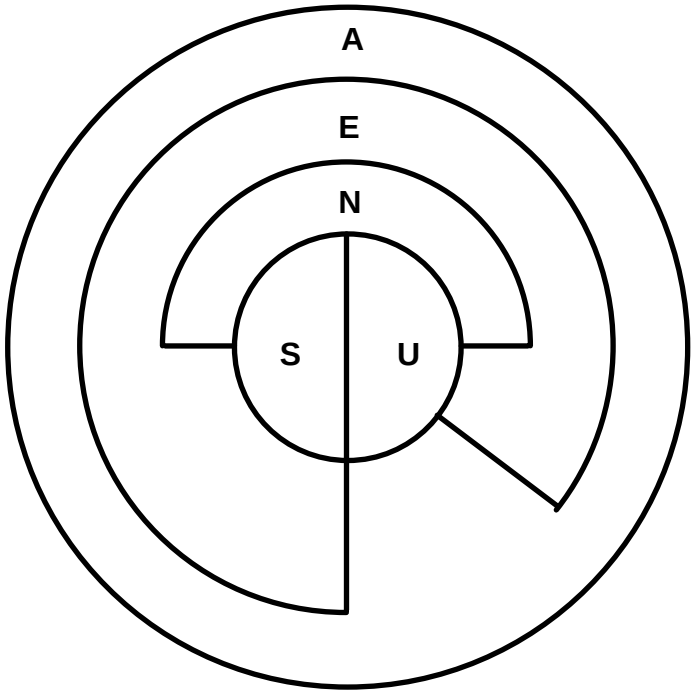
Exemple


message
requête
contrôleur
température
chauffage
pression
vannes
procédé

Les Tâches

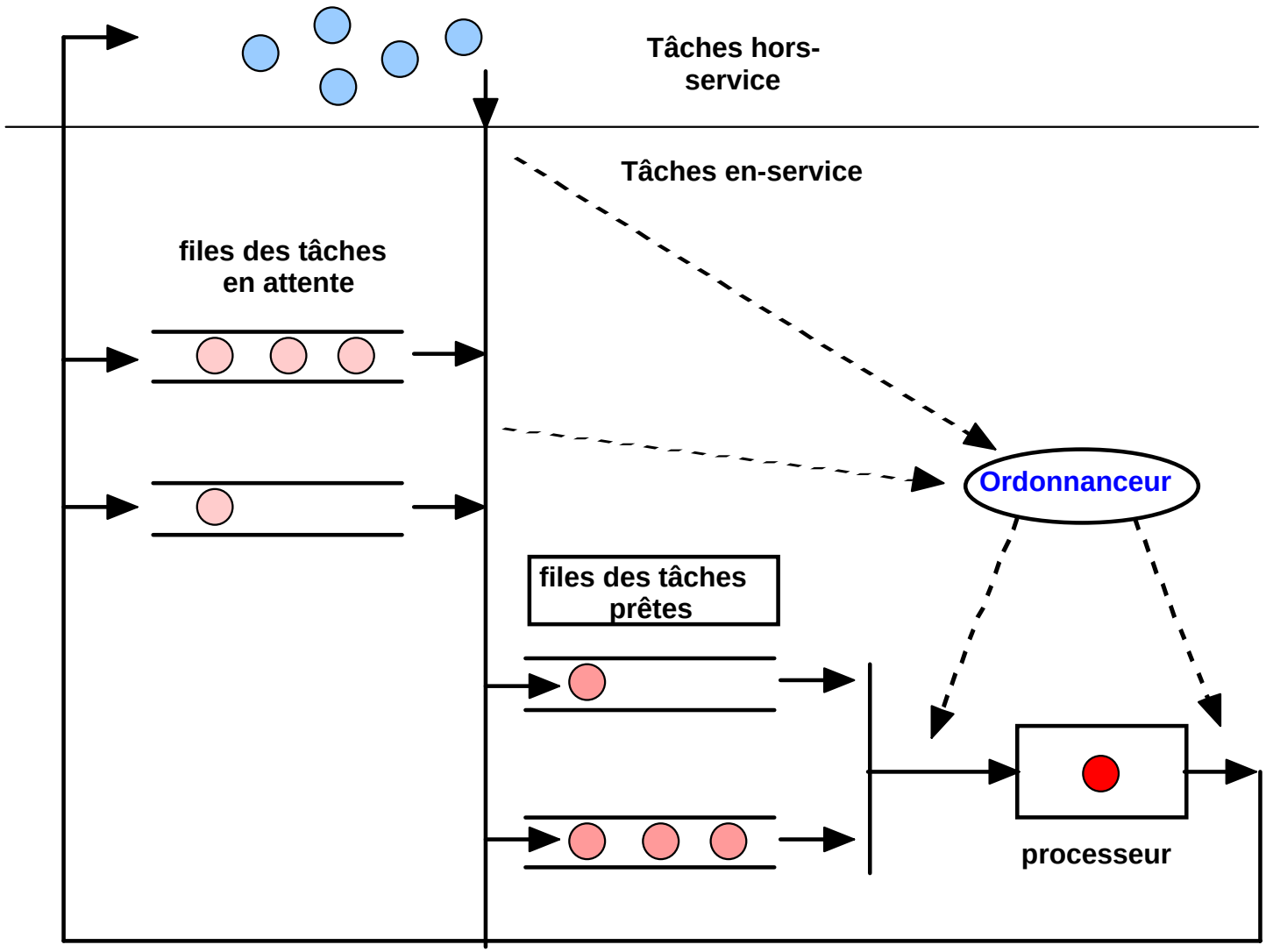

clavier
capteur de température
Htempér.
Interprète de commandes
Hpression
capteur de pression
régulation
réglage
chauffage
température
affichage
régulation
réglage
pression
vannes
écran

Structure en couches

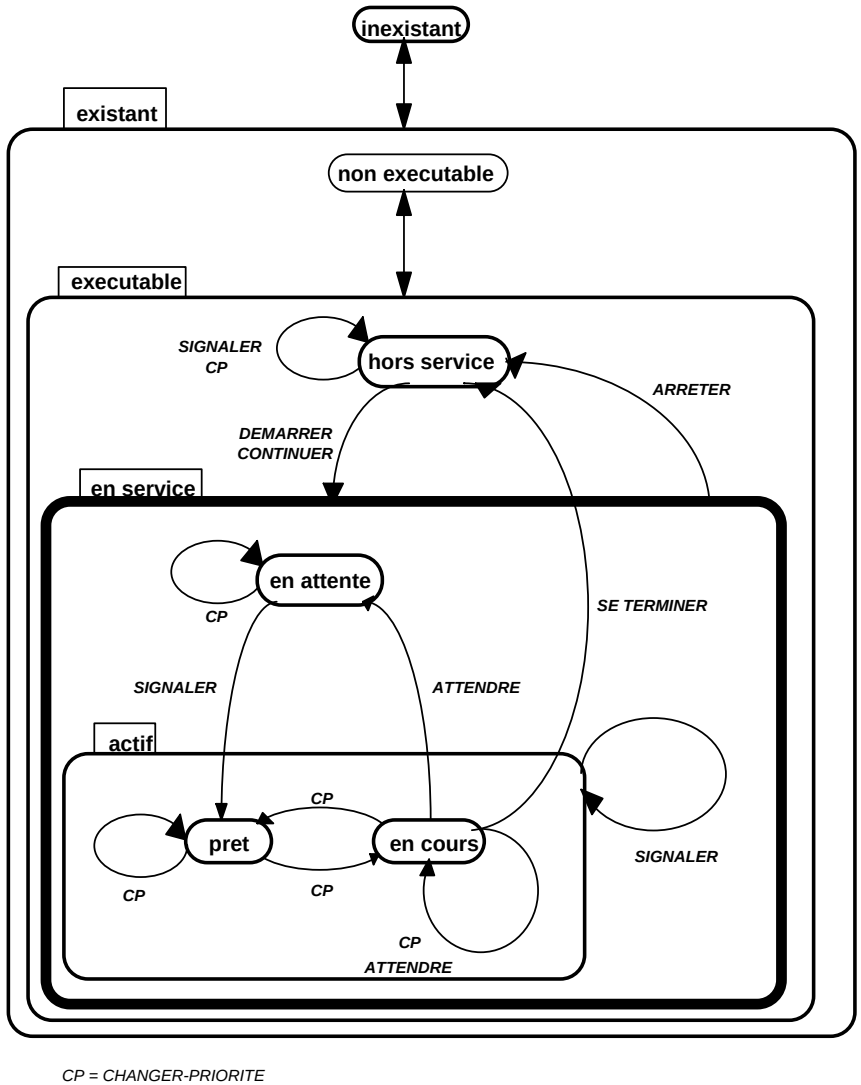


A : Application
E : Exécutif
N : Noyau
S : mode superviseur
U : mode utilisateur

Architecture d'un RTOS

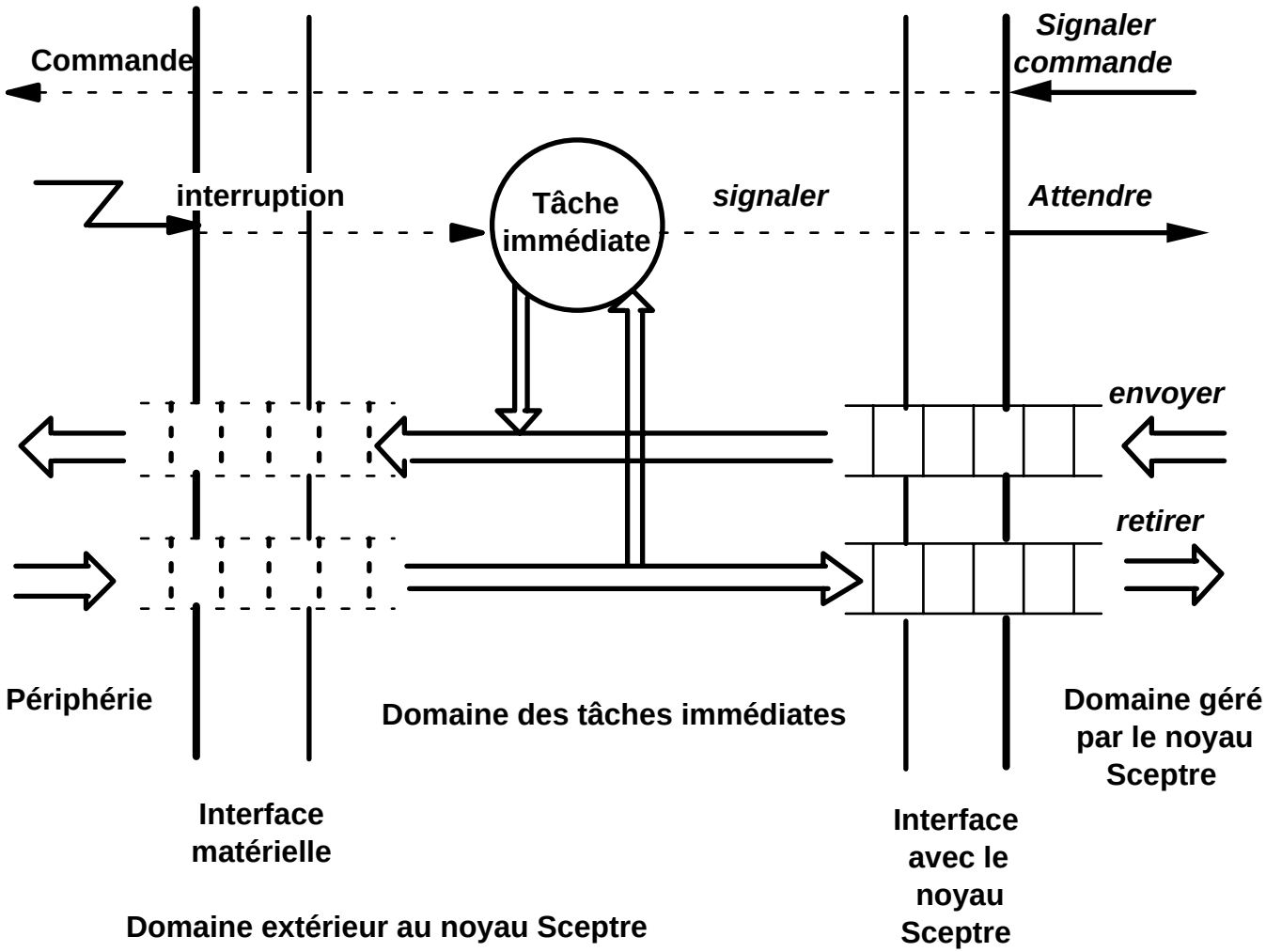


Etats des tâches



CP = CHANGER-PRIORITE

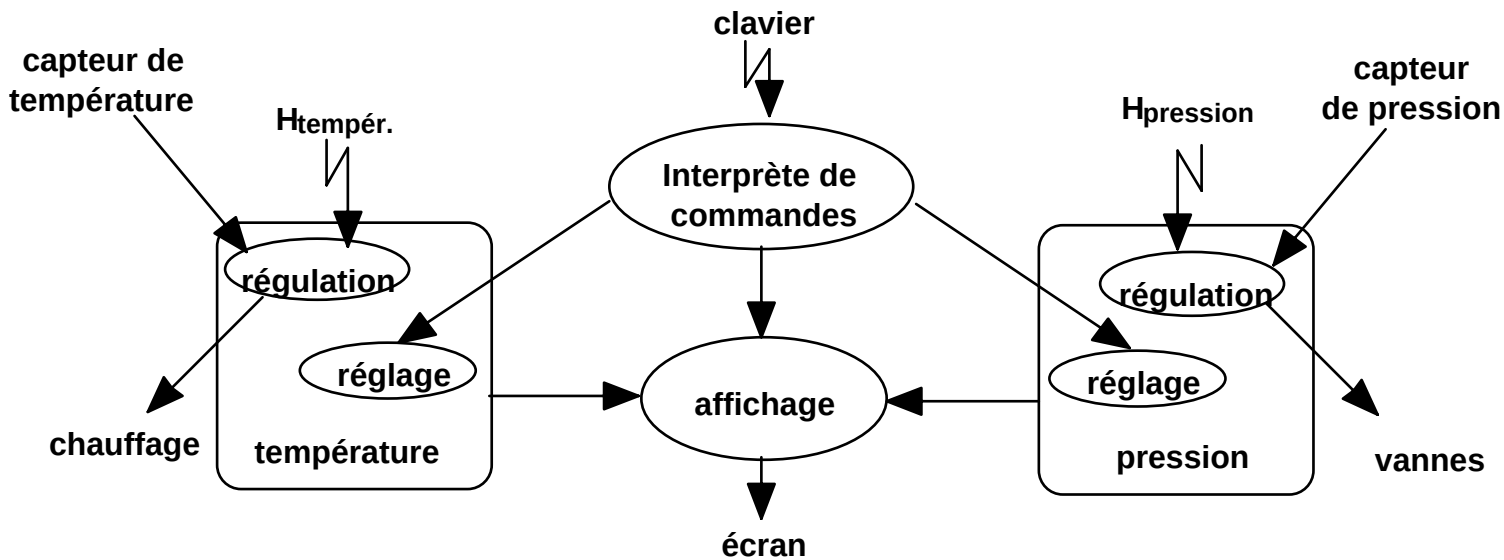
Exemple d'E/S


Commande
Signaler commande
interruption
Tâche immédiate
signaler
Attendre
envoyer
retirer
Périphérie
Domaine des tâches immédiates
Domaine géré par le noyau Sceptre
Interface matérielle
Interface avec le noyau Sceptre
Domaine extérieur au noyau Sceptre

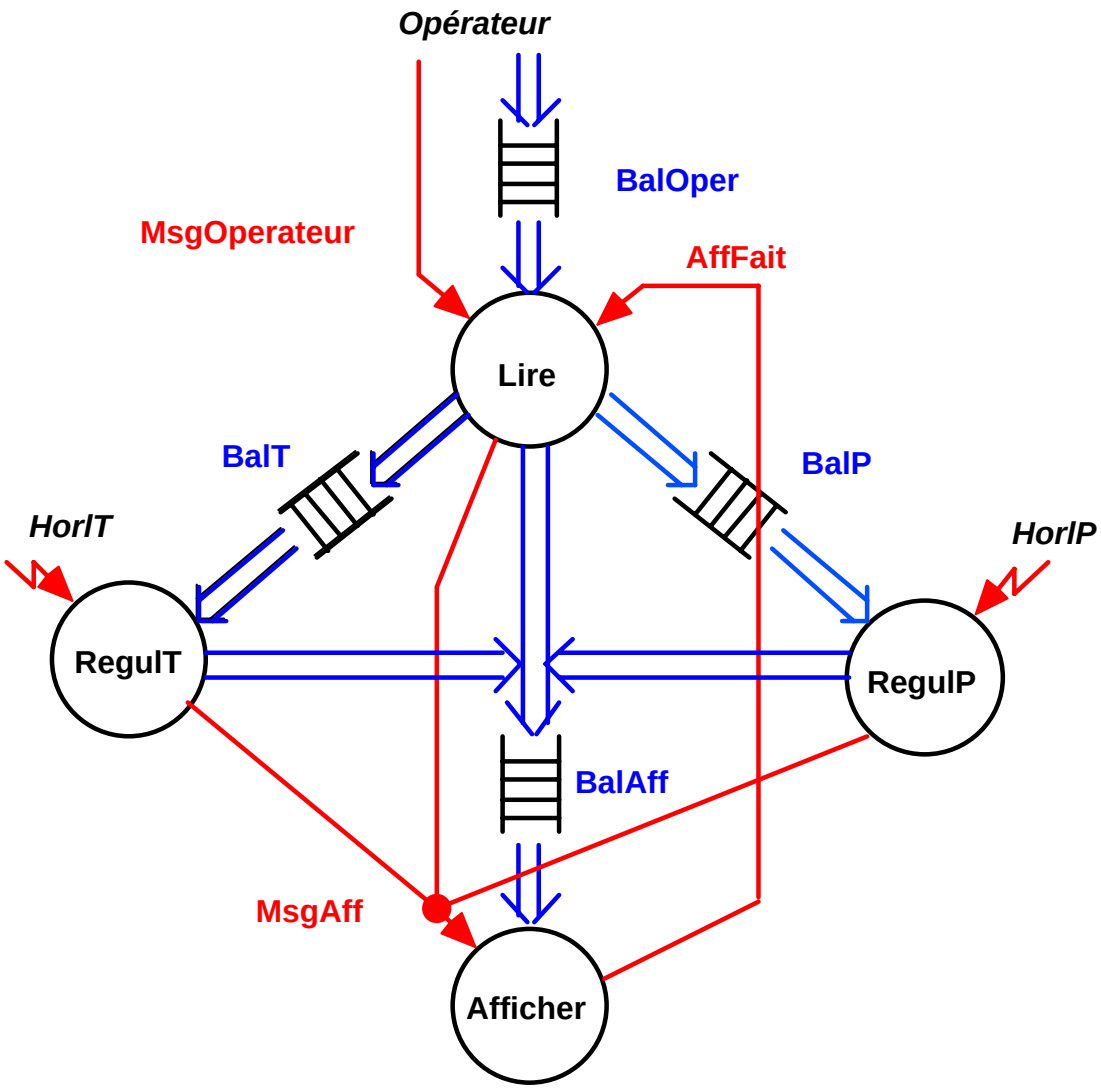
Application

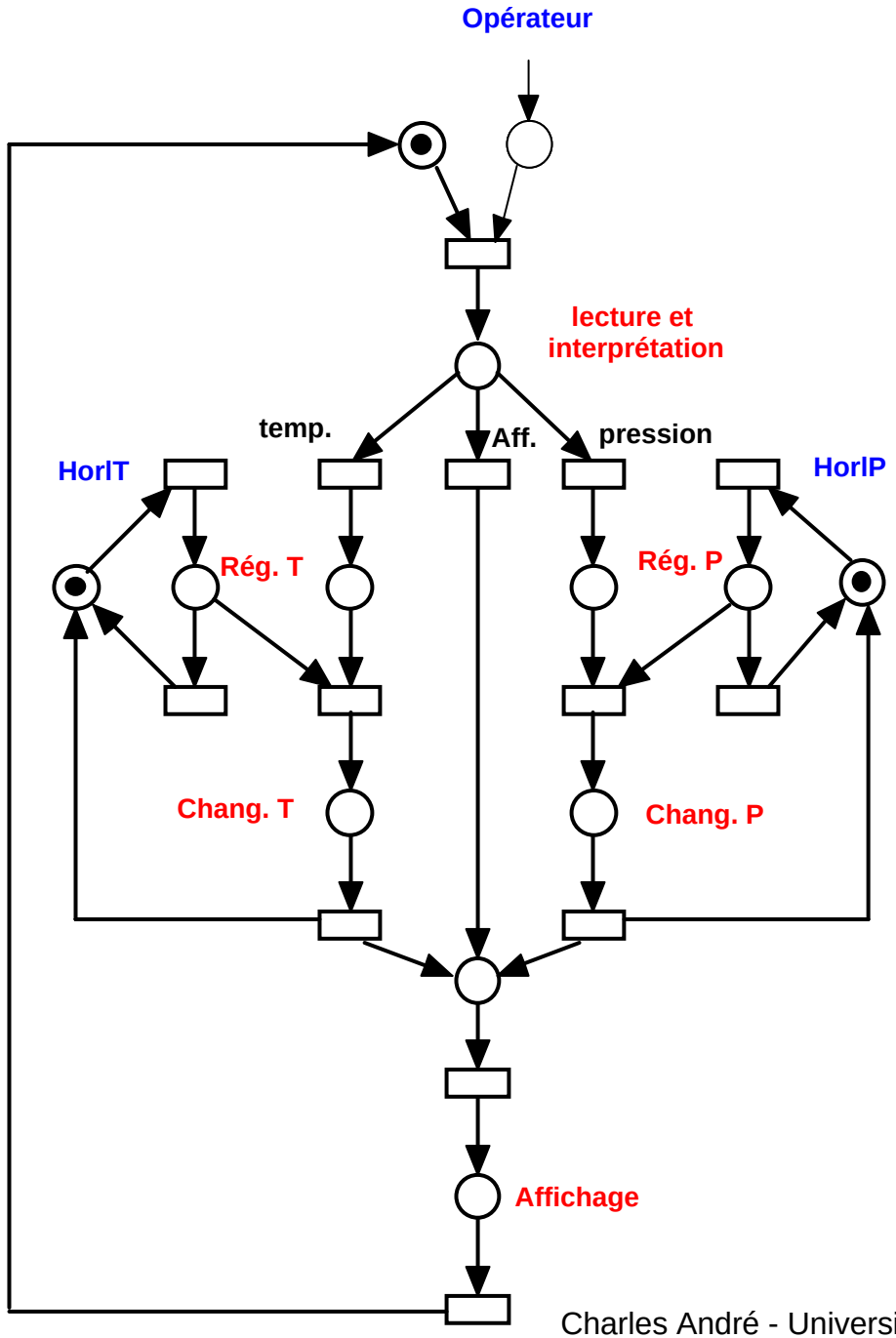
1. Les tâches
2. Les objets
3. Le comportement

Les tâches (rappel)



Les objets


Opérateur
BalOper
MsgOperateur
AffFait
Lire
BalT
BalP
HorlT
HorlP
RegulT
RegulP
BalAff
MsgAff
Afficher


Opérateur
lecture et
interprétation
temp.
Aff.
pression
HorlT
HorlP
Rég. T
Rég. P
Chang. T
Chang. P
Affichage

Programmation (1)

```
// Déclarations
Init, Lire, RegulT, RegulP,
  Afficher: TACHE;
MsgOperateur, AffFait, MsgAff,
  HorlT, HorlP: EVENEMENT;
BalAff, BalOper, BalT, BalP:
  FILE;
RC, RA, R: REGION;
```

Programmation (2)

```
TACHE Init:
Debut
    ENTRER(R);
        DEMARRER(RegulT);
        DEMARRER(RegulP);
        DEMARRER(Lire);
        DEMARRER(Afficher);
    SORTIR(R);
fin
```

Programmation (3)

```
TACHE RegulT:
Tconsigne: float;
CompteRendu: Message_t;
Debut
   EFFACER([HorlT]);
   boucle
      ATTENDRE([HorlT]);
      EFFACER([HorlT]);
      // algorithme de régulation
      // changement de consigne ?
```

Programmation (4)

```
        si non VIDE(BalT) alors
            RETIRER(Tconsigne,BalT);
            // modifier les paramètres
            ENTRER(RA);
                ENVOYER(CompteRendu,BalAff);
                SIGNALER(MsgAff,Afficher);
            SORTIR(RA);
          fin si
      fin boucle
fin
```

Programmation (5)

```
TACHE RegulP:
Pconsigne: float;
CompteRendu: Message_t;
Debut
   EFFACER([HorlP]);
   boucle
      ATTENDRE([HorlP]);
      EFFACER([HorlP]);
      // algorithme de régulation
      // changement de consigne ?
```

Programmation (6)

```
        si non VIDE(BalP) alors
            RETIRER(Pconsigne,BalP);
            // modifier les paramètres
            ENTRER(RA);
                ENVOYER(CompteRendu,BalAff);
                SIGNALER(MsgAff,Afficher);
            SORTIR(RA);
          fin si
      fin boucle
fin
```

Programmation (7)

```
TACHE Lire:
Cmd, Consigne, CompteRendu: Message_t;
Debut
   boucle
      ENTRER(RC);
      si VIDE(BalOper) alors
         EFFACER([MsgOperateur]);
         SORTIR(RC);
         ATTENDRE([MsgOperateur]);
      sinon
         SORTIR(RC)
      fin si;
```

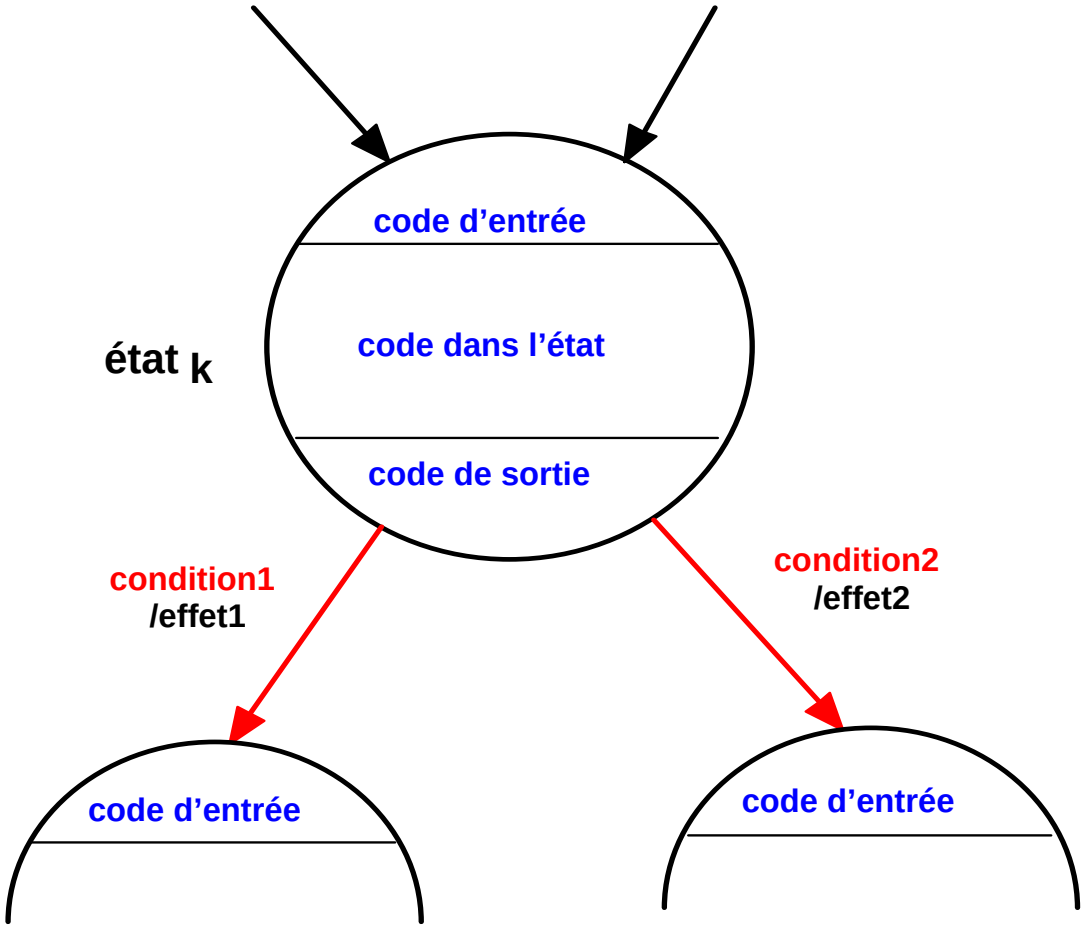
Programmation (8)

```
        RETIRER(Cmd, BalOper);
        // décoder et interpréter
        // en déduire Cmd et Consigne
        EFFACER([AffFait]);
        cas où
           Cmd = 'température':
                ENVOYER(Consigne,BalT);
           Cmd = 'pression':
                ENVOYER(Consigne,BalP);
           Cmd = 'affichage':
                ENTRER(RA);
                   ENVOYER(CompteRendu,BalAff);
                   SIGNALER(MsgAff,Afficher);
                SORTIR(RA);
        fin cas;
        ATTENDRE([AffFait]);
    fin boucle
fin
```

Multitâche coopératif

- Les diverses tâches exécutent du code
  - non bloquant (lecture ou écriture)
  - et de durée connue (pas de tant que avec nombre d'itérations non prévisible)
- Pas d'exécutif particulier pour gérer ces tâches
- Cas particulier intéressant en contrôle : découpage en états avec actions d'entrée, de sortie et activités.

Multitâche coopératif avec états



États : programmation

On distingue 3 parties dans le code :

1. Partie entrée : ce code n'est exécuté que lors de l'entrée dans l'état
2. Partie régime permanent : ce code est exécuté à chaque retour dans l'état. Cette partie de code est suivie du test des conditions de sortie de l'état. Si une condition est vrai, on sélectionne l'état suivant correspondant. Si aucune condition n'est satisfaite, l'état suivant reste l'état courant.
3. Partie sortie : ce code est exécuté quand on change d'état

États : programmation en C

- Un état est codé par une fonction

  `void State_Xxx (void)`

- L'état courant et l'état suivant sont des variables du type pointeur sur fonction :

```
void (* currentState) (void);
void (* nextState) (void);
```

États : programmation en C

```
void State_Xx ( void )
 {
   static Boolean entering = true;
   // other local variables
   if (entering) {
      entering = false;
      // actions to execute when entering
      return;
   }
   // code to execute when IN the state
     ...
   // check transitions for leaving (see next page)
```

États : programmation en C

```
if ( cond1 ) {
    // code to execute when leaving
    Trans_Action1(…);
    nextState = State_X1;
} else {
    if ( cond2 ) {
        // code to execute when leaving
        Trans_Action2(…);
        nextState = State_X2;
    } else {
        // stay in this state
        return;
    }
}
entering = true;
}
```