IFT6223
Systèmes reconfigurables

ASM

Jean Pierre DAVID

david@iro.umontreal.ca

Université de Montréal

Plan

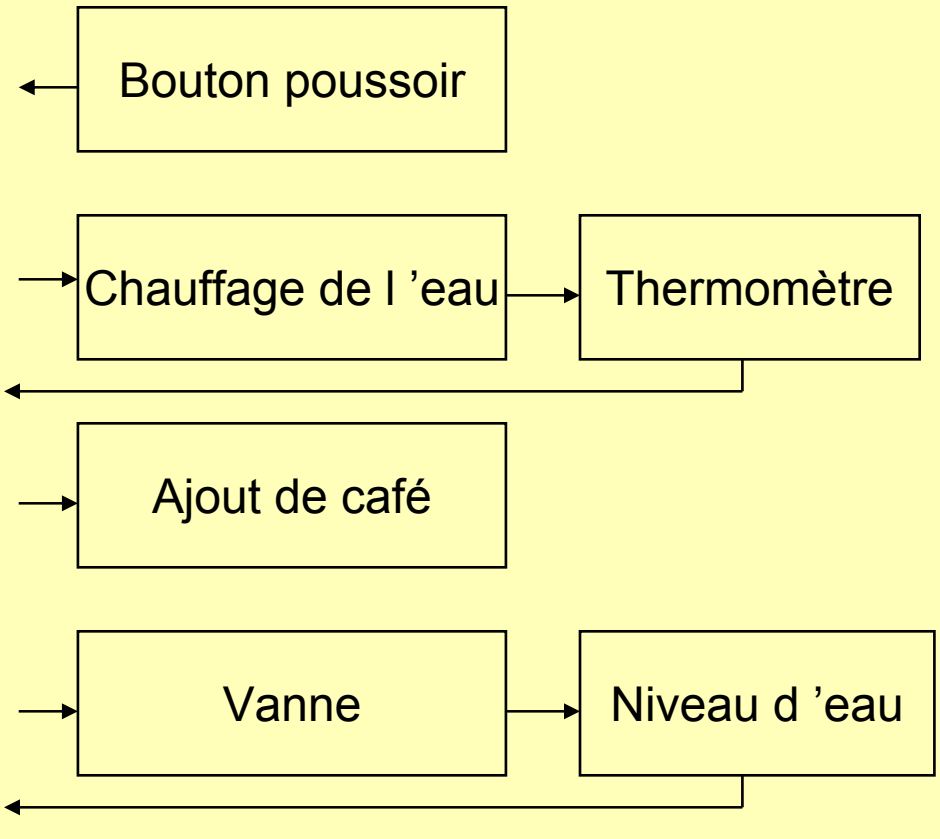
- Introduction à la synthèse de systèmes numériques.
- Rappels sur la logique synchrone
- La notion d'ASM
- Le passage d'un langage de haut niveau (C) à l'ASM
  - Un exemple illustratif (Algorithme de Fermat)
- Une première optimisation
  - Application à l'exemple
- Synchronisation des transferts de données
  - Application à l'exemple
- Perspectives

Exemple: Une machine à café automatique

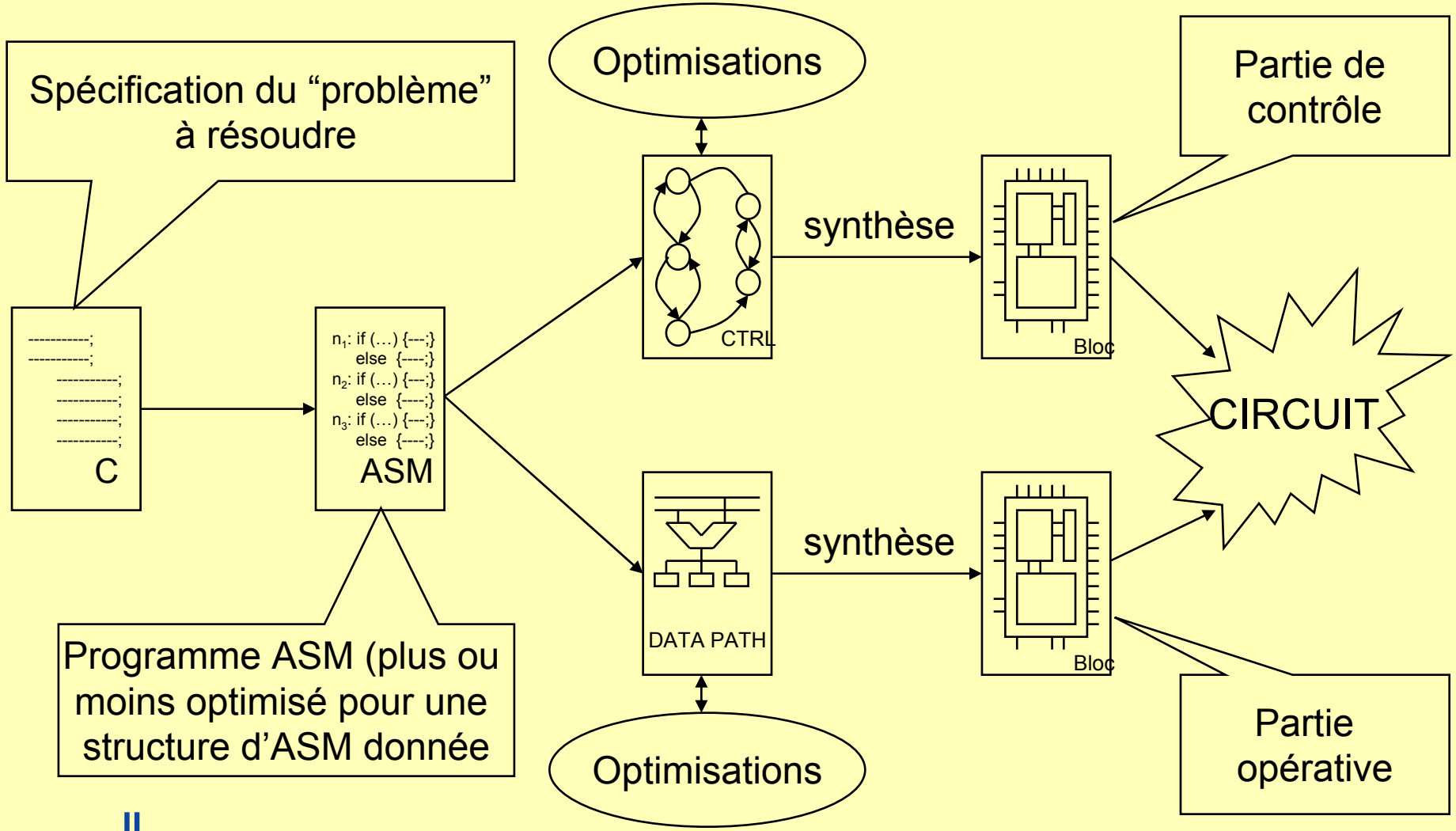
Partie de contrôle

- Attendre que l 'on pousse sur le bouton
- Chauffer l 'eau
- Attendre qu 'elle soit chaude
- Mettre une dose de café
- Ouvrir la vanne d 'eau
- Attendre que la tasse soit remplie
- Fermer la vanne
- Arrêter le chauffage de l 'eau

Partie opérative

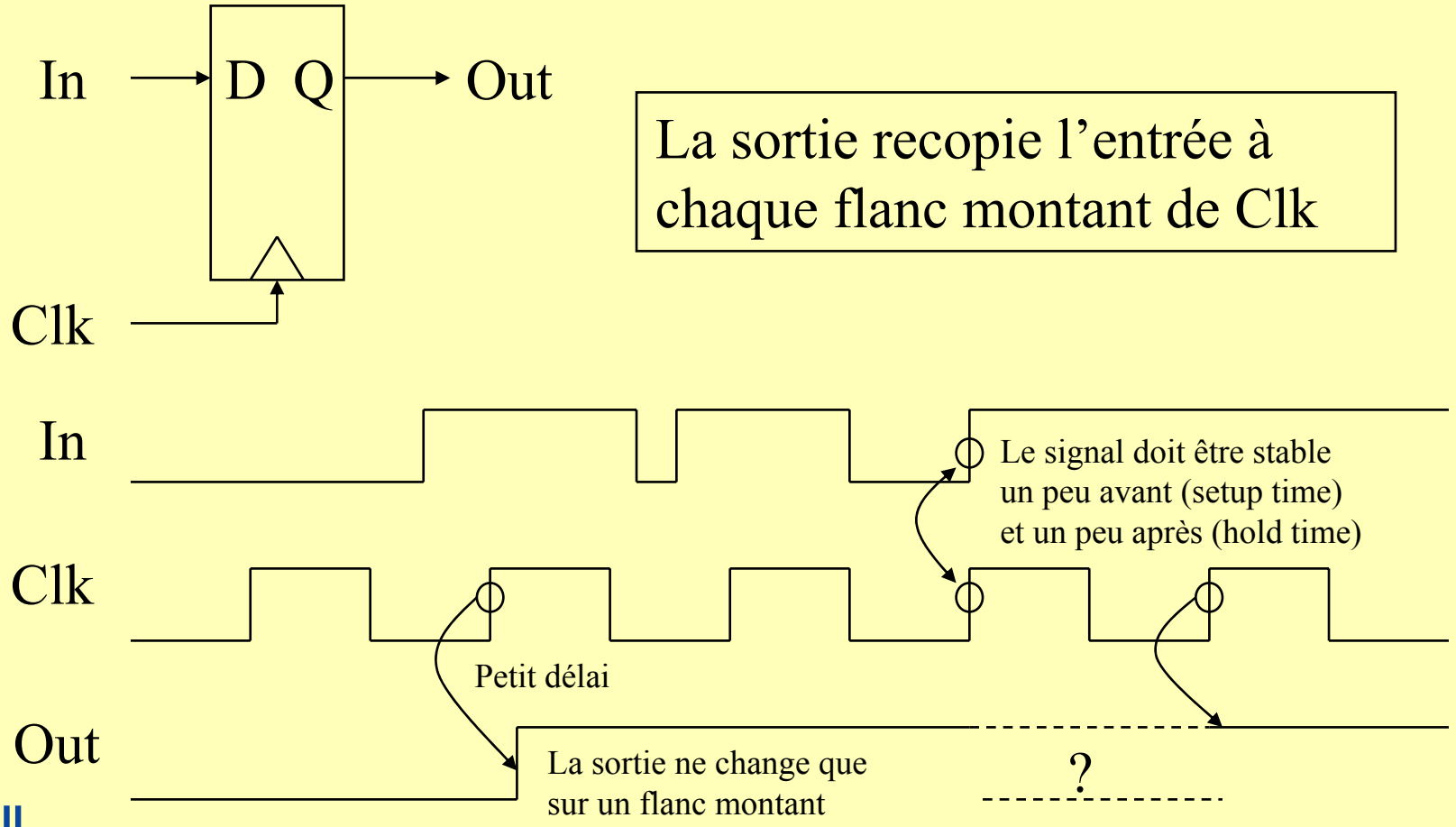


La conception/réalisation d'un circuit

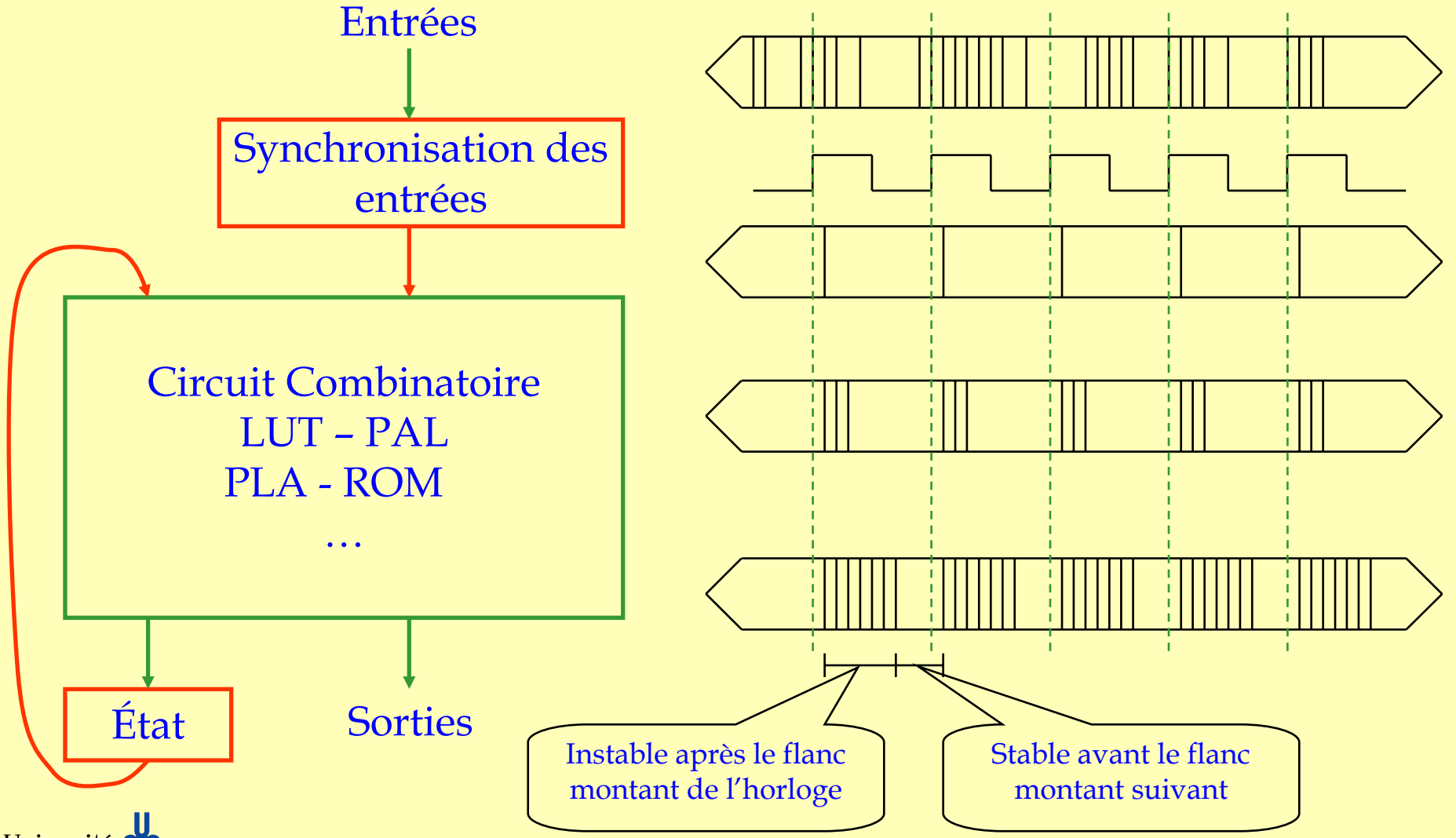


La logique synchrone (1/2)

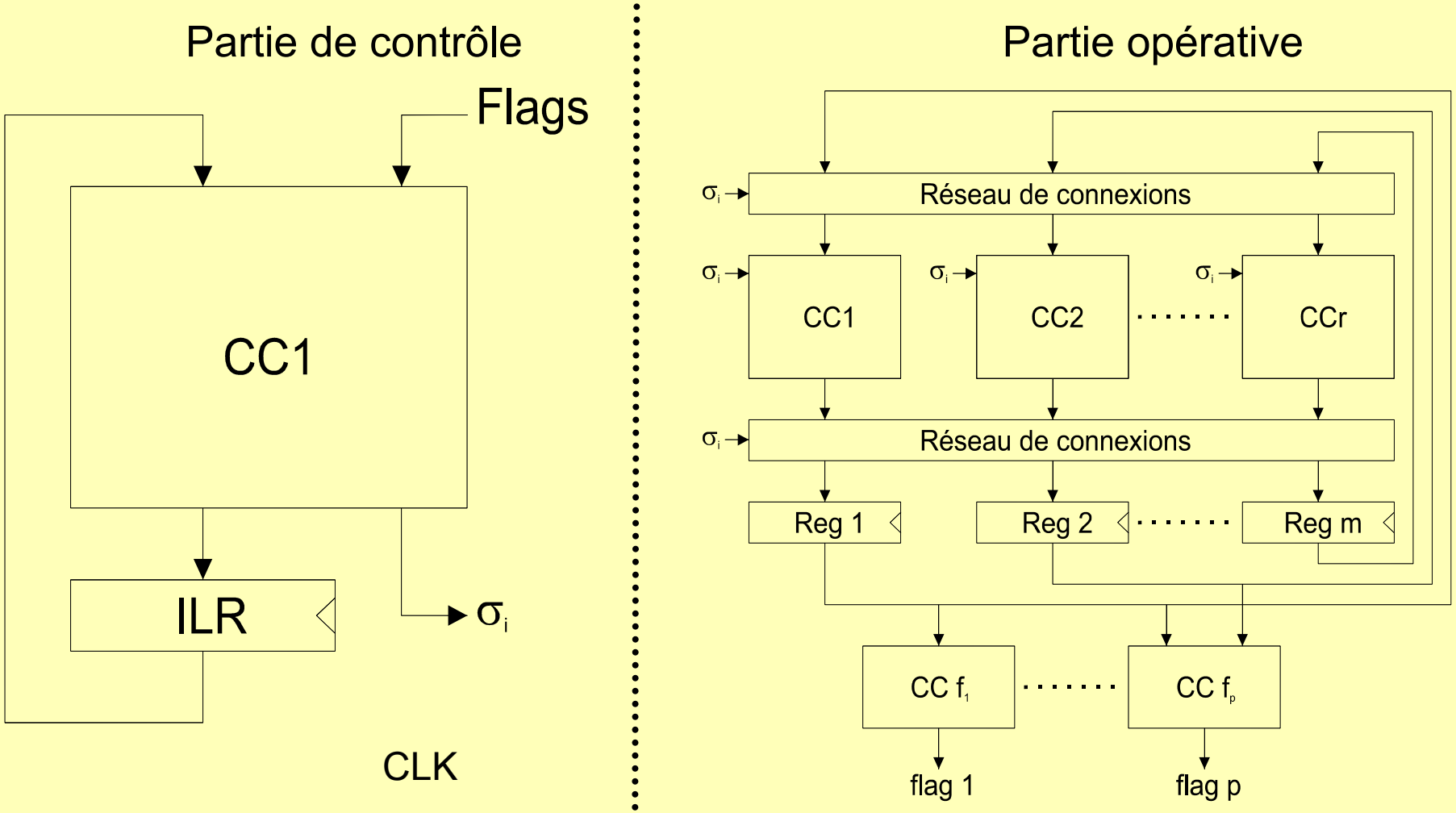
Le flip-flop D (ou Registre)



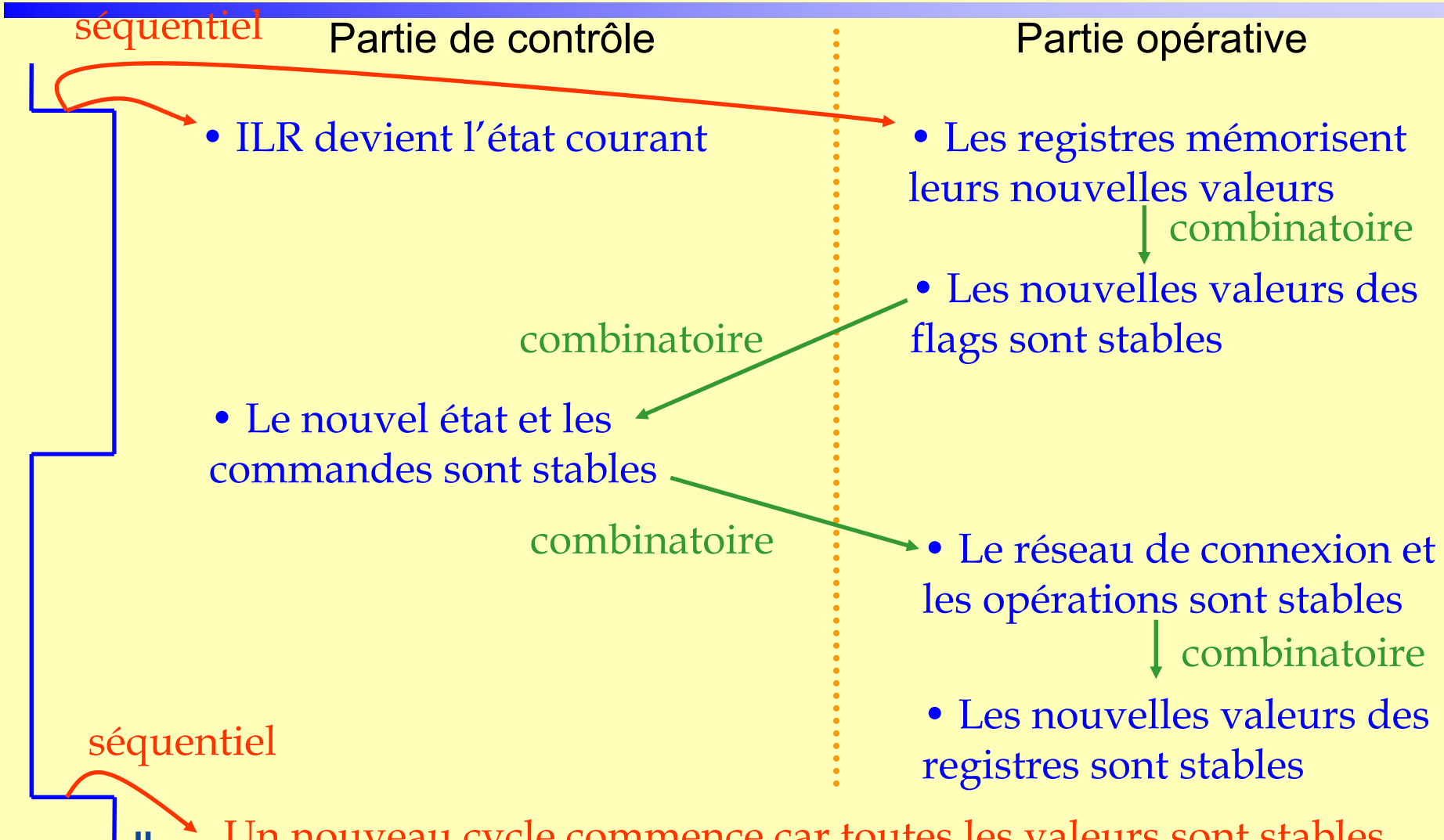
La logique synchrone (2/2)



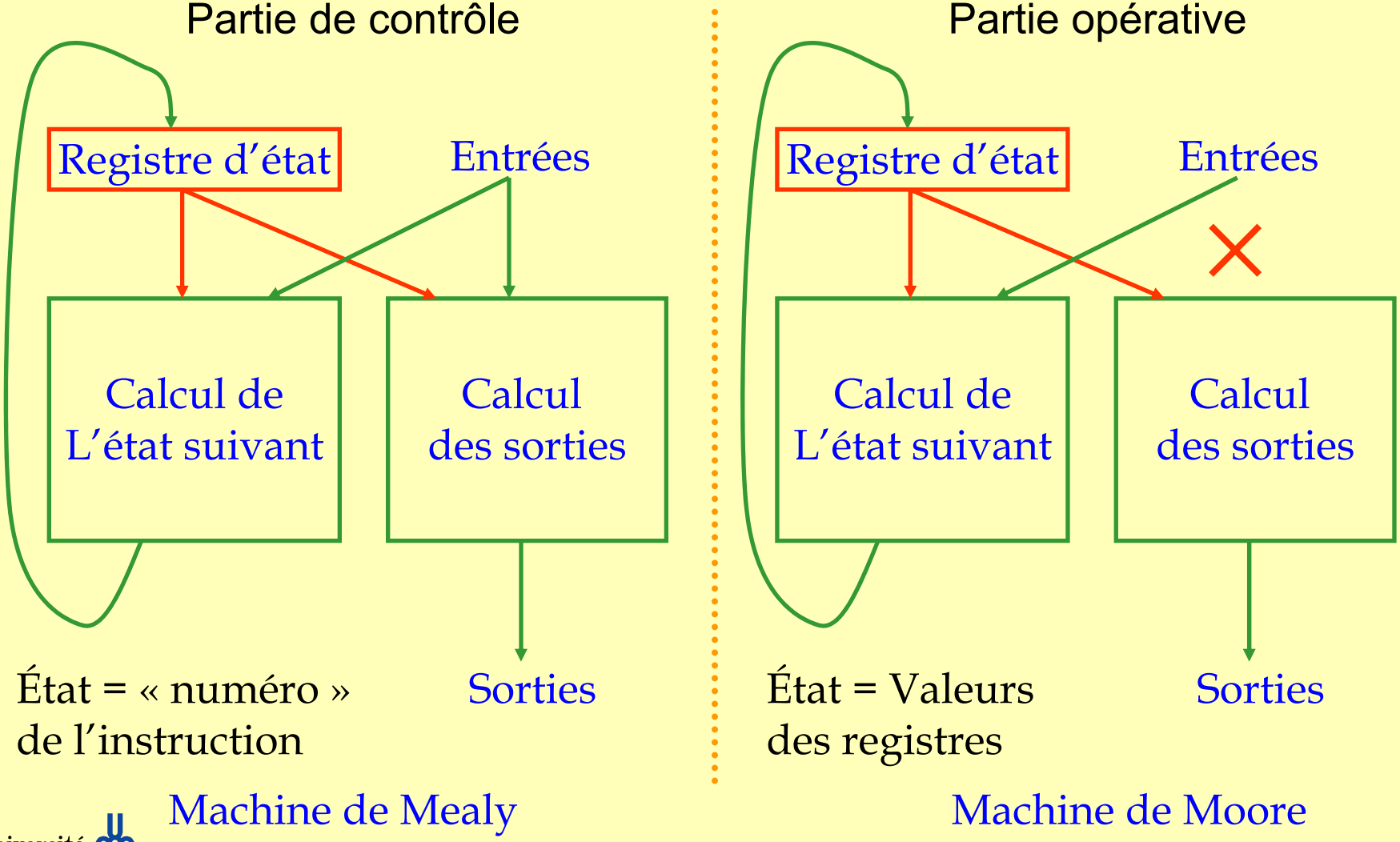
Algorithmic State Machine (ASM) (1/4)



Algorithmic State Machine (ASM) (2/4)



Algorithmic State Machine (ASM) (3/4)



Algorithmic State Machine (ASM) (4/4)

Forme générale

```
n_i: switch (t_{n-1}..t_0) {
       case 0   :a=F(a,b,c);b=G(a,b,c);c=H(a,b,c); goto n_j;}
       case …   :a=I(a,b,c);b=J(a,b,c);c=K(a,b,c); goto n_k;}
       case 2^{n-1} :a=X(a,b,c);b=Y(a,b,c);c=Z(a,b,c); goto n_l;}
}
```

Forme conditionnelle

```
n_i: if (t0)    {a=F(a,b,c); b=G(a,b,c); c=H(a,b,c); goto n_j;}
    else        {a=X(a,b,c); b=Y(a,b,c); c=Z(a,b,c); goto n_l;}
```

Forme inconditionnelle

```
n_i: if (true) {a=F(a,b,c); b=G(a,b,c); c=H(a,b,c); goto n_j;}
```

Convention : l'opérateur 'a=F' signifie : au prochain flanc montant de l'horloge (globale), a prendra la valeur présente de F.

Du langage C vers l'ASM (1/3)

1 Élimination des structures « for »

```
for (exp1;exp2;exp3) exp4        F0:    exp1;
                                        while (exp2) {
                                          exp4
                                          exp3;
                                        }
```

2 Élimination des structures « while »

```
while (exp1) exp2                W0:    if (exp1) {exp2 goto W0;}

do exp2 while (exp1);            D0:    exp2 if (exp1) goto D0;
```

Du langage C vers l'ASM (2/3)

3 Élimination des tests imbriqués

```
if (exp_test) exp1;
else exp2;
```

```
I0:     if (exp_test) goto I0i;
        else goto I0e;
I0i:    exp1; goto I0n;
I0e:    exp2; goto I0n;
I0n:
```

```
switch  (exp_test) {
        case 0: exp0;
        …
        case P: expP;
}
```

```
S0:     switch (exp_test) {
                case 0: goto S0_0;
                …
                case P: goto S0_P;
S0_0:   exp0;
…
S0_P:   expP;
```

Du langage C vers l'ASM (3/3)

4 Simplifier et paralléliser les expressions

Règle: En ASM, toutes les opérations d'un même état se font en parallèle. Les valeurs « à droite » sont évaluées en parallèle avant d'être affectées au variables « à gauche ». Chaque instruction C doit donc théoriquement faire l'objet d'un état de l'ASM.

✓ Simplifier les expressions en supprimant les {} inutiles, les étiquettes multiples et les branchements inaccessibles.

✓ Augmenter le parallélisme :

- o Toutes les opérations successives qui sont indépendantes peuvent être effectuées en parallèle dans un seul état de l'ASM
- o Certaines dépendances triviales peuvent être parallélisées en tenant compte de la règle ci-dessus.
  - o Ex: a=b; c=a+d (en C); devient {a=b; c=b+d;} (en ASM)
- o Les branchements vers des instructions n'impliquant pas de test peuvent être regroupées dans un seul état.

Exemple : L'algorithme de Fermat

```
int Prime(unsigned long N) {
    long x,y,r,N2;
    int OK;

    if (N%2==0) OK=0;
    else {
        N2=sqrt(N); x=1+2*N2; y=1; r=N2*N2-N;
        while (r!=0) {
            if (r<0) {r=r+x; x=x+2;}
            else {r=r-y; y=y+2;}
        }
        OK=(x-y==2);
    }
    if (N==2) return 1;
    else return OK;
}
```

Éliminer les structures while

```
if (N%2==0) OK=0;
     else {
          N2=sqrt(N); x=1+2*N2; y=1; r=N2*N2-N;
W0: if (r!=0) {{
               if (r<0) {r=r+x; x=x+2;}
               else {r=r-y; y=y+2;}
          }; goto W0;}

          OK=(x-y==2);
     }
     if (N==2) return 1;
     else return OK;
}
```

Éliminer les tests imbriqués (1/2)

```
I0:       if (N%2==0) goto I0i;
          else goto I0e;
I0i:      OK=0; goto I0n;
I0e:{
        N2=sqrt(N); x=1+2*N2; y=1; r=N2*N2-N;
W0: if (r!=0) {{
              if (r<0) {r=r+x; x=x+2;}
              else {r=r-y; y=y+2;}
        }; goto W0;}

        OK=(x-y==2);
    } goto I0n;
I0n:
    if (N==2) return 1;
    else return OK;
}
```

Éliminer les tests imbriqués (2/2)

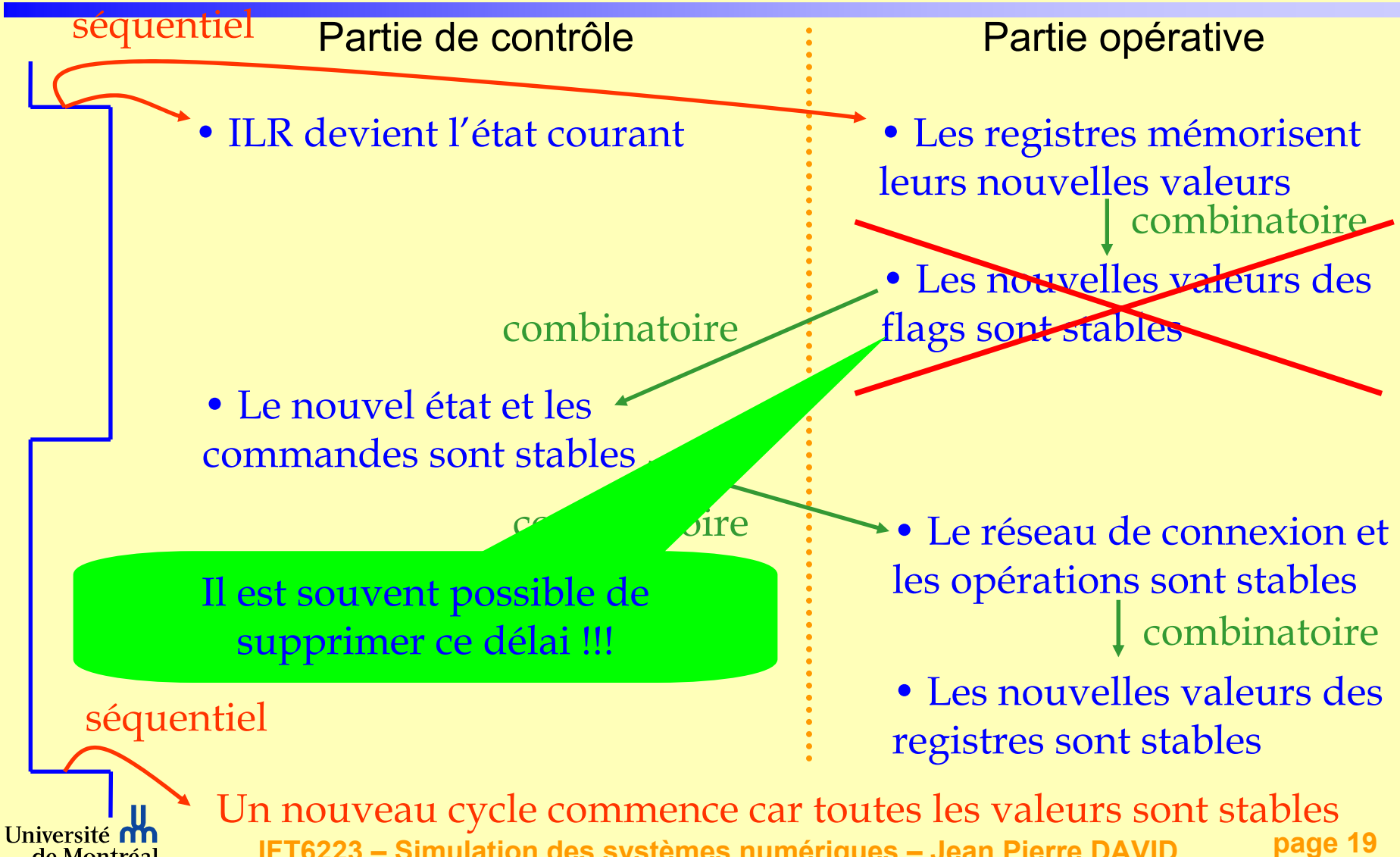
```
I0:       if (N%2==0) goto I0i; else goto I0e;
I0i:      OK=0; goto          I0n;
I0e:      {N2=sqrt(N); x=1+2*N2; y=1; r=N2*N2-N;
W0:I1:    if (r!=0) goto I1i; else goto I1e;
I1i:      {{
I2:       if (r<0) goto I2i;
          else       goto I2e;
I2i:      {r=r+x; x=x+2;} goto I2n;
I2e:      {r=r-y; y=y+2;} goto I2n;
I2n:        }; goto W0;} goto I1n;
I1e:      goto I1n;
I1n:        OK=(x-y==2);}
I0n:I3:   if (N==2) goto I3i;
          else goto I3e;
I3i:      return 1; goto I3n;
I3e:      return OK; goto I3n;
I3n:      end;}
```

Simplifier les expressions et augmenter le paral.

```
Entrées :       long N;
Variables :     long x,y,r,N2;
                int OK;

I0:     if (N%2==0)     {OK=0; goto I3;}
        else            {N2=sqrt(N); y=1; goto N0;}
N0:     if (true)       {x=1+2*N2; r=N2*N2-N; goto W0}
W0:     if (r!=0)       {goto I1i;}
        else            {OK=(x-y==2); goto I3;}
I1i:    if (r<0)        {r=r+x; x=x+2; goto W0};
        else            {r=r-y; y=y+2; goto W0};
I3:     if (N==2)       {return 1;}
        else            {return OK};
```

Augmenter la fréquence (réduire les délais) (1/2)



Augmenter la fréquence (réduire les délais) (2/2)

5 Anticiper le calcul des tests

Soit l'état N_i d'un ASM qui évalue un test Ti:

```
Ni: if (Ti)    {a=F(a,b,c); b=G(a,b,c); c=H(a,b,c); goto Nj;}
    else       {a=X(a,b,c); b=Y(a,b,c); c=Z(a,b,c); goto Nl;}
```

- L'idée consiste en ce que Ti ne soit plus le résultat d'une fonction combinatoire des registres de la partie opérative (car cela prend un certain délai à évaluer) mais qu'il soit lui aussi un registre (1 bit). Il doit donc avoir été calculé à l'étape précédente.

- Il faut donc calculer la valeur de Ti dans tous les états qui peuvent directement précéder N_i (tous les goto N_i).

- Attention, il faut tenir compte des valeurs qu'auront les registres APRES l'exécution des instructions liées à ces états.

Application à l'exemple

```
Entrées :      long N;
Variables :    long x,y,r,N2;
               int OK;

I0:    if (N%2==0)     {OK=0; T2=(N==2); goto I3;}
       else            {N2=sqrt(N); y=1; goto N0;}
N0:    if (true)       {x=1+2*N2; r=N2*N2-N; T1=0; goto W0}
W0:    if (T1)         {goto I1i;}
       else            {OK=(x-y==2); T2=(N==2); goto I3;}
I1i:   if (r<0)        {r=r+x; x=x+2; T1=((r+x) !=0); goto W0};
       else            {r=r-y; y=y+2; T1=((r-y) !=0); goto W0};
I3:    if (T2)         {return 1;}
       else            {return OK};
```

Déjà un registre (le bit de poids faible de N)

Déjà un registre (le bit de poids fort de r)

Synchronisation des transferts de données

6 Ajouter des signaux de synchronisation

1) Le circuit doit avoir un signal qui va déclencher son exécution.
2) Une fois le résultat disponible, le circuit devra transmettre son résultat.

De nombreux protocoles existent. Dans un premier temps, nous considérerons seulement le cas synchrone simple :

- Un signal d'entrée « start » charge les registres d'entrée de la fonction avec leurs valeurs et démarre l'exécution de l'algorithme

```
INIT:  if (Start)    {Ready=0; Initialiser les registres;
                      goto N0;}
       else          {goto INIT;}
```

- Un signal de sortie « ready » est validé lorsque les résultats sont disponibles. L'instruction return est donc remplacée par

```
…      …             {Result= …; Ready=1; goto INIT;}
```

Application à l'exemple

```
Entrée(s) :     long Input;
Variables :     long N,x,y,r,N2;
                int OK;
Sortie(s) :     int Result;

INIT:   if (Start)          {Ready=0; N=Input; goto I0;}
        else                {goto INIT;}
I0:     if (N%2==0)         {OK=0; T2=(N==2); goto I3;}
        else                {N2=sqrt(N); y=1; goto N0;}
N0:     if (true)           {x=1+2*N2; r=N2*N2-N; T1=0; goto W0}
W0:     if (T1)             {goto I1i;}
        else                {OK=(x-y==2); T2=(N==2); goto I3;}
I1i:    if (r<0)            {r=r+x; x=x+2; T1=((r+x) !=0); goto W0};
        else                {r=r-y; y=y+2; T1=((r-y) !=0); goto W0};
I3:     if (T2)             {Result=1; Ready=1; goto INIT;}
        else                {Result=OK; Ready=1; goto INIT;}
```

Perspectives

- Comment compiler un ASM pour un FPGA ?
  - Langage de description de matériel
- Existe-t-il plusieurs implémentations possibles ?
  - Partie de contrôle
  - Partie opérative
- Comment le logiciel de CAO travaille-t-il ?
- Comment interconnecter plusieurs ASM entre eux ?
- Comment gérer des accès à la mémoire ?
- Le processeur est-il un ASM particulier ?
- Comment utiliser plusieurs horloges ?