12. Principes de logique séquentielle

Principes de fonctionnement des ordinateurs

Jonas Lätt

Centre Universitaire d'Informatique

Trouvé une erreur sur un transparent? Envoyez-moi un message

- *sur Twitter @teachjl ou*
- *par e-mail jonas.latt@unige.ch*

Contenu du cours

Partie I: Introduction

Partie II: Codage de l'information

Partie III: Circuits logiques

Partie IV: Architecture des ordinateurs

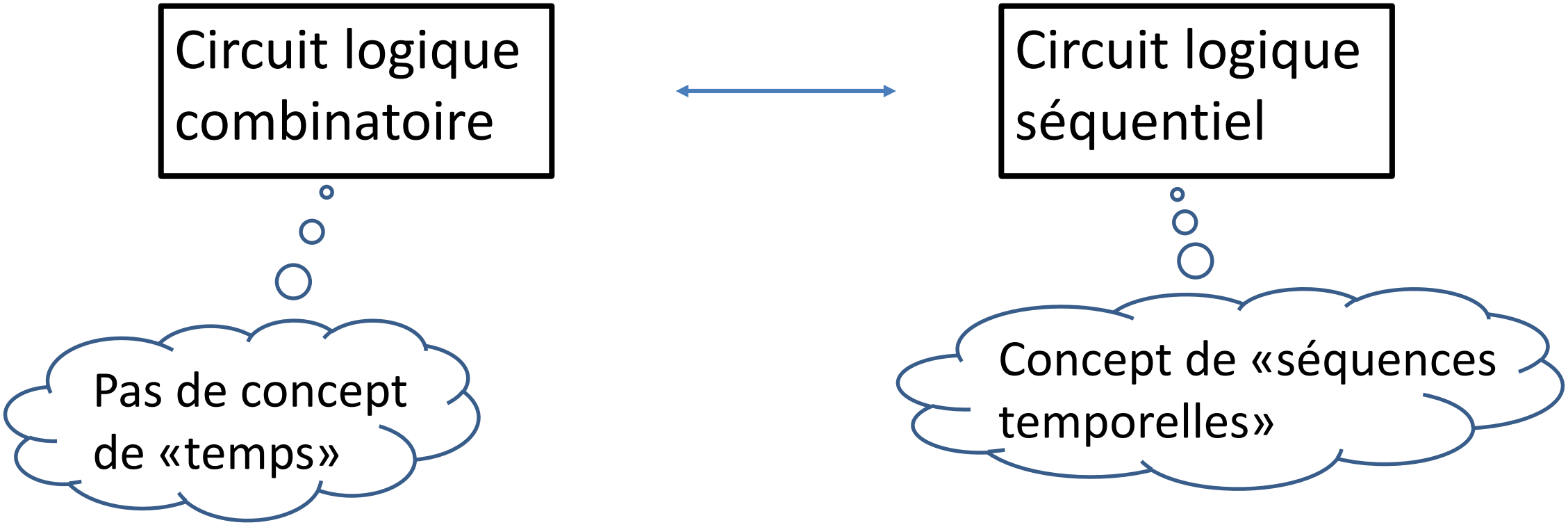
1. Introduction
2. Histoire de l'informatique
3. Information digitale et codage de l'information
4. Codage des nombres entiers naturels
5. Codage des nombres entiers relatifs
6. Codage des nombres réels
7. Codage de contenu média
8. Portes logiques
9. Circuits logiques combinatoires et algèbre de Boole
10. Réalisation d'un circuit combinatoire
11. Circuits combinatoires importants
12. **Principes de logique séquentielle**
13. Réalisation de la bascule DFF
14. Architecture de von Neumann
15. Réalisation des composants
16. Code machine et langage assembleur
17. Architecture d'un processeur
18. Performance et micro-architecture
19. Du processeur au système

Introduction:

Mémoire, circuits synchrones et horloges

Jonas Lätt

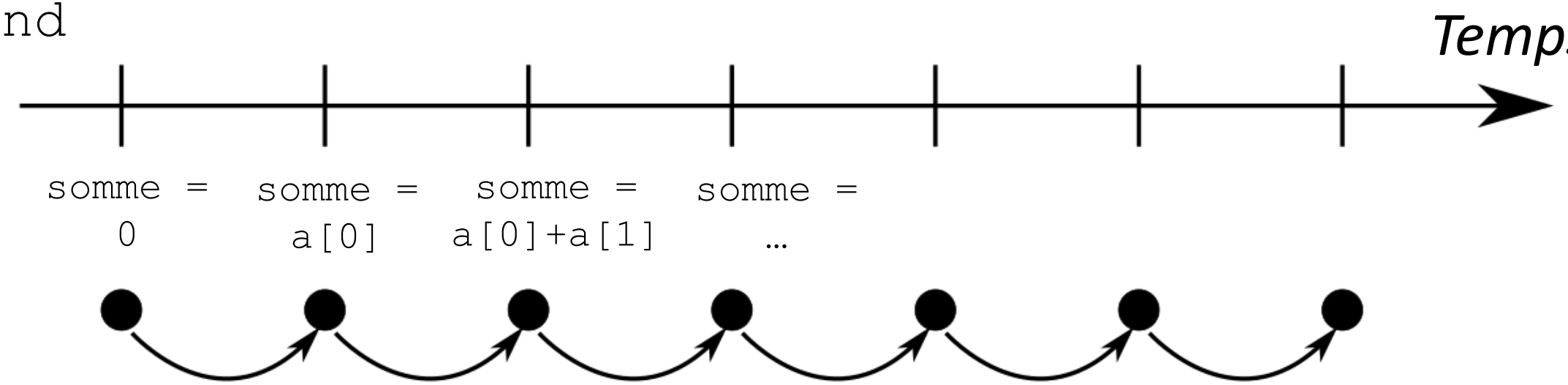
Rappel: logique combinatoire



Exemple: exécution d'un programme

```
int a[100] = {…} // tableau ("array") à 100 éléments
somme=0;
for (int i=0; i<100; ++i)
    somme = somme + a[i];
end
```

Il nous faut une "mémoire" dont la valeur varie dans le temps.



Qu’est-ce qui nous manque?

Le concept de **mémoire**

Le concept de **progression temporelle**

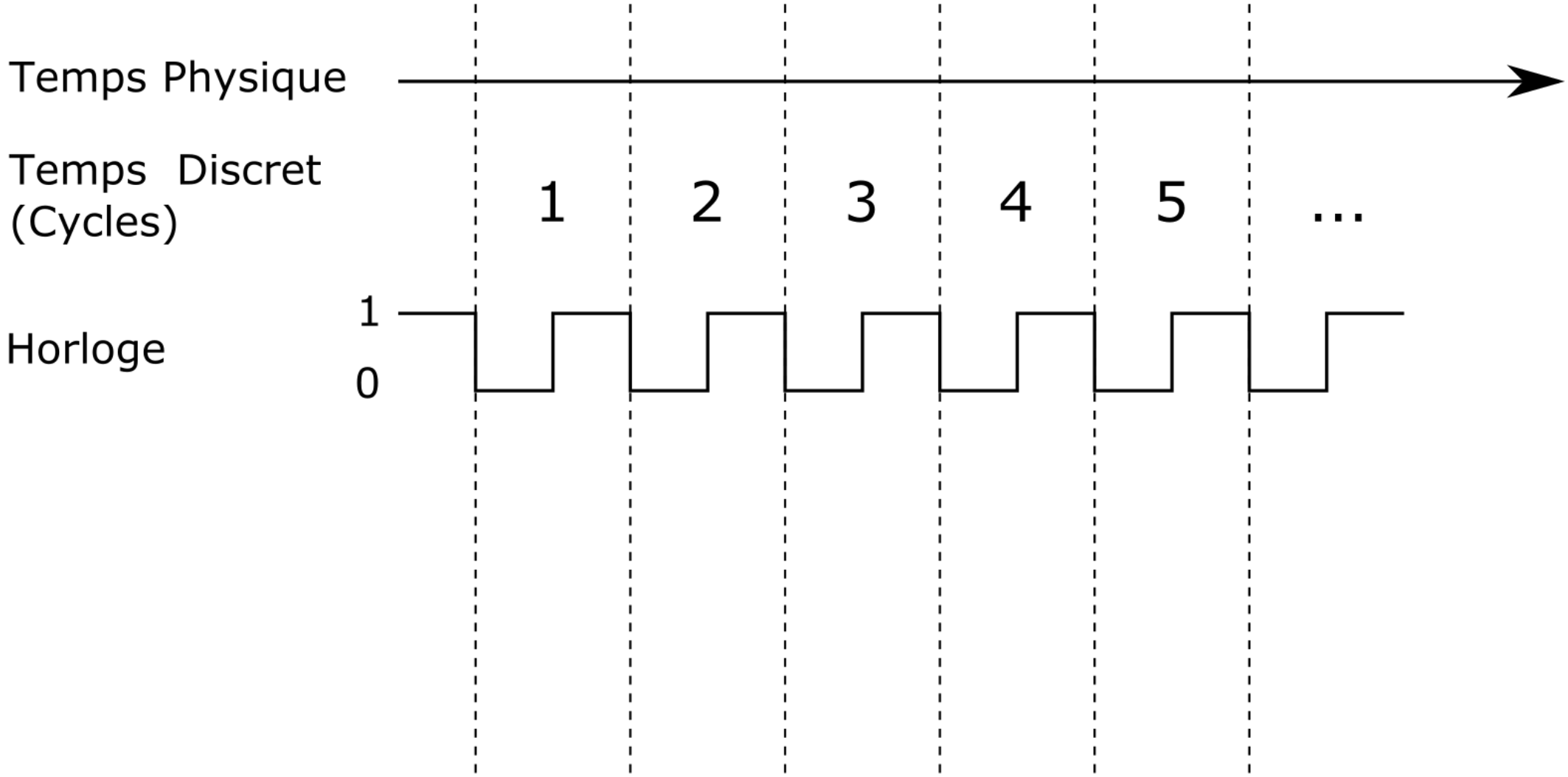
Le concept de **réutilisation des circuits logiques combinatoires**

Circuits séquentiels synchrones

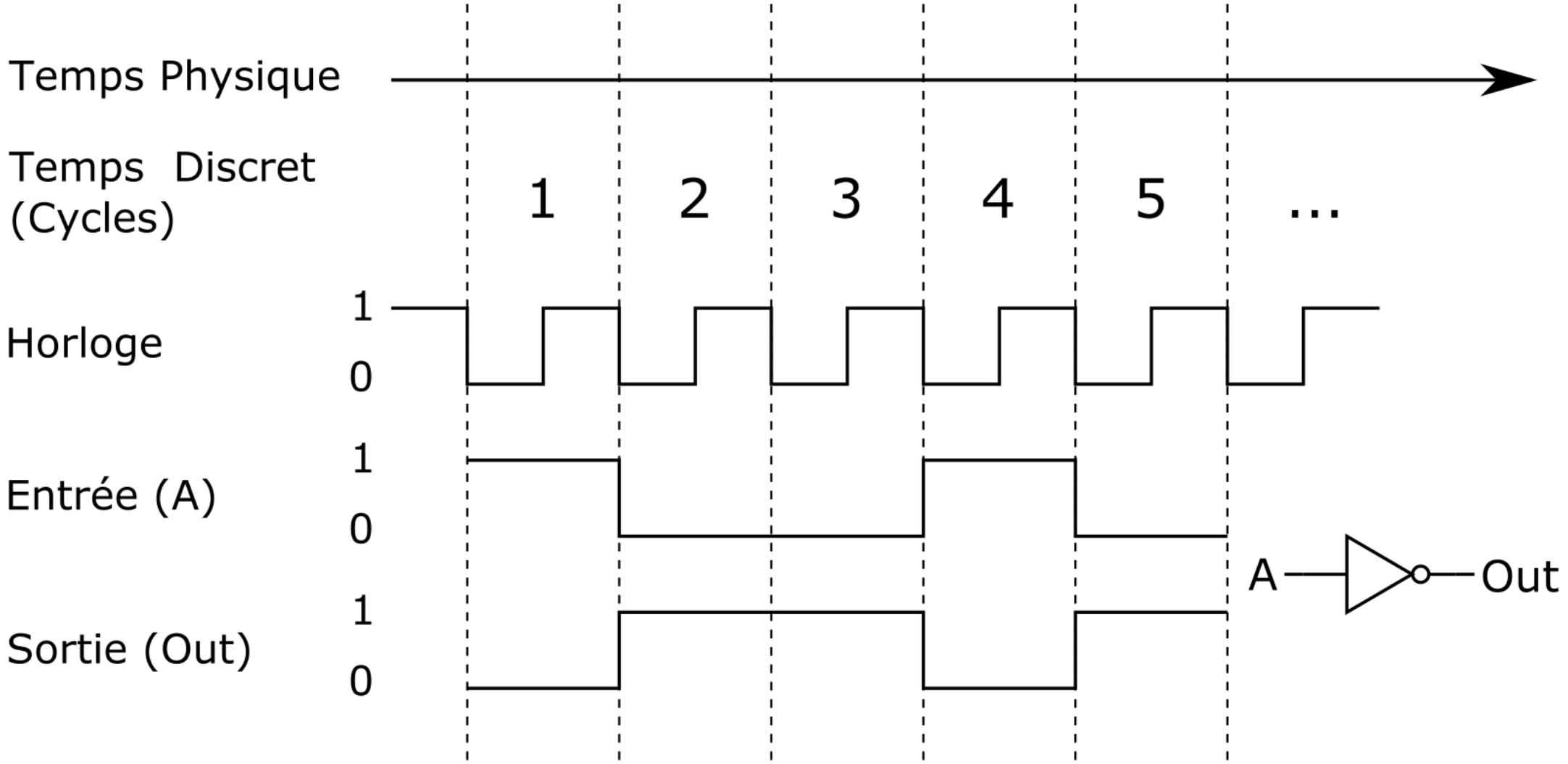
Presque tous les circuits séquentiels modernes utilisent un concept de **logique synchrone**.

- Idée: **sous-division de l'axe du temps en intervalles réguliers**.
- Les circuits fonctionnent à un rythme dicté par une horloge.
- **L'horloge** (en anglais: clock) est un oscillateur électronique qui génère une séquence de pulsations répétées: le signal d'horloge.

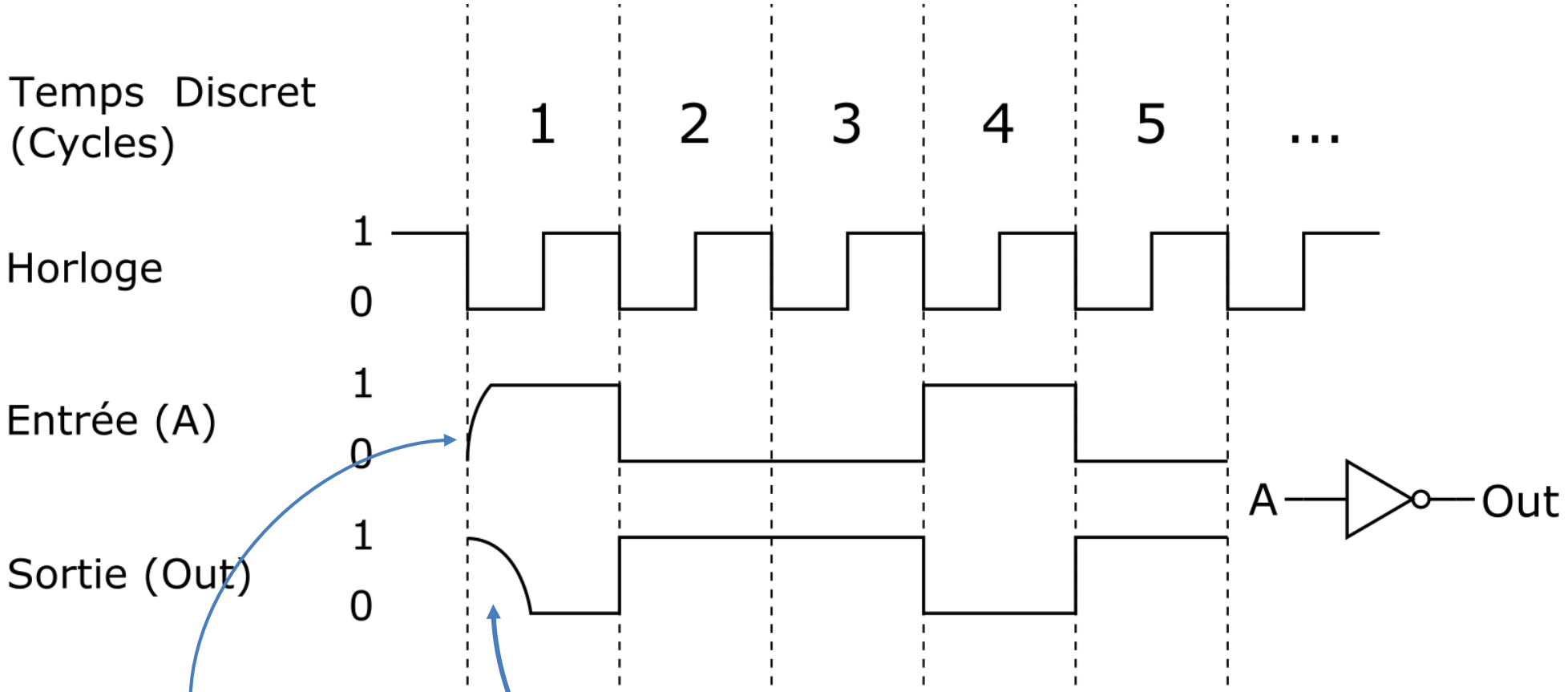
L’horloge



L’horloge


Temps Physique
Temps Discret
(Cycles)
1
2
3
4
5
...
Horloge
1
0
Entrée (A)
1
0
Sortie (Out)
1
0
A
Out

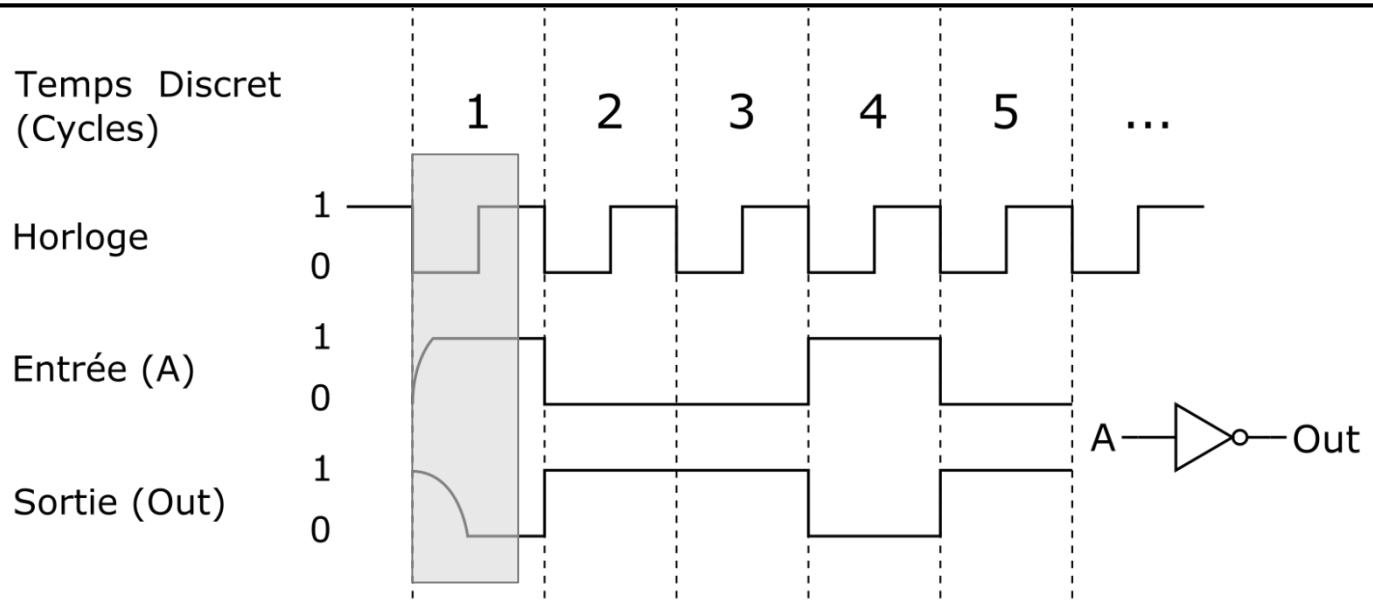
La réalité des signaux électroniques



Premier délai: construction du signal d'entrée (voltage)

Deuxième délai: construction du signal de sortie (premier délai + délai de propagation)

Signaux analogiques / Etat discret digital



- But: utiliser un circuit séquentiel sans se préoccuper des délais.
- Idée: Etat du système = valeur des signaux à la fin du cycle, quand le système est stabilisé.
- On fait abstraction de ce qui se passe dans le rectangle gris.

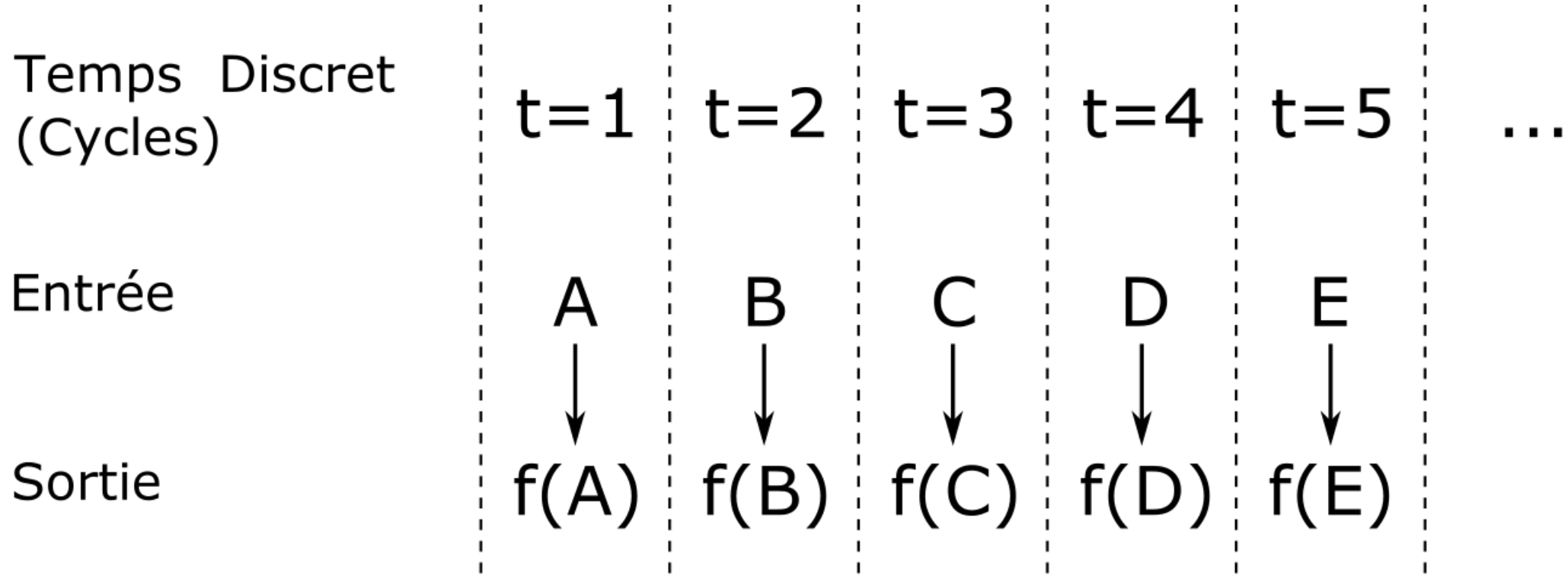
Exercice

Pourquoi utilisons-nous des circuits synchrones, avec signal d'horloge?

1. Pour laisser au système le temps de trouver un état stable.
2. Pour rendre le circuit plus rapide.
3. Dans un circuit synchrone, les composants électroniques n'ont pas de délai de propagation.
4. Parce que le temps physique est discret, comme cela est prédit par la mécanique quantique.

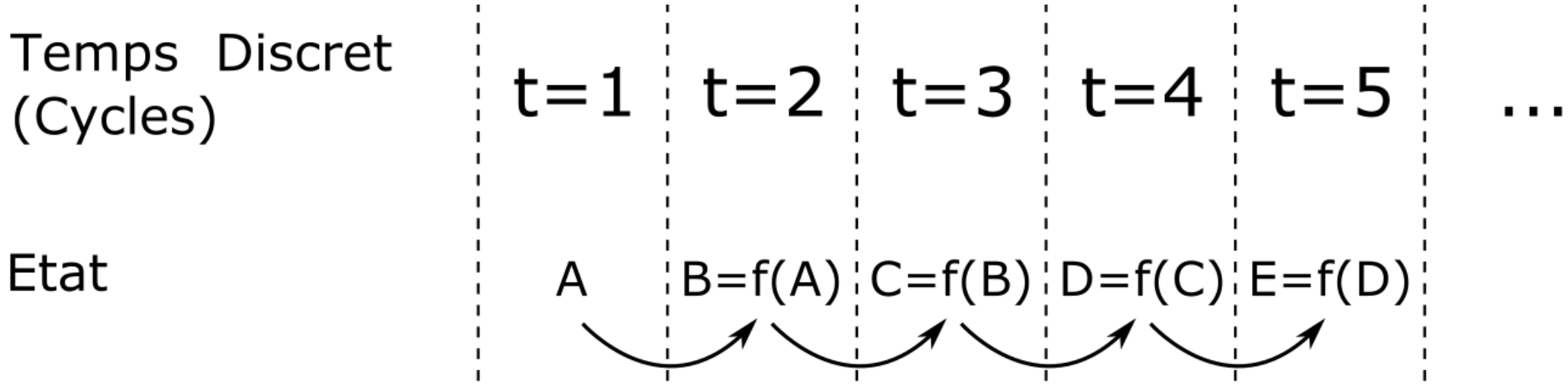
Logique combinatoire vs. séquentielle

- Combinatoire: `Q(t) = fonction( A(t) )`
- Séquentiel: `Etat(t+1) = fonction( Etat(t) )`



Logique combinatoire vs. séquentielle

- Combinatoire: `Q(t) = fonction( A(t) )`
- Séquentiel: `Etat(t+1) = fonction( Etat(t) )`



Le Delay Flip-Flop DFF:

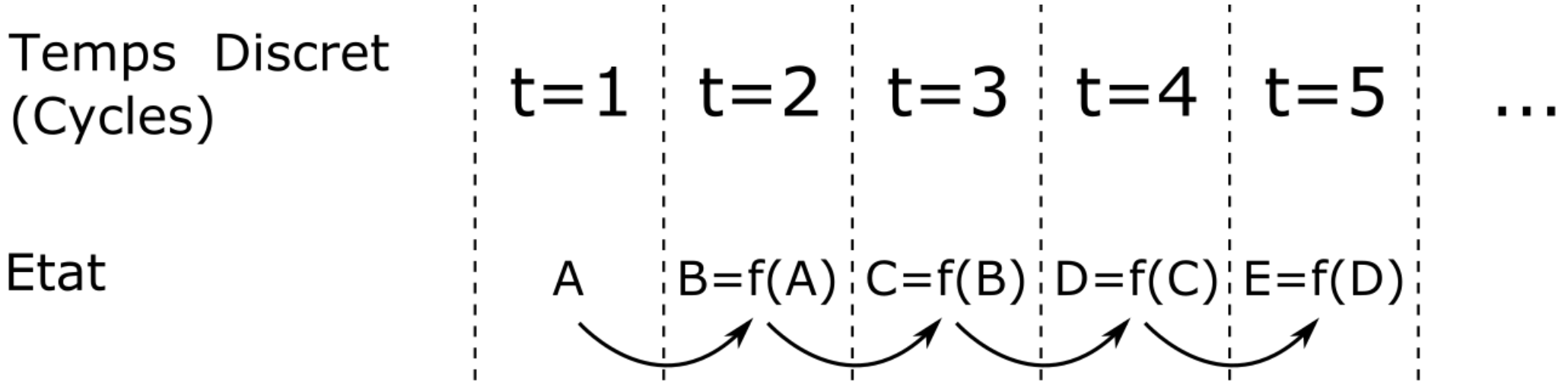
Porte logique séquentielle élémentaire

Jonas Lätt

Logique séquentielle: rappel

Une porte logique séquentielle doit pouvoir

- Mémoriser un état
- Transporter cet état d'un pas de temps vers le prochain



Le Delay Flip-Flop DFF

- Porte logique séquentielle élémentaire: Le Delay Flip-Flop DFF.
- Tout circuits logique sera une combinaison de DFF et de circuits combinatoires uniquement.

Idée: la porte combinatoire la plus élémentaire est l'**identité**:

`Out(t) = In(t)`

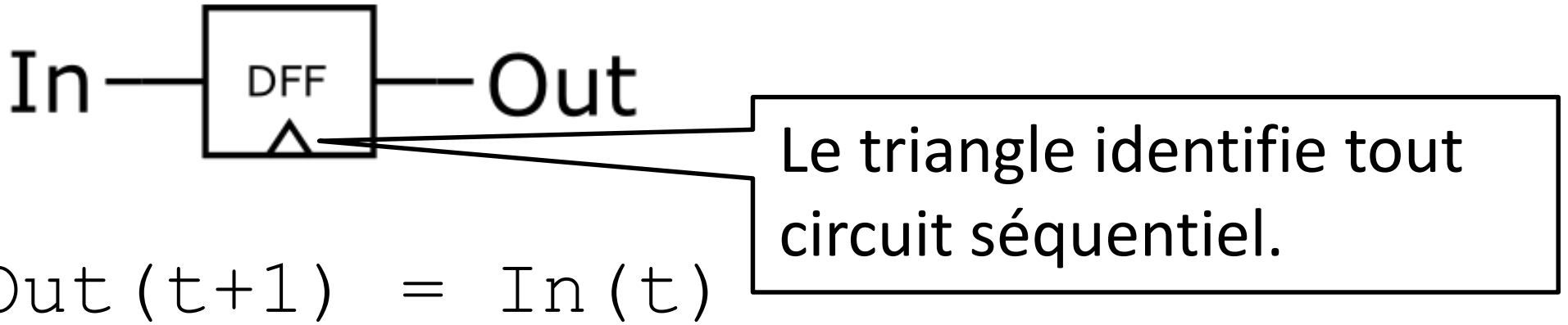
La **porte séquentielle élémentaire**, Le Delay Flip-flop DFF, se construit par extension:

`Out(t+1) = In(t)`

- Le DFF mémorise un seul bit d'information sur une durée d'un seul pas de temps.

Le Delay Flip-Flop Synchronisé DFF

«Flip-Flop», de l'anglais «flip» = «changer d'état»



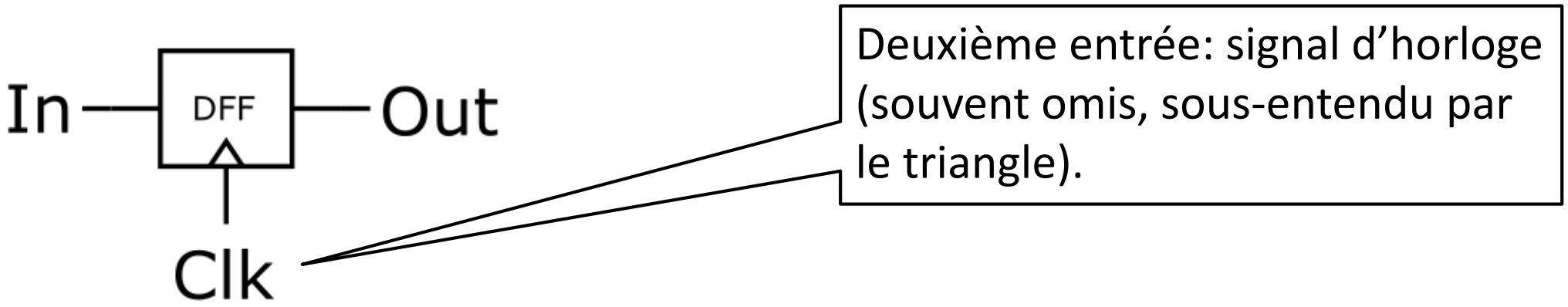

```
Out(t+1) = In(t)
```

Le DFF est un **circuit à délai**:

- Tout signal en entrée au temps t arrive à la sortie au prochain temps t+1.
- Délai d'un cycle

Le Delay Flip-Flop Synchronisé DFF

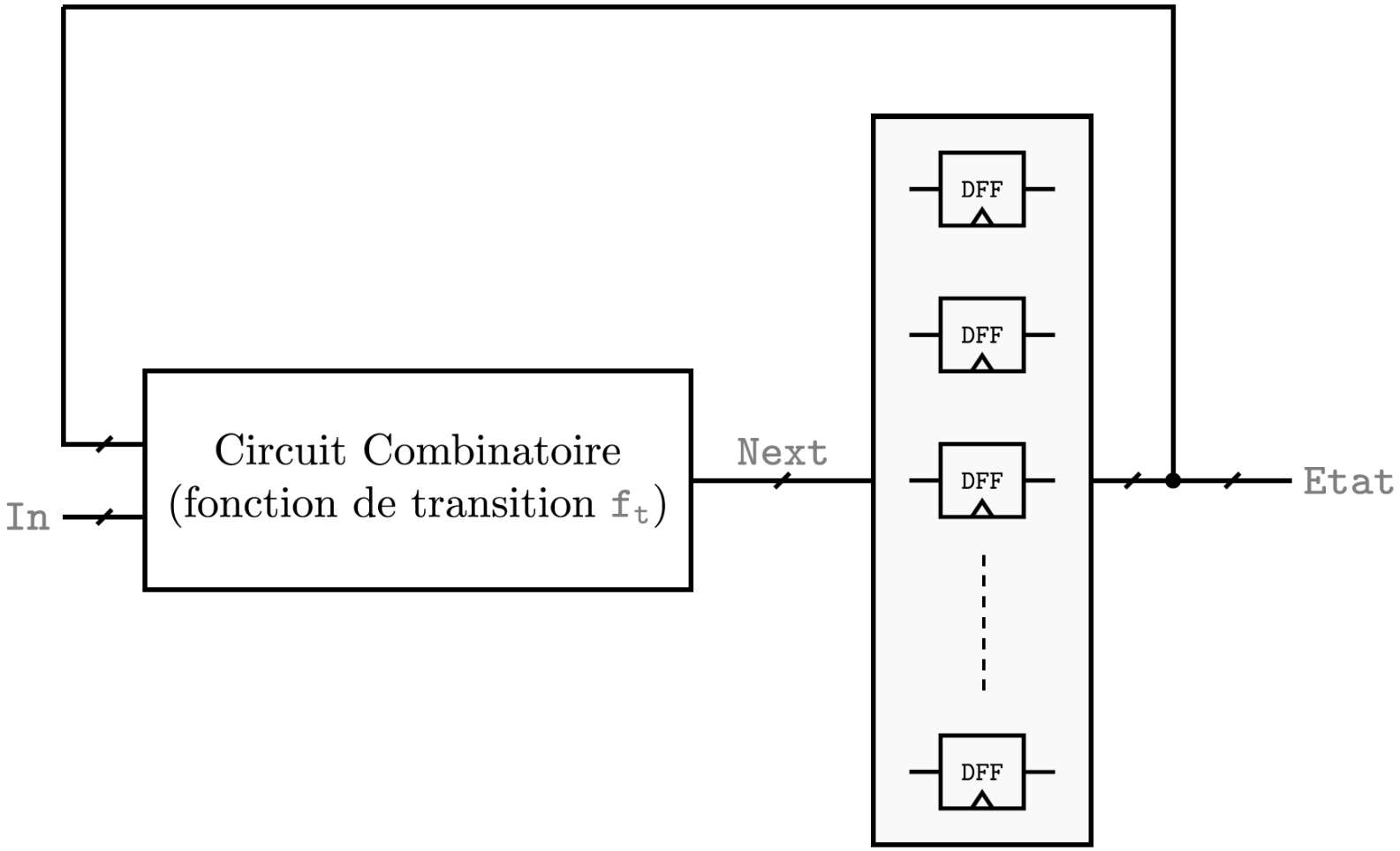
Nous utilisons le DFF synchronisé: utilisation d'un signal d'horloge.



Concept: un circuit (exemple: processeur) utilise une seule horloge et distribue son signal à tous les éléments de mémoire du circuit. Cela permet de coordonner l'opération séquentielle du circuit.

Circuits logiques séquentiels: le cas général

$$Etat(t+1) = Next(t) = f_t(\ In(t), Etat(t)\ )$$



Réalisation d’un circuit séquentiel

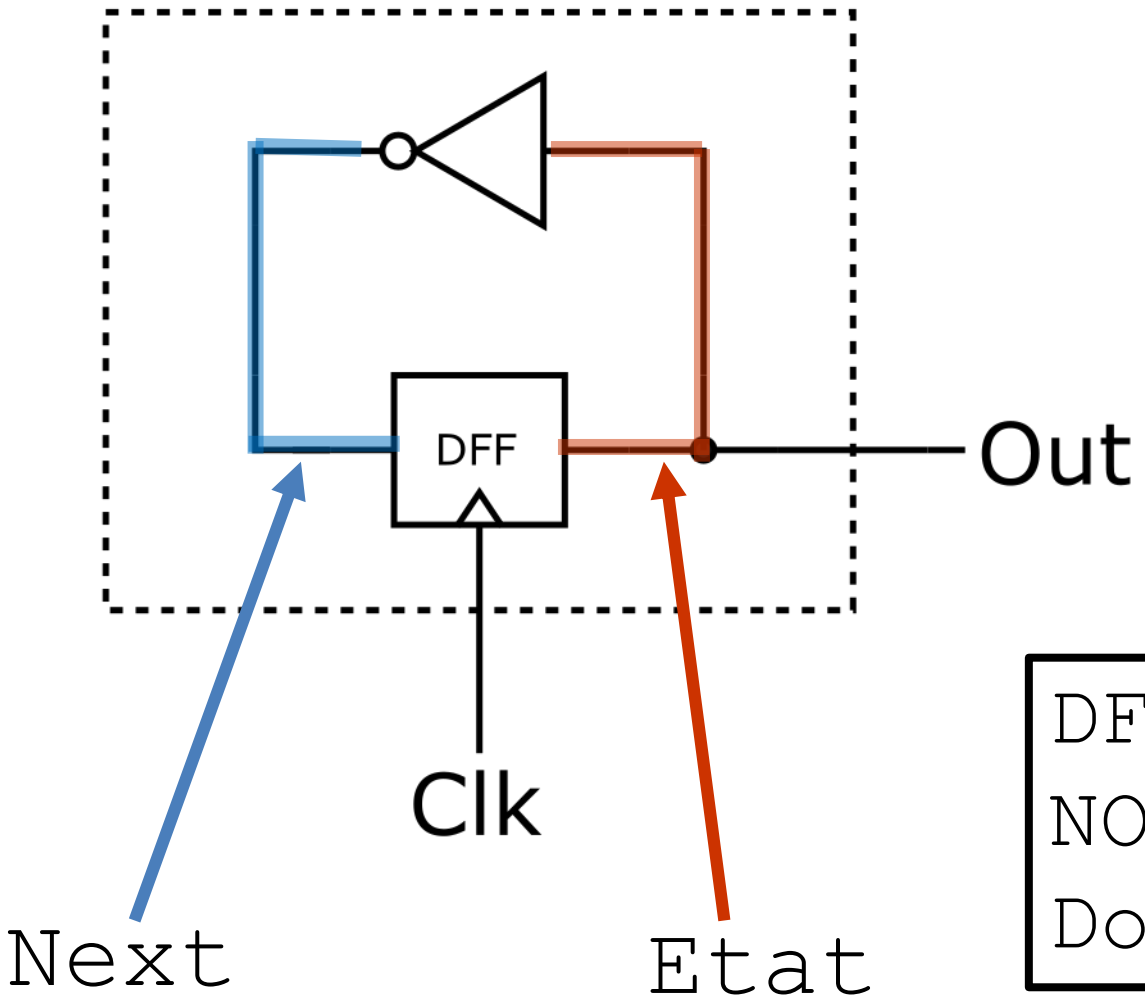
1) **Définition de l'état.** Identification du nombre minimal de bits nécessaires pour définir de manière unique l'état du système: Correspond au nombre de DFF.

2) **Construction du circuit combinatoire.** Description du rapport entre Next(t) et l'état Etat(t), sous forme de **fonction de transition** (la fonction logique f) ou de **table de transition** (la table de vérité de f).

Exemples de circuits logiques séquentiels

Jonas Lätt

Premier exemple: signal alterné



Le signal `Etat`:

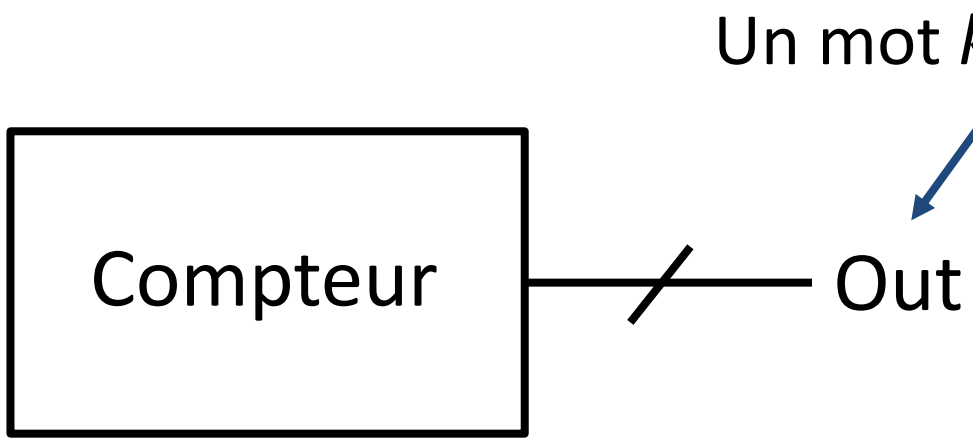
- `Etat` = `Out`
- `Etat`: Sortie du DFF
- `Etat`: Entrée de la porte NOT

Le signal `Next`:

- `Next`: Entrée du DFF
- `Next`: Sortie de la porte NOT

```
DFF:  Next(t)  =  Etat(t+1)
NOT:  Next(t)  =  !Etat(t)
Donc: Etat(t+1) = !Etat(t)
```

Deuxième exemple: compteur digital



Fonctionnement: `Out(t+1) = Out(t)+1`

Exemple, un compteur 4-bit:

- Compte de 0 à 15
- Ensuite, recommence à zéro, sans jamais s'arrêter

Compteur digital: table de transition

	$Etat_3(t)$	$Etat_2(t)$	$Etat_1(t)$	$Etat_0(t)$	$Next_3(t)$	$Next_2(t)$	$Next_1(t)$	$Next_0(t)$
t=0	0	0	0	0				
t=1	0	0	0	1				
t=2	0	0	1	0				
t=3	0	0	1	1				
t=4	0	1	0	0				
t=5	0	1	0	1				
t=6	0	1	1	0				
t=7	0	1	1	1				
t=8	1	0	0	0				
t=9	1	0	0	1				
t=10	1	0	1	0				
t=11	1	0	1	1				
t=12	1	1	0	0				
t=13	1	1	0	1				
t=14	1	1	1	0				
t=15	1	1	1	1				

Deuxième exemple: compteur digital

	$Etat_3(t)$	$Etat_2(t)$	$Etat_1(t)$	$Etat_0(t)$	$Next_3(t)$
t=0	0	0	0	0	0
t=1	0	0	0	1	0
t=2	0	0	1	0	0
t=3	0	0	1	1	0
t=4	0	1	0	0	0
t=5	0	1	0	1	0
t=6	0	1	1	0	0
t=7	0	1	1	1	1
t=8	1	0	0	0	1
t=9	1	0	0	1	1
t=10	1	0	1	0	1
t=11	1	0	1	1	1
t=12	1	1	0	0	1
t=13	1	1	0	1	1
t=14	1	1	1	0	1
t=15	1	1	1	1	0

La plupart du temps, les bits ne changent pas: $Next_3(t) = Etat_3(t)$

Si tous les bits inférieurs valent 1, une retenue est propagée, et le bit change:

si $Etat_0(t) \cdot Etat_1(t) \cdot Etat_2(t))$ vaut 1

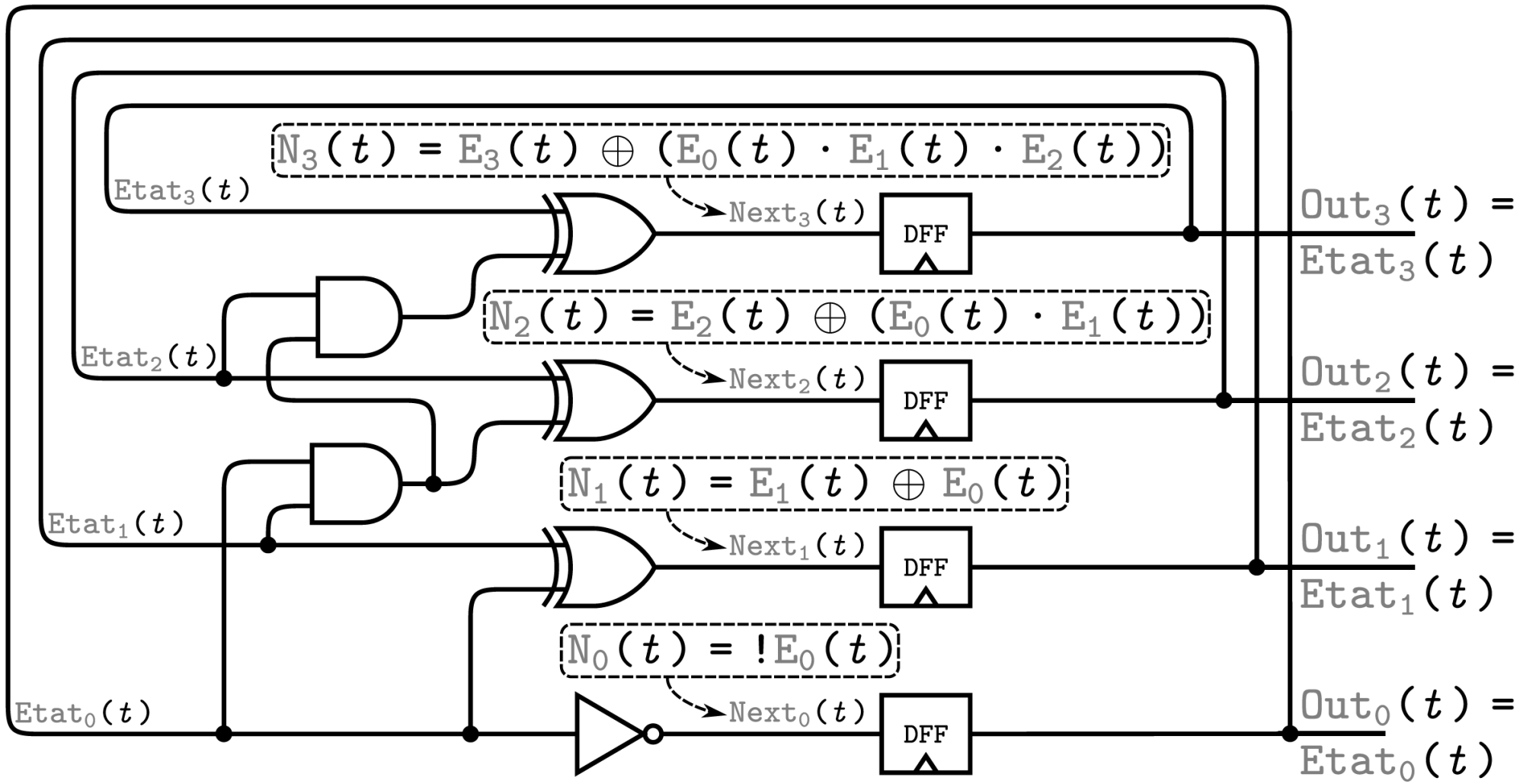
$Next_3(t) = !Etat_3(t)$

autrement

$Next_3(t) = !Etat_3(t)$

$$Next_3(t) = Etat_3(t) \oplus (Etat_0(t) \cdot Etat_1(t) \cdot Etat_2(t))$$

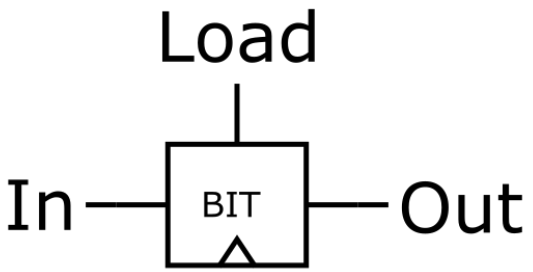
Deuxième exemple: compteur digital



Mémoire de longue durée: Le registre 1-bit

- But: mémoriser un bit pour davantage qu'un seul pas de temps.
- Description algorithmique du registre 1-bit:

```
si Load(t)vaut 1
     Out(t+1) = In(t)
autrement
     Out(t+1) = Out(t)
```



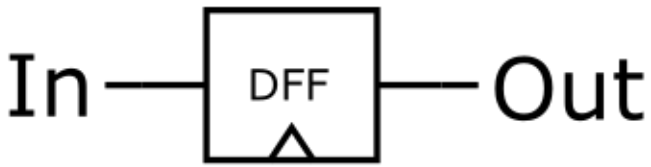
Rappel: le signal d'horloge est implicite

Réalisation du registre 1-bit

```
si Load(t)vaut 1
     Out(t+1) = In(t)
autrement
     Out(t+1) = Out(t)
```

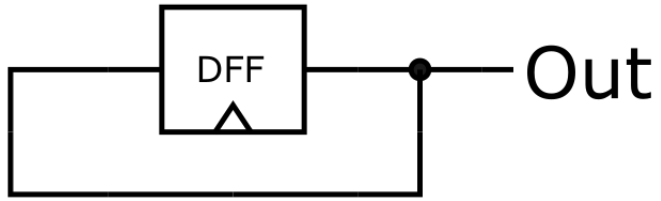
Réalisation partielle:

```
Out(t+1) = In(t)
```



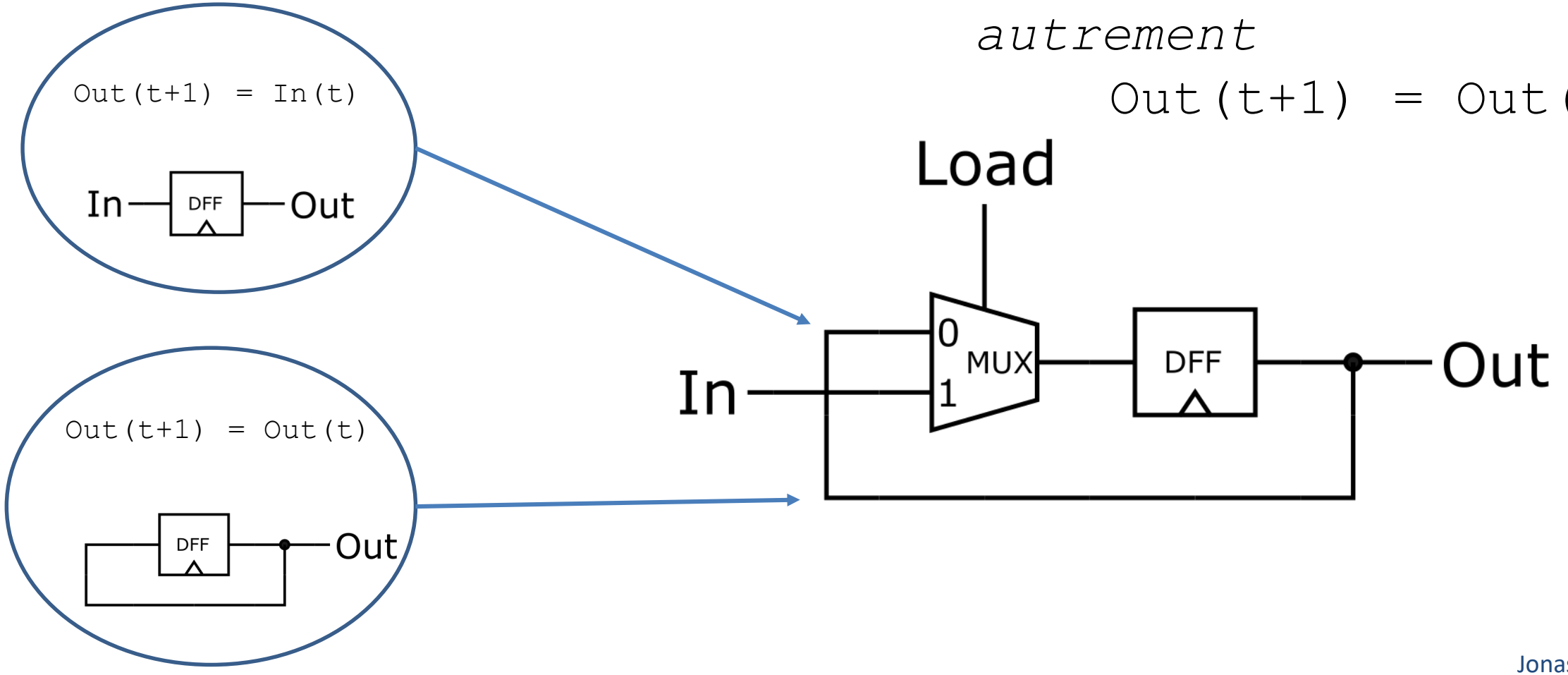
Réalisation partielle:

```
Out(t+1) = Out(t)
```



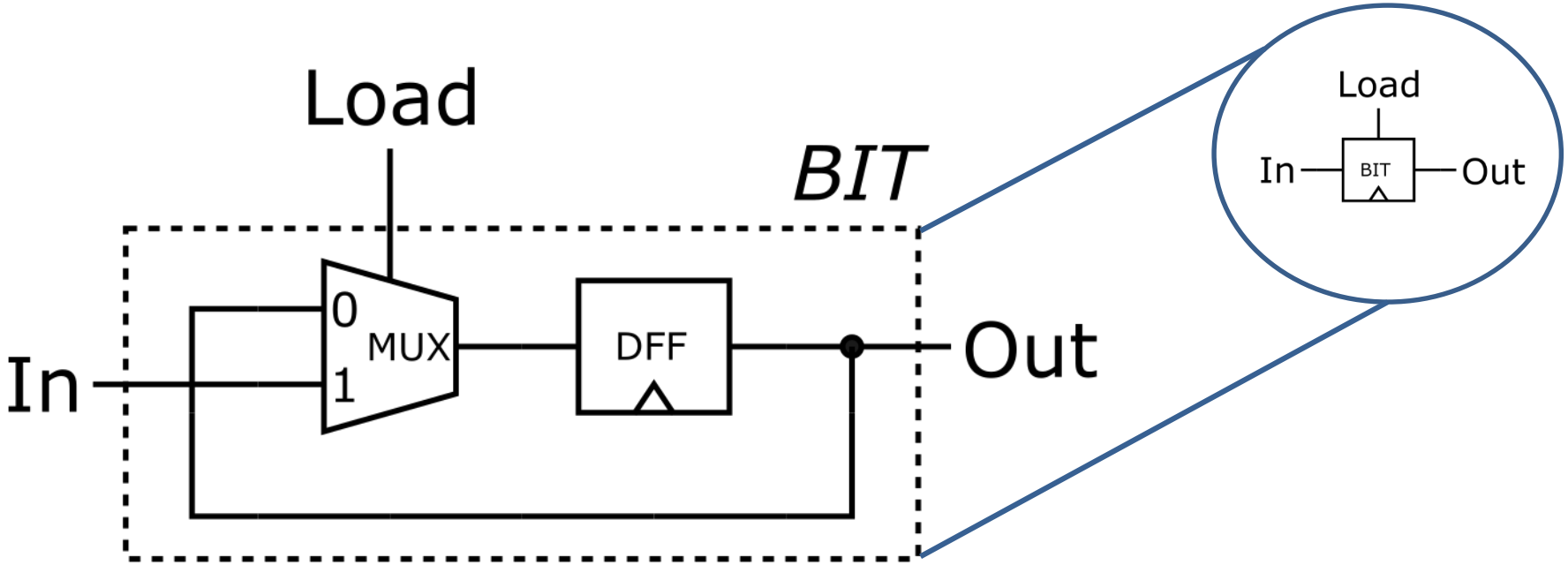
Réalisation du registre 1-bit

si Load(t)vaut 1
    Out(t+1) = In(t)
autrement
    Out(t+1) = Out(t)



Réalisation du registre 1-bit

si Load(t)vaut 1
    Out(t+1) = In(t)
autrement
    Out(t+1) = Out(t)



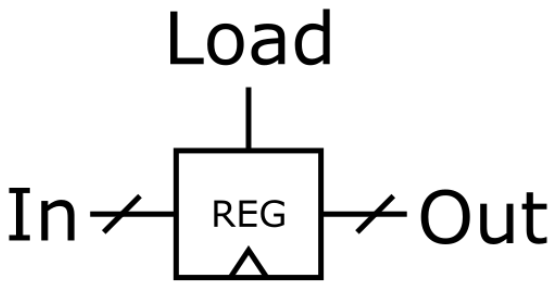
Registre k-bit

Le registre k-bit mémorise un mot k-bit. Sa description algorithmique est la même que celle du registre 1-bit:

```
si Load(t)vaut 1
    Out(t+1) = In(t)
autrement
    Out(t+1) = Out(t)
```

Les entrées/sorties `In/Out` sont des *mots* k-bit.

Symbole de diagramme:



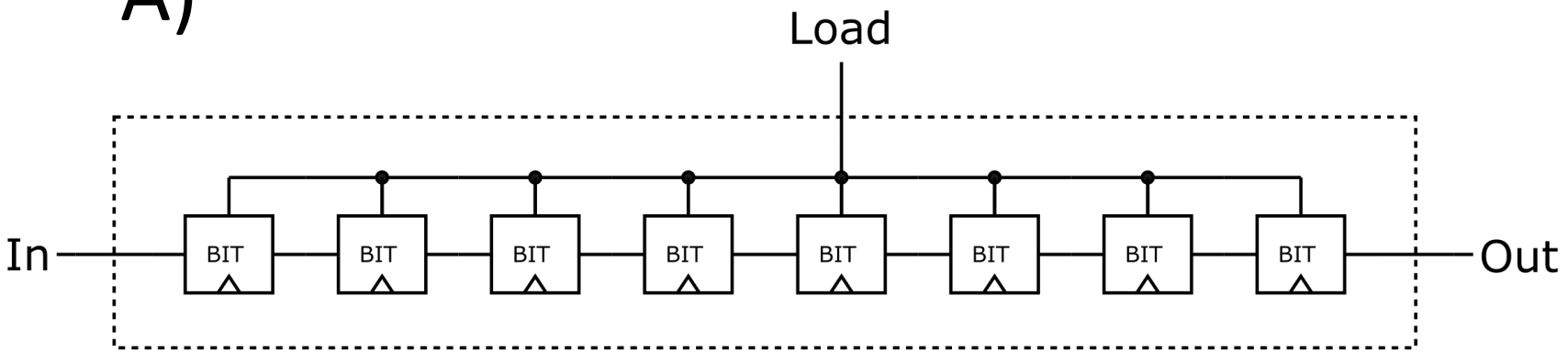
Exercice instantané

Lequel de ces deux circuits réalise-t-il un registre k-bit?

www.votamatic.ch

LKZJ

A)



B)