

Cours Windows IoT

2 – Kernel Objects

Plan du chapitre

- ◆ Objets kernel et handles
- ◆ Ordonnancement
- ◆ Gestion des processus
- ◆ Gestion des threads

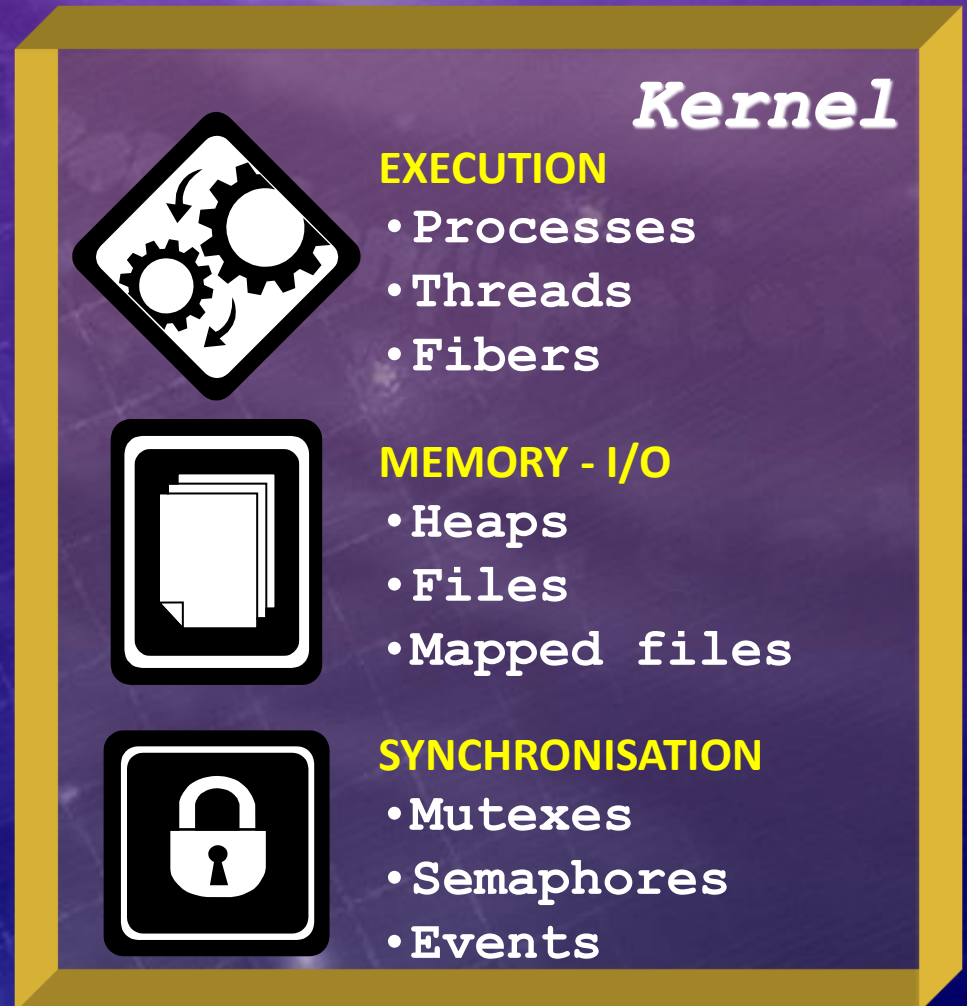
Kernel objects WinNT

◆ Services

- Exécution
- Mémoire – I/O
- Synchronisation

◆ Accès API

- HANDLE
- Nom

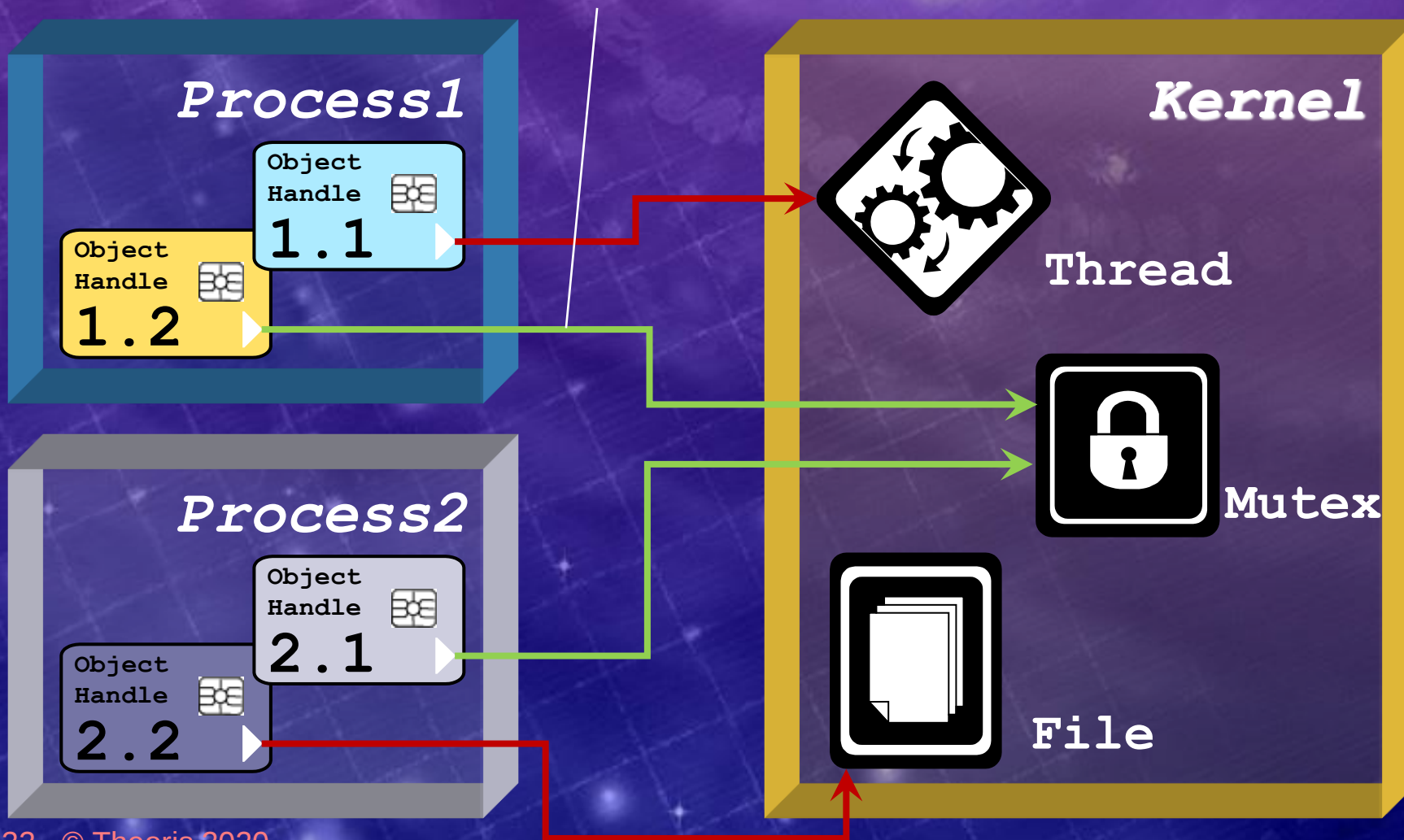


Le type HANDLE

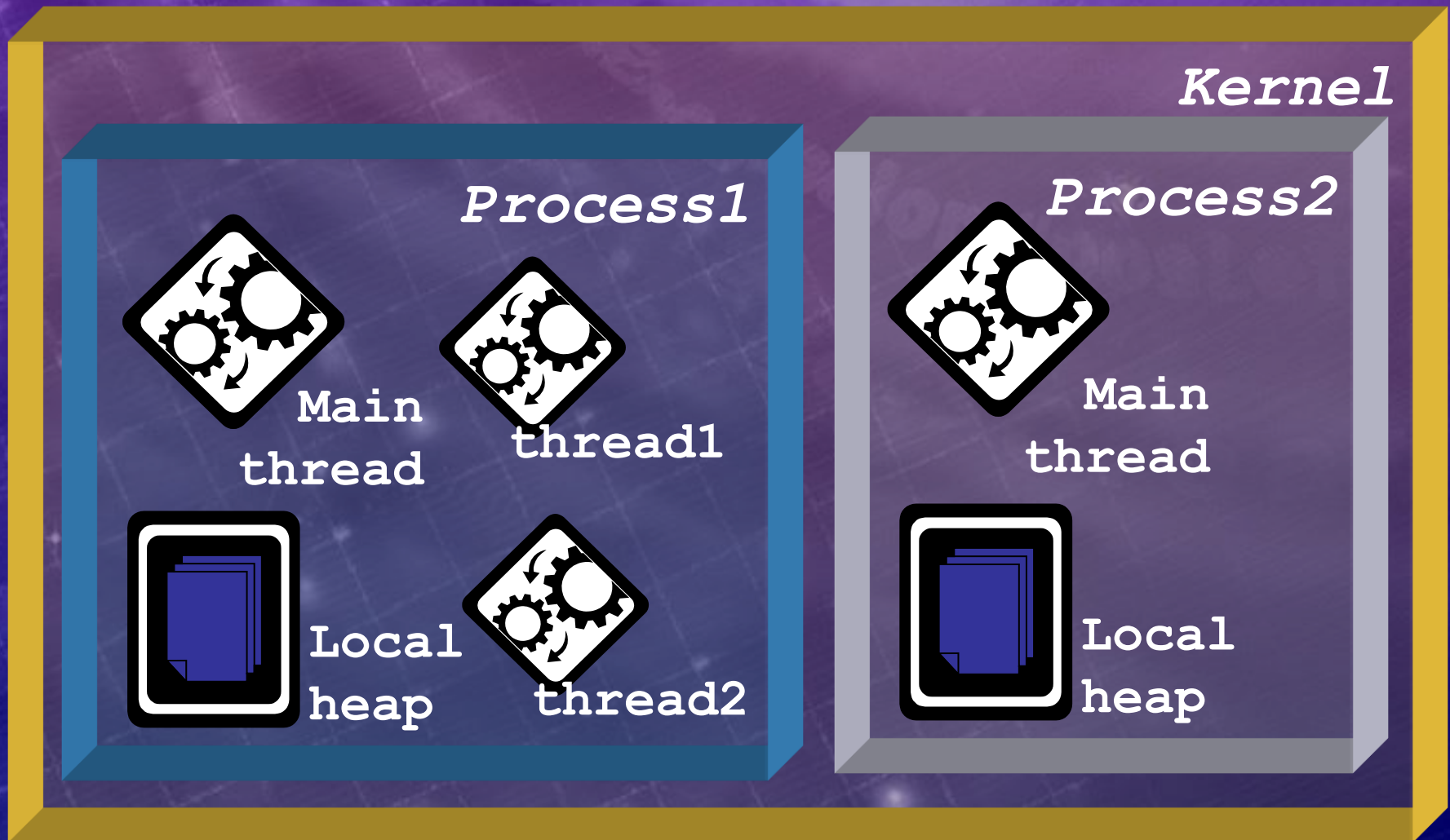
- ◆ Accès à un Kernel Object
 - Référence Unique sur 32 bits
 - Paramètre des appels d'API WinNT
- ◆ Spécifique à un processus
 - Stocké dans la table des HANDLE
 - Comptage de références associé
 - Libéré par un appel à:
`BOOL CloseHandle(HANDLE hObject);`

Handles, Processes & Objects

Les HANDLES 1.2 and 2.1 référencent le même objet (Mutex) mais ils sont différents en ne peuvent pas être échangés entre les processus 1 et 2



Process vs Thread



Application & Processus

◆ L'objet kernel «processus»

- Un espace d'adressage privé et virtuel
- Un exécutable (EXE)
- Des handles d'objets kernel

◆ Avec ou sans console par défaut:

- /SUBSYSTEM:CONSOLE → **main**
- /SUBSYSTEM:WINDOWS → **WinMain**
- On peut toujours ouvrir une console a posteriori avec la fonction **AllocConsole**

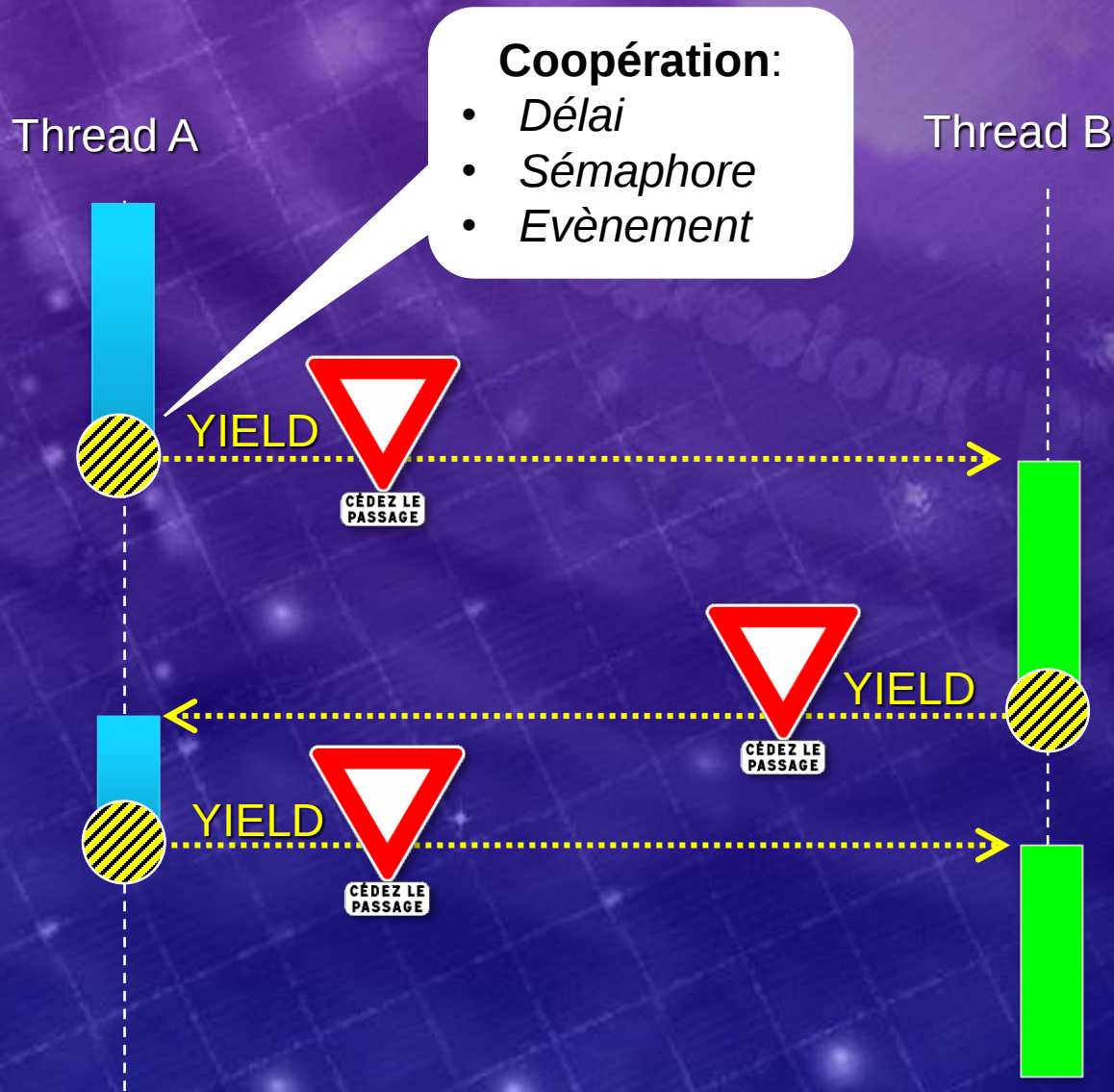
Vie et mort d'un processus

- ◆ WinNT: «père» et «fils» indépendants
- ◆ **CreateProcess**
 - L'exécutable associé et la ligne de commande
 - Les attributs de création
 - L'environnement
- ◆ **ExitProcess** et **TerminateProcess**
 - Threads terminés, objets libérés ou fermés
- ◆ La «vie» après la mort
 - Le code retour d'un processus, sa signalisation

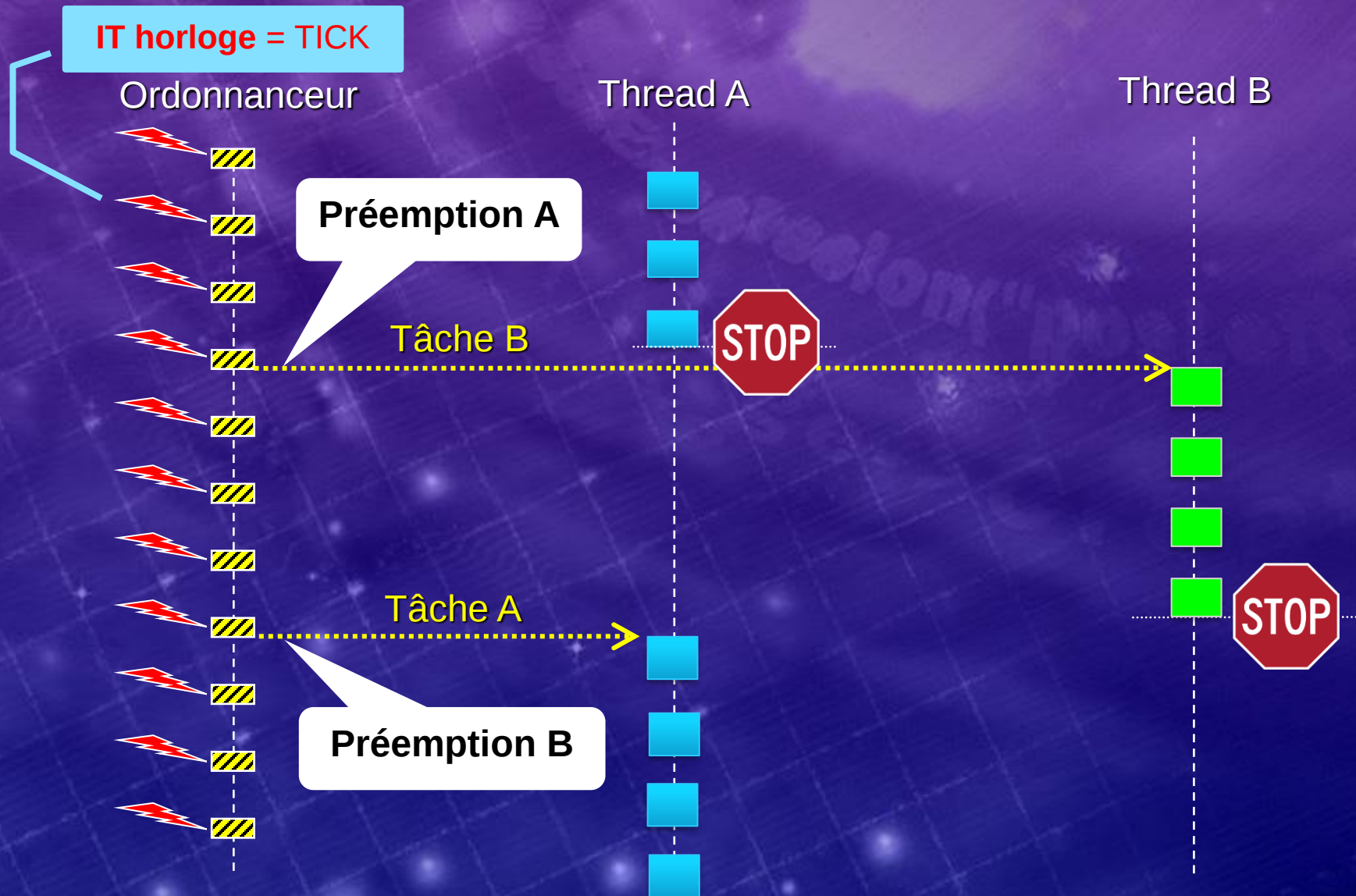
Environnement d'un processus

- ◆ **Les variables d'environnement**
 - Dupliquées ou spécifiées à la création d'un processus fils
- ◆ **Le lecteur et le répertoire courant**
 - Limité au lecteur courant mais peut être étendu au travers de variables d'environnement un peu particulières.
- ◆ **La gestion des erreurs critiques**
- ◆ **Les mécanismes d'héritage**
 - L'héritage intervient à la création d'un processus fils
 - C'est une importante source d'erreur car le processus fils ne sait pas de quoi il hérite.

Multitâche Coopératif

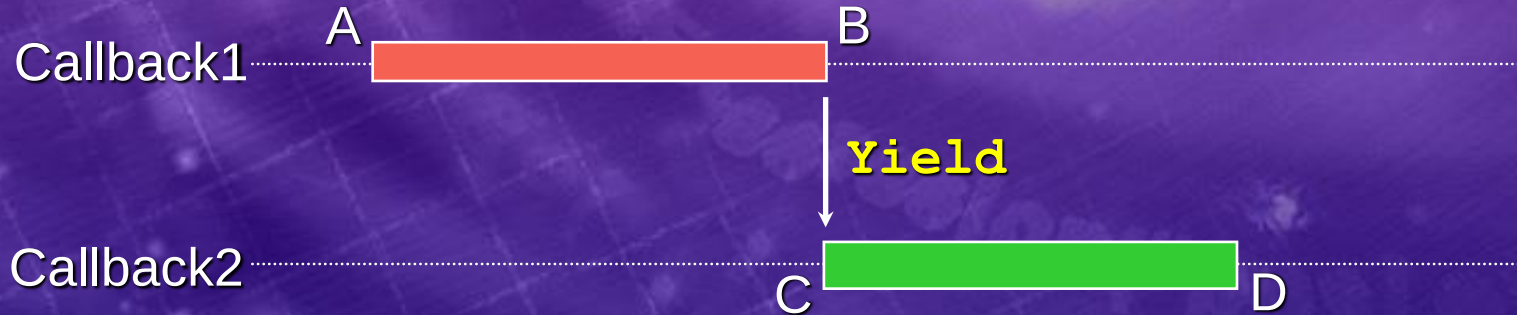


Multitâche Préemptif

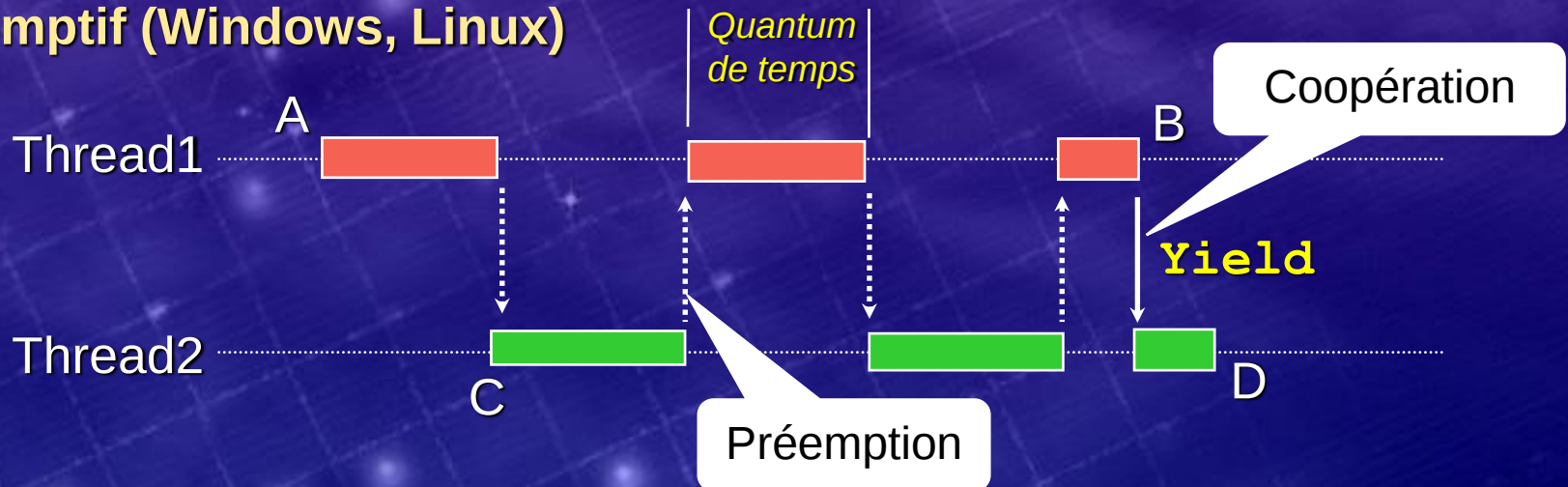


Coopératif / Préemptif

Coopératif (RTOS sans Round Robin)



Préemptif (Windows, Linux)



Threads

- ◆ **L'objet kernel «thread»**
 - Un contexte (registres, stack, exceptions...)
 - L'espace d'adressage du processus
- ◆ **Le point d'entrée du thread**
 - Une simple procédure dont le nom n'est pas imposé mais le prototype (paramètre et retour).
- ◆ **Les états d'un thread**
 - les transitions automatiques
 - les transitions explicites
 - les synchronisations

Vie et mort d'un thread

◆ CreateThread

- Le point d'entrée et le paramètre 32 bits
- Les attributs de création

◆ ExitThread et TerminateThread

◆ La bibliothèque C

- Utiliser `_beginthreadex`, `_endthreadex` et `_beginthreadNT`
- Eviter `_beginthread` et `_endthread`

Ordonnanceur WinNT

- ◆ **Préemptif, quantum de temps géré par l'OS**
 - Possibilité de Quantum multiples (I/O)
- ◆ **Priorité au niveau du thread**
 - Gestion groupée au niveau du processus
 - Modification dynamique par l'ordonnanceur
- ◆ **Le thread qui tourne a:**
 - Besoin de l'UC
 - La plus forte priorité
 - Le quantum en cours

Ordonnancement (suite)

- ◆ Principes d'ordonnancement ...
 - Les priorités variables
 - le coup de pouce d 'avant plan
 - Les priorités temps réel : accès Administrateur
 - Threads dépendants de leur processus

- ◆ ... et réalités
 - Une stratégie qui change avec les versions...
 - L'ordonnanceur NT est très efficace dans sa gestion des capacités hardware
 - *Les optimisations spécifiques au niveau de l'application sont délicates*

Priorités des processus

- ◆ WinNT a 6 classes de priorité applicatives
 - IDLE_PRIORITY_CLASS
 - BELOW_NORMAL_PRIORITY_CLASS
 - NORMAL_PRIORITY_CLASS
 - ABOVE_NORMAL_PRIORITY_CLASS
 - HIGH_PRIORITY_CLASS
 - REALTIME_PRIORITY_CLASS
- ◆ Fonctions d'accès
 - SetPriorityClass
 - GetPriorityClass

Priorités des threads

- ◆ 32 niveaux de priorités
 - de 0 priorité minimale à 31 priorité maximale
 - découpés en 2 groupes:
 - groupe 0-15 priorité variable
 - groupe 16-31 priorité fixe (Microsoft parle de «temps réel»)
- ◆ Priorités = ordre d'allocation du processeur
 - Le système alloue le(s) processeur(s) au(x) thread(s) prêt(s) en respectant strictement les priorités.
 - En cas d'égalité de priorité, l'allocation se fait à part égale.

Priorités d'un thread Win32

- ◆ Win32 propose 7 niveaux de priorité
 - `THREAD_PRIORITY_LOWEST` (-2)
 - `THREAD_PRIORITY_BELOW_NORMAL` (-1)
 - `THREAD_PRIORITY_NORMAL` (défaut)
 - `THREAD_PRIORITY_ABOVE_NORMAL` (+1)
 - `THREAD_PRIORITY_HIGHEST` (+2)
 - `THREAD_PRIORITY_IDLE` (1 ou 16)
 - `THREAD_PRIORITY_TIME_CRITICAL` (15 ou 31)
- ◆ `SetThreadPriority`
- ◆ `GetThreadPriority`

Priorités Win32

