Cours Programmation Concurrente

Master MIAGE M1

Jean-François Pradat-Peyre,
Université Paris Nanterre - UFR SEGMI

2023-2024

1 : Introduction

Objectifs du cours

- Appréhender les notions de programmation concurrente
- Paradigmes de la concurrence et modèles de la programmation concurrente
- Mises en application en C/Posix, Java et Python

Déroulement du cours

- Le cours est composé de 6 séances de cours – CM / TD / TP
  - S1 : Intérêt de la programmation concurrente, OS, les différentes solutions
  - S2 : Créer des entités concurrentes, sémantique et mise en pratique
  - S3 : Synchronisation entre entités concurrentes, sémantique et illustrations en Java et C/Posix
  - S4 : Paradigmes de la concurrence 1
  - S5 : Paradigmes de la concurrence 2
  - S6 : Les coroutines avec Python et les « Future » en Java

- L'évaluation est constituée
  - D'un examen final (50%)
  - De QMC, travaux personnels et partiel (50%) au cours des séances

Bibliographie

- Java Concurrency in Practice, B. Goetz et al., 2006, Addison Wesley
- Mastering Concurrency Programming with Java 8, J. Fernadez Gonzalez, 2016
- Hands-on Design Patterns with Kotlin: Build scalable applications using traditional, reactive and concurrent design patterns in Kotlin, Alexey Soshin, 2018

Cours Programmation Concurrente

Master MIAGE M1

Jean-François Pradat-Peyre,
Université Paris Nanterre - UFR SEGMI

2023-2024

S1 : Intérêt de la programmation concurrente, bref historique des OS, les différentes solutions

Cours Programmation Concurrente

Master MIAGE M1

Jean-François Pradat-Peyre,
Université Paris Nanterre - UFR SEGMI

2023-2024

S1.1 : Quelques rappels sur les Systèmes d'exploitation (OS)

Rôles d'un OS (Operating System)

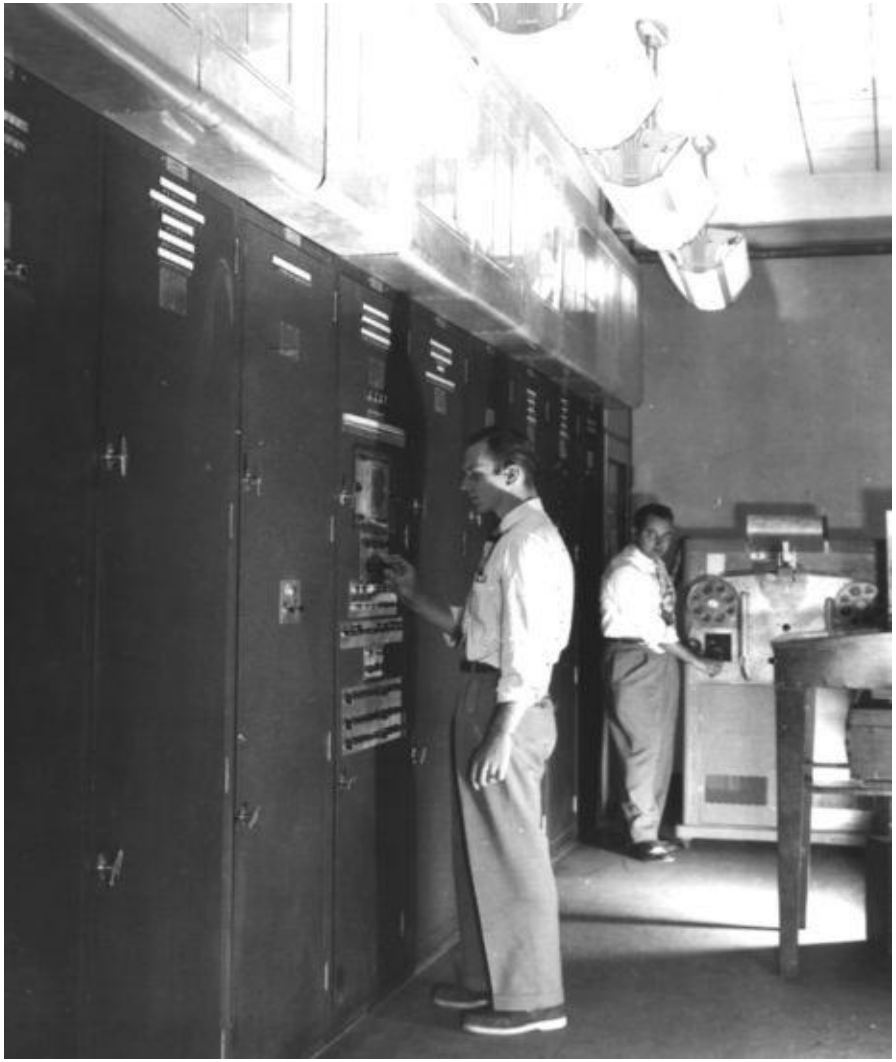
- Fournir des interfaces d'accès aux ressources de haut niveau
  - Abstraction du matériel
  - Abstraction des activités
- Permettre un partage des ressources physiques
- Gérer/simplifier les entrées sorties
- Assurer la sécurité et l'intégrité des données
  - Identification
  - Contrôle d'accès aux données
- Synchroniser les activités
- Assurer la communication inter activités

Concepts apparus peu à peu !

Bref historique (1)

- 1950 : EDVAC (Electronic Discrete Variable Automatic Computer) une des premières machines avec programme chargeable
- Usage militaro-scientifique
- Mémoire de 5,5 KO
- *l'OS se limite à un « lanceur » : notion d'amorçage (bootstrap)*

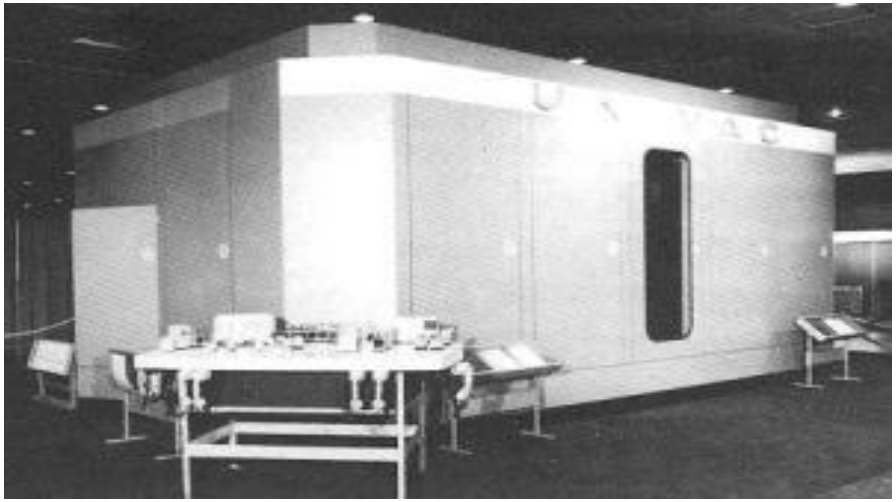
- voir http://en.wikipedia.org/wiki/EDVAC



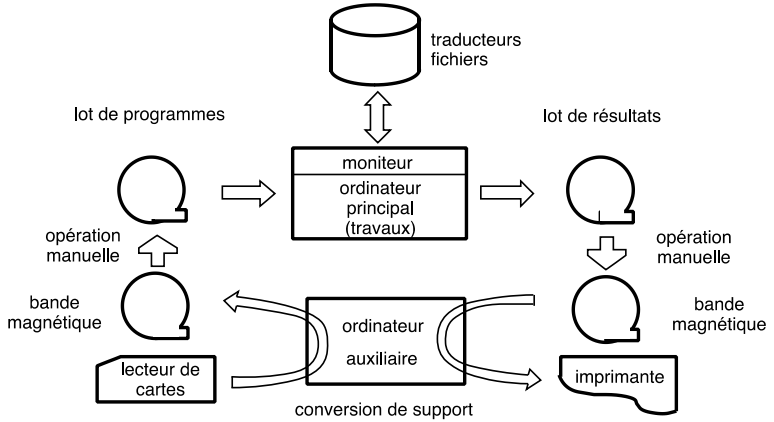
The EDVAC as installed in Building 328
at the Ballistics Research Laboratory.

Bref historique (2)

- 1960 : Machine UNIVAC (***UNIV**ersal **A**utomatic **C**omputer)* de Control Data Corporation
- *pas encore de réel OS : système batch (notion d'enchaînement automatique de travaux (jobs))*
- Puis amélioration des E/S grâce à des ordinateurs spécialisés d'entrées-sorties

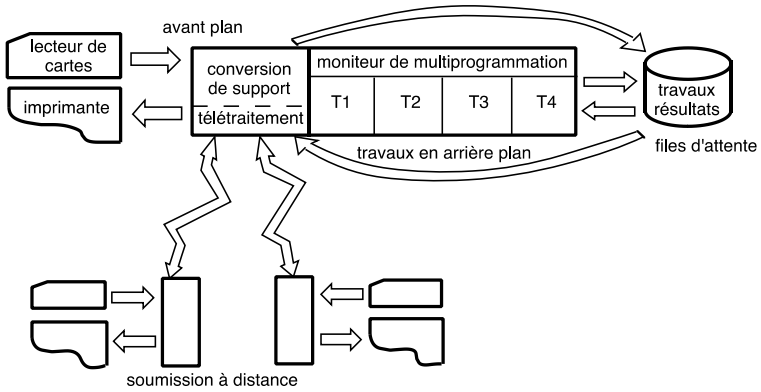


UNIVAC I Central Complex, containing the central processor and main memory unit.



Bref historique (3)

- 1964 : début des mainframes (IBM) et premiers OS
  - IBM OS /360 pour la famille des mainframes de la série System/360 (6 machines différentes)
  - Premiers *disques dur* (accès direct), *E/S tamponnées*
  - Introduction du parallélisme (*multitasking*) avec partage de temps (*time sharing*)



An IBM System 360/20 (front panels removed), with IBM 2560 MFCM (Multi-Function Card Machine)

Bref historique (4)

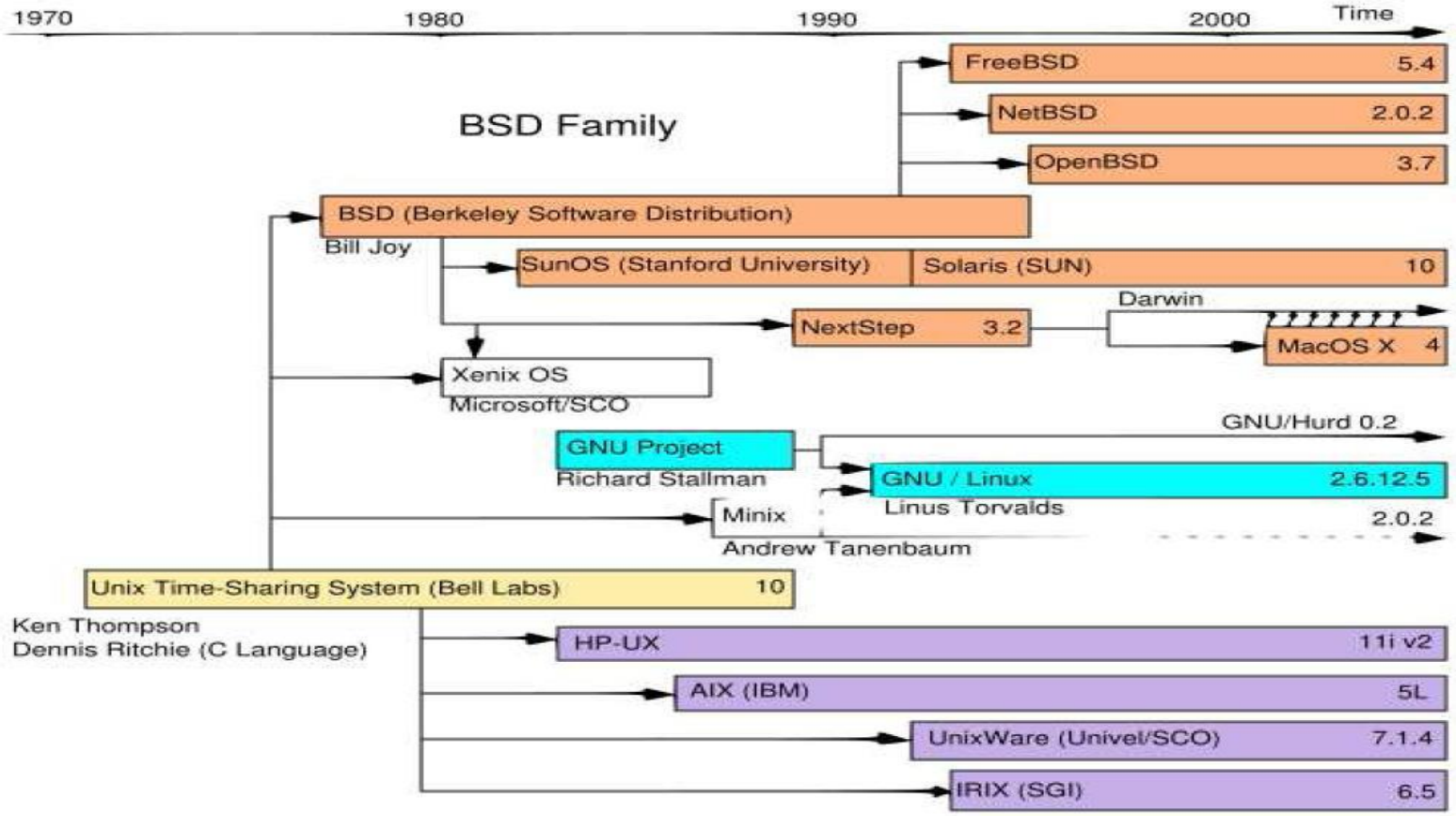
- 1964 : début de Multics (**Mult**iplexed **I**nformation and **C**omputing **S**ervice)
- 1969 : apparition d'Unix (initialement Unics) par (entre autres) Ken Thompson (langage B) et Denis Ritchie (langage C) chez AT&T /Bell Labs
- Système interactif
- Réécrit en langage C en 1973
- Très nombreux concepts modernes
  - Pas de distinction pour les programmes entre mémoire et fichiers (mmap)
  - Distinction noyau / espace utilisateur
  - Système de fichier hiérarchisé
  - Multi tâches multi utilisateurs
  - Tubes (pipe)
  - Configuration par fichier textes (non binaires)

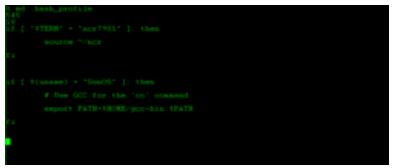
Ken Thompson (left) with Dennis Ritchie

```
16087 ?? IW      0:00.04 bash /home/users/t/ta/tanders/src/net-snmp-main/dist/ns
16200 ?? DW      0:00.00 grep -v ^ *+ conftest.erl
16610 ?? IW      0:04.82 /bin/bash /home/users/t/ta/tanders/src/net-snmp-V5-1-pa
16929 ?? I       0:00.16 sh Compile cvs/RELEASE wxGTK
17066 ?? I       0:00.00 sh Compile cvs/RELEASE wxGTK
17686 ?? IW      0:00.03 bash /home/users/t/ta/tanders/src/net-snmp-main/dist/ns
17774 ?? IW      0:06.28 /bin/bash /home/users/t/ta/tanders/src/net-snmp-V5-3-pa
18239 ?? I       0:00.00 sh Compile cvs/RELEASE wxGTK
19497 ?? IWs     0:00.12 sshd: tanders@notty
23894 ?? S       0:00.07 gmake
24011 ?? S       2:09.33 snmpd -d -r -U -p /tmp/snmp-test-31-11960/snmpd.pid.num
24544 ?? S       0:00.00 /bin/sh ../../../../../bk-deps g++ -c -o ogledit_view.o
26353 ?? IW      0:00.04 bash /home/users/t/ta/tanders/src/net-snmp-main/dist/ns
27935 ?? DE      0:00.01 (bash)
28366 ?? IWs     0:00.13 sshd: tanders@notty
29812 ?? IWs     0:00.03 bash /home/users/t/ta/tanders/src/net-snmp-main/dist/ns
 2336 p0 R+      0:00.00 ps ax
21906 p0 Is      0:00.01 -bash
24304 p0 S       0:00.03 sh
  697 00 IWs+    0:00.01 /usr/libexec/getty Pc console
  639 E1 IWs+    0:00.01 /usr/libexec/getty Pc ttyE1
  734 E2 IWs+    0:00.01 /usr/libexec/getty Pc ttyE2
  762 E3 IWs+    0:00.01 /usr/libexec/getty Pc ttyE3
$
```

A partial list of simultaneously running processes on a Unix system.

Bref historique (5) : aperçu de la famille Unix





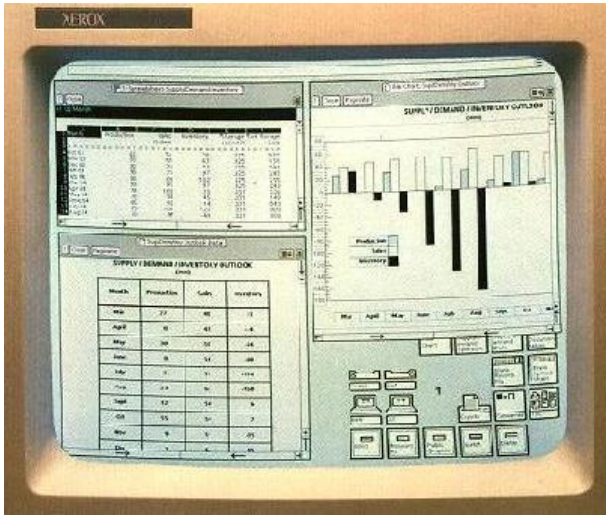
Bref historique (6) : aperçu de la famille Unix (suite)

- Systèmes partiellement compatibles
- Standards : Posix, X /Open, BSD, System V, …
- Systèmes modulaires
- Sources accessibles (modèle « Open Source ») et Code de bonne qualité
- Peuvent être réduits fortement en taille (micro noyau)
- Offrent un ensemble d'outils efficaces
  - Compilateurs (gcc, cc, ar, …)
  - Langages de commandes (bash, csh, …)
  - Éditeurs (ed, vi, emacs, …)
  - Traitements de texte (LaTex, …)
  - Fenêtrage (X, kde, ….)
  - …

Bref historique (7) : la micro-informatique

- 1974: un des premiers micro-ordinateurs (PC) Xerox PARC, puis STAR 8010
- OS: Alto OS *Premier système interactif avec bureau, fenêtres barre d'outils et souris (développé en 1981)*
- Pas de succès commercial
- Idées reprises par la suite par Apple et Microsoft

La console avec le clavier et la souris



Une photo d'écran

Bref historique (8) : la micro-informatique (suite)

- 1975 : Fondation de Microsoft
- 1977 : Lancement de l'Apple II
  - Lecteur de cassette puis de disquette 5'1/4
  - Pas vraiment d'OS
- 1979 : application VisiCal sur Apple II
  - premier tableur
  - très forte augmentation de ventes
- 1981 : IBM PC (IBM) système MS-DOS
- 1984 : Lancement du Macintosh et système MacOS
- 1987 : Windows 2.0 et IBM OS/2
- 1991 : Linux kernel 0.01,
- 1992 : Windows 3.1 et 3.11
- …

IBM PC

Apple II

Macintosh

Bref historique (9) : les systèmes embarqués puis mobiles

- Dès le début des années 1950
  - Systèmes embarqués
    - ✓ Avions, sous-marins, fusées, etc.
  - Systèmes dits aussi « temps-réel »
    - ✓ Peu de ressources
    - ✓ Des contraintes temporelles fortes

- Puis avec l'évolution de l'électronique
  - Systèmes domestiques
    - ✓ Lecteurs CD puis DVD, puis …
  - Appareils mobiles
    - ✓ Smartphone, tablette, montres
  - Et maintenant les systèmes pervasifs
    - ✓ Capteurs, puces rfid, …

Les différents modes d'utilisation d'un OS

- mode interactif: suite de commandes brèves, réponses rapides, réflexion - décision
  - temps partagé = machine pour soi seul,
    création, test, mise au point, exécution de programmes quelconque
  - transactionnel = exécution de programmes prédéfinis
    consultation et mise à jour de bases de données
  - conception assistée = puissance de machine, aspects graphiques

- mode différé: traitement par lot, travail important, nombreuses ressources, résultats en quelques heures

- mode temps réel: contrôle de procédés industriels,
  saisie et commande faites à des moments imposés

Système général ou spécialisé

- Multics (Bull), VMS (Dec), VM/CMS (Ibm)
  - privilégient le temps partagé (mode interactif)
  - bon fonctionnement en transactionnel et en différé
  - Multics et VM non adaptés au temps réel ou à la conception assistée
  - VMS configurable pour temps réel et conception assistée
- Unix, Linux, Windows
  - privilégient le temps partagé (mode interactif)
  - configurables pour la conception assistée, non adaptés au temps réel
- Gcos3 (BULL), MVS (IBM)
  - privilégient le mode différé
  - autres modes obtenus par sous-systèmes (TSO, CICS, IMS sur MVS)
- QNX, PalmOS, Windows CE, Java Card, IOS, LynxOS, µc Linux, …
  - Privilégient le mode temps réel avec intégration de contraintes de taille

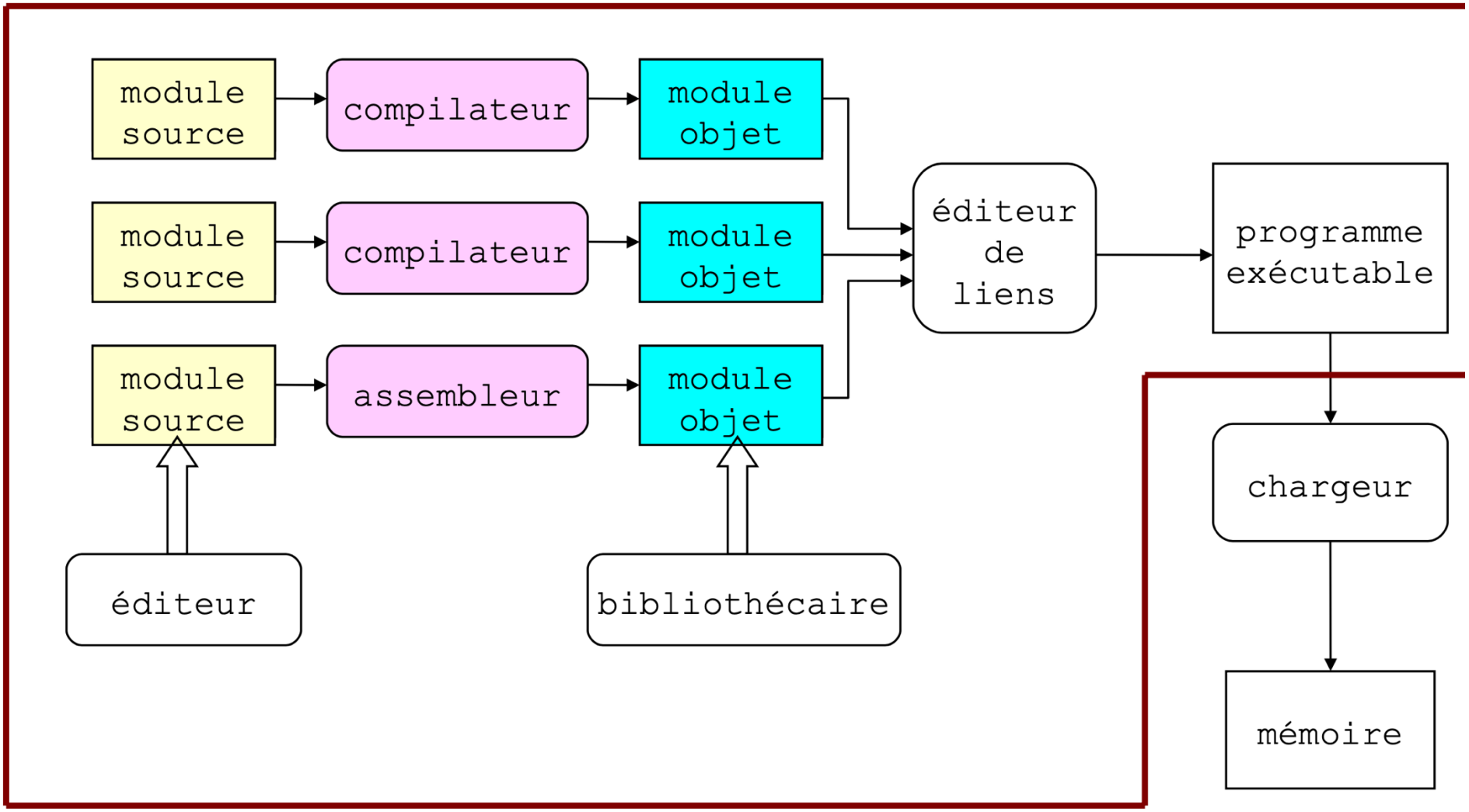
Production de programmes (1)

- Outils logiciels utiles pour la réalisation des programmes
- Interpréteur de commande (Shell, exécution de programmes / scripts)
- Éditeur de texte (création et modification de programmes)
  - éditeurs lignes => découpage de texte en lignes, opérations sur les lignes
    - ✓ fonctions déduites de celles sur cartes perforées
    - ✓ actuellement utilisation réservée à des traitements automatiques
  - éditeurs pleine page => découpage texte en pages = écran
    - ✓ déplacement et modification sur l ’écran, texte = chaîne de caractères
  - éditeurs syntaxiques => tenir compte de la structure du langage
    - ✓ manipulations guidées par la syntaxe du langage

Production de programmes (2)

- compilateurs, interpréteurs, assembleurs (traduction)
  - programme dans un langage donné => programme en langage machine
- éditeurs de liens, chargeur
  - découpage des logiciels de bonne taille en modules
  - éditeur de liens rassemble les modules en un programme exécutable
  - chargeur met en mémoire le programme exécutable
- metteur au point (aider à la mise au point des programmes)
- paragrapheur (mise en forme du texte source)
- bibliothécaire (gestion des modules déjà compilés)
- etc...

Production de programmes (3)



Abstractions offertes par le système (1)

- Le système offre une vue abstraite de la machine à l'utilisateur et aux programmes

- Pourquoi fournir des abstractions ?
  - Assurer la portabilité
  - Simplifier le développement en fournissant une machine virtuelle correspondant aux besoins du programmes
  - Assurer l'indépendance des usagers
  - Simuler les ressources qui manquent

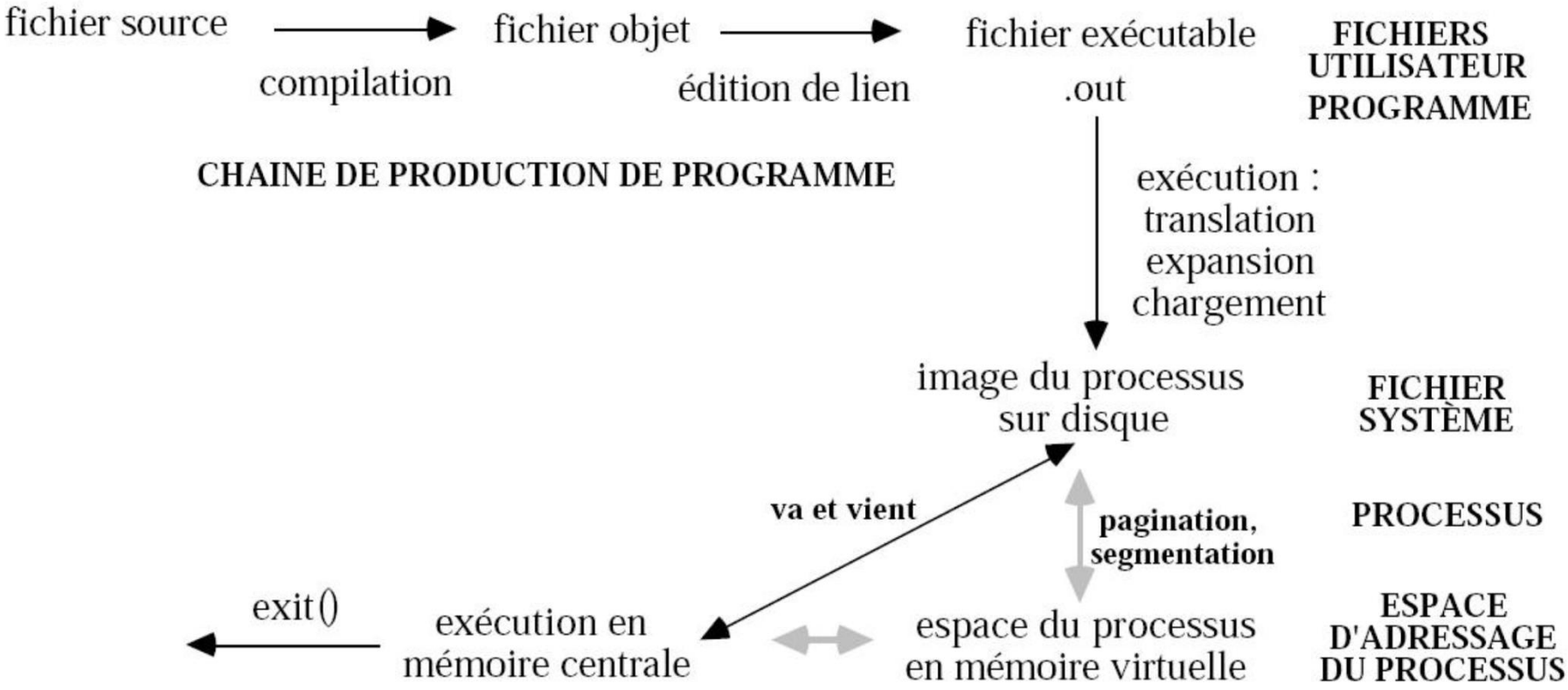
Abstractions offertes par le système (2)

- Désignation des ressources
- Gestion des fichiers et système de fichiers
  - En Unix l'arborescence unique est construite à l'aide de points de montage
- Accès aux périphériques et dialogue avec les pilotes de périphérique
  - En Unix ce sont des fichiers spéciaux (type block ou caractères)
- Accès à la mémoire physique avec la Mémoire virtuelle
  - Dissociation entre la mémoire physique et celle manipulée par les programmes
  - Les adresses sont virtuelles, la translation est faite par le CPU à l'aide de circuits MMU (Memory Management Units) et de cache pour la table des pages TLB (Translation Lookaside Buffer)
  - Pagination, segmentation (ou les deux à la fois)
- Manipulation et gestion de Programmes grâces aux processus
  - Un processus exécute un programme dans un espace d'adressage privé
  - Plusieurs processus peuvent se dérouler simultanément (meilleur efficacité du système, conception plus simple)
  - Un processus est caractérisé par son état, son espace d'adresssage et son contexte
  - Lorsque les processus partagent leur espace d'adressage on parle de *thread*

Notion de processus : Un pas vers l'abstraction

- Un processus = un support d'exécution d'un programme
- Les actions d'un processus sont exécutés séquentiellement
- Chaque processus pense disposer de toutes les ressources pour lui seul : notion de « virtualisation » des ressources
  - Processeur
  - Mémoire
  - Périphériques
- Offre un moyen de protection entre les programmes
  - Une erreur dans un processus n'a pas de conséquences sur les autres
- Permet de se concentrer sur la solution d'un problème spécifique ; exemple :
  - Un processus gère les E/S réseaux (ex. driver réseau)
  - Un autre (ex httpd) dispache les données vers des processus « consommateurs » (ex. Navigateur, Mail, Transfert de fichier, etc.)
  - Chaque processus consommateur a sa propre logique (programme)

Du programme au processus (vue assez détaillée)



Abstractions offertes par le système (1)

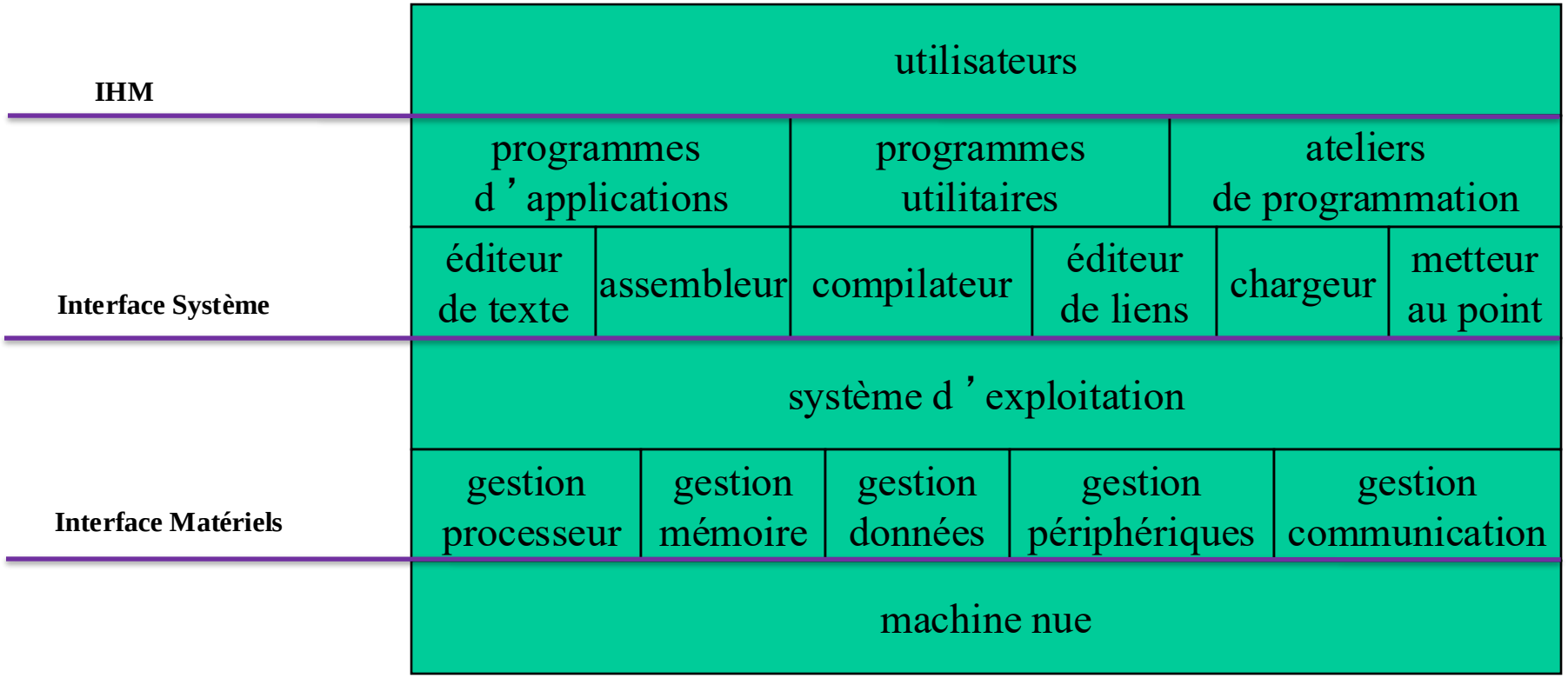
- Le système offre une vue abstraite de la machine à l'utilisateur et aux programmes

- Pourquoi fournir des abstractions ?
  - Assurer la portabilité
  - Simplifier le développement en fournissant une machine virtuelle correspondant aux besoins du programmes
  - Assurer l'indépendance des usagers
  - Simuler les ressources qui manquent

Abstractions offertes par le système (2)

- Désignation des ressources
- Gestion des fichiers et système de fichiers
  - En Unix l'arborescence unique est construite à l'aide de points de montage
- Accès aux périphériques et dialogue avec les pilotes de périphérique
  - En Unix ce sont des fichiers spéciaux (type block ou caractères)
- Accès à la mémoire physique avec la Mémoire virtuelle
  - Dissociation entre la mémoire physique et celle manipulée par les programmes
  - Les adresses sont virtuelles, la translation est faite par le CPU à l'aide de circuits MMU (Memory Management Units) et de cache pour la table des pages TLB (Translation Lookaside Buffer)
  - Pagination, segmentation (ou les deux à la fois)
- Manipulation et gestion de Programmes grâces aux processus
  - Un processus exécute un programme dans un espace d'adressage privé
  - Plusieurs processus peuvent se dérouler simultanément (meilleur efficacité du système, conception plus simple)
  - Un processus est caractérisé par son état, son espace d'adresssage et son contexte
  - Lorsque les processus partagent leur espace d'adressage on parle de *thread*

Architecture générale d'un système (simplifiée)



Exécution concurrente : OS multi tâches

- Origine : besoin de partager la machine entre différents programmes / utilisateurs
- Intérêts : permet de « partager » de façon virtuelle la machine entre différentes activités et/ou utilisateurs → économie d'échelle, meilleure utilisation des ressources
- Virtualisation du processeur (un par processus)
- Plusieurs modes possibles :
  - Les uns après les autres (pas de concurrence)

  - Partage du processeur (pseudo parallélisme) entre les processus

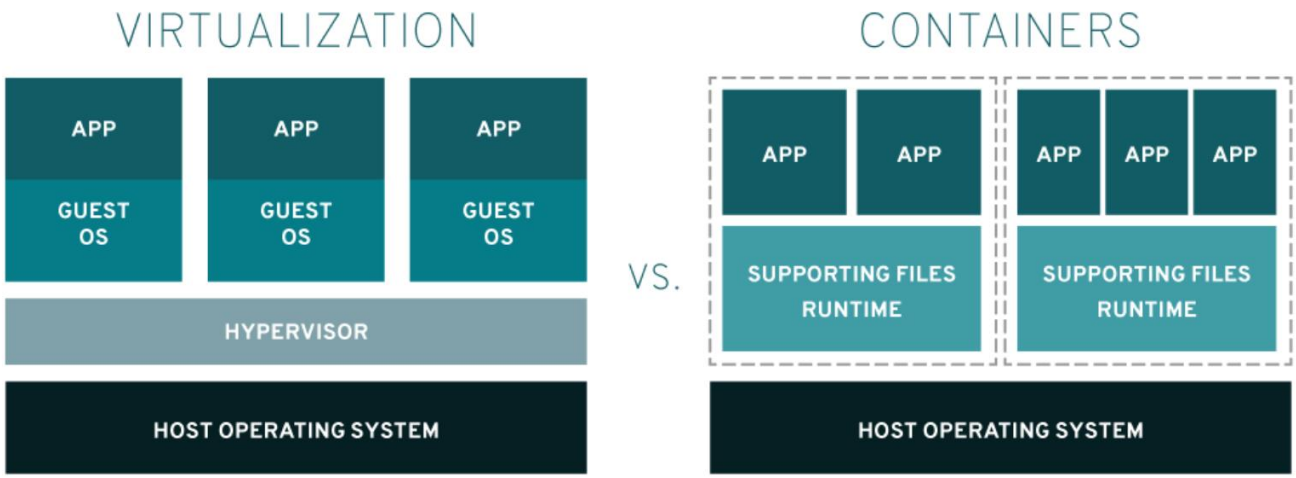
  - Utilisation simultanée de plusieurs processeurs (parallélisme sur multi coeurs)

Virtualisation de la mémoire

- Définir la représentation de la mémoire vue par l'utilisateur
  - Mémoire linéaire
  - Mémoire segmentée
  - Vue comme infinie
- Définir la représentation de la mémoire vue au niveau physique
  - Cases en mémoire physique
  - Limitation physique naturelle
- Définir une correspondance entre ces deux vues
  - Notion de « pagination » à la demande
  - Notion de zone de « swap » avec utilisation de la mémoire secondaire (disque) pour évacuer des pages qui ne servent plus au profit de nouvelles pages nécessaires à un processus actif
- Peut conduire à des phénomènes d'écroulement (***trashing***)

Machines Virtuelles / conteneurs

- Donner à des applications binaires l'impression de s'exécuter sur un OS particulier
- Utilisation d'un hyperviseur
  - Présente une interface uniforme aux différentes VM
  - Encapsule l'état des VM
  - Isole et protège les VM les unes des autres
- Les conteneurs sont une solution plus légère mais se limite à un OS (OS natif)



Cours Programmation Concurrente

Master MIAGE M1

Jean-François Pradat-Peyre,
Université Paris Nanterre - UFR SEGMI

2023-2024

S1.2 : Premières notions sur la Programmation Concurrente

Programmation Concurrente ?

- Le terme **Concurrence** doit se comprendre comme
  - Concourir à un même objectif, travailler à plusieurs dans un même sens
  - Eventuellement se gêner mais pas nécessairement de compétition
  - En Anglais Concurrency
- Un programme concurrent comprend donc
  - Différentes activités qui s'exécutent en même temps
  - Du vrai ou du pseudo parallélisme
  - Des activités locales ou distantes
- Une activité d'un programme concurrent
  - Peut être appelée tâche (thread), processus, activité, coroutine
  - Possède des données propres et des données partagées
  - Peut collaborer avec d'autres activités (communication, synchronisation)

Intérêts de la Programmation Concurrente (1/2)

- Un programme vise à automatiser / simplifier / améliorer des activés humaines
  - Envoyer un courrier personnalisé à de nombreuses personnes
  - Éditer les notes d'une promotion
  - Produire un graphique représentant l'évolution des températures sur une période
- Une activité humaine se décompose en
  - Des séquences d'actions élémentaires : mesurer une donnée puis l'enregistrer
  - Des collaborations : la saisie des données courantes est indépendante de la construction d'un graphique sur des données anciennes et ces deux actions peuvent être faites par différents acteurs (avec différentes compétences)
  - Des synchronisations implicites ou explicites : une donnée ne peut être enregistrée que si elle a été mesurée (implicite), lorsque toutes les données d'une période sont enregistrées on signale que le graphique peut être produit (explicite)
- Construire un programme consiste à traduire la réalité en abstractions
  - Plus le langage d'abstraction est proche de la réalité plus le programme est simple à concevoir et à réaliser : s'appuyer sur une abstraction des activités humaines

Intérêts de la Programmation Concurrente (2/2)

- Optimiser l'utilisation des ressources
  - Temps humain >> temps processeur
  - Temps E/S (mémoire, disque, réseau) >> temps processeur
  - → Utiliser le **temps processeur libre** entre 2 actions humaines ou 2 E/S
- Franchir des barrières technologiques
  - La densification des circuits atteint les limites physiques (qqs centaines d'atomes)
  - L'augmentation de la fréquence horloge augmente exponentiellement la dissipation de chaleur (et de la consommation électrique)
  - → Concevoir des puces **multi-cœurs** plutôt que des processeurs monolithiques
- Concevoir des architectures plus robustes
  - Panne matérielle non négligeable
  - Dépendance de plus en plus grande aux systèmes informatiques
  - → **Répartir** les données et les calculs sur plusieurs machines augmente la fiabilité

Les différentes architectures de la concurrence

- Modèles Synchrones
  - Toutes les entités concurrentes sont rythmées par une horloge commune
  - Les temps de calculs élémentaires sont assimilés à un temps nul
  - Destinés aux systèmes embarqués (avion, missiles, voiture, etc.)
  - De grandes réussites industrielles (en particulier Françaises ou Européennes)
- Modèles Asynchrones
  - Chaque entité concurrente avance à son propre rythme
  - La synchronisation se fait explicitement ou implicitement
  - C'est le modèle plus répandu (et que nous utiliserons)
- Modèles Centralisés ou Répartis
  - En modèle centralisé les entités s'exécutent sur la même machine
  - En modèle réparti les entités s'exécutent sur des machines reliées en réseaux
  - C'est par nature un modèle asynchrone

Nous travaillerons dans ce cours sur le modèle Centralisé Asynchrone

Les prochaines séances

- S2 : Créer des entités concurrentes, sémantique et mise en pratique
- S3 : Synchronisation entre entités concurrentes, sémantique et illustrations en Java et C/Posix
- S4 : Paradigmes de la concurrence 1
- S5 : Paradigmes de la concurrence 2
- S6 : Les coroutines avec Kotlin (et programmation concurrente sous Android)