Introduction aux systèmes GNU/Linux

Module IntroLinux – S25E03

Philippe Latu / Université de Toulouse
inetdoc.net

Cours 3
Le plan

- Environnements graphiques
- Shell BASH
- Permissions Unix
- Initialisation système

Environnements graphiques

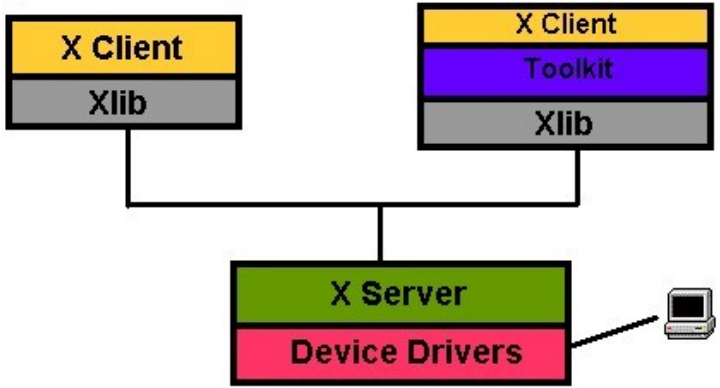
Environnements graphiques – Historique

Consortium X-Window

- 1986 : première version diffusée par le MIT
- 1992 : projet XFree86
  - Au départ : processeurs Intel i386
  - Ensuite : processeurs du projet GNU

Fondation X.Org

- 2004 : lancement http://www.x.org
  - Architecture modulaire
- 2008 : Projet wayland



X = Protocole réseau

Environnements graphiques – X.Org

Architecture X.org – freedesktop.org

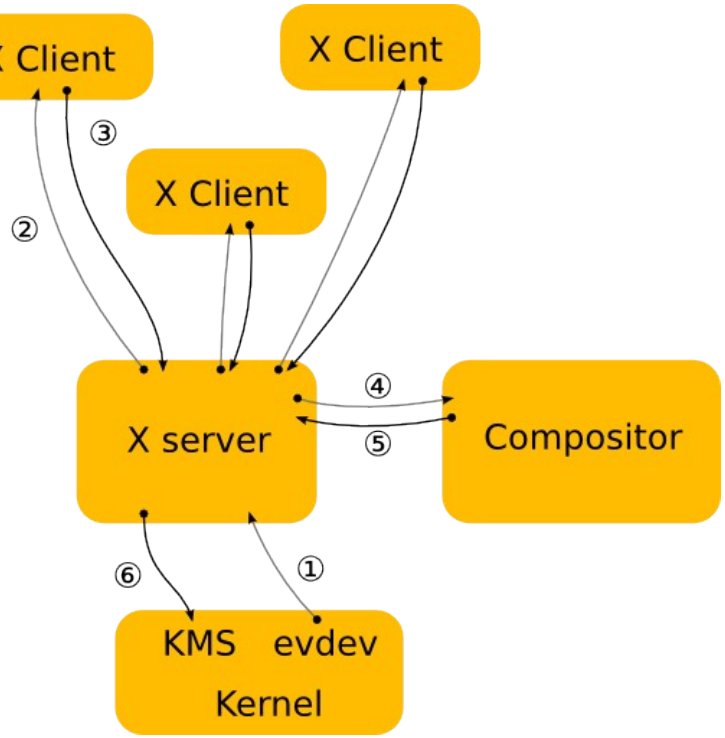
- Architecture client/serveur
- Serveur X = service mandataire ou proxy
- Tous les traitements ont lieu côté Compositor

Solution modulaire

- ⊕ Ajout de fonctions supplémentaires
- ⊖ Accumulation dans le temps

Fonctions réseau

- Application graphiques accessibles à distance



Environnements graphiques – Wayland

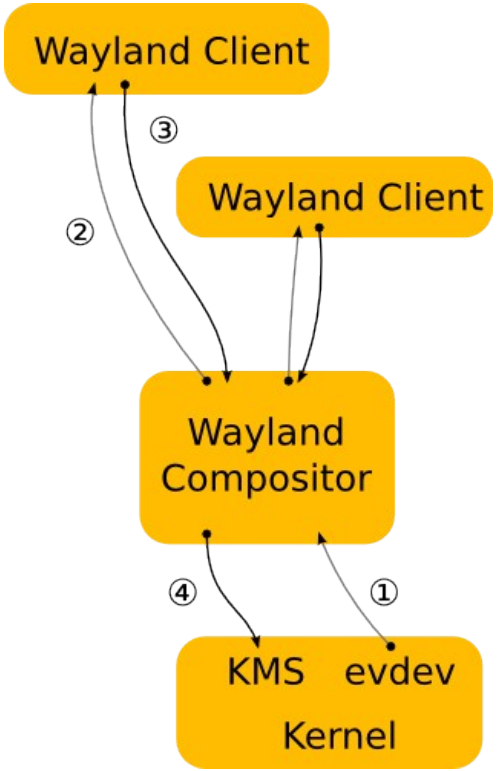
Architecture Wayland – freedesktop.org

- Architecture «simplifiée»
- Plus de service mandataire côté Compositor

Solution adoptée par GNOME / Ubuntu

Sponsors actifs : Qt, Intel

Travaux en cours : NVidia



Environnements graphiques – Développement

- Un écosystème beaucoup plus structuré qu'il n'y paraît
  - Environnement graphique == Chaîne de développement
  - Démonstrateur de l'état courant des avancées techniques

- Très(Trop) grande diversité des développements / bibliothèques

- Modèle OpenSource – avantages et inconvénients

  ⊕ Introduction facile de nouvelles fonctionnalités

  ⊖ Coordination difficile entre projets «concurrents»

Environnements graphiques – Exemples

- LXDE & XFCE

  Solutions orientées faibles capacités de traitement

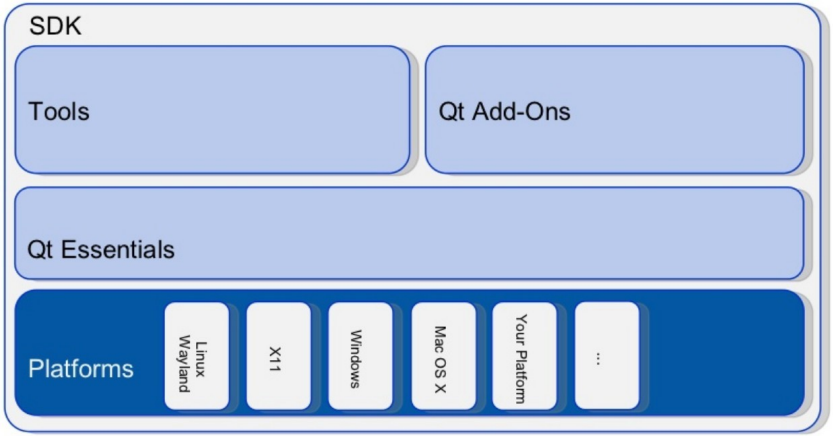
  Systèmes embarqués, Kali Linux, Xubuntu : https://xubuntu.org/

- Gnome & GTK

  RAD - Glade

- KDE & Qt

  RAD – Qt Creator



RAD : Rapid Application Development

Bilan Environnements Graphiques

- Cloud et Virtual Desktop Infrastructure (VDI)
- Compromis performances consommation énergétique
- Rapid Application Development

Shell BASH

Shell BASH – Historique

1976 : Unix version 7 → Bourne Shell

1989 : GNU Bash → Bourne again Shell (Born again Shell)

- Interpréteur de commandes : brace expansion, command line completion, etc.
- IEEE POSIX P1003.2/ISO 9945.2 Shell and Tools standard

Shell : Environnement de développement

- Algorithmique : tests, conditions et boucles, fonctions, arithmétique et tableaux
- Chaînes de caractères : alias, filename globbing, brace expansion ...

Documentation

- Reference manual ou Advanced Bash Scripting

Shell BASH – Ligne de commande

- Auto-correction : commande `'shopt -s dirspell'`
- Auto-completion : touche tabulation
- Historique et rappel des commandes antérieures
  - Commande `'history'`
- Séquences de touches
  - Rappel de l'historique : `'Ctrl r +[caractères commande]'`
  - Curseur en début de ligne : `'Ctrl a'`
  - Curseur en fin de ligne : `'Ctrl e'`
  - Défilement de l'historique : flèches haut et bas

Shell BASH – Séquencement

- Tubes ou pipes : `ip nei ls | grep de:b4:8a:f8:f1:4d`
- Redirection dans un nouveau fichier : `echo "syntax on" > ~/.vimrc`
- Redirection avec ajout en fin de fichier : `echo "syntax on" >> ~/.vimrc`
- Enchaînements logiques entre commandes
  - Inconditionnel : exécution de la commande 1 puis de la commande 2

    `commande1 ; commande2`
  - Si la commande 1 se termine avec succès, on passe à la commande 2

    `commande1 && commande2`
  - Si la commande 1 échoue, on passe à la commande 2

    `commande1 || commande2`

Shell BASH – Heredocument

- Formater une commande sur plusieurs lignes

```
cat << EOF
The current working directory is: $PWD
You are logged in as: $(whoami)
EOF
```

- Créer un script sans éditeur

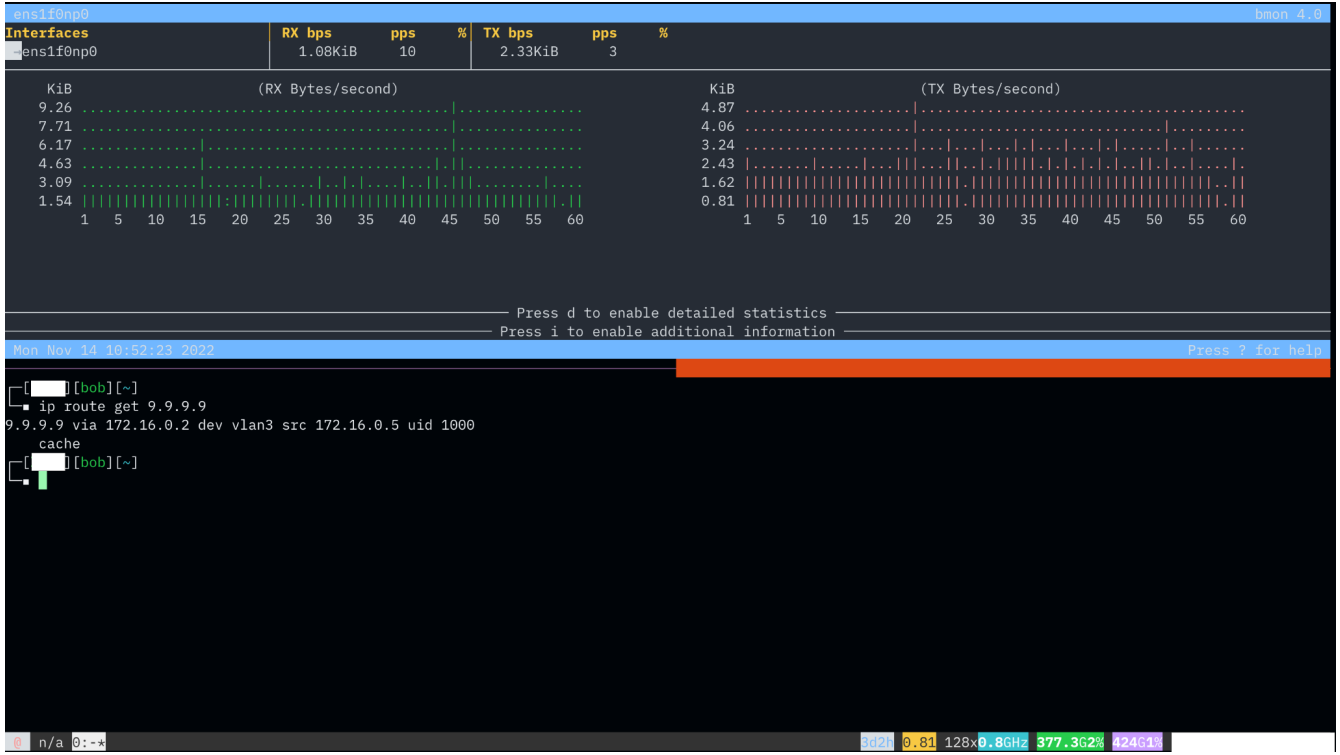
```
for i in {0..2}
do
  cat << EOF >> start3vms.sh
  echo "Start vm $i ..."
  $HOME/vm/scripts/ovs-startup.sh \
  vm$i.qcow2 512 $(($i + 10))
  EOF
done
```

- Envoyer une suite d’instructions via SSH

```
ssh -t user@server.com sh -s << 'EOF'
  echo "My own VM list"
  cd $HOME/vm
  ls -lAh *.qcow2
  echo "That’s all folks!"
EOF
```

Shell BASH – Gestionnaire de fenêtres

- Conserver les fenêtres ouvertes entre différentes sessions ou connexions
- Byobu
  basé sur tmux
- https://zellij.dev/
  plus récent



Shell BASH – Personnalisation

Tous les éléments de personnalisation en un seul outil
→ Oh My Bash
https://github.com/ohmybash/oh-my-bash

Paramètres prédéfinis

- Thème
- Alias
- Invite de commandes
- Intégration git

Bilan Shell Bash
Carrefour du monde DevOps

- Dev : algorithmique de base
  traitements non idempotents
- Préambule :
  aux outils de l'automatisation
  aux traitements idempotents
- Ops : pilier essentiel de l'administration système

Idempotence : f(f(x)) = f(x) pour tout x

Préparation aux travaux pratiques

Analysez le code du script ci contre

- Que signifie “*Bash shebang*” ?
- Pourquoi “set -e” ?
- Qu’est-ce qui distingue les conditions des deux tests if [...] ?
- À quel répertoire correspond ~ ?
- Pourquoi “cat << EOT” au lieu de echo ?
- Pourquoi “exit 0” ?

```
#!/bin/bash

set -e

GREEN='\E[32m'
NC='\e[0m' # No Color

echo -e "${GREEN}Préparation à l’utilisation des machines virtuelles${NC}"

# Tests sur l’existence du répertoire et des liens
if [ ! -L ~/masters ]; then
        cat << EOT
        Création du lien vers le catalogue des
        fichiers images de machines virtuelles.

        EOT
        ln -s /var/cache/kvm/masters ~
fi

if [ ! -d ~/vm ]; then
        cat << EOT
        Création du dossier de dépôt des copies
        des fichiers images de machines virtuelles.

        EOT
        mkdir ~/vm
        ln -s ~/masters/scripts ~/vm/
fi

echo -e "${GREEN}Arborescence prête${NC}"
tput sgr0

exit 0
```

Permissions Unix

ghost from the past

Permissions Unix – postulat de départ

Paradigme Unix :

“Everything is a file descriptor or a process”

Objectif :

Une interface uniforme pour
les applications et/ou les utilisateurs

Objets représentés sous forme de fichiers :

Processus, fichiers, répertoires, sockets, tubes (*pipes*)

Permissions Unix – définitions

Tout fichier possède 3 attributs

- Propriétaire – ***Owner/User***
  Actions possibles pour le propriétaire du fichier

- Groupe – ***Group***
  Actions possibles pour les membres du groupe défini pour le fichier

- Autre – ***Other***
  Actions possibles pour tous les utilisateurs du système

Permissions Unix – modes d’accès

Chaque attribut possède 3 permissions

- Lecture – ***Read***
  Lire ou visualiser le contenu du fichier
- Écriture – ***Write***
  Modifier ou effacer le contenu du fichier
- Exécution – ***eXecute***
  Exécuter le fichier comme une application

Permissions Unix – exemple

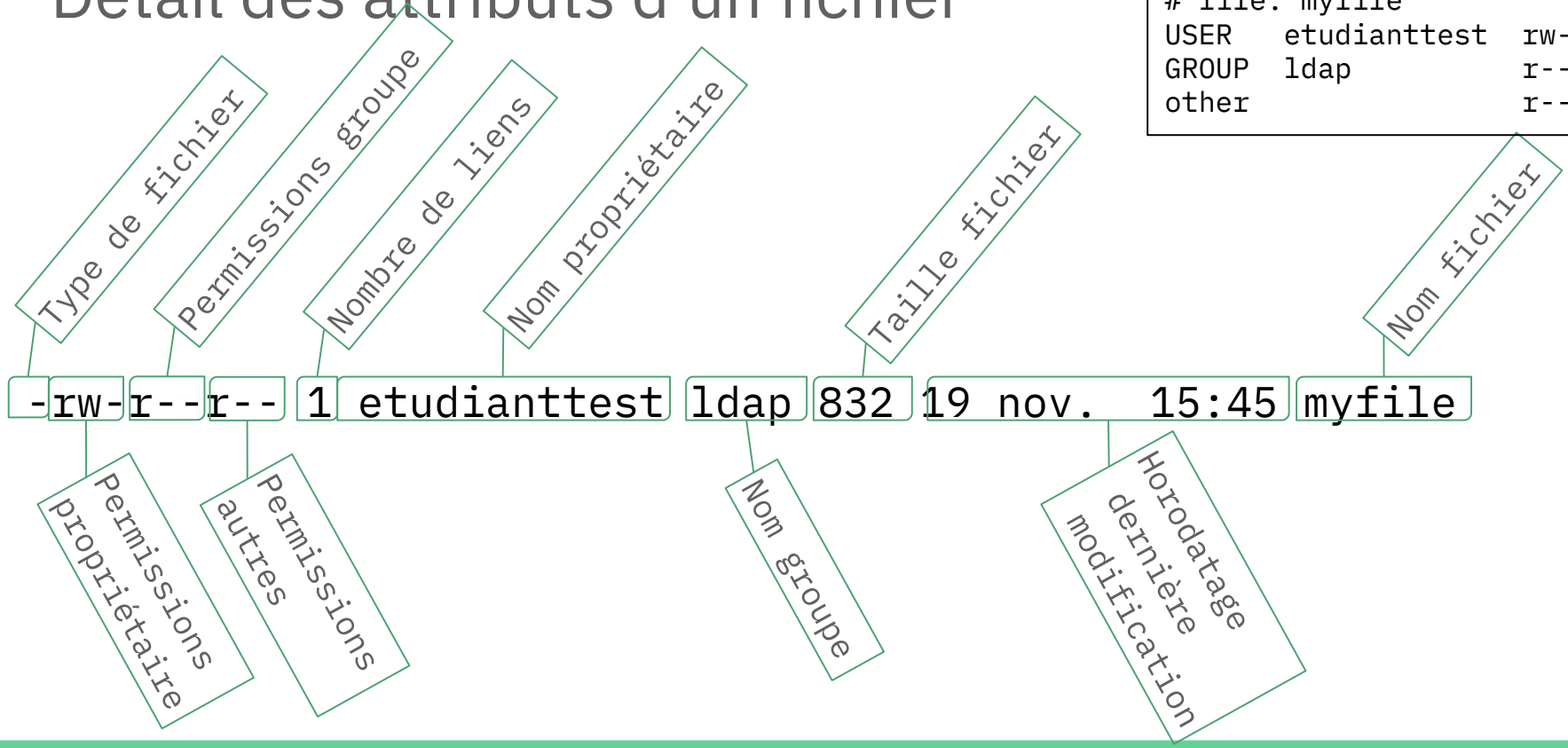
Visualisation des permissions avec la commande `ls`

```
┌─[etudianttest][bob][~]
└─▪ ls -lAh
total 48K
-rw-------  1 etudianttest ldap 1,2K 19 nov.  15:18 .bash_history
-rw-r--r--  1 etudianttest ldap  220 27 oct.  10:55 .bash_logout
-rw-r--r--  1 etudianttest ldap 4,4K 12 nov.  16:32 .bashrc
lrwxrwxrwx  1 etudianttest ldap   22 19 nov.  15:17 masters -> /var/cache/kvm/masters
drwxr-xr-x 15 etudianttest ldap 4,0K 12 nov.  16:32 .oh-my-bash
-rw-r--r--  1 etudianttest ldap   17 12 nov.  16:32 .osh-update
-rw-r--r--  1 etudianttest ldap  807 27 oct.  10:55 .profile
-rw-r--r--  1 etudianttest ldap    0 19 nov.  15:32 .sdirs
drwx------  2 etudianttest ldap 4,0K 27 oct.  10:58 .ssh
-rw-------  1 etudianttest ldap 8,0K 19 nov.  15:18 .viminfo
-rwxr-xr-x  1 etudianttest ldap  609 19 nov.  15:18 virt-tree.sh
drwxr-xr-x  2 etudianttest ldap 4,0K 19 nov.  15:17 vm
```

Permissions Unix – tous les attributs

Détail des attributs d'un fichier

```
getfacl -t myfile
# file: myfile
USER   etudianttest  rw-
GROUP  ldap          r--
other                r--
```



Permissions Unix – type de fichiers

–	fichier standard
b	fichier accessible en mode bloc : échange de blocs de bits
c	fichier mode caractère : échange de caractères
d	répertoire
l	lien symbolique
p	*named pipe* : mécanisme de communication interprocessus
s	*socket* : mécanisme de communication interprocessus

Permissions Unix – notations

Notation symbolique ou numérique

Chaque “bit” de permission d’un attribut correspond à une puissance de 2

Codage du masque des permissions sous deux formes équivalentes

- notation symbolique
- notation numérique

r	w	x
2^2	2^1	2^0
4	2	1

```
rw- r-- --- = 4+2   4   0   : masque des permissions = 640
rwx r-x r-x = 4+2+1 4+1 4+1: masque des permissions = 755
```

Permissions Unix – exemple

Manipulations simples sur les permissions d’un fichier script Bash

Commande : chmod

```
cat << EOF >hello.sh
#!/bin/bash

echo "Hello, World!"

exit 0
EOF
```

```
ls -l hello.sh
-rw-r--r-- 1 etudianttest ldap 42 19 nov.  16:57 hello.sh

chmod +x hello.sh

ls -l hello.sh
-rwxr-xr-x 1 etudianttest ldap 42 19 nov.  16:57 hello.sh

chmod 750 hello.sh

ls -l hello.sh
-rwxr-x--- 1 etudianttest ldap 42 19 nov.  16:57 hello.sh
```

Permissions Unix – permissions étendues

3 permissions supplémentaires :

SETUID	le processus qui exécute le fichier prend les privilèges du propriétaire du fichier
SETGID	le processus qui exécute le fichier prend les privilèges du groupe propriétaire du fichier
Directory Sticky bit	seul son propriétaire peut créer/modifier/supprimer un fichier dans un répertoire avec cette permission activée

Exemples

```
ls -lAh /usr/bin/passwd
-rwsr-xr-x 1 root root 67K 11 nov.  09:28 /usr/bin/passwd

ls -ld /var/local
drwxrwsr-x 2 root staff 4096  3 oct.  23:30 /var/local

ls -ld /var/tmp
drwxrwxrwt  5 root root  4,0K 19 nov.  09:25 tmp
```

Bilan Permissions Unix

- Paradigme Unix
  Everything is a file descriptor or a process
- Permissions : savoir essentiel de l'administration système
- Utilisateur, Groupe et permissions sont indissociables
- Piliers de la sécurité système

Stockage objet : métadonnées systèmes

Préparation aux travaux pratiques

À l'aide de la commande "ls", rechercher et afficher les permissions d'un "objet" du type :

- Programme exécutable
- Lien symbolique
- Périphérique accessible en mode caractère
- Périphérique accessible en mode bloc
- Socket de la famille Unix

Quels sont les traitements assurés par les commandes suivantes ?

- `chmod`
- `chown`
- `chgrp`
- `setfacl`
- `umask`

Initialisation système

Initialisation système – les étapes

POST + UEFI/BIOS	Power On Self Test Unified Extensible Firmware Interface Basic Input Output System	Test des composants à la mise sous tension
MBR/GPT	Master Boot Record Grand Partition Table	Unité de stockage d'initialisation
GRUB	Grand Unified Bootloader	Gestionnaire d'amorçage
Kernel	Linux	Pilotage de l'architecture matérielle
Systemd	System and service manager	Pilotage de l'initialisation
Target	Target unit configuration	État attendu du système après initialisation

Initialisation système – POST

POST = *Power On Self Test*

- Premiers codes lancés à la mise sous tension
- Tests de fonctionnement des composants et détection de pannes
- Initialisation de tous les micrologiciels (*firmwares)* des sous-systèmes de l'architecture matérielle : *"binary blobs"*
  - Processeurs
  - Interfaces réseau : filaire ou Wifi
  - Périphériques de stockage
  - GPU = Graphical Processor Unit
  - NPU = Neural Processor Unit

Tendance : multiplication des micrologiciels assimilables à des systèmes complets

Initialisation système – BIOS/UEFI

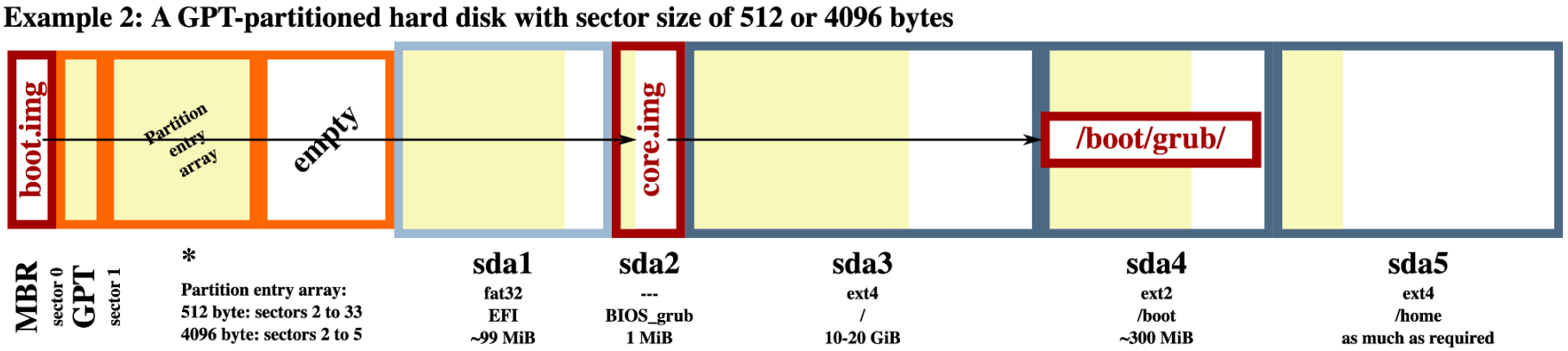
BIOS = *Basic Input Output System*

- IBM PC 1980
- Service “d’abstraction du matériel” à partir de zones mémoire prédéfinie
- Définition de l’ordre des périphériques d’amorçage
  UEFI = *Unified Extensible Firmware Interface*
- Partitions *Master Boot Record* (MBR) et *Grand Partition Table* (GPT)
- Shell + système de fichiers
- Service réseau standard : Preboot Execution Environment (PXE)
- *SecureBoot* : empêche le chargement de noyau et de pilotes non signés

Initialisation système – GRUB

GRUB = *GRand Unified Bootloader*

- *Bootloader* = Gestion de l'amorçage
- Sélection du noyau
- Identification des partitions et des système de fichiers

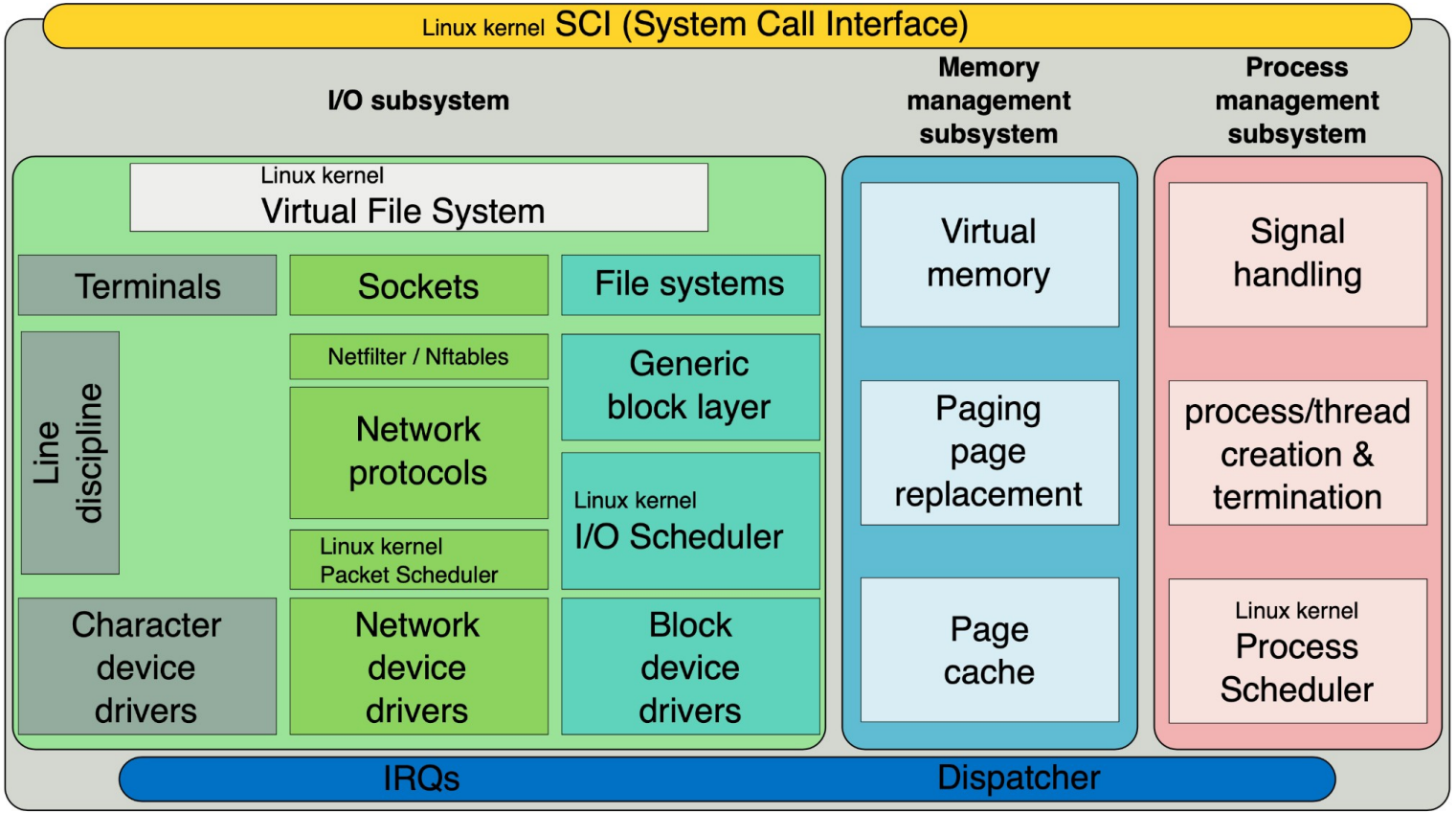


Initialisation système – Linux Kernel

Séquence d'initialisation

1. Détection architecture matérielle
   ARM / AMD64 / RISC-V
2. Organisation de la mémoire virtuelle
   Pagination / Contraintes d'ordre (Memory barrier)
3. Identification des ressources de l'ordonnanceur
   Horloges / Interruptions
4. Application des paramètres de la ligne de commande
   Configuration gestion d'amorçage
5. Ouverture du disque RAM initial (*initramfs*)
   Boîte à outils BusyBox / Modules propres au matériel
6. Montage des systèmes de fichiers
   Filesystem Hierarchy Standard
7. Lancement du processus init

Initialisation système – Linux Kernel

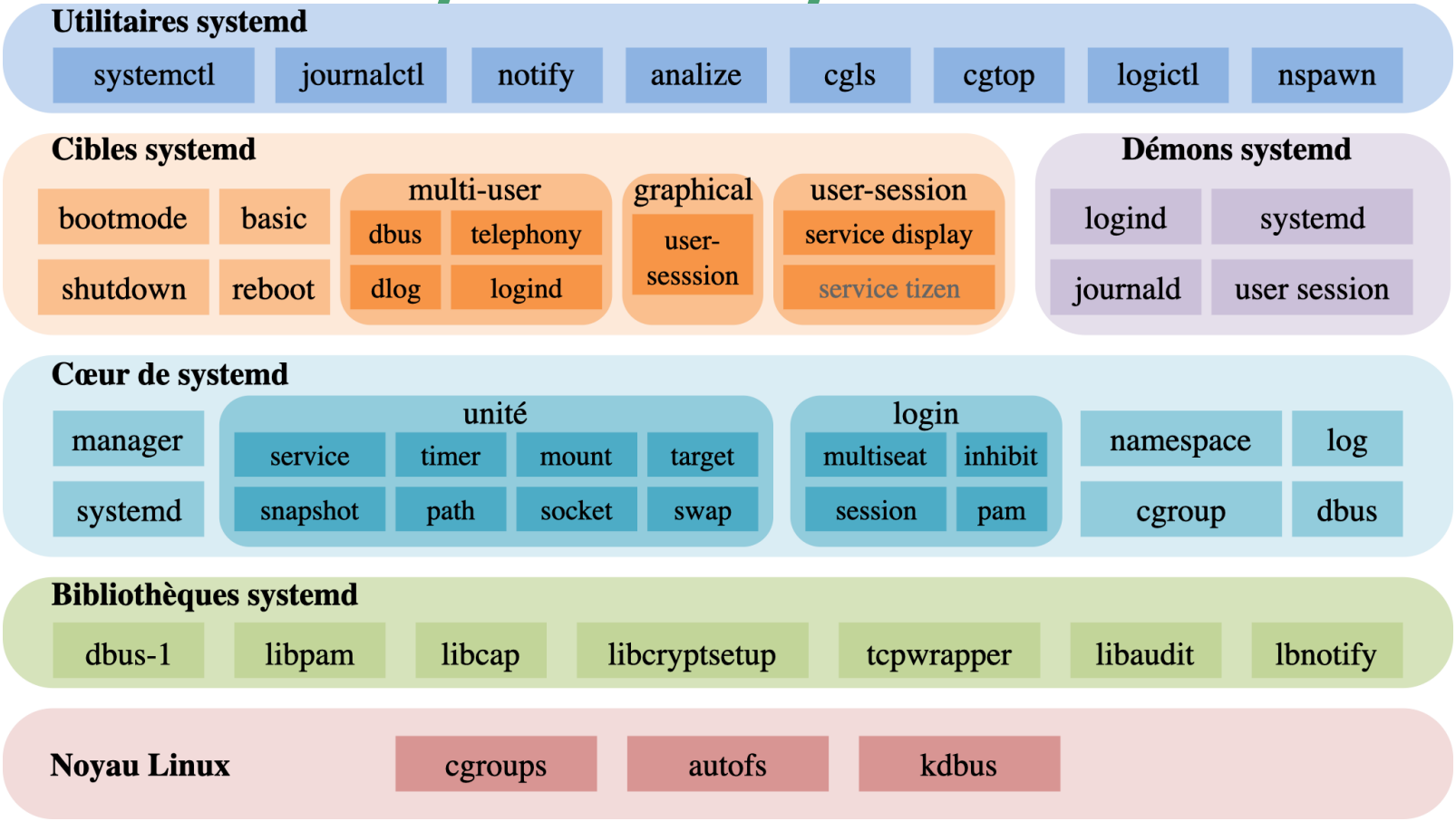


Initialisation système – Systemd

Gestion des services et des processus dans l'espace utilisateur

- Parallélisme
  - Initialisation rapide des services utilisateur
- Gestion dynamique des périphériques matériel
  - udev / hotplug
- Gestion des contraintes unifiées
  - cgroups : % CPU | % mémoire | % bande passante réseau
- Découpage hiérarchique et groupes logiques
  - Arborescence système != Arborescence utilisateur
- Remplacement de services "historiques"
  - Planification : cron ➟ systemd timers
  - Journalisation : syslog ➟ journald

Initialisation système – Systemd



Bilan Initialisation Système

- Classer les étapes de l’initialisation dans le bon ordre
- Gestion de l’amorçage
  On Premises
- Noyau Linux
  Applications & événements matériels
- Systemd
  Socle de la gestion des services