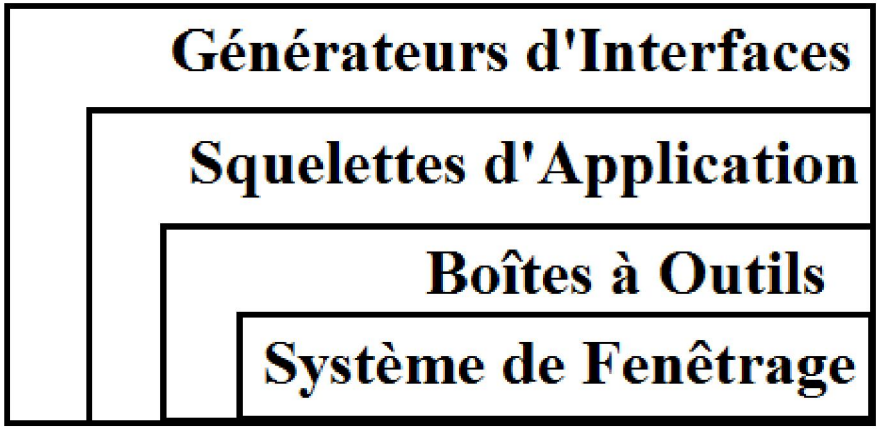
OUTILS DE GENERATION D'INTERFACES

SYSTÈME DE FENETRAGE

Outils de Génération d'Interfaces

- Mis en couches hiérarchiques pour produire des applications interactives. Implantés au-dessus du **système de fenêtrage**, ce sont:

  - Boîtes à outils (ToolBox)
  - Systèmes génériques (squelettes d'application)
  - Générateurs d'Interfaces



Interfaces textuelles et graphiques

Interfaces textuelles

- unicité de la source: stdin
- unicité de la sortie: stdout,stderr
- granularité (unité d'échange) élevée:
  - ligne entière en entrée
  - message en sortie
- aide laborieuse, explicite

Interfaces graphiques

- multiplicité des sources : clavier, souris, avec interprétation selon le lieu:
  - fenêtre
  - menu, barre de bouton
  - raccourci, équivalent clavier
- multiplicité des sorties : texte, messages, dialogues, clignotements
- granularité fine:
  - l'évènement élémentaire (KeyPress, KeyRelease)
  - évènement symbolique (activate)
- aide implicite facilitée par
  - bulles d'aide
  - changement de curseur et/ou de couleur
  - réaction aux actions (cliignotement)

Les interfaces graphiques sont plus difficiles et plus longues à réaliser

Système de Fenêtrage

- Bibliothèque qui regroupe des fonctions facilitant la production d'interfaces graphiques
- On considère le Système **XWindow** développé au MIT (Athena 1984), devenu un standard sur les stations graphiques.

Système de Fenêtrage

- **XWindow** est indépendant :
  - → du matériel (CPU, écran, …)
  - → du système d'exploitation

- Disponible sur de nombreux systèmes:
  - → Unix et variantes; Windows NT, Server…
  - → Stations Sun, DEC, terminaux X…

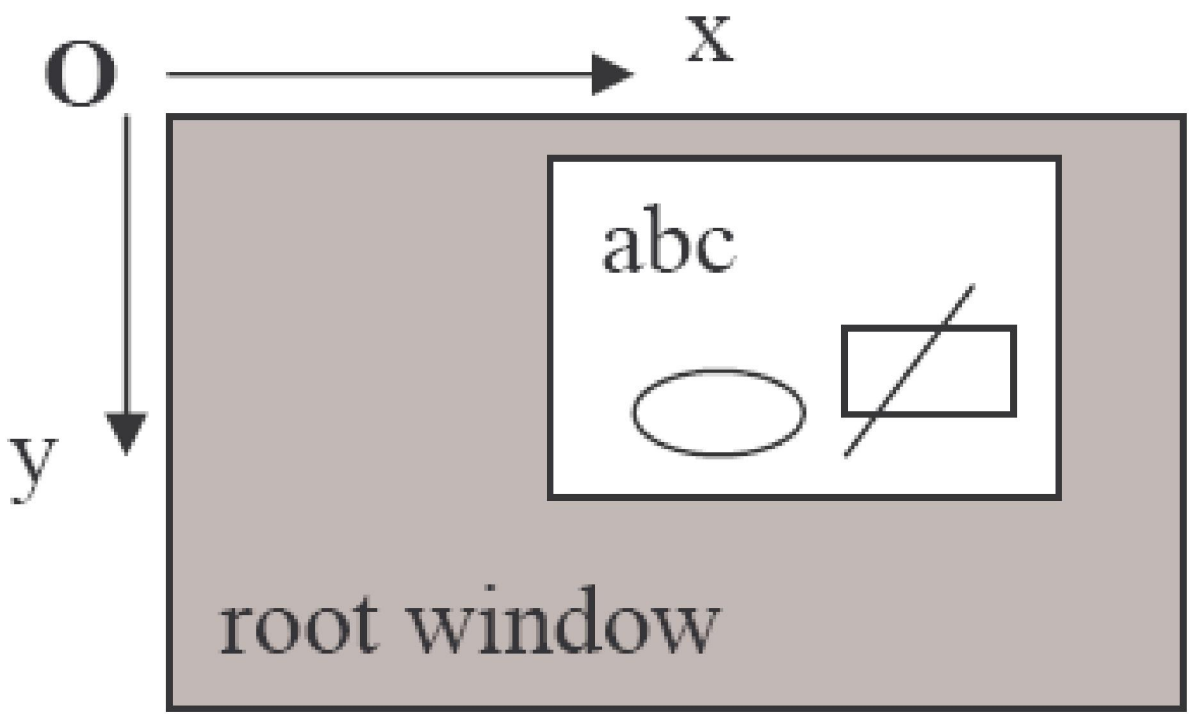
I/ Caractéristiques Générales

- Couche au-dessus du système d'exploitation
- Gestion de l'affichage en mode graphique
  - Fenêtres, dessins, textes, images
  - Gestion des entrées (clavier, souris)
- Utilisation en réseau transparente et optimisée
- Modèle Client/serveur
- Une application peut ouvrir N fenêtres, pas forcément sur la même machine

II/ FENETRES

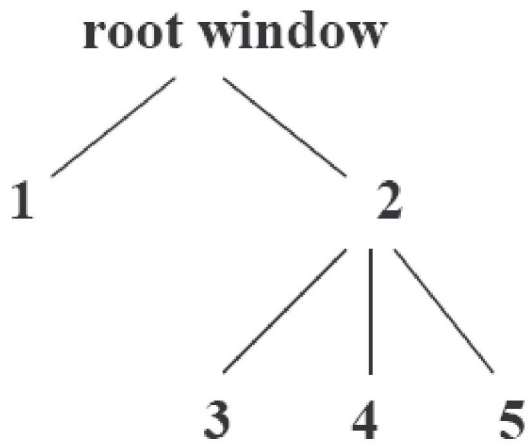
- Zones d'écrans virtuelles
- Contrôlées par le serveur X:
  - En entrée: des évènements X …
  - En sortie: affichage…
- **Espaces graphiques** avec des arguments:
  - Couleur de fond, de bordure
  - A quels événements la fenêtre va-t-elle réagir?
  - A Chaque fenêtre → **structure** stockée dans le serveur

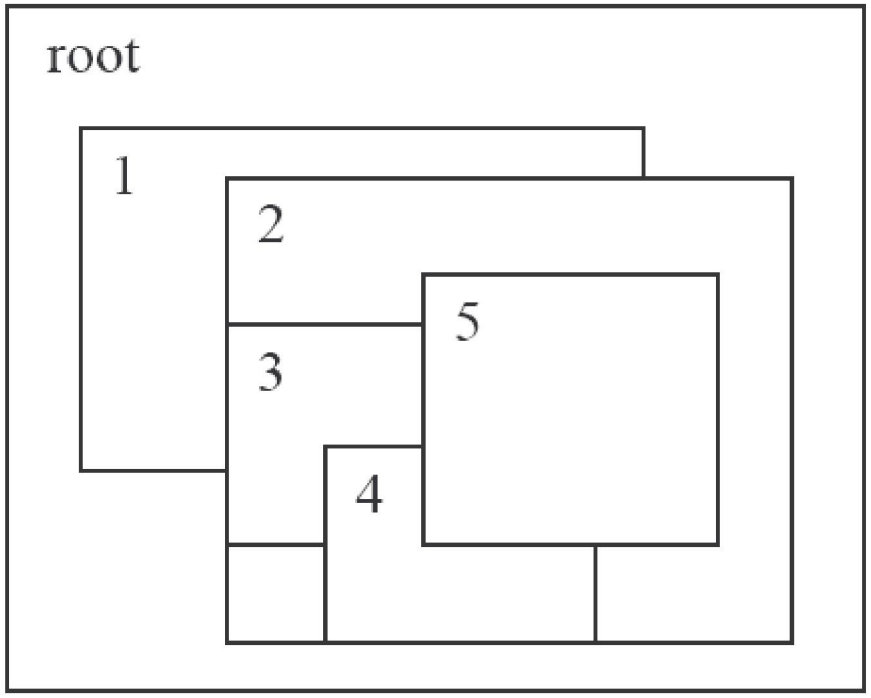
II/ FENETRES



II/ FENETRES

Structure arborescente





- Fenêtres enfant 3, 4, 5 ou *ChildWindow (windows)*
- Hiérarchie des fenêtres: système de coordonnées relatives

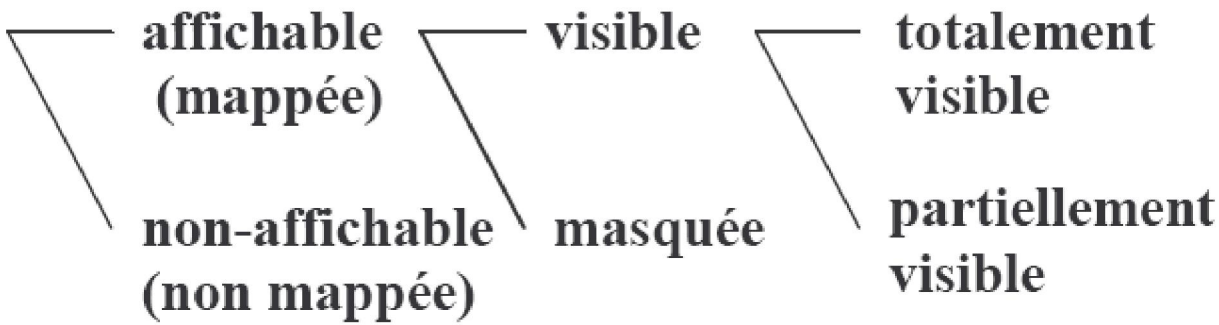
II/ FENETRES

- Une fenêtre principale **"root window"** (créée lors de l'initialisation du serveur) couvre la totalité de l'écran

- **"Window manager"** est un **client** particulier. Il s'occupe des différentes fenêtres de la station et permet de les déplacer, iconifier, retailler ..

II/ FENETRES

Etat d'une fenêtre



III/ MODELE CLIENT/SERVEUR

- **Objectif** : Fonctionnement en réseau → afficher sur l'écran d'une même station des applications tournant sur différentes machines.

- **Serveur X** : programme qui gère **l'écran** d'affichage, le **clavier** et la **souris**. Intermédiaire entre:
  - Les clients (applications graphiques)
  - Les ressources de la machine locale (clavier, souris, écran(s) …)

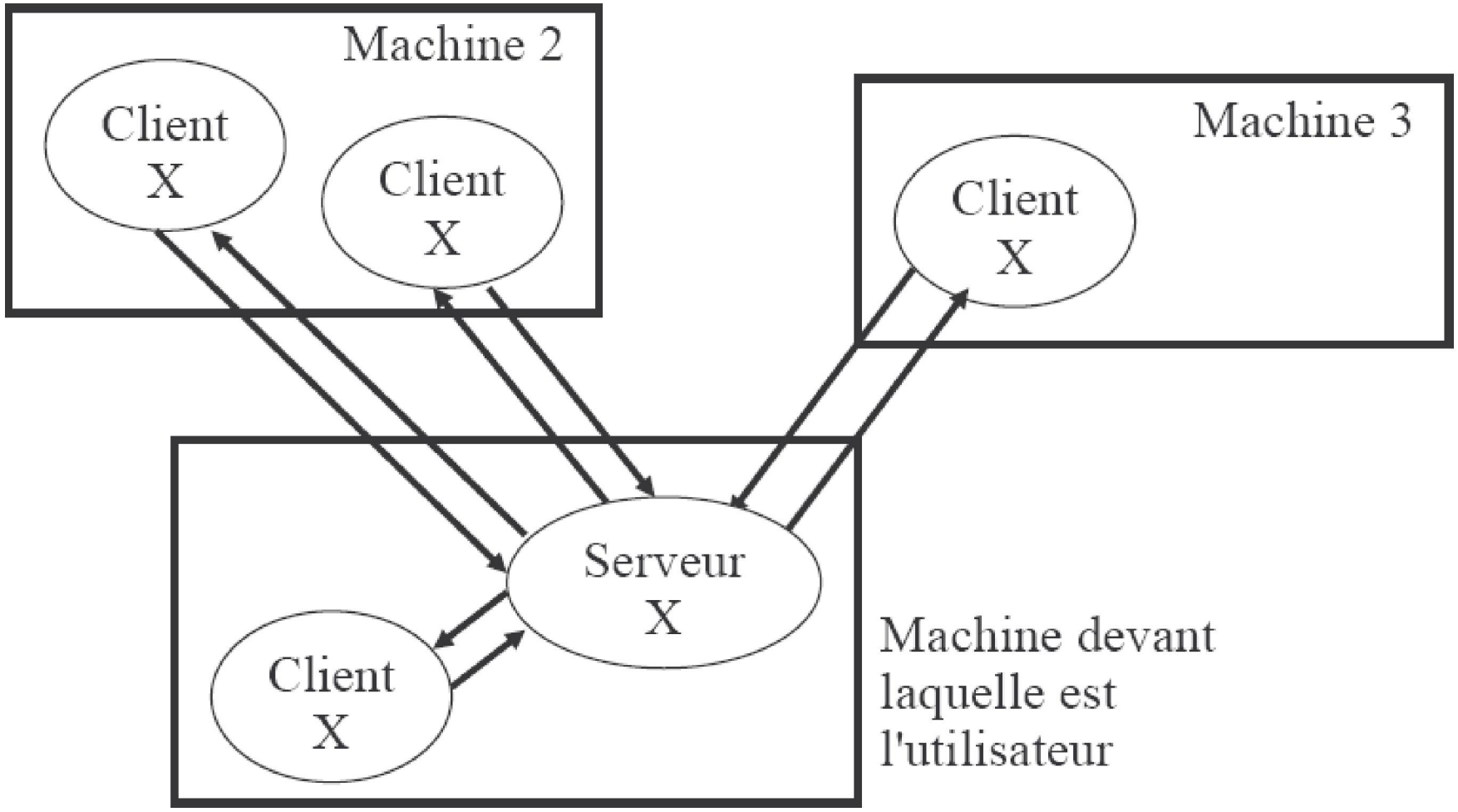
.

III/ MODELE CLIENT/SERVEUR

- **Client X**: programme application (*comme la calculatrice, horloge, bloc-notes*…) qui peut s'exécuter sur la même machine que le serveur ou ailleurs, et communique avec le serveur par le protocole (**XProtocol**)

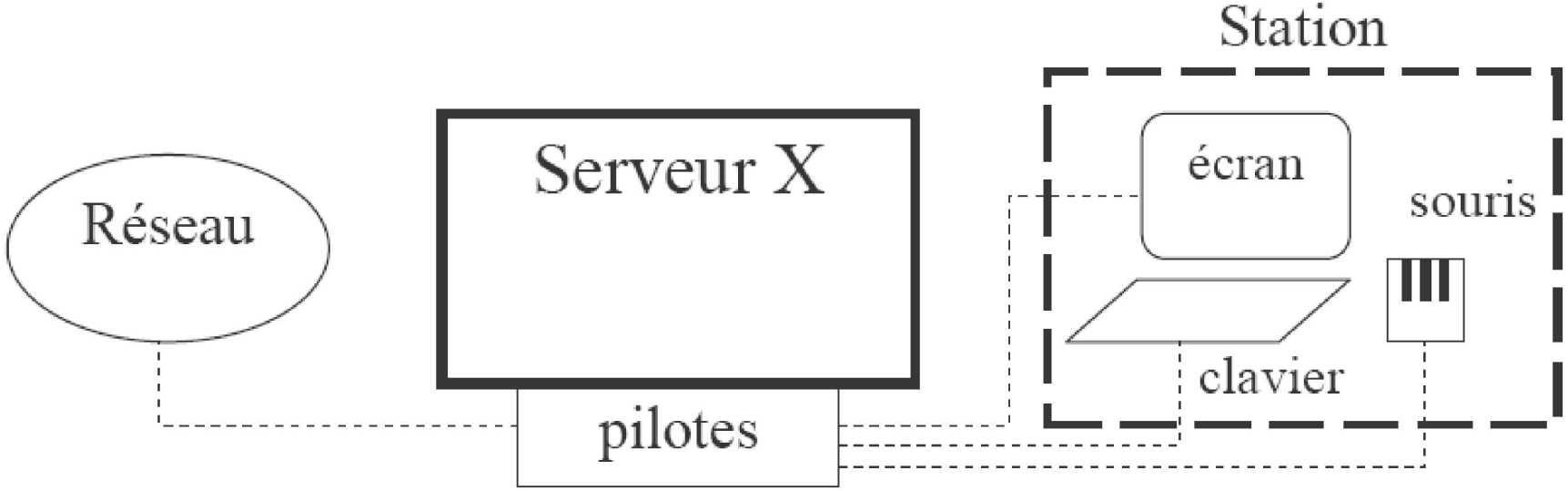
- Un **Serveur** peut être accédé par **plusieurs clients** et un **client** peut accéder à **plusieurs serveurs**

III/ MODELE CLIENT/SERVEUR



III/ MODELE CLIENT/SERVEUR

- Composantes:
  - Pilote d'écran
  - Pilotes de clavier et de souris
  - Pilote de communication (en réseau)



IV/ DISPLAYS ET ECRANS

- Un **display** correspond à un **serveur X** ou au canal de communication menant à ce serveur→ plusieurs display(s) sont sur une machine

- Un **display** peut gérer plusieurs écrans

  adressage: **nom_host (adresse IP) : n°_display.n°_écran**

  Ex : **fermi**:**0.0**, **192.70.50.2**:**0.1**

- En général **un seul display et un seul écran**

IV/ DISPLAYS ET ECRANS

Un ***Display*** comprend :

- une unité centrale,
- un ou plusieurs écrans(*screens*),
- un clavier,
- un dispositif de pointage (souris),
- un protocole de communication

V/ EVENEMENTS

Un **évènement** est engendré par le serveur pour indiquer un *changement* :

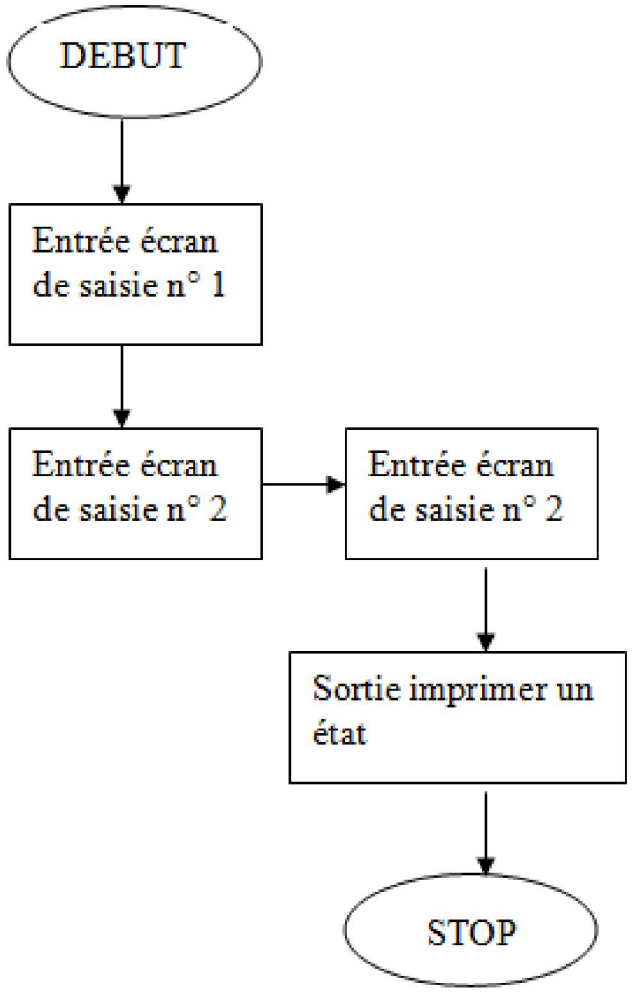
- entrée du clavier
- clic de la souris
- changement d'état dans une fenêtre

→ L'envoi des évènements est ***asynchrone***. Il se fait dès que possible.

→ Le déroulement synchrone est très coûteux, utilisé seulement lors du débogage

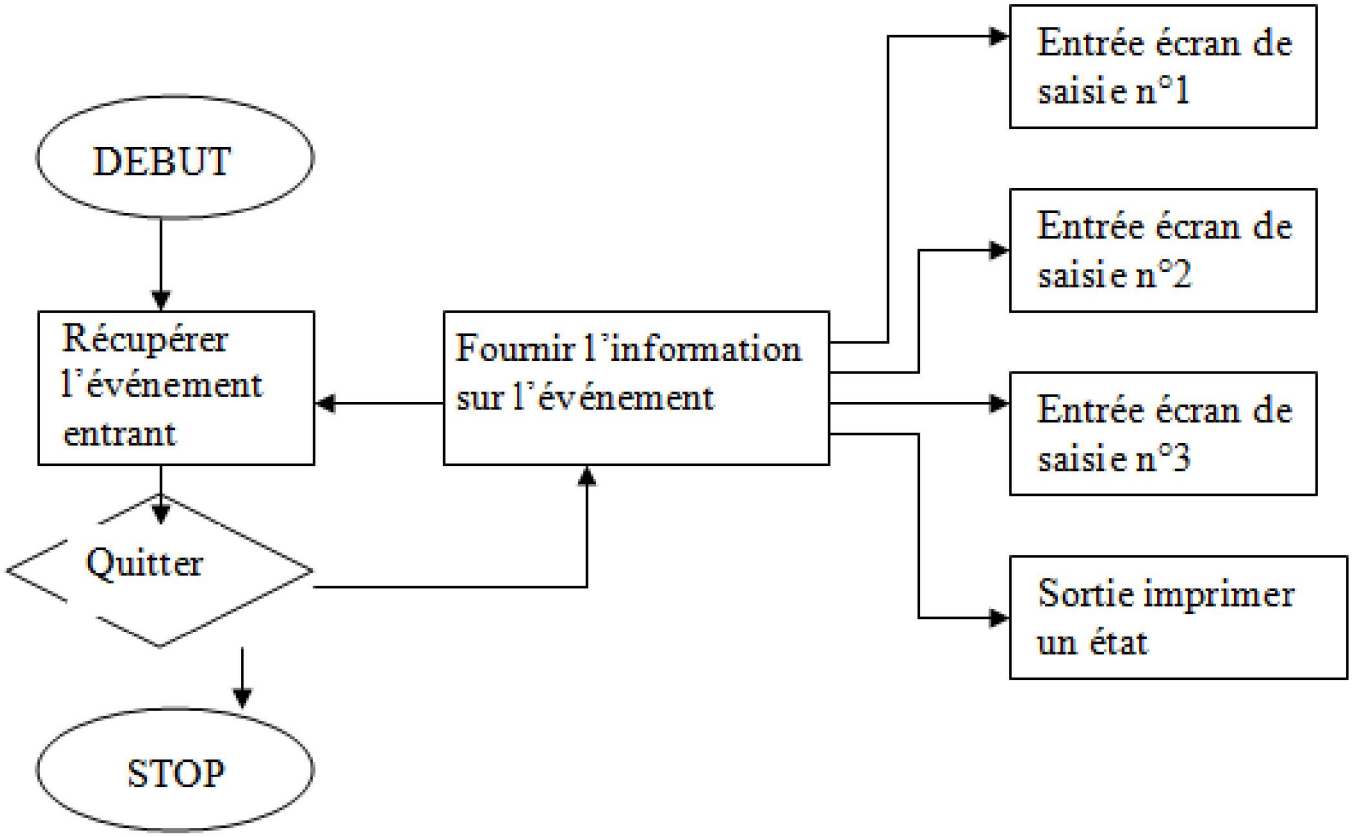
PROGRAMME CLASSIQUE (exemple)

- Organigramme représentant un programme utilisé par les agents de voyages pour délivrer des billets d'avion (3 écrans de saisie et une impression)



PROGRAMME EVENEMENTIEL (exple)

- Programme piloté par événement:



PROGRAMME CLASSIQUE PROCEDURAL (non EVENEMENTIEL)

- Le déroulement se fait par une séquence d'instructions. L'application a le **contrôle**.
- L'utilisateur **au service** de l'application
- Structure générale du programme:
  - **Programme principal**
  - Initialisations
  - **Lire et traîter commande 1 (ou Ecran 1)**
  - Lire et traîter **commande 2 (ou Ecran 2)**
  - … Lire et traîter **commande i (ou Ecran i)**
  - **Fin du Programme**

PROGRAMME EVENEMENTIEL

- L'utilisateur **garde** le contrôle. Application **esclave** de l'utilisateur
- **Boucle principale** de gestion des **événements** (parfois enfouie dans la librairie):
- **Programme Principal**
- Initialisations (fenêtre, objets graphiques…)

while (! fin) {

attendre événement suivant **E**

traiter événement **E** **}**

- **Fin du Programme**

BOUCLES D'EVENEMENTS

Au niveau du serveur X : **boucle infinie** qui :

- Détecte les actions graphiques (clic souris, touche du clavier, …) sous forme de requêtes des clients
- Envoie les événements **XEvents** aux clients concernés

Au niveau du client X : **boucle infinie** qui :

- Récupère ces événements **XEvents**
- Appelle les fonctions correspondantes

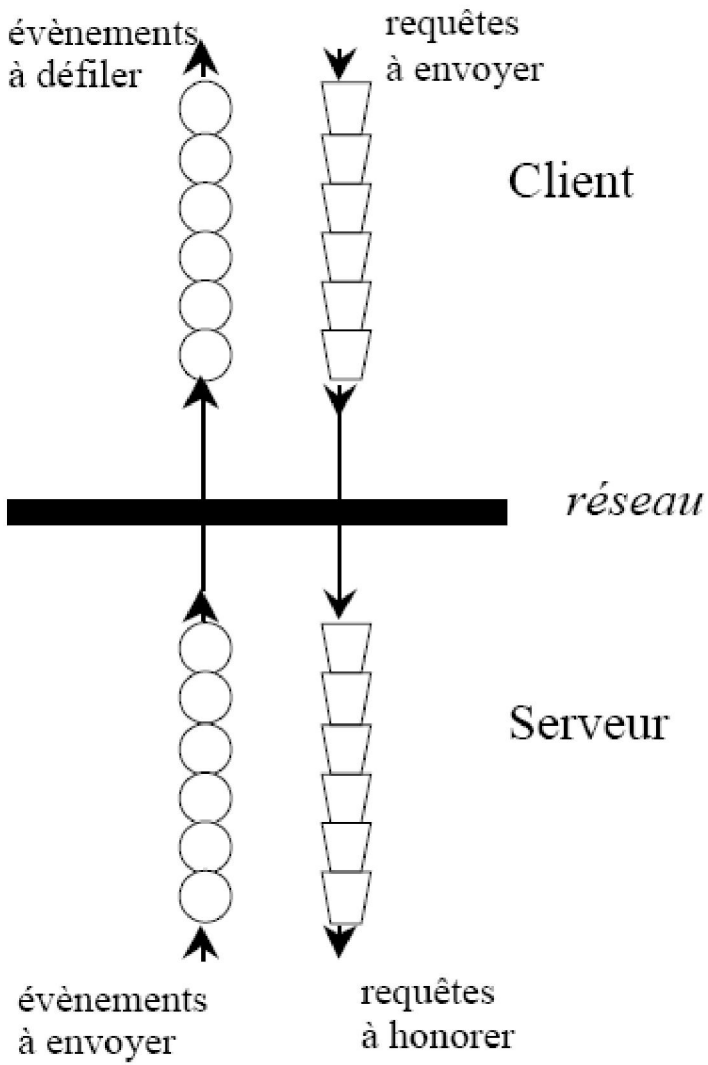
FILES DES EVENEMENTS

Les Requêtes d'un client sont:

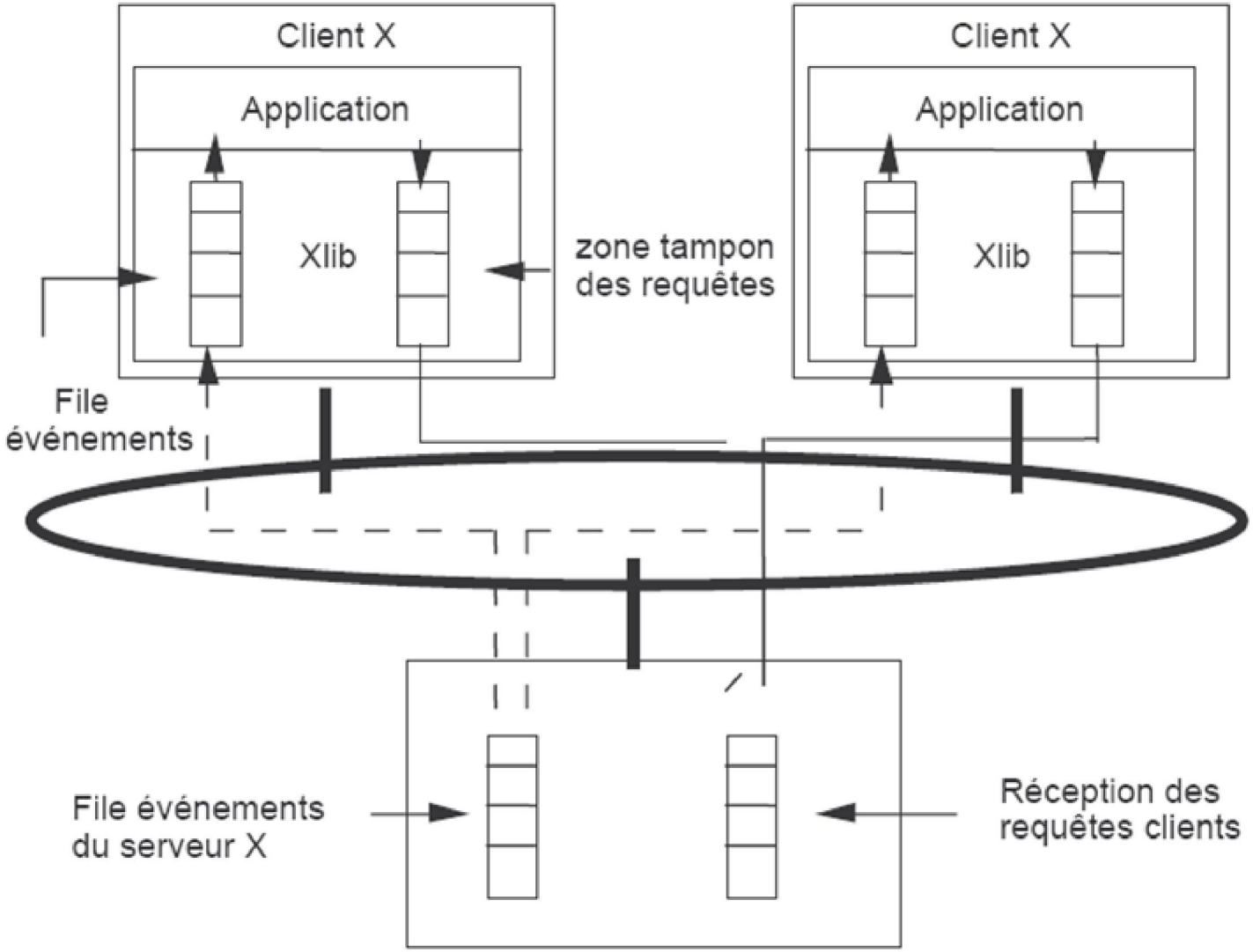
- mises en **file d'attente**, par chaque **application**, puis envoyées au **serveur**;
- à la réception par le serveur, les requêtes sont mises en **file d'attente** puis traitées dans l'ordre;

Les évènements sont:

- mis en **file** par le serveur, pour chaque client *intéressé*
- Puis envoyés au client qui les lit dans une **file**



FILES DES EVENEMENTS



V/ EVENEMENTS

- Les clients lisent les évènements qui les concernent par

  XNextEvent (dpy, &evmt);

- Les clients peuvent forcer l'envoi des requêtes par

  Xflush (dpy);

- mais ne peuvent pas forcer leur exécution

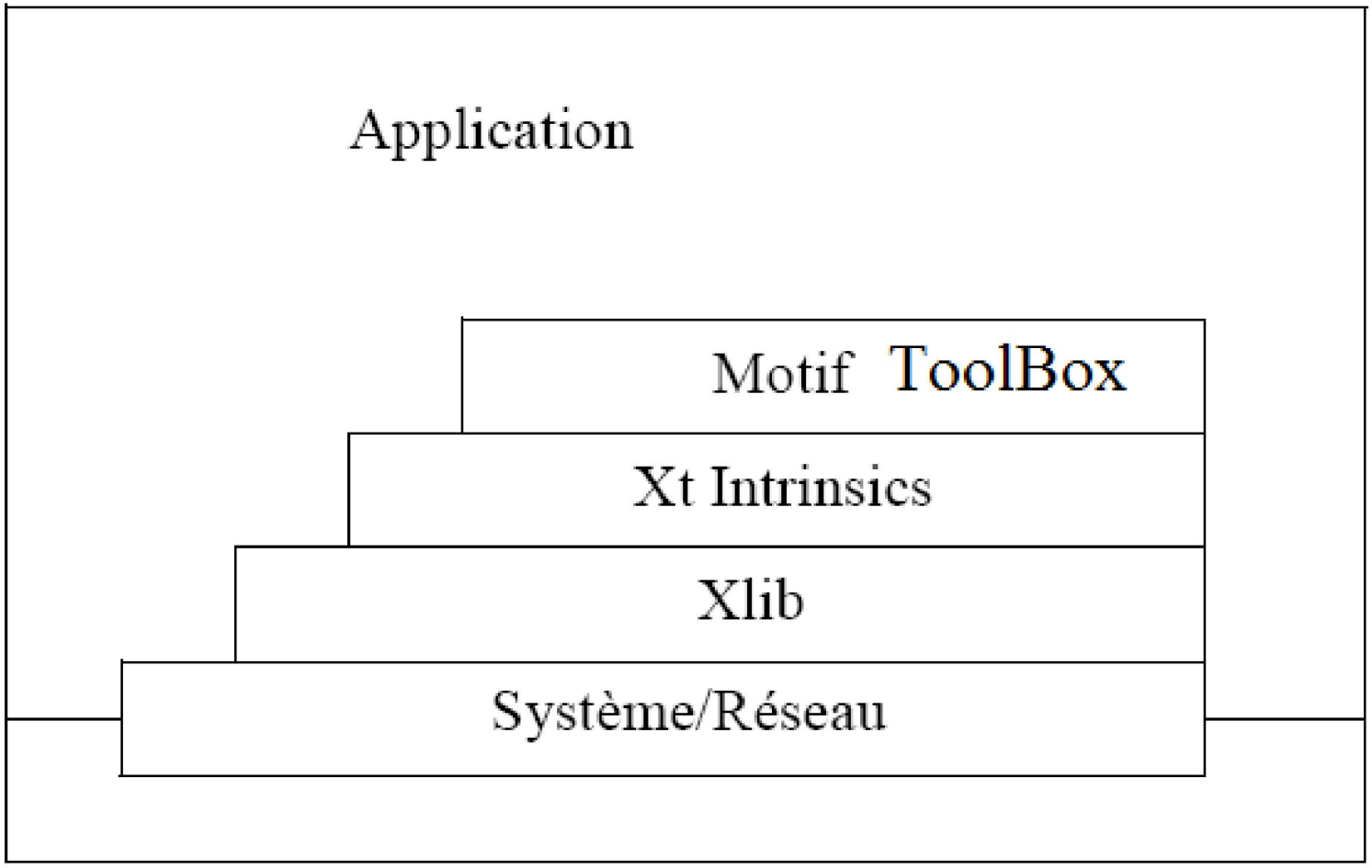
VI/ Structure d'un programme XWindow

Un prog. XWindow comporte 3 parties :

- Etablissement de la **connexion au serveur**.
- **Création** de la hiérarchie des **fenêtres** (resp. des widgets)
- Gestion de la **boucle d'évènements** :
  - Lire un évènement (***read***);
  - En fonction de la nature de l'évènement et la fenêtre où il a lieu, entreprendre l'action appropriée (***eval***).
  - envoyer les requêtes correspondantes au serveur *(print)*

VII/ Bibliothèques de XWindow

X est bâti sur un système de couches.



BIBLIOTHEQUE XLIB

- La communication entre les clients et le serveur avec le **protocole «X»** est efficace mais peu pratique d'emploi

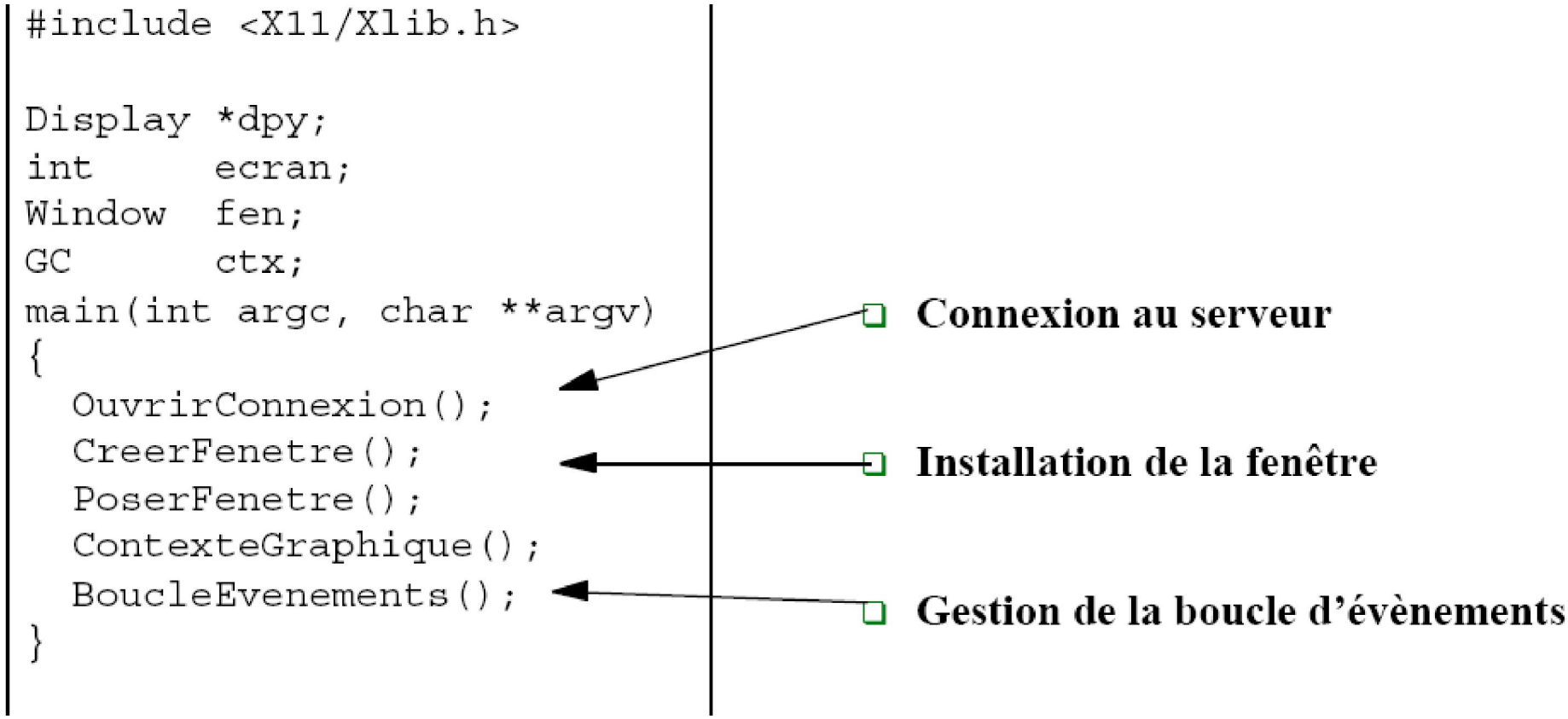
- La bibliothèque **Xlib** est une **interface au-dessus** du protocole «X» et permet de communiquer avec le serveur de manière plus simple. Elle assure la gestion :

- des fenêtres, fontes
- des contextes graphiques
- des événements
- des régions, images et couleurs

Exemple de programme Xlib

```
#include <X11/Xlib.h>

Display *dpy;
int      ecran;
Window   fen;
GC       ctx;
main(int argc, char **argv)
{
  OuvrirConnexion();
  CreerFenetre();
  PoserFenetre();
  ContexteGraphique();
  BoucleEvenements();
}
```



BIBLIOTHEQUE XT (INTRINSICS)

Xt (appelée «Intrinsics ») propose des fonctions spécialisées dans la construction d'interfaces graphiques au-dessus de XWindow.

Permet la construction facile d'une large variété de **widgets** (**boutons, zones de texte, listes** …)

Xt → orientée objet

VIII/ CONTEXTE GRAPHIQUE

- **Xlib** offre des fonctions de dessin graphiques
- **Trousse de dessin** (canevas): utilisée pour transmettre les paramètres graphiques à ces fonctions (couleur, épaisseur de trait, police, style de remplissage, etc…)
- **Contexte Graphique (GC)** = identificateur de trousse de dessin.
- Sous **Windows**, c'est le Device Context (DC)

VIII/ CONTEXTE GRAPHIQUE

- On peut créer plusieurs **GC** dans un client et indiquer à quel tracé il est destiné
- **Indépendant du périphérique**: le dessin graphique peut être envoyé vers **l'imprimante**, ou vers le **réseau**.

VIII/ CONTEXTE GRAPHIQUE

- **Création du contexte graphique:**

- GC **XCreateGC**(display, drawable, masque, AdrValeursGC)
- AdrValeursGC: adresse d'une structure qui contient les paramètres graphiques
- masque: identifie les paramètres utilisés

Exemple de Graphic Context

- GC gc;
- ...
- gc = XCreateGC (display, RootWindow (display, screen), 0, NULL);
- XSetForeground (display, gc, BlackPixel (display, screen));
- XSetLineWidth (display, gc, 4);
- ...
- /* on peut désormais utiliser ce GC */

Exemple d'utilisation (G.C.)

La structure `XGCValues`

Les principaux champs :

- `foreground` : couleur du trait
- `background` : couleur de fond
- `font` : police de caractères
- `line_width` : épaisseur de trait
- `line_style` : style de trait
- `fill_style` : style de remplissage d'une zone
- `function` : fonction de dessin