

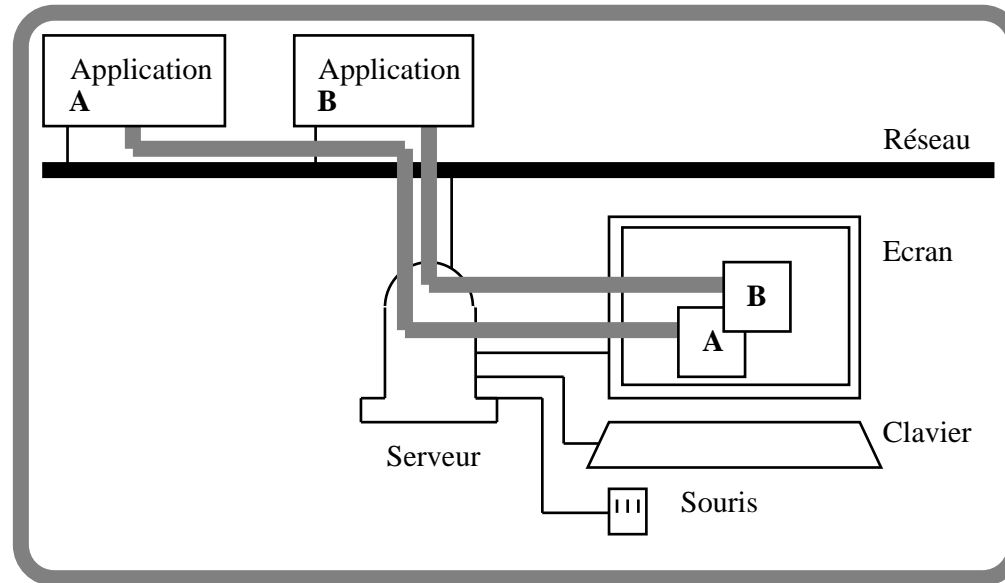
**X-I**

# Introduction au systeme X

- o **Présentation générale**
- o **Serveurs et clients**
- o **La programmation**

X-I

## Présentation générale



- Fondé sur le modèle *client-serveur*.
- Comporte 3 composantes :
  - | Le *serveur* - programme qui contrôle le dispositif d'affichage (display) : écran, clavier, souris.
  - | Les *clients* - programmes d'application.
  - | Le dispositif de *communication* entre clients et serveur.

X-I

## Le serveur

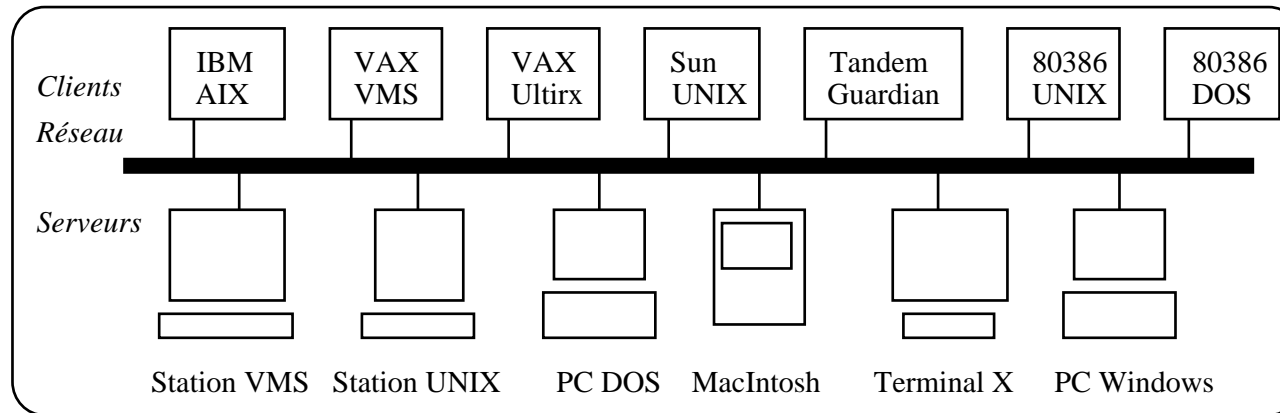
- Contrôle le dispositif d'affichage (display)
- Accepte des *requêtes* émises par les clients, via le canal de communication :
  - | créer une fenêtre;
  - | changer taille et position;
  - | tracer texte et dessins.
- Emet des *événements* à destination des clients,
  - | indiquant les entrées sur clavier et souris;
  - | notant changement dans l'état des fenêtres.

## Le client

- En plus de ses fonctionnalités propres, contient les instructions X nécessaires
  - | pour formuler les requêtes;
  - | pour analyser les événements et en tenir compte.
- Est indépendant et distinct du serveur, et peut se connecter à tout display.

## X-I

## La communication



**Clients et serveurs sont logiquement séparés.**

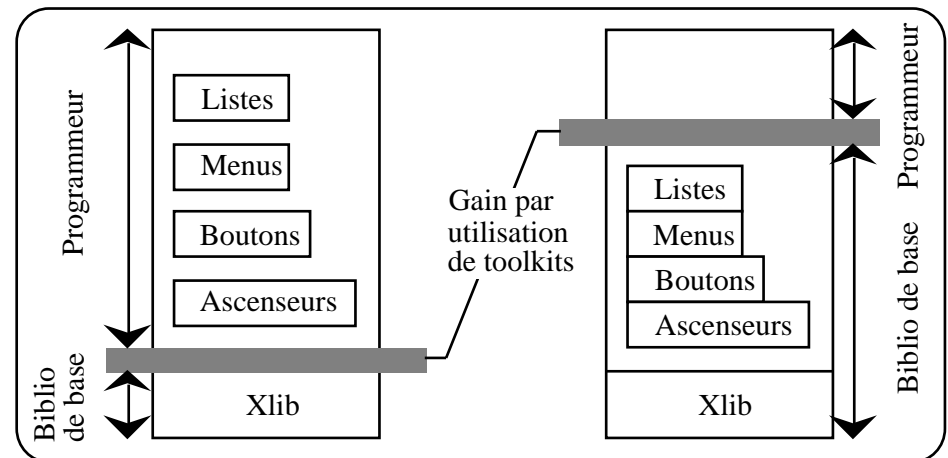
- **Sur une *même machine*, communication inter-processus.**
- **Sur *machines distinctes*, communication par réseau**
  - | **indépendant du réseau;**
  - | **indépendant de l'architecture.**
- **Le fonctionnement de X est *transparent* au réseau.**
- **Des clients sur des machines d'architectures différentes simultanément sur le même serveur**
- **Un même client simultanément sur plusieurs serveurs**

## X-I

## GUI, API, et les autres

### Terminologie

- **GUI (Graphical User Interfaces) ou *interface graphique utilisateur* : c'est l'interface graphique, telle qu'elle se présente à un utilisateur.**
- **API (Application Programmer's Interface) ou *interface du programmeur d'application* : c'est l'interface qu'utilise un programmeur pour créer des interfaces utilisateur.**
- **Interface Builders ou *créateurs d'interfaces* : ce sont des outils (graphiques en général) permettant au programmeur d'application de développer rapidement ses applications**
- **Une API évoluée fournit des composants d'interfaces à comportement intégré, comme boutons, menus, ascenseurs.**
- **Le niveau de l'API influe considérablement l'effort du programmeur**
- **Les outils spécifiques de création d'interfaces (composant et fonction) sont des *toolkits* (boîtes à outils).**



X-I

## Interfaces du programmeur d'applications

- Les composants d'interfaces, comme menus, ascenseurs, boutons ne font pas partie du système X de *base* (Xlib).
- X ne fournit pas d'API en standard, mais des *mécanismes*, et tous les outils nécessaires pour créer des interfaces.
- Ces interfaces se construisent donc *au-dessus* de X.
  
- Les principales boîte à outils au-dessus de X sont
  - | Athena Widget Set, fourni par le MIT avec la distribution de X.
  - | Open Look (Sun)
  - | Motif (OSF), la plus répandue des interfaces commerciales.
- Chaque interface définit son propre "*look and feel*", mais toute application X construite sur une interface tourne sur tout serveur. Ainsi, le système X n'a pas de guide de style.
- Ne pas confondre les deux composantes intervenant dans les applications:
  - | L'*interface de l'application* : développée pour l'application particulière.
  - | Un *gestionnaire de fenêtres* (Window manager) : gère les fenêtres sur l'écran.

## Autres interfaces

- o **Macintosh**
  - | **interface intégrée au système.**
  - | **ne supporte que des applications locales.**
- o **Microsoft Windows**
  - | **interface intégrée, au dessus du système.**
  - | **ne supporte que des applications locales.**
- o **De nombreuses API sont développées au dessus de Xlib, mais sans utiliser les Intrinsics**
  - | **Java**
  - | **Tcl/Tk**
  - | **Ilog Views**

X-I

## Discussion

- **Avantages de tout système de fenêtrage**
  - | manipulation *directe* (par boutons, ascenseurs, déplacements);
  - | menus et dialogues remplacent langage de commande.
- **Avantages spécifiques à X**
  - | plusieurs applications exécutables simultanément;
  - | copier - coller et communication entre applications.
- **Applications indépendantes du matériel:**
  - | Toute machine possédant un serveur X peut exécuter toute application X.
  - | Inversement, X possède un émulateur de terminaux (*xterm*) utile pour UNIX et les programmes "classiques".
- **Transparence et indépendance du réseau.**
- **Serveur indépendant du matériel : existe sur PC, Mac,....**
- **Extensions prévues et réalisées (PEX- Phigs sous X).**
- **Pas de réglementation ("Style Guide").**
- **X est gratuit et disponible.**

**X-I**

## Terminaux X

- **Un terminal X est composé**
  - | **du dispositif d'affichage (écran, clavier, souris),**
  - | **d'un support pour le serveur X;**
  - | **d'un support pour la communication.**
- **Il est simple à configurer, et simple à administrer.**
- **Il est bon marché.**

**X-I**

## Serveurs et clients

### Le serveur

- **Les fonctions du serveur**
- **Ce que le serveur ne fait pas**
- **Composition du serveur**
- **Gestion des sorties**
- **Gestion des entrées**

### Le client

- **Le client envoie des requêtes**
- **Le client reçoit des évènements**

## Les fonctions du serveur

- Traiter les *sorties graphiques* en exécution des requêtes :
  - | Création, configuration, destruction des *fenêtres*;
  - | Gestion des *polices* et des *textes*;
  - | Tracés *graphiques* (lignes, arcs, régions);
  - | Gestion des *images* (pixmap) et *curseurs* .
- Pour cela, le serveur dispose d'*informations* locales:
  - | ressources (au sens X);
  - | fenêtres, polices, curseurs, copies d'écran (pixmap);
  - | table des couleurs;
  - | contextes graphiques;
  - | propriétés, atomes, tampons.
- Traiter les *entrées* et les transmettre aux clients :
  - | détection des frappes de clavier;
  - | détection des mouvements de souris.
- Gérer les *communications* avec le réseau.

X-I

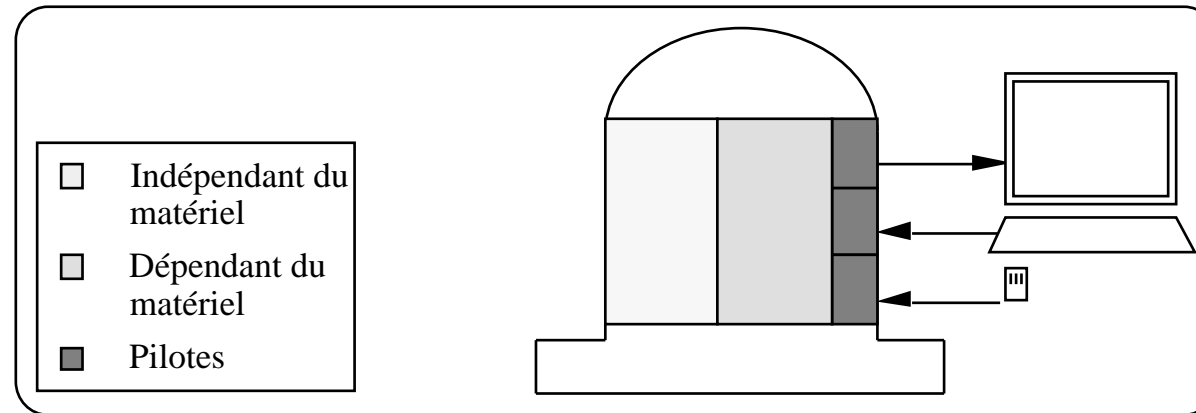
## Ce que le serveur ne fait pas

**Le serveur n'interprète d'aucune manière les évènements ou modifications intervenues :**

- o **Ne redessine pas le contenu d'une fenêtre redécouverte. En revanche, envoie un évènement *expose* ;**
- o **Ne fait pas de zoom;**
- o **Ne contient pas de code pour menus, ascenseurs, etc.**
- o **Ne fait pas de gestion logique des fenêtres (comme agrandissement, élastique, etc.) : c'est le rôle du *gestionnaire de fenêtres* ;**
- o **N'interprète pas les entrées au clavier;**
- o **N'interprète pas les mouvement de la souris.**

## X-I

## Composition du serveur

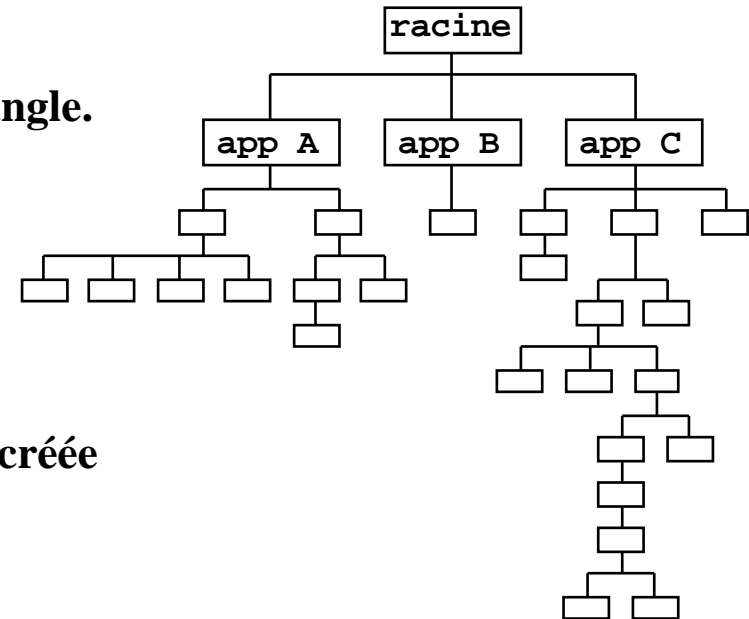


- **Le serveur est composé (au niveau logiciel), de trois couches:**
  - | **une partie indépendante du matériel;**
  - | **une partie dépendante du matériel (taille écran, couleurs, résolution);**
  - | **une partie qui communique avec les pilotes d'entrées et de sorties.**
- **Les clients sont indépendants du matériel, mais doivent pouvoir s'adapter en fonction des capacités (de l'écran, du clavier).**

## X-I

## Gestion des sorties

- L'objet de base est la *fenêtre* , essentiellement un rectangle.
- Les fenêtres d'un écran sont organisées en arbre :
  - | la fenêtre racine (*root window*) occupe l'écran;
  - | toute autre fenêtre a une *mère*;
  - | chaque application a une *fenêtre principale* et des sous-fenêtres.
- Les fenêtres ont des attributs et des états. Une fenêtre créée peut être affichée ou non, visible ou non, etc.
- Toute sortie graphique se fait dans une fenêtre.
- Le serveur contient ou charge les informations pertinentes :
  - | polices de caractères;
  - | curseurs
  - | pixmaps (images)
- C'est le serveur qui maintient les informations concernant les fenêtres.



X-I

## Textes et polices

**Pour écrire, il faut disposer d'une police.**

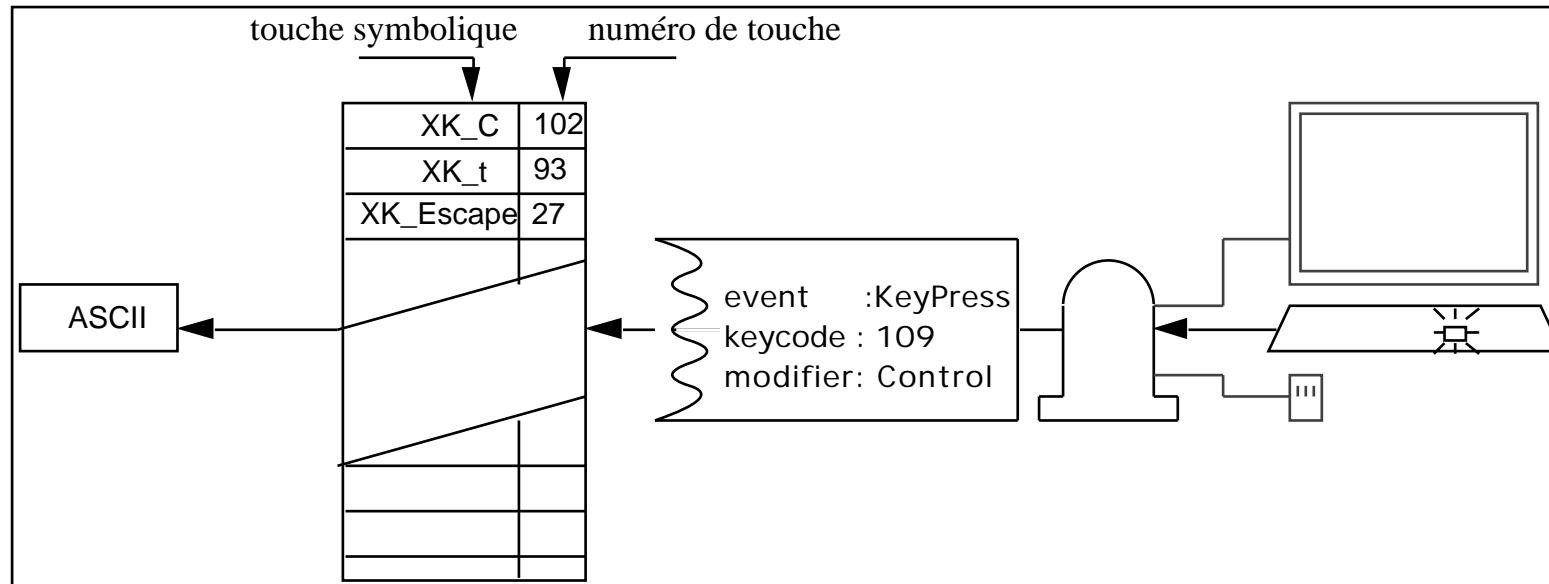
- **Les polices ont des noms, et sont stockées sous des formats variés.**
- **Jusqu'à la version 4, les polices sont stockées en bitmap sur le serveur.**  
**Conséquences:**
  - | **pas de transfert via le réseau;**
  - | **requiert de la place pour chaque taille d'une même police;**
  - | **formes des textes limitées.**
- **Depuis la version 5 (actuellement, on en est à la version 6),**
  - | **chaque police peut être stockée par contour, et convertie en bitmap quand c'est nécessaire;**
  - | **plus de choix dans les polices.**
  - | **un serveur de polices, chargé de la gestion des polices pour tout le réseau.**

## Bitmaps, pixmapes et curseurs

**Bitmaps, pixmapes et curseurs résident dans le serveur. Toutes les primitives de dessin s'appliquent aux fenêtres et aux pixmapes (*drawables*).**

- ***bitmap* : image décrite par un tableau de pixels.**
  - | chaque pixel représenté par un bit (noir et blanc)
  - | usage pour icônes, etc.
  - | création facilitée par le programme `bitmap`.
  - | représentation possible comme tableau de caractères.
- ***pixmap* :**
  - | les pixels sont représentés par plus de bits (donc couleurs ou niveaux de gris).
  - | permet de sauvegarder des parties d'écran.
- ***curseur* : chaque fenêtre peut avoir son curseur. Chaque curseur a**
  - | une forme, un masque
  - | un *hotspot* . (point de référence)

## X-I

Entrée clavier

Lorsque l'on enfonce (ou que l'on relâche) une touche sur le clavier

- le *serveur* engendre un évènement **KeyPress**, dans lequel sont consignés
  - | le numéro de la touche enfoncée (*keycode*);
  - | l'état des touches d'altération.
- le *client* peut transformer le numéro de la touche en une touche symbolique (*keysym*), au moyen d'une table de correspondance stockée sur le serveur, mais consultée par le client au moment de la connexion.
- le *client* obtient aussi facilement un équivalent ASCII de la touche symbolique, s'il existe.

X-I

## Les requêtes du client

**Le client envoie des requêtes au serveur.**

- **Les requêtes concernent :**
  - | **tracés de textes ou graphiques;**
  - | **création, affichage, gestion de fenêtres;**
  - | **demandes d'informations (tables, états);**
  - | **polices.**
- **Les requêtes sont formulées à travers les fonctions Xlib, en s'appuyant sur un protocole propre, le *protocole X*. La formulation est invisible à l'utilisateur.**
  - | **une fonction Xlib provoque une ou plusieurs requêtes;**
  - | **chaque requête est formulée dans un format propre, codée comme chaîne de caractères.**
- **Les Intrinsic et Widget Sets font appel à Xlib.**

X-I

## Transmission des requêtes

- Les requêtes sont transmises selon un protocole de réseau qui assure la séquentialité, comme TCP/IP. A l'intérieur d'une station, la communication interprocessus est utilisée.
- L'envoi des requêtes est *asynchrone* : on n'attend pas la réponse (en général).
- Certaines requêtes demandent une *réponse* . Dans ce cas, l'écriture est bloquante. Ce sont des demandes d'informations.
- Chaque client possède sa *file de requêtes* où sont rangées les requêtes en attente d'envoi: l'envoi se fait par nécessité.

X-I

## Les évènements

- Un évènement est engendré par le serveur pour indiquer un *changement* :
  - | entrée sur clavier
  - | entrée sur souris
  - | changement d'état dans une fenêtre.
- L'envoi des évènements est *asynchrone*. Elle se fait dès que possible.
- Le serveur *filtre* les évènements. En effet :
  - | chaque fenêtre est "sensibilisée" (par le client) à une famille d'évènements;
  - | le serveur n'envoie, aux clients, que les évènements auxquels le client s'est déclaré intéressé, pour chaque fenêtre.
  
- Il existe 33 évènements différents.
- Chaque évènement comporte des informations générales:
  - | le type de l'évènement;
  - | la fenêtre où il a eu lieu;
  - | l'heure, en millisecondes, où il a eu lieu.
- Chaque évènement comporte aussi des informations spécifiques:
  - | le numéro de touche enfoncée pour un `KeyPress`
  - | le bouton enfoncé pour un `ButtonPress`,...

X-I

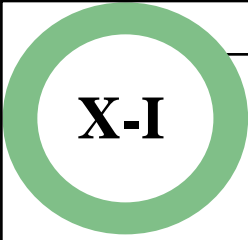
## Les files d'attente

- o **Les requêtes d'un client**
  - | **sont mises en file, par chaque application, puis envoyées au serveur;**
  - | **à la réception par le serveur, les requêtes sont mises en file d'attente puis traitées, à tour de rôle;**
  - | **elles sont traitées dans l'ordre, mais sans délai garanti (sauf pour les requêtes bloquantes)**
- o **Les évènements**
  - | **sont mis en file, par le serveur, pour chaque client *intéressé*;**
  - | **puis sont envoyés au client qui les lit dans une file.**
- o **Les clients lisent les évènements qui les concernent par**  

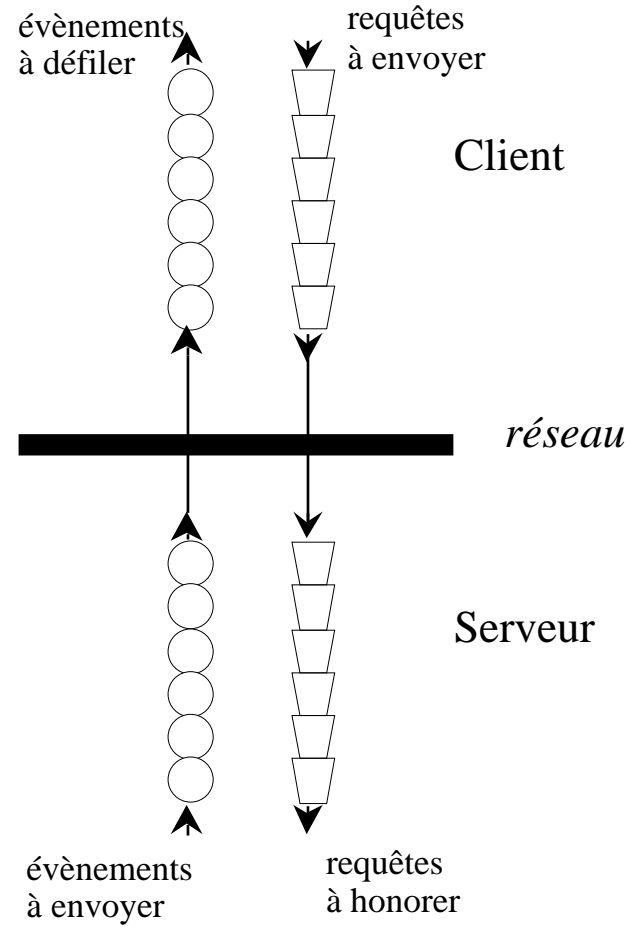
```
XNextEvent ( dpy , &evmt ) ;
```
- o **Les clients peuvent forcer l'envoi des requêtes par**  

```
XFlush ( dpy ) ;
```

**mais ne peuvent pas forcer leur exécution.**



X-I



**X-I**

## La programmation

- **Structure d'un programme X**
- **Les trois niveaux de programmation**
- **Programmation Xlib**
- **Programmation Intrinsics**
- **Programmation Toolkits**
- **Un exemple Xlib**
- **Un exemple Athena Widgets**

## Structure d'un programme X

**Un programme X comporte trois 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 de la fenêtre où il a eu lieu, entreprendre l'action appropriée (*eval* ).**
  - | **envoyer les requêtes correspondantes au serveur(*print* ).**

**La gestion des évènements est particulière parce que**

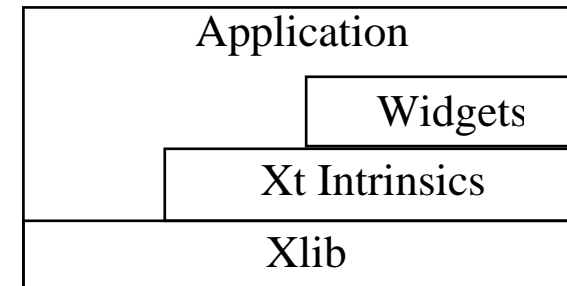
- | **l'ordre d'arrivée des évènements n'est pas séquentielle**
- | **il y a interaction avec d'autres clients, et notamment le gestionnaire de fenêtres**

## X-I

## Les niveaux de programmation

**X est structuré en trois niveaux :**

- **Le niveau *Xlib* : niveau de base, très flexible, nécessaire pour la gestion graphique.**
- **Le niveau *Xt* ou *Intrinsics* : fournit le concept de widget et les outils pour l'emploi d'ensembles de widgets. Ne contient que quelques widgets de base.**
- **Le niveau *Widget sets* . N'est pas fourni par le Consortium X (sauf Athena Widget Set).**
- ***X Toolkit* est le nom collectif le deux derniers niveaux : les *Intrinsics* + un ensemble de widgets.**
- **Il existe des ensembles de widgets qui ne sont pas basés sur les *Intrinsics* (*XView* par exemple).**



X-I

## Aspects spécifiques de la programmation Xlib

- o Gestion explicite de la boucle d'évènements.
- o Grande souplesse des primitives.
- o Gestion fine de tous les paramètres.
- o Grand courant d'échanges avec le serveur.

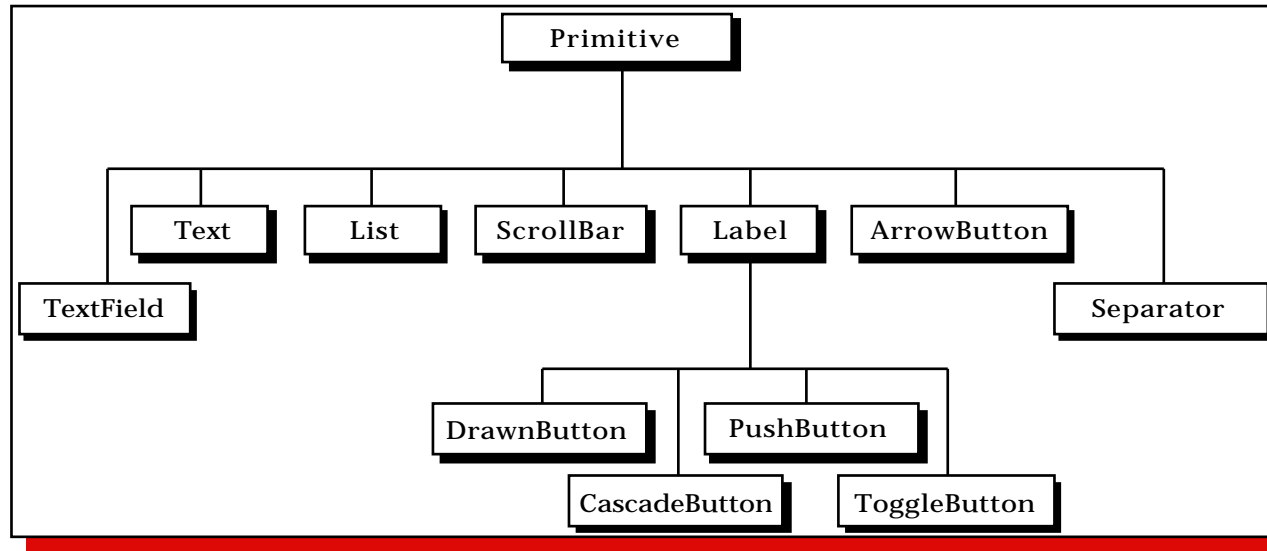
```
XEvent evmt;  
for(;;){  
    XNextEvent(dpy,&evmt);  
    switch (evmt.type) {  
        case ButtonPress :  
        case ButtonRelease :  
        case Expose :  
    }  
}
```

## X-I

## Aspects de la programmation Intrinsics

- Gestion intégrée de la boucle d'évènements : `XtMainLoop( ) ;`
- Introduction du concept de *widget* (fenêtre avec comportement prédéfini), avec trois familles de classes :
  - | les *widgets shell*, responsables de la communication avec le gestionnaire de fenêtres;
  - | les *widgets conteneurs*, responsable de la gestion géométrique de leurs filles;
  - | les *widgets d'affichage*, objets d'interface et de dialogue avec l'utilisateur.
- Trois concepts supplémentaires pour augmenter le niveau d'abstraction :
  - | les *actions* (translations) permettent d'associer des actions à des suites d'évènements;
  - | les *ressources* permettent de préciser les aspects d'une fenêtre;
  - | les *réflexes* (callback) sont des fonctions exécutées dans des situations précises (par des actions).
- Outils pour accéder de façon précise aux fonctionnalités Xlib.

## Aspects de la programmation par widgets



- Un ensemble de widgets fixe un “look and feel” :
  - | l’aspect visuel (“look”) est fixé;
  - | le comportement (“feel” : réaction aux boutons de la souris,...) est codifié.
- Contient de plus des widgets *composées* à fonctionnalités complexes. Par exemple, listes, FileSelectionBoxes, menus déroulants, boutons radio.
- Les widgets sont instances de *classes de widgets* qui respectent le mécanisme d’héritage (simple) des classes.
- Déplace la programmation par événements vers la programmation objets.

**X-I**

## Un 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();
}
```

o **Connexion au serveur**

o **Installation de la fenêtre**

o **Gestion de la boucle d'évènements**

## X-I

```
void CreerFenetre(void)
{
    fen=XCreateSimpleWindow(
        /* le display */ dpy,
        /* fenetre mere */ DefaultRootWindow(dpy),
        /* geometrie */ 30,40,500,300,
        /* largeur bord */ 6,
        /* couleur bord */ BlackPixel(dpy,ecran),
        /* couleur fond */ WhitePixel(dpy,ecran));
    XStoreName(dpy,fen,"Bonjour, monde !");
}
```

○ Création simplifiée  
d'une fenêtre

○ Numéro d'écran

```
void PoserFenetre(void)
{
    XMapWindow(dpy,fen);
}
```

○ Poser la fenêtre

○ Comment dessiner

```
void ContexteGraphique(void)
{
    ctx = DefaultGC(dpy,ecran);
}
```

X-I

```
void OuvrirConnexion(void)
{
    dpy=XOpenDisplay(0);
    ecran = DefaultScreen(dpy);
}
```

o Connexion au serveur

```
void BoucleEvenements(void)
{
    XEvent evmt;
    char * nom = "Hello, world !";
    XSelectInput(dpy, fen, ButtonPressMask);
    for(;;){
        XNextEvent(dpy, &evmt);
        if (evmt.type == ButtonPress) {
            if (evmt.xbutton.button == Button1)
                exit(0);
            XDrawString(dpy, fen, ctx, 100, 100, nom, strlen(nom));
        }
    }
}
```

o Evènement

o Sensibilisation de la  
fenêtre

o Lecture de l'évènement

o Actions.

## X-I

## Un exemple de programme (Athena Widgets)

```
#include <X11/Intrinsics.h>
#include <X11/StringDefs.h>
#include <X11/Xaw/Label.h>

main(int argc, char **argv)
{
    Widget racine, etiquette;

    racine = XtInitialize( argv[0], "XSimple",
        NULL, 0, &argc, argv);
    etiquette = XtVaCreateManagedWidget("Etiq",
        labelWidgetClass, racine,
        XtNlabel, "Hello, world !", NULL);

    XtRealizeWidget(racine);
    XtMainLoop();
}
```

widget = fenêtre + comportement prédéfini;

une widget s'occupe de l'interface

une widget de la classe "label" comme fille de racine;

création de l'arbre des widgets;

boucle principale.

## Livres en français

- **Sur le langage C**
  - | **J.-P. Braquelaire, *Méthodologie de la programmation en langage C*, Masson, 2e édition 1994**
- **Sur UNIX**
  - | **J.-M. Rifflet, *La programmation sous UNIX*, Ediscience, 3e édition 1993**
- **Sur Xlib**
  - | **C. Recanati, *X-WINDOW, manuel de programmation*, Eyrolles, 1993, 385 pages**
  - | **O. Jones, *Le système X WINDOW*, InterEditions 1992, 602 pages**
  - | **A. Janssens, *Système X Window, la bible*, Eyrolles 1993, 796 pages**
- **Sur Xlib, Xt et Open Look**
  - | **R. Stoeckel, *X Window, programmation*, Armand Colin, 1991, 486 pages**
- **Sur Xt et Motif, et aussi Xlib**
  - | **D. A. Young, *X Window, programmation avec les Xt Intrinsics*, Masson 1993, 494 pages**
- **Sur l'utilisation de X, principalement des gestionnaires de fenêtres.**
  - | **M. Lamboulé, *Objectif X Window*, Masson, 1995, 176 pages.**

## Livres en anglais

- **Sur X**
  - | N. Mansfield, *The Joy of X*, Addison-Wesley 1992, 368 pages (existe en français)
- **Référence sur Xlib**
  - | R. W. Scheifler, J. Gettys, *X Window System*, Digital Press, 3e édition 1992, 998 pages et aussi:
  - | C-Q. Yang, M. S. Ali, *Xlib by Example*, AP Professional, 1994, 635 pages
- **Référence sur Xt Intrinsics**
  - | P. J. Asente, R. R. Swick, *X Window System Toolkit*, Digital Press 1990, 967 pages
- **Sur Motif**
  - | D. L. McMinds, *Mastering OSF/Motif Widgets*, Addison-Wesley 1992, 660 pages
  - | T. Berlage, *OSF/Motif, Concepts and Programming*, Addison-Wesley, 1991, 487 pages
  - | *Motif/OSF, Programmers Reference Guide*
  - | D. A. Young, *Object-oriented programming with C++ and OSF/Motif*, Prentice Hall 1995, 445 pages.
- **Et aussi:**
  - | E. Cutler, D. Gilly, T. O'Reilly, *The X Window System in a Nutshell*, O'Reilly and Associates, 1992.

**X-I****et aussi**

- o **O'Reilly & Associates**
  - | **Vol 0, X Protocol Reference Manual, 504 pages**
  - | **Vol 1, Xlib Programming Manual, 635 pages**
  - | **Vol 2, Xlib Reference Manual, 765 pages**
  - | **Vol 3, X Window System User's Guide, 924 pages**
  - | **Vol 4, X Toolkit Intrinsic Programming Manual, 584 pages**
  - | **Vol 5, X Toolkit Intrinsic Reference Manual, 764 pages**
  - | **Vol 6A, Motif Programming Manual, 972 pages**
  - | **Vol 6B, Motif Reference Manual, 908 pages**