Lifecycle of an X11 Window

XI033

Patrick Mahan

TGV Window Washer

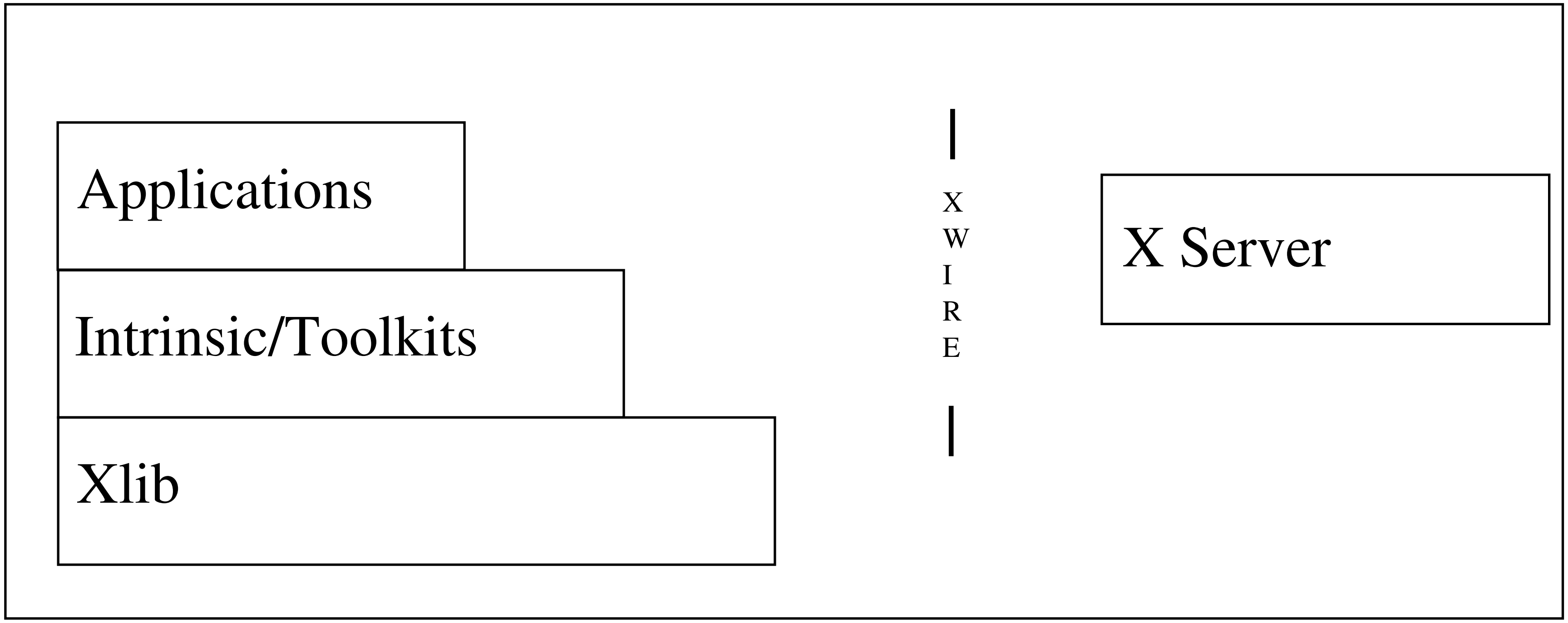
mahan@tgv.com

TGV, Incorporated

9:00 AM, June 8, 1993

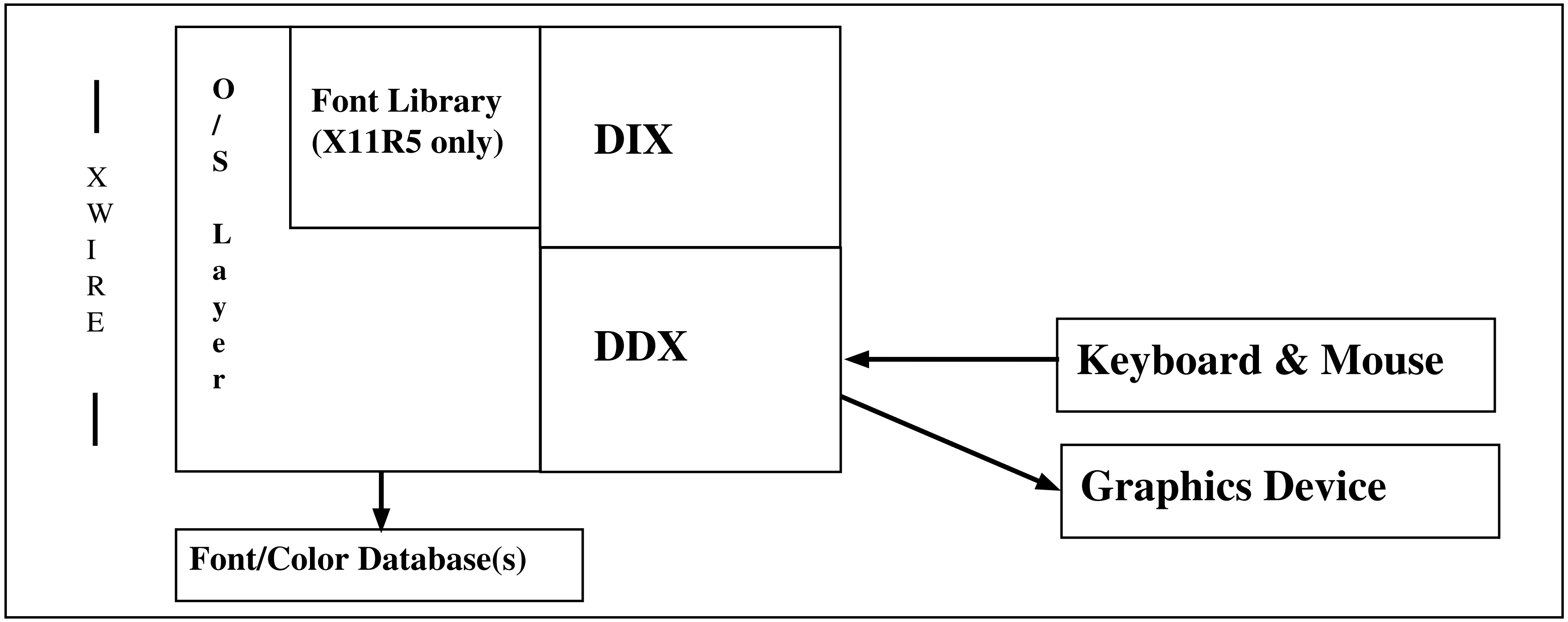
Lifecycle of an X11 Window
Spring DECUS Symposium 1993

- **Architecture Overview**



Lifecycle of an X11 Window
Spring DECUS Symposium 1993

❑ X Server Architecture



❑ Major parts of the X Server

- Device Independent X layer (DIX)
- Device Dependent X Layer (DDX)
- Operating System Layer (OS)
- Font Management

❑ Device Independent Layer

- Generic functions not dependent on O/S, Hardware or Input Devices
- Responsible for Request Dispatching
- Responsible for Event Queues
- Responsible for over all control of the server

❑ Device Dependent Layer

- Interface to Graphics Hardware and Input Devices
- Responsible for creating/manipulation of Pixmaps, Clipping Regions, Colormaps, fonts, etc.
- Heart of the server

Operating System Layer

- Host dependent functions
- Manages the client connections
- Memory Allocation/Deallocation
- Loading the RGB color database
- Loading fonts (pre X11R5)

Font Management Layer (X11R5)

- Loading fonts
- Accessing font server(s) (via OS Layer)

❑ Server Data Structures

- ♦ Drawables
- ♦ Windows
- ♦ Pixmaps
- ♦ Screens

❑ Drawables

- Generic representation of objects of drawing operations
- Contains info common to both Windows and Pixmaps
- Actually contained inside the Window/Pixmap structures

❑ Drawable Structure Layout

```
typedef struct _Drawable {
        unsigned char type;         /* Pixmap, Window */
        unsigned char class;        /* InputOutput, InputOnly */
        unsigned char depth;        /* depth of display */
        unsigned char bitsPerPixel;
        unsigned long id;           /* resource id */
        short    x;                 /* (x,y) position of drawable */(*)
        short    y;
```

```
        unsigned short width;       /* size of drawable */
        unsigned short height;
        ScreenPtr    pScreen;       /* pointer to screen */
        unsigned long serialNumber;
} DrawableRec;
```

* For Pixmaps this value is (0,0)

Drawables

- Serial Number is used for Graphics Context validation
- (x,y) position must be in screen coordinates and specifies the inside of the window
- For Pixmaps, (x,y) must be (0,0)

Windows

- Representation of a viewable object
- Represented by two structures
  - Window Structure
  - Window Optional Structure

Lifecycle of an X11 Window

Spring DECUS Symposium 1993

- ## Window structure

```
typedef struct _Window {
    DrawableRec     drawable;
    WindowPtr       parent;        /* ancestor chain */
    WindowPtr       nextSib;       /* next lower sibling */
    WindowPtr       prevSib;       /* next higher sibling */
    WindowPtr       firstChild;    /* top-most child */
    WindowPtr       lastChild;     /* bottom-most child */
    RegionRec       clipList;      /* clipping rectangle for output */
    RegionRec       borderClip;    /* NotClippedByChildren + border */
```

```
union _Validate      *valdata;
 RegionRec           winSize;
 RegionRec           borderSize;
 DDXPointRec         origin;          /* position relative to parent */
 unsigned short      borderWidth;
unsigned short       deliverableEvents;
 Mask                eventMask;
 PixUnion            background;
PixUnion             border;
 pointer             backStorage;  /* null when BS disabled */
WindowOptPtr         optional;
unsigned             backgroundState:2; /* None, Relative, Pixel, Pixmap */
```

```
unsigned        borderIsPixel:1;
unsigned        cursorIsNone:1;     /* else same as parent */
unsigned        backingStore:2;
unsigned        saveUnder:1;
unsigned        DIXsaveUnder:1;
unsigned        bitGravity:4;
unsigned        winGravity:4;
unsigned        overrideRedirect:1;
unsigned        visibility:2;
unsigned        mapped:1;
unsigned        realized:1;    /* ancestors are all mapped */
unsigned        viewable:1;    /* realized && InputOutput */
```

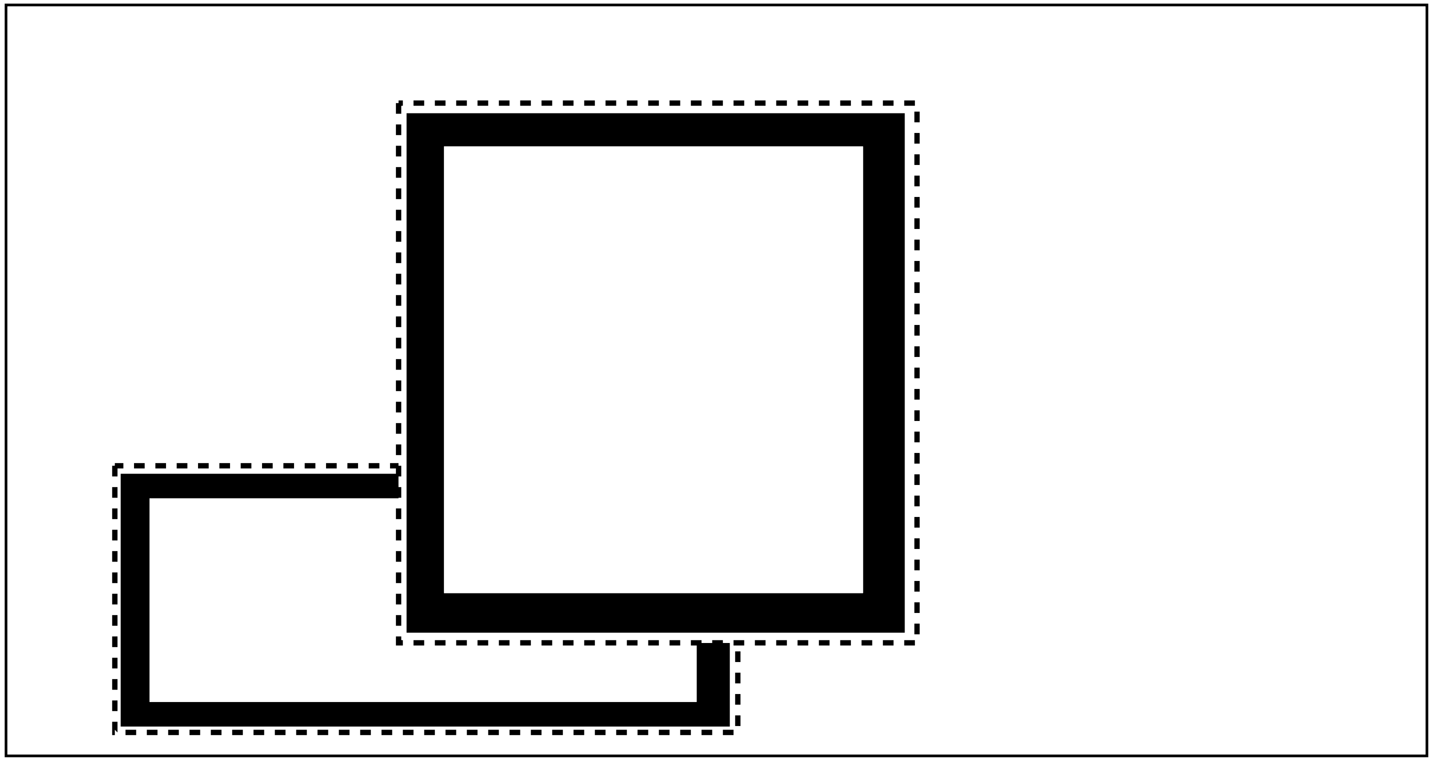
```
    unsigned        dontPropagate:3;/* index into DontPropagateMasks */
    DevUnion        *devPrivates;
} WindowRec;
```

Windows

- The origin of a window is the value stored in the Drawable structure but offset by the border width
- The Window Optional Structure (*optional*) may or may not be created

Lifecycle of an X11 Window
Spring DECUS Symposium 1993

- **Window layout**



Lifecycle of an X11 Window
Spring DECUS Symposium 1993

- **Window Optional Structure**

```
typedef struct _WindowOpt {
    VisualID                 visual;            /* default: same as parent */
    CursorPtr                cursor;            /* default: window.cursorNone */
    Colormap                 colormap;          /* default: same as parent */
    Mask          dontPropagateMask;            /* default: window.dontPropagate */
    Mask          otherEventMasks;              /* default: 0 */
    struct _OtherClients *otherClients;         /* default: NULL */
```

```
    struct _GrabRec        *passiveGrabs;         /* default: NULL */
    PropertyPtr            userProps;             /* default: NULL */
    unsigned long          backingBitPlanes;      /* default: ~0L */
    unsigned long          backingPixel;          /* default: 0 */
#ifdef SHAPE
    RegionPtr              boundingShape;         /* default: NULL */
    RegionPtr              clipShape;             /* default: NULL */
#endif
#ifdef XINPUT
    struct _OtherInputMasks *inputMasks;          /* default: NULL */
#endif
} WindowOptRec;
```

❑ Window Optional Structure

- Rarely created, since most of the information is inherited from the parent
- Generally used when the client installs a colormap other than the default or requests a cursor other than the parent's
- Also contains info for maintaining the SHAPE extension*

* SHAPE extension is currently only support under OpenVMS AXP Servers (This does not include Ultrix based machines)

Pixmaps

- Represents an off screen object that can be used for drawing operations
- Implementation is Server specific, the Pixmap can be in offscreen memory or located in VRAM on the Graphics Hardware

Lifecycle of an X11 Window

Spring DECUS Symposium 1993

- **Pixmap structure**

```
typedef struct _Pixmap {
        DrawableRec  drawable;
        int          refcnt;
        int          devKind;*
        DevUnion     devPrivate;*
} PixmapRec;
```

* These represent information that is private to the Device Dependent Layer (DDX) of the server

Screens

- Server consists of a single display containing a keyboard, pointing device and one or more screens
- Differ in size, depth and visual classes
- Represented by two structures
  - ScreenInfo structure
  - ScreenRec structure

- ## ScreenInfo structure

```
#define MAXSCREENS 4        /* from MISC.H */

#define MAXFORMATS 8        /* from MISC.H */


typedef struct _ScreenInfo {

     int            imageByteOrder;

     int            bitmapScanlineUnit;

    int             bitmapScanlinePad;

    int             bitmapBitOrder;
```

```
    int             numPixmapFormats;
    PixmapFormatRec     formats[MAXFORMATS];
    int             arraySize;
    int             numScreens;
    ScreenPtr       screens[MAXSCREENS];
} ScreenInfo;
```

❑ ScreenRec structure

```
#define MAXFORMATS 8        /* from MISC.H */


typedef struct _Screen {
    int         myNum;      /* index of this instance in Screens[] */
    ATOM        id;
    short       width, height;
    short       mmWidth, mmHeight;
    short       numDepths;
```

```
unsigned char        rootDepth;
DepthPtr        allowedDepths;
unsigned long        rootVisual;
unsigned long defColormap;
short            minInstalledCmaps, maxInstalledCmaps;
char    backingStoreSupport, saveUnderSupport;
unsigned long whitePixel, blackPixel;
unsigned long rgf;    /* array of flags; she's -- HUNGARIAN */
GCPtr            GCperDepth[MAXFORMATS+1];
                 /* next field is a stipple to use as default in
                  a GC. we don't build default tiles of all depths
                  because they are likely to be of a color
```

```
                 different from the default fg pixel, so
                 we don't win anything by building
                 a standard one.
                 */
PixmapPtr            PixmapPerDepth[1];
pointer          devPrivate;
short    numVisuals;
VisualPtr            visuals;
int                  WindowPrivateLen;
unsigned             *WindowPrivateSizes;
unsigned             totalWindowSize;
int                  GCPrivateLen;
```

```
    unsigned                *GCPrivateSizes;
    unsigned                totalGCSize;

/* Random screen procedures */

    Bool (* CloseScreen)();          /* index, pScreen */
    void (* QueryBestSize)();        /* class, pwidth, pheight, pScreen */
    Bool (* SaveScreen)();           /* pScreen, on */
    void (* GetImage)();             /* pDrawable, sx, sy, w, h, format,
                                      * planemask, pdestbits */
    void (* GetSpans)();             /* pDrawable, wMax, ppt, pwidth,
                                      * nspans, pdstbits */
```

```
    void (* PointerNonInterestBox)();    /* pScr, BoxPtr */


    void (* SourceValidate)();           /* pDrawable, x, y, w, h */


/* Window Procedures */

    Bool (* CreateWindow)();             /* pWin */

    Bool (* DestroyWindow)();            /* pWin */

    Bool (* PositionWindow)();           /* pWin, x, y */

    Bool (* ChangeWindowAttributes)();   /* pWin, mask */

    Bool (* RealizeWindow)();            /* pWin */

    Bool (* UnrealizeWindow)();          /* pWin */

    int (* ValidateTree)();              /* pParent, pChild, kind */
```

```
    void (* PostValidateTree)();          /* pParent, pChild, kind */
    void (* WindowExposures)(); /* pWin: WindowPtr, pRegion: RegionPtr */
    void (* PaintWindowBackground)(); /* pWin, pRgn, which */
    void (* PaintWindowBorder)();         /* pWin, pRgn, which */
    void (* CopyWindow)();                /* pWin, oldPt, pOldRegion */
    void (* ClearToBackground)();         /* pWin, x,y,w,h, sendExpose */
    void (* ClipNotify)();                /* pWin, dx, dy */

/* Pixmap procedures */

    PixmapPtr (* CreatePixmap)();         /* pScreen, width, height, depth */
    Bool (* DestroyPixmap)();             /* pPixmap */
```

```
/* Backing store procedures */

    void (* SaveDoomedAreas)();          /* pWin, pRegion, dx, dy */
    RegionPtr (* RestoreAreas)();        /* pWin, pRegion */
    void (* ExposeCopy)();               /* pSrc, pDst, pGC, pRegion, */
                                         /* srcx, srcy, dstx, dsty, plane */
    RegionPtr (* TranslateBackingStore)();/* pWin, dx, dy, pOldClip */
    RegionPtr (* ClearBackingStore)();   /* pWin, x, y, w, h, sendExpose */
    void (* DrawGuarantee)();            /* pWin, pGC, guarantee */

/* Font procedures */
```

```
    Bool (* RealizeFont)();            /* pScr, pFont */
    Bool (* UnrealizeFont)();          /* pScr, pFont */

/* Cursor Procedures */
    void (* ConstrainCursor)();        /* pScr, BoxPtr */
    void (* CursorLimits)();           /* pScr, pCurs, BoxPtr, BoxPtr */
    Bool (* DisplayCursor)();          /* pScr, pCurs */
    Bool (* RealizeCursor)();          /* pScr, pCurs */
    Bool (* UnrealizeCursor)();        /* pScr, pCurs */
    void (* RecolorCursor)();          /* pScr, pCurs, displayed */
    Bool (* SetCursorPosition)();      /* pScr, x, y */
```

```
/* GC procedures */

   Bool (* CreateGC)();                /* pGC */

/* Colormap procedures */

   Bool (* CreateColormap)();          /* pcmap */
   void (* DestroyColormap)();         /* pcmap */
   void (* InstallColormap)();         /* pcmap */
   void (* UninstallColormap)();       /* pcmap */
   int (* ListInstalledColormaps) ();  /* pScreen, pmaps */
```

```
    void (* StoreColors)();            /* pmap, ndef, pdef */
    void (* ResolveColor)();           /* preg, pgreen, pblue */

/* Region procedures */

    RegionPtr (* RegionCreate)();      /* rect, size */
    void (* RegionInit)();             /* pRegion, rect, size */
    Bool (* RegionCopy)();             /* dstrgn, srcrgn */
    void (* RegionDestroy)();          /* pRegion */
    void (* RegionUninit)();           /* pRegion */
    Bool (* Intersect)();              /* newReg, reg1, reg2 */
    Bool (* Union)();                  /* newReg, reg1, reg2 */
```

```
Bool (* Subtract)();                /* regD, regM, regS */
Bool (* Inverse)();                 /* newReg, reg1, invRect */
void (* RegionReset)();             /* pRegion, pBox */
void (* TranslateRegion)();         /* pRegion, x, y */
int (* RectIn)();                   /* pRegion, pRect */
Bool (* PointInRegion)();           /* pRegion, x, y, pBox */
Bool (* RegionNotEmpty)();          /* pRegion: RegionPtr */
void (* RegionEmpty)();             /* pRegion: RegionPtr */
BoxPtr (* RegionExtents)();         /* pRegion: RegionPtr */
Bool (* RegionAppend)();            /* pRegion, pRegion */
Bool (* RegionValidate)();          /* pRegion, pOverlap */
RegionPtr (* BitmapToRegion)();     /* PixmapPtr */
```

```
    RegionPtr (* RectsToRegion)();       /* nrects, pRects, ordering */
    void (* SendGraphicsExpose)();       /* client, rgn, draw, major, minor */


 /* os layer procedures */
    void (* BlockHandler)();             /* data: pointer */
    void (* WakeupHandler)();            /* data: pointer */
    pointer blockData;
    pointer wakeupData;


/* anybody can get a piece of this array */
    DevUnion      *devPrivates;
} ScreenRec;
```

Screens

- ScreenInfo structure
  - There is only 1 of these per display
  - Maintains information global to the server
  - Link to the ScreenRec structure(s)
- ScreenRec structure
  - As many as defined during the build of the server*
  - The data is maintain by the DIX
  - Contains the entry points into the DDX for use by DIX

* Currently Digital provides support for up to 4 screens

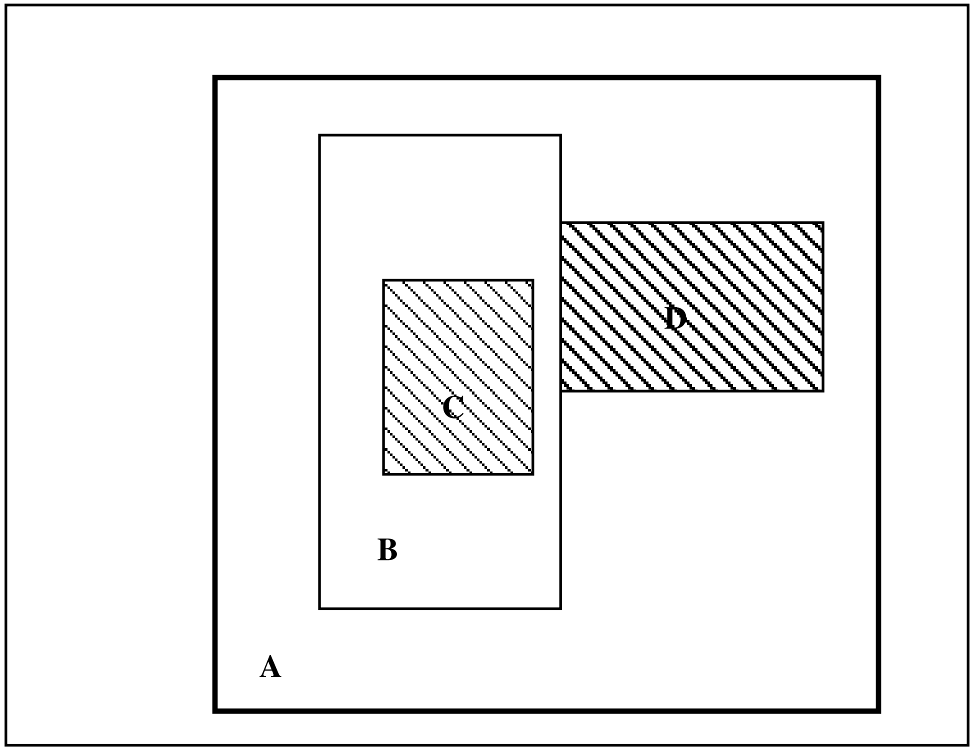
❑ Windows

- Windows are kept as a hierarchical data structure in the server which is refered to as the *Window Tree*
- Each Window has one *Parent* window
- Each Window can have one or more *Child* windows
- Windows are *Clipped* to its parent
- Windows which have the same parent are called *Siblings*
- All Windows under a given branch of the hierarchy

are known as *Descendants*

- All Parents in a given branch of the hierarchy are known as *Ancestors*
- Descendants are also known as *Inferiors*
- There is one special window that exists as the Ancestor of all windows and is known as the *Root Window*
- A Window cannot be both an Ancestor and Descendant of another window

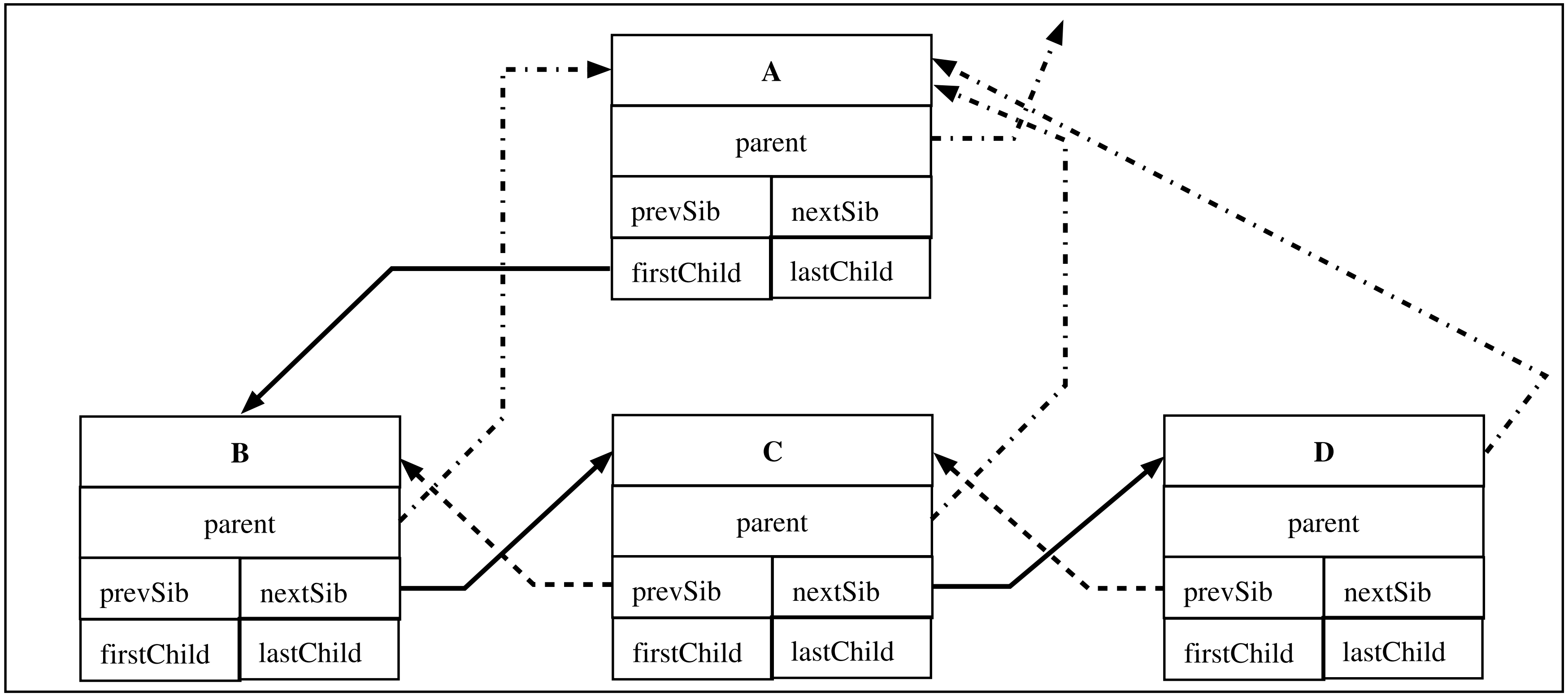
- **Window Tree**



Lifecycle of an X11 Window

Spring DECUS Symposium 1993

- **Window Tree**



❑ Window Creation

- The Client request the creation of a window
  - via XCreateWindow()
  - via XtRealize()
  - via XtCreateWindow()

CreateWindow Request

```
struct createWindow {
        unsigned char   opcode;     /* value is 1 */
        unsigned char   depth;
        short           length;     /* length of this request in longwords */
        unsigned long   id;         /* window id , set by Xlib */
        unsigned long   parent;     /* parent window id */
        short           x;          /* (x,y) */
        short           y;
        unsigned short  width;      /* size of the window */
```

```
        unsigned short height;
        unsigned short bwidth;      /* border width */
        unsigned short class;
        unsigned long visual;
        unsigned long mask;  /* attributes set in XSetWindowAttributes */
};
```

CreateWindow Request

- The size of the Request is 8 longwords (32 bytes) plus any values set in the associated XSetWindowAttributes structure (each value is 1 longword)
- Visual and Class may be copied from the Parent

Window Creation in the Server

- CreateWindow Request is dispatched to ProcCreateWindow() (byte-swapping as needed)
  - Length of Packet is verified (must be >= 32 bytes)
  - Parent Window Id is verified
  - width and height are validated
- ProcCreateWindow() in turn calls CreateWindow()
  - All information that is inherited from Parent is copied
  - Class and Visual are validated
  - Window structure is allocated

- If Visual is different from parent, then allocate a WindowOpt structure
- Regions are initialized
- Window is inserted into the Window Tree
- DDX routine for window creation is called
- DDX routine for window positioning is called
- Event mask is established on the window
- Window Attributes are set

Lifecycle of an X11 Window
Spring DECUS Symposium 1993

- **Mapping a Window**
  - The Client requests that the window be mapped
    - via XMapWindow()

- **MapWindow Request**

```
struct mapWindow {
        unsigned char   opcode;         /* this will be 8 */
        char            pad;            /* unused */
        unsigned short length;          /* will be 2 longwords */
        unsigned long   id;             /* resource id(*) */
};
```

* This can be a Window, Pixmap, Font, GC, etc

❑ Mapping a window in the server

- MapWindow Request is dispatch to ProcMapWindow() (byte-swapping as needed)
  - Length of Packet must = 8 bytes
  - Window Id is validated
- ProcMapWindow() in turn calls MapWindow()
  - Check to make sure it hasn't already been mapped
  - Mark the window as mapped
  - If Parent window is not yet mapped, then defer till later
  - Map the window and its descendents, done by calling into the DDX

Window Clipping

- Windows can overlap, reducing visiblity of a window
- Visible portions of the window are defined by 4 Regions
  - Border Size Region
  - Window Size Region
  - Border Clip Region
  - Window Clip Region

Lifecycle of an X11 Window
Spring DECUS Symposium 1993

- **Window Clipping**