

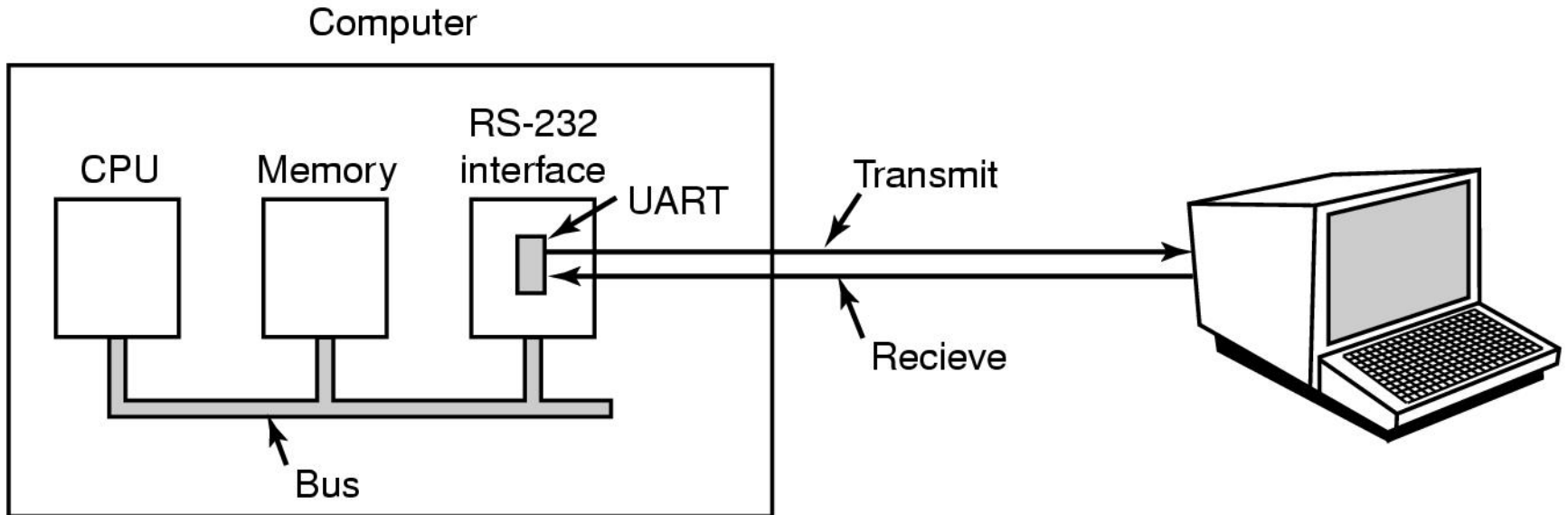
Benutzerschnittstellen

Ein-/Ausgabe

Prof. Dr.-Ing. Torben Weis
University Duisburg-Essen

ASCII Terminals

RS-232 Terminal Hardware



- Ein RS-232 Terminal tauscht Daten Bit-weise mit dem Computer aus
- Es wird nur eine zeichenbasiertes UI unterstützt
- Im Terminal selbst wird keine Applikations-Logik ausgeführt

ASCII Terminals

- Eingabe wird vom Terminaltreiber vorverarbeitet
- Raw Mode
 - Alias: Nicht-kanonischer Modus
 - Jedes Zeichen wird so übergeben, wie es auf der Tastatur eingegeben wurde
 - Gilt auch für Steuerzeichen
 - Cursor, Return, ESC, ...
- Cooked Mode
 - Alias: Kanonischer Modus
 - Alias: Zeilenorientierter Modus
 - Löschoptionen etc. werden vom Treiber realisiert
 - Übergabe einer kompletten Zeile nach RETURN

ASCII Terminals

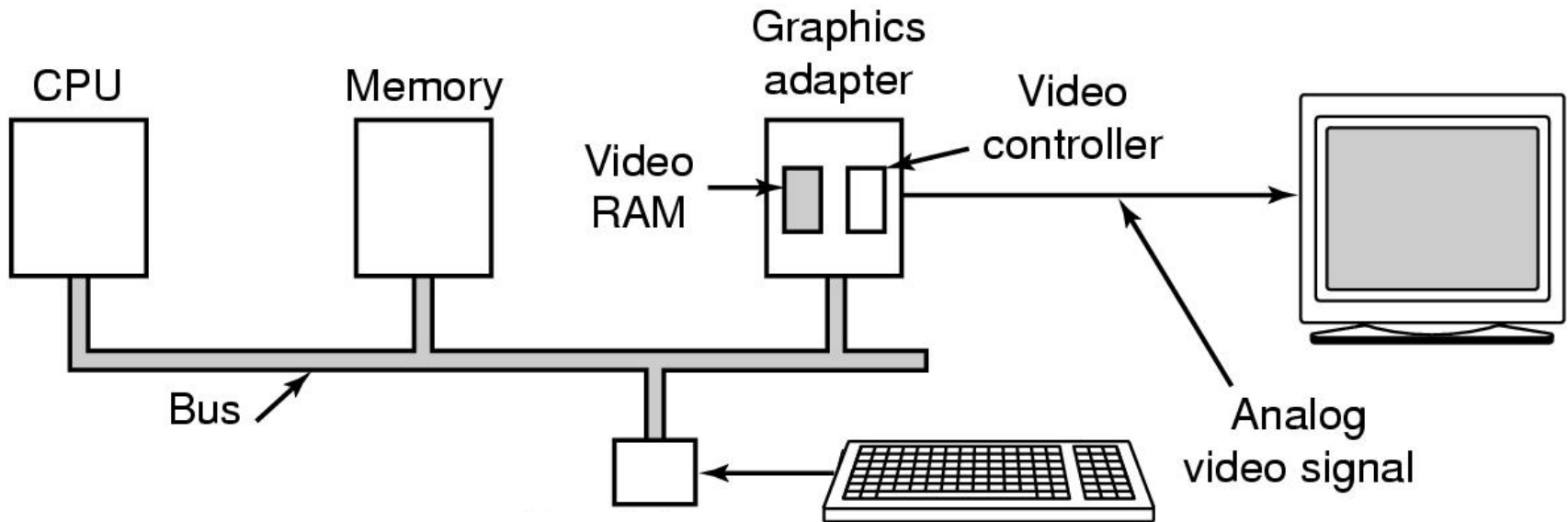
Character	POSIX name	Comment
CTRL-H	ERASE	Backspace one character
CTRL-U	KILL	Erase entire line being typed
CTRL-V	LNEXT	Interpret next character literally
CTRL-S	STOP	Stop output
CTRL-Q	START	Start output
DEL	INTR	Interrupt process (SIGINT)
CTRL-\	QUIT	Force core dump (SIGQUIT)
CTRL-D	EOF	End of file
CTRL-M	CR	Carriage return
CTRL-J	NL	Linefeed

Steuerzeichen für den zeilenorientierten Modus

ANSI Escape Sequenzen

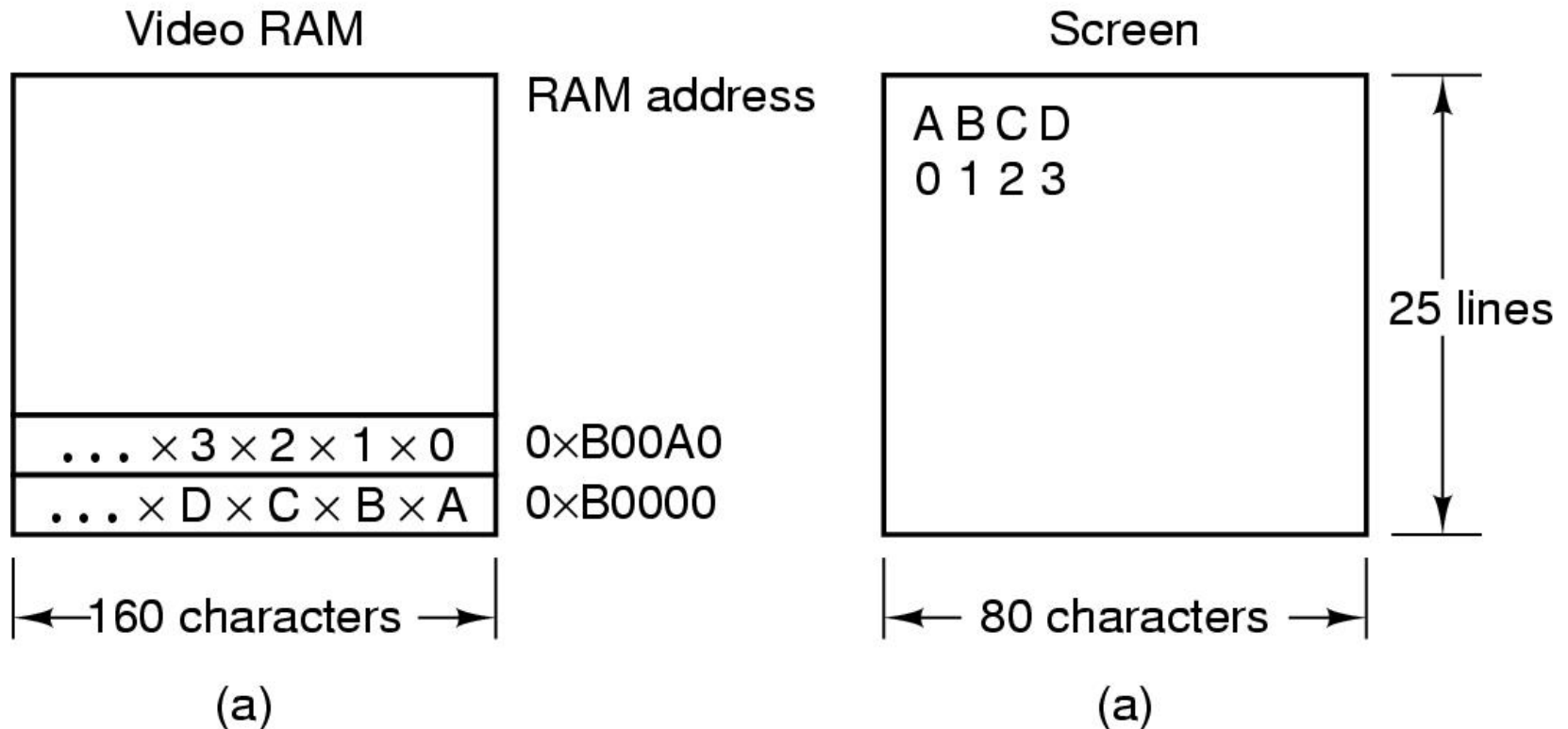
Escape sequence	Meaning
ESC [<i>n</i> A	Move up <i>n</i> lines
ESC [<i>n</i> B	Move down <i>n</i> lines
ESC [<i>n</i> C	Move right <i>n</i> spaces
ESC [<i>n</i> D	Move left <i>n</i> spaces
ESC [<i>m</i> ; <i>n</i> H	Move cursor to (<i>m</i> , <i>n</i>)
ESC [<i>s</i> J	Clear screen from cursor (0 to end, 1 from start, 2 all)
ESC [<i>s</i> K	Clear line from cursor (0 to end, 1 from start, 2 all)
ESC [<i>n</i> L	Insert <i>n</i> lines at cursor
ESC [<i>n</i> M	Delete <i>n</i> lines at cursor
ESC [<i>n</i> P	Delete <i>n</i> chars at cursor
ESC [<i>n</i> @	Insert <i>n</i> chars at cursor
ESC [<i>n</i> m	Enable rendition <i>n</i> (0=normal, 4=bold, 5=blinking, 7=reverse)
ESC M	Scroll the screen backward if the cursor is on the top line

Grafik Hardware



- Grafikkartenspeicher
 - Auf der Grafikkarte selbst
 - Mitbenutzung des Hauptspeichers
 - Führt zu Datenstau auf dem Bus
 - Hybrides Modell
 - Hauptspeicher wird insbesondere für Texturen genutzt

Zeichenbasierte Grafik Hardware



- Video RAM
 - Speichert 2 Byte: Das Zeichen und sein Attribut
 - Schreiben auf das Video RAM führt sofort zu Änderungen auf dem Bildschirm
 - Uralt Technik aus Zeiten des Commodore C64

Framebuffer

- Grafikspeicher wird in den Hauptspeicher eingeblendet
- BS (oder eine Applikation) kann Pixel einzeln setzen und löschen
- Sehr langsam
 - Zeichnen einer 640 Pixel langen schrägen Linie bedarf aufwendiger Berechnungen und 640 Speicherzugriffen
- Video RAM Layout
 - 24 Bit: Red, Green, Blue (RGB) jeweils 1 Byte
 - 32 Bit: RGB + Transparenz

Framebuffer

- **Anwendungsgebiete**
 - Energiesparende Geräte
 - GPU verbraucht zuviel Energie
 - Billige Geräte
 - GPU ist zu teuer
 - GPU nimmt zuviel Platz auf der Platine in Anspruch
 - Mobiltelefone, PDAs, Unterhaltungselektronik
- **Geringe Auflösung**
 - VGA oder SVGA
 - Für höhere Auflösungen sind Framebuffer zu langsam
- **Keine 3D Effekte**
 - Mangels Hardware Beschleunigung zu langsam

Framebuffer

- Anwendungen schreiben direkt in das Video RAM
- Das BS hat keine Kontrolle
 - Anwendungen können überall hinschreiben
 - Anwendung muss alles selbst machen
 - „Fenster“ sind nicht durchsetzbar
- Sicherheitsrisiko
 - Beispiel Mobiltelefon: „Bitte geben Sie ihre PIN ein“
 - Wer fragt das?
 - Betriebssystem? Es muss die PIN wissen
 - Anwendung? Darf die PIN nicht erfahren

Framebuffer

- Nur das BS hat Zugriff auf den Framebuffer
- BS stellt Grafik API zur Verfügung
 - Beispiel: Windows GDI
 - Pixels, Linien, Rechtecke, Ellipsen, Füllen
 - Grafikpfade (Polygone)
 - Bitmaps
 - Schriften
- BS kann „Fenster“ durchsetzen
 - Clipping: Applikation kann nur in ihrem Fenster zeichnen
 - Alles andere wird abgeschnitten

2D Beschleuniger

- Mit Framebuffern muss das BS immer noch jedes Pixel einzeln setzen
 - Viel zu langsam
- Graphics Processing Unit (GPU)
 - Hardwareunterstützung für Zeichenoperationen
 - BS schickt nur noch Zeichenbefehle an die GPU
 - Entlastet die CPU
- Hardwareabstraktion
 - BS bietet Anwendungen einheitliche API
 - Manchmal schwierig, weil eine Zeichenoperation unterschiedlich durchgeführt werden kann

3D Beschleuniger

- 3D Szenen kann man mit 2D Primitiven zeichnen, aber das ist ineffizient
- Polygone, Schatten, Kantenglättung und Texturen brauchen enorme Rechenzeit
 - Von der CPU auf die GPU verlagern
 - Das BS übermittelt nur noch Gittermodelle, Texturen und Beleuchtungen
 - Die GPU berechnet daraus einzelne Pixel
- Herausforderung für das BS
 - Einheitliche API trotz verschiedener GPUs
 - Lösung: Erst die API definieren, dann die Chips bauen
 - Beispiel: MS DirectX

Fenster

- Anwendungen dürfen in einem Fenster ...
 - Zeichnen (2D oder 3D)
 - Ereignisse empfangen und verarbeiten
 - Maus, Tastatur, ...
- Zeichnen erfolgt (in der Regel) synchron
 - BS blockiert die Applikation solange, bis der Zeichenbefehl an die GPU geschickt wurde
- Ereignisse werden asynchron zugestellt
 - Pro Fenster wird eine Ereignis-Warteschlange verwaltet
 - Wenn die Anwendung zu langsam ist, ergibt sich ein Stau
 - Das BS kann unter Umständen die Ereignisse zusammenfassen
 - 3 Mouse-Move-Events -> die ersten 2 kann man wegwerfen

Win32 API (1)

```
#include <windows.h>
```

```
int WINAPI WinMain(HINSTANCE h, HINSTANCE, hprev, char *szCmd, int iCmdShow)
{
    WNDCLASS wndclass;           /* class object for this window */
    MSG msg;                    /* incoming messages are stored here */
    HWND hwnd;                  /* handle (pointer) to the window object */

    /* Initialize wndclass */
    wndclass.lpfWndProc = WndProc; /* tells which procedure to call */
    wndclass.lpszClassName = "Program name"; /* Text for title bar */
    wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION); /* load program icon */
    wndclass.hCursor = LoadCursor(NULL, IDC_ARROW); /* load mouse cursor */

    RegisterClass(&wndclass); /* tell Windows about wndclass */
    hwnd = CreateWindow ( ... ) /* allocate storage for the window */
    ShowWindow(hwnd, iCmdShow); /* display the window on the screen */
    UpdateWindow(hwnd); /* tell the window to paint itself */
}
```

Win32 API (2)

```
while (GetMessage(&msg, NULL, 0, 0)) { /* get message from queue */
    TranslateMessage(&msg); /* translate the message */
    DispatchMessage(&msg); /* send msg to the appropriate procedure */
}
return(msg.wParam);
}

long CALLBACK WndProc(HWND hwnd, UINT message, UINT wParam, long lParam)
{
    /* Declarations go here. */

    switch (message) {
        case WM_CREATE:    ... ; return ... ; /* create window */
        case WM_PAINT:    ... ; return ... ; /* repaint contents of window */
        case WM_DESTROY:  ... ; return ... ; /* destroy window */
    }
    return(DefWindowProc(hwnd, message, wParam, lParam)); /* default */
}
```

Schriften

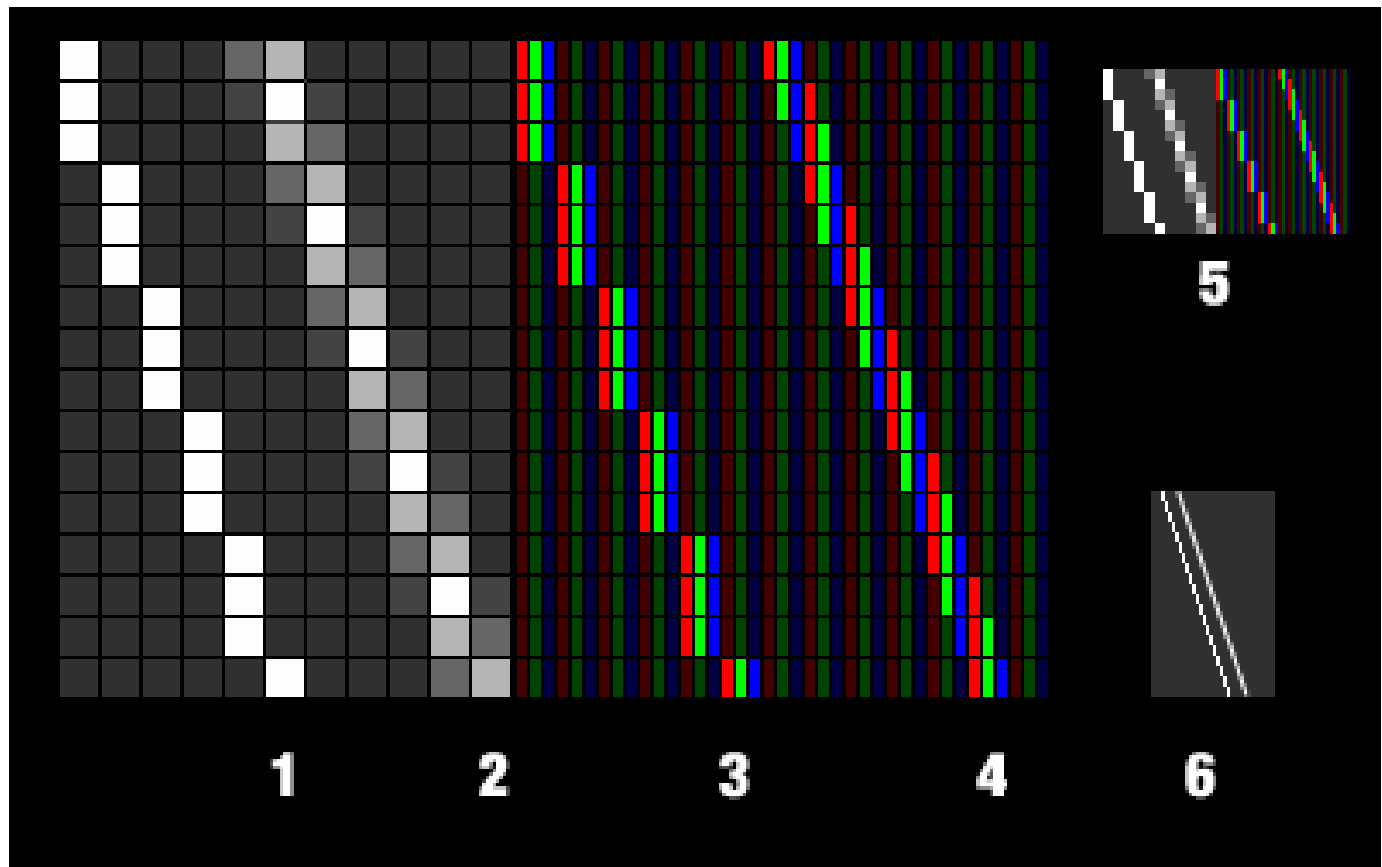
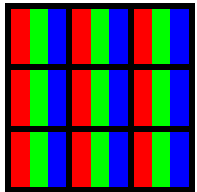
20 pt: abcdefgh

53 pt: abcdefgh

81 pt: abcdefgh

Pixels und Subpixels

- Auf LCDs unterteilt sich ein Pixel in 3 Subpixel



Videos & Video-Beam-Synchronisation

- Die GPU tastet den Speicher Zeile für Zeile ab
 - Daraus wird das Monitor Signal erzeugt
 - Wenn alle Zeilen durch sind, kommt die Austastlücke
 - Das Bild verweilt kurzzeitig unverändert
- Das BS kann Videos mit dem Video-Beam synchronisieren
 - Videos werden Bild für Bild an die Grafikkarte übermittelt
 - Idealerweise werden die Bilder „hinter“ dem Video-Beam geschrieben
 - Anderenfalls sieht man oben Bild $n+1$ und unten noch Bild n

Windows Vista

- Win32: GDI+
 - Anwendung wird aufgefordert, ein Fenster neu zu zeichnen (WM_Paint Event)
 - Wenn die Anwendung blockiert, bleibt das Fenster weiß oder schmutzig (dirty)
- Vista: Desktop Composition Engine
 - Anwendungen sagen dem BS, was sie darstellen wollen und wo
 - Text, Bild, Polygone
 - Das ergibt eine Baumstruktur
 - Das BS kann jederzeit neu zeichnen ohne die Anwendung zu fragen

Session Management

- Was passiert beim Ausloggen/Herunterfahren ?
 - a) Alle Anwendungen bekommen ein STOP Signal. Anwendungen beenden sich.
 - Die Session wird beendet
 - b) Das BS speichert den kompletten Systemzustand auf der Platte
 - Das BS speichert die Session auf Betriebssystem-Ebene, d.h. Speicherbelegung, Prozesse, etc.
 - c) Ein Session Manager speichert laufende Prozesse und Position der Fenster
 - Danach werden alle Anwendungen beendet
 - Beim Neustart werden die Anwendungen wieder gestartet
 - Die Fenster werden wieder positioniert wie vorher
 - Der Session Manager speichert die Session auf höherer Abstraktionsebene

Kern oder nicht Kern?

- Wohin gehören die GUI Fähigkeiten des Betriebssystems
- Windows
 - GUI ist integraler Bestandteil des Betriebssystems
 - Ohne ein Window Handel kommt man nicht weit
 - Instabiler Grafiktreiber kann BS korrumpieren
- UNIX
 - X11 Server ist ein fast normaler Prozess
 - X11 Server kann man abbrechen und neu starten ohne dass das BS Schaden nimmt
 - Standard UI sind altmodische ANSI-kompatible Terminals

Betriebssystem-unabhängige GUI

- Jedes BS hat (mindestens) ein Look&Feel
 - Aussehen von Standard-Controls
 - Verhalten der Controls
 - Layout von Dialogen
 - Position von Controls
 - Abstände zwischen Controls und Rändern
- Anwendungen sollten portabel sein, d.h. unter verschiedenen Betriebssystemen laufen
 - a) Abstraktion des BS durch eine virtuelle Maschine:
Java, .NET (teilweise)
 - b) Abstraktion durch eine Klassenbibliothek:
Qt

Betriebssystem-unabhängige GUI

- Ansatz 1: Ein Look & Feel für alle
 - Anwendung sieht unter jedem BS gleich aus
 - Anders formuliert: Anwendung sieht unter keinem BS wie eine native Anwendung aus
 - Beispiel: Java Swing
- Ansatz 2: Anpassen an das jeweilige Look&Feel
 - Anwendung sieht unter jedem BS anders aus
 - Nicht von einer nativen Anwendung unterscheidbar
 - „Look“ ist relativ einfach
 - „Feel“ ist kompliziert, weil die Anwendung sich im jeweiligen BS anders verhalten muss

Remote Desktops & Grafik-Terminals

- Bisher: Anwendung und Grafikanzeige laufen auf dem selben Rechner
- Nachteile
 - Anwendung muss auf dem PC des Nutzers installiert sein
 - Erhöhte Administrationskosten
- Remote Desktops
 - Anwendung läuft auf einem Server
 - Grafik-Terminal leitet Eingabe Ereignisse an den Applikations-Server weiter
 - Der Applikations-Server sagt dem Terminal, was es darstellen soll

VNC: x11

Persönlicher Ordner Müllleimer System Druckaufträge USB-Stick/ Diskette

Unbenannt1 - OpenOffice.org Writer

Datei Bearbeiten Ansicht Einfügen Format Tabelle Extras Fenster Hilfe x

Standard Times New Roman 12 F K U

Hallo von einer VNC-Sitzung.

Seite 1 / 1 Standard 100% EINFÜG STD HYF * 16:16:24

- Persönlich Ordner
- USB/Kamer
- Webbrowse
- Médecins Sans Fron
- green.pn
- Fischbach Wappen.sv
- MTK Wappen.sv

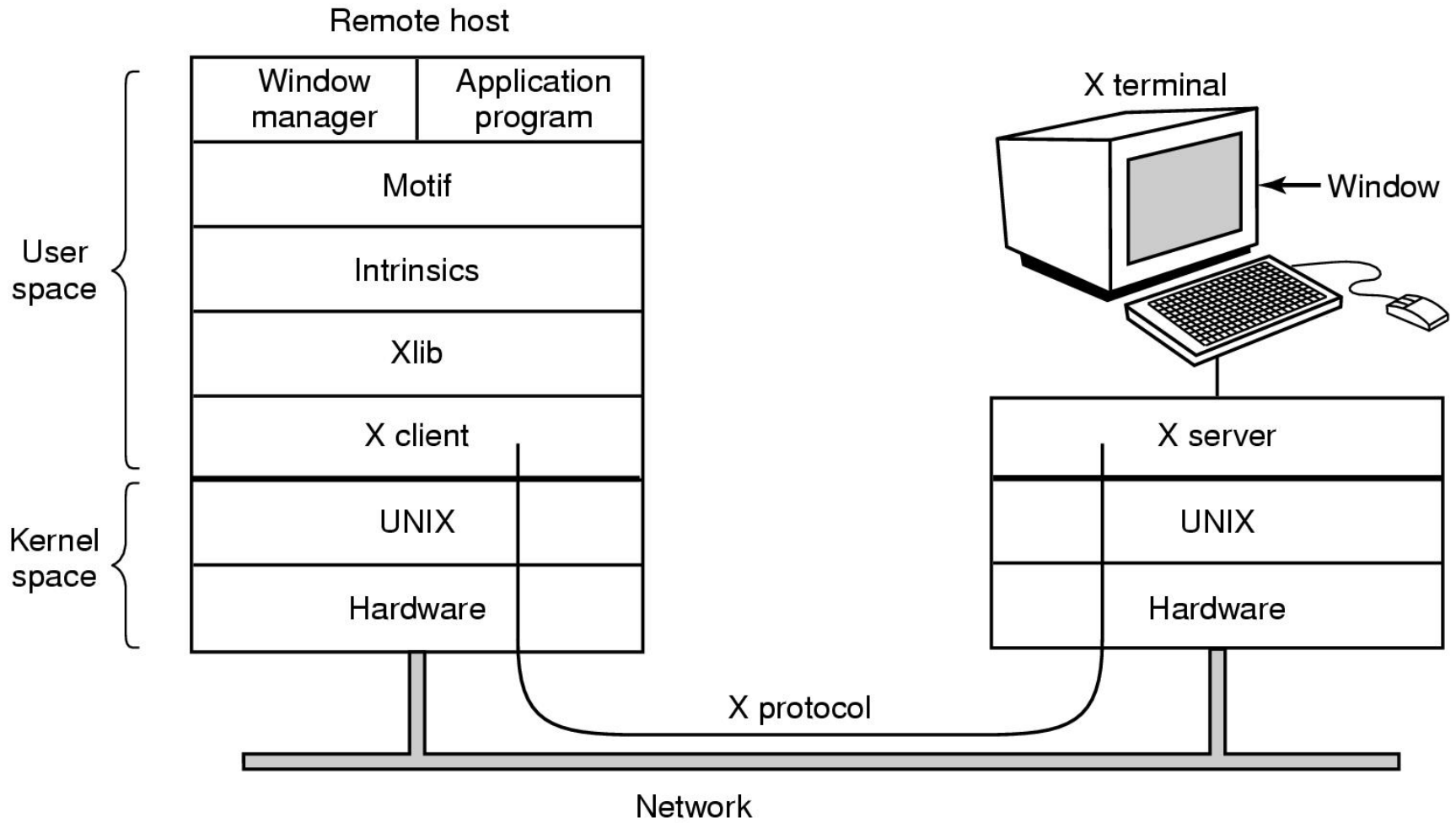
- ta- hload
- ctbot- Projekt
- nglisch
- Wormhole. svg
- heim- en.svg
- Area of a circle.svg
- Overhead. wiki.txt
- Commons: User:Sven...
- matrox_ driver-x86...

Remote Desktop Protokolle

- ALP
 - Appliance Link Protocol ist ein Bitmap-basiertes Protokoll für Sun Rays
- XML over HTTP
 - Benutzt von Web Applikationen, die AJAX einsetzen
- X11
 - Unter UNIX verbreitetes Protokoll
- X11 over SSH
 - Sichere Variante von X11
- NX technology
 - Komprimierte Variante von X11
- VNC
 - Bitmap-basiertes Protokoll für Remote Desktops
- RDP
 - Bitmap-basiertes Protokoll für Remote Desktops von Microsoft
- HTML over HTTP
 - Web Applikationen

Network Terminals

X Windows



Network Terminals

X Windows

- Das X11 Protokoll übermittelt ...
 - Fenster-Management
 - Erzeugen, zerstören, verschieben
 - Zeichenbefehle
 - Linie, Rechteck, Ellipse, Polygone, Bilder
 - Bilder werden nur einmal übertragen und zwischengespeichert
 - Eingabeereignisse
 - Maus, Tastatur, ...
- Nachteile
 - Sehr hohe Netzlast bei Animationen (viele Zeichenbefehle) und Video (alle Bilder müssen über das Netzwerk)

Thin-Clients versus Fat-Clients

- Fat-Client
 - Windows oder Linux PC
 - Eigene Festplatte, Anwendungen werden auf dem PC installiert
 - Wartungsintensiv
 - Entlastet die Server und ermöglicht anspruchsvollste Nutzeroberflächen
- Thin-Client
 - Das neue Wort für Grafik-Terminal
 - Benutzt Remote Desktop Technologie
 - Beschränkte Nutzeroberflächen, besonders im Hinblick auf Animationen, Video, 3D
 - Keine Festplatte, keine installierte Anwendungen
 - Weniger Administrationsaufwand
 - Höhere Sicherheit durch weniger Angriffspunkte

Beispiel für einen Thin-Client

