

1. Window-Systeme

1.1 Grundlagen von Windowsystemen

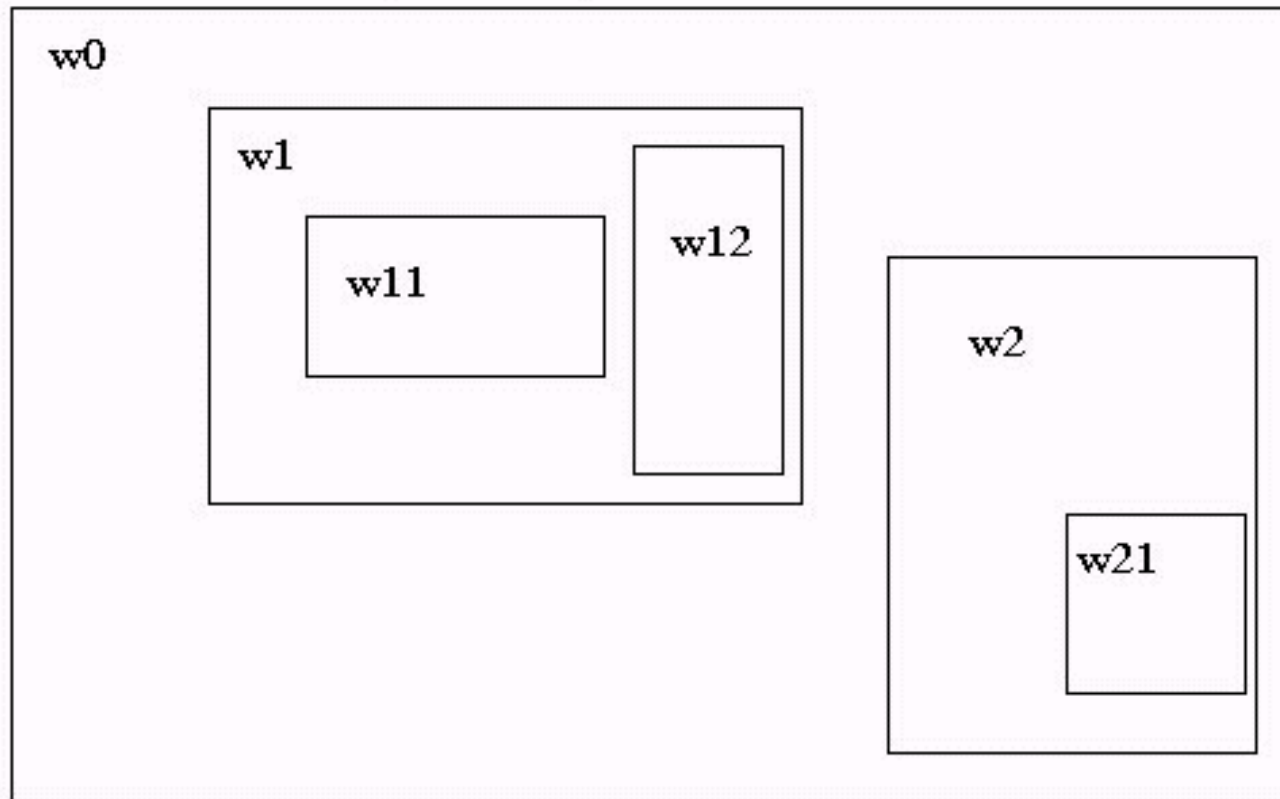
Vorbemerkung

- dieser Abschnitt abstrahiert die Konzepte, die in unterschiedlichen Window-Systemen vorkommen (Xwindows, Java-Awt, MSWindows)
- in den unterschiedlichen Systemen sind diese Konzepte unterschiedlich umgesetzt
 - mit unterschiedlicher Technik (prozedural, o-o)
 - mit unterschiedlichen Fähigkeiten (z.B. Window-Eigenschaften, Graphik-Primitiven, client-server-Fähigkeiten)
 - mit unterschiedlicher Behandlung der Event-Loop

1.1 Grundlagen von Windowsystemen

Ein Window-System erlaubt Einteilung des Bildschirms in rechteckige Bereiche (Windows).

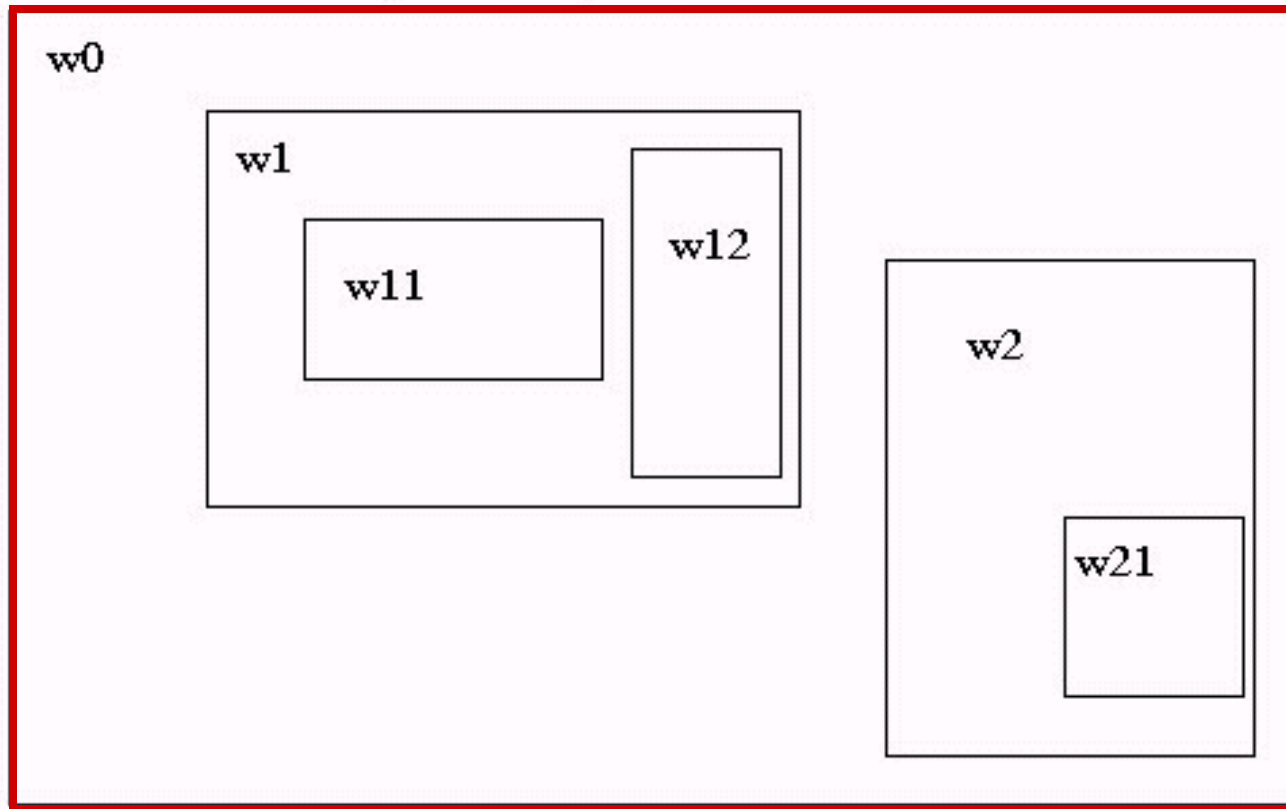
ROOT-WINDOW (SCREEN)



1.1 Grundlagen von Windowsystemen

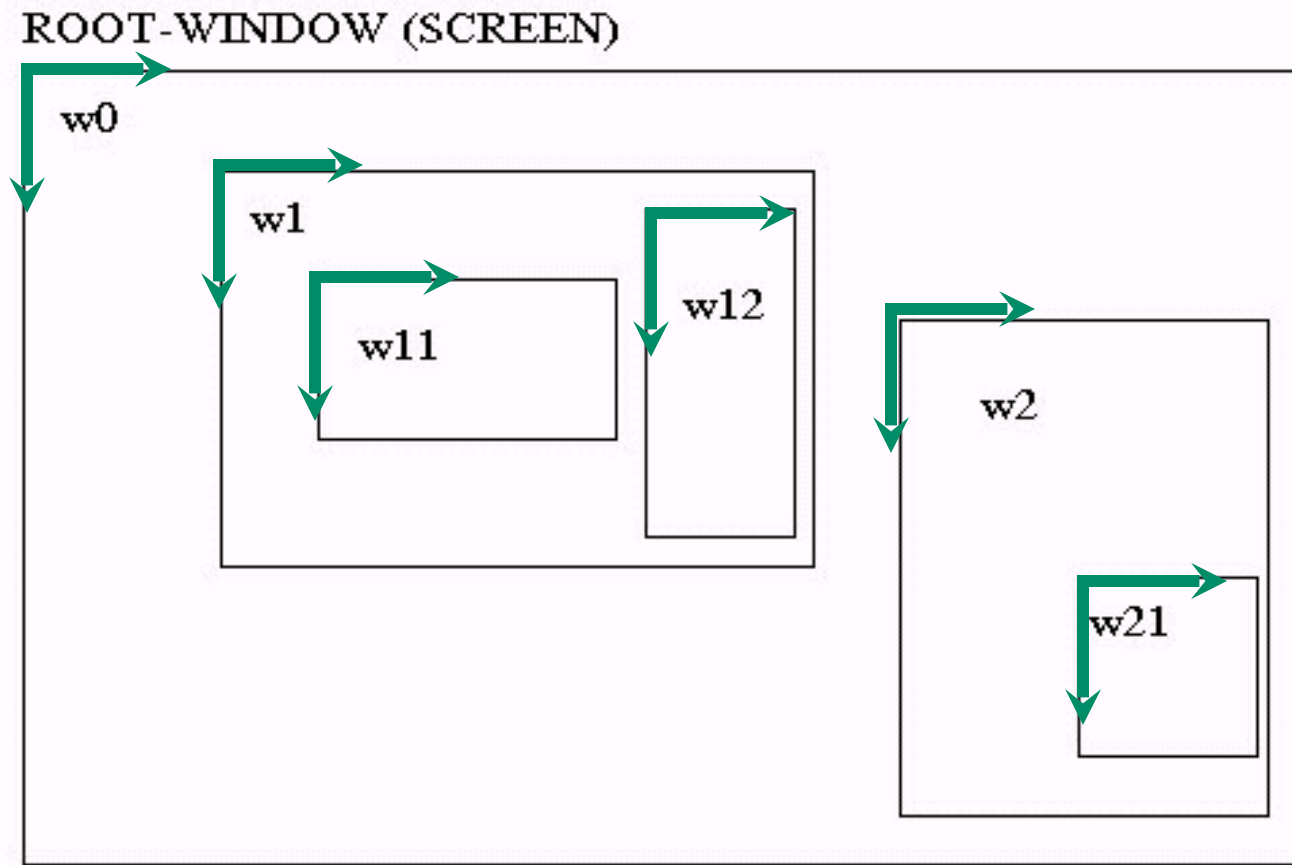
Den kompletten Bildschirm (-Hintergrund) bezeichnet man als **root window**.

ROOT-WINDOW (SCREEN)



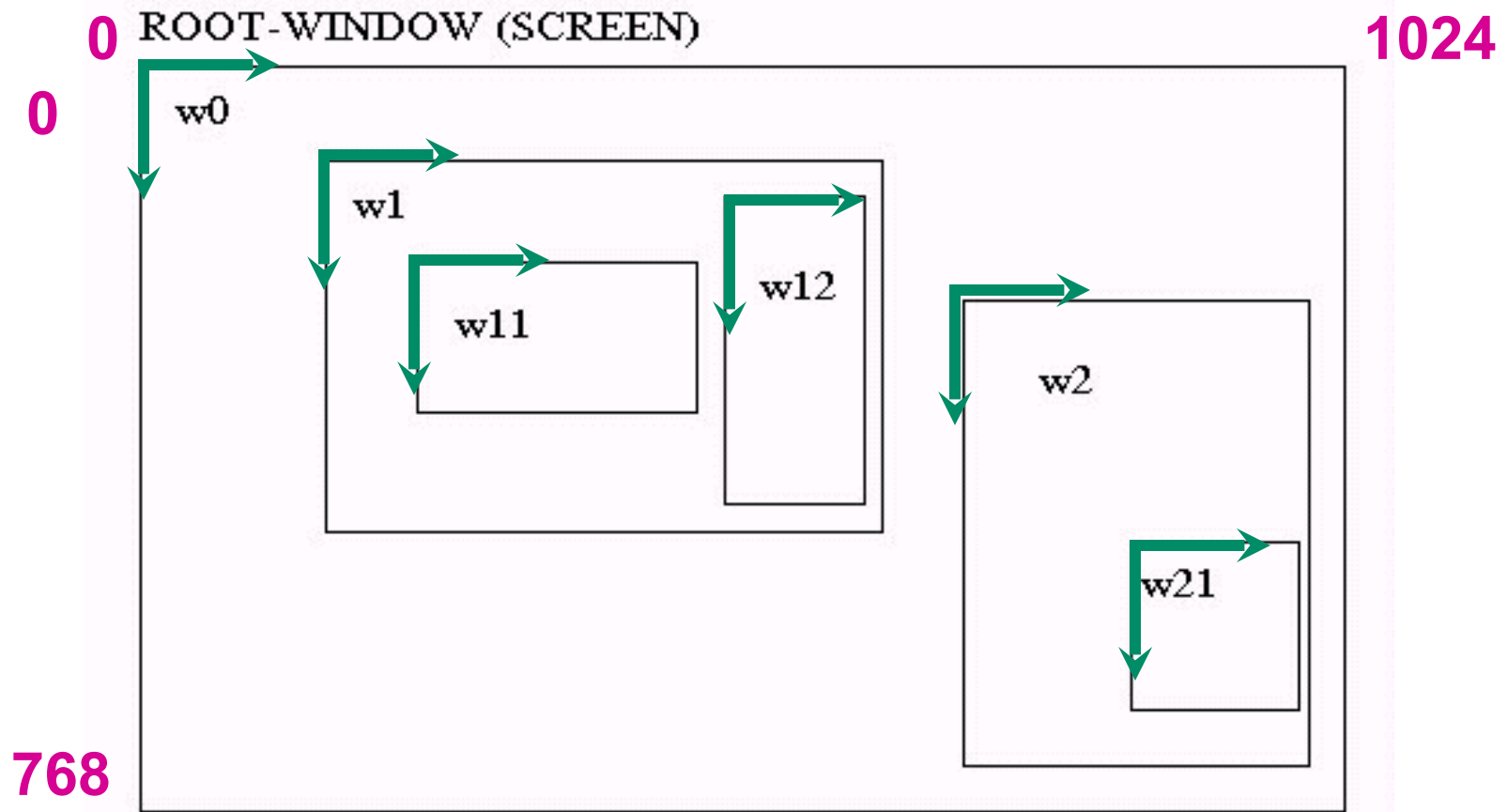
1.1 Grundlagen von Windowsystemen

Jedes Window besitzt ein eigenes Koordinatensystem.



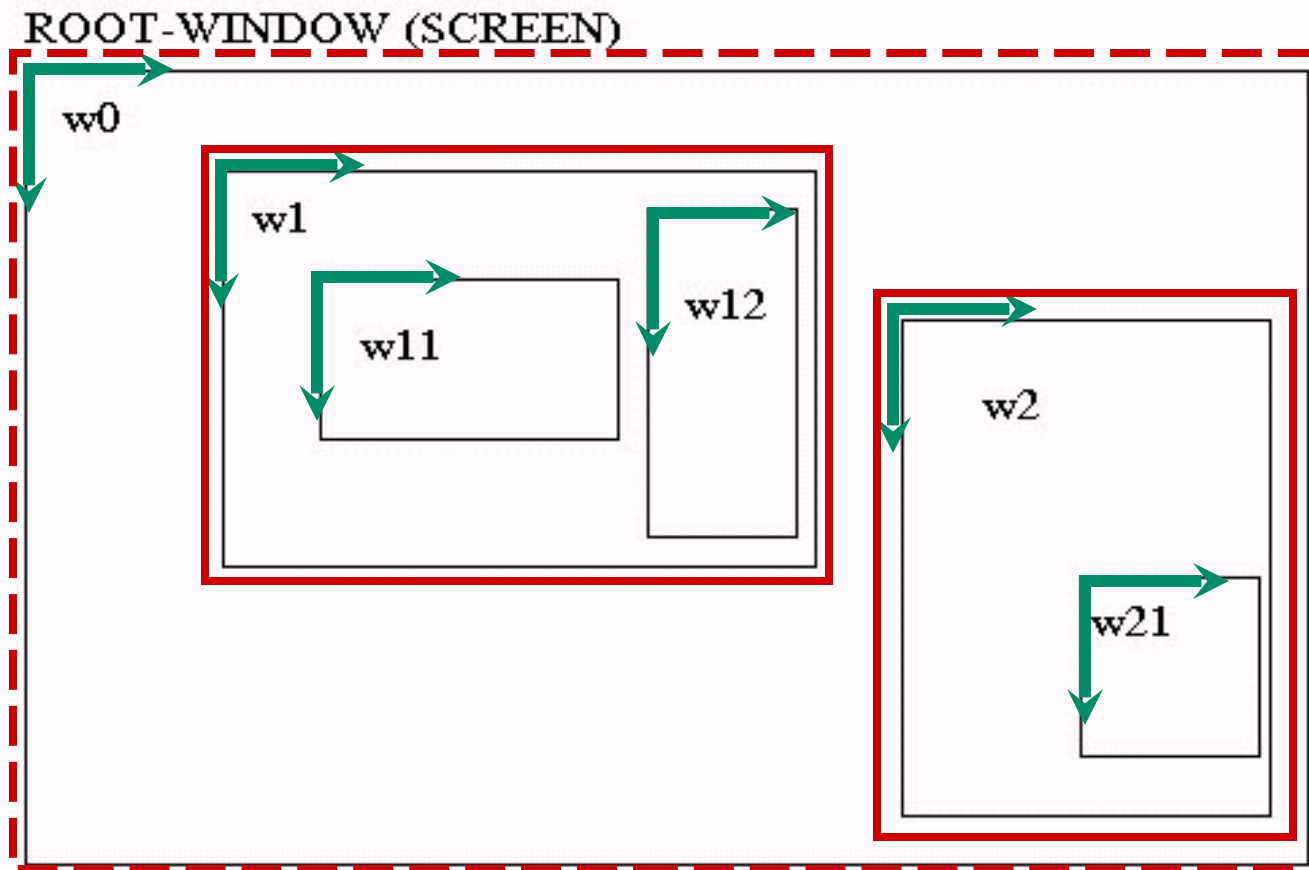
1.1 Grundlagen von Windowsystemen

Die Koordinatensysteme sind integer-wertig und repräsentieren Bildschirm-Pixel. Y wächst nach unten.



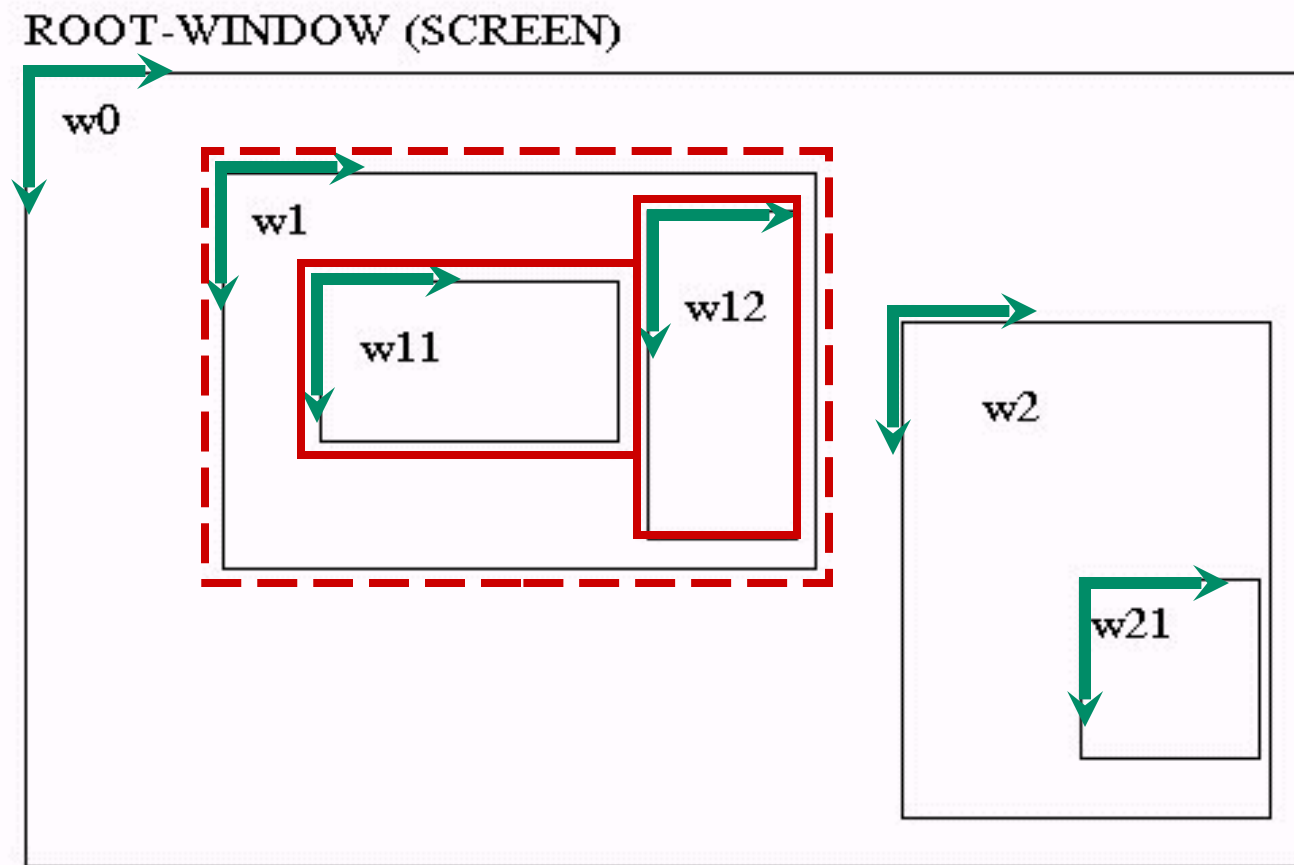
1.1 Grundlagen von Windowsystemen

Windows stehen zueinander in einer Parent-Child-Beziehung.



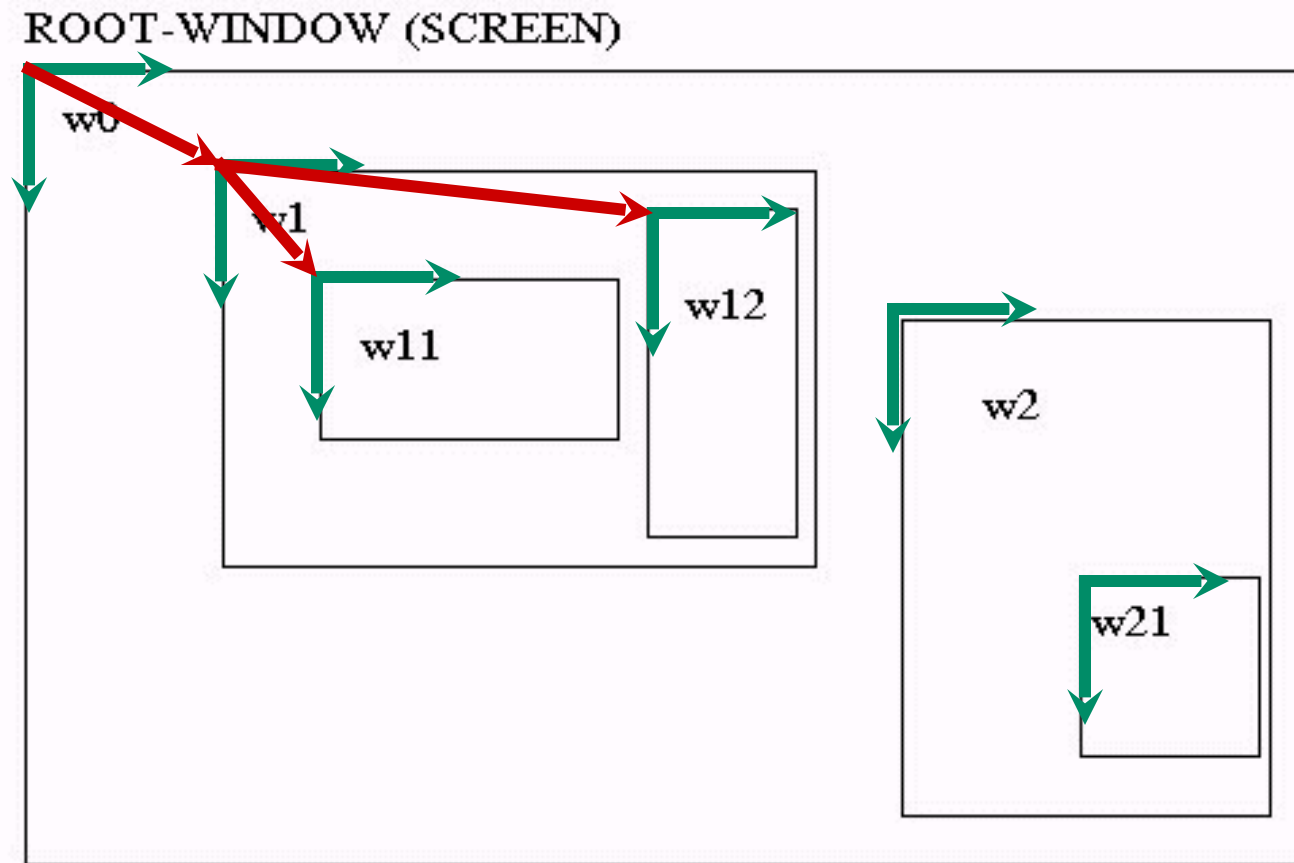
1.1 Grundlagen von Windowsystemen

Die Parent-Child-Beziehung ist rekursiv definiert.



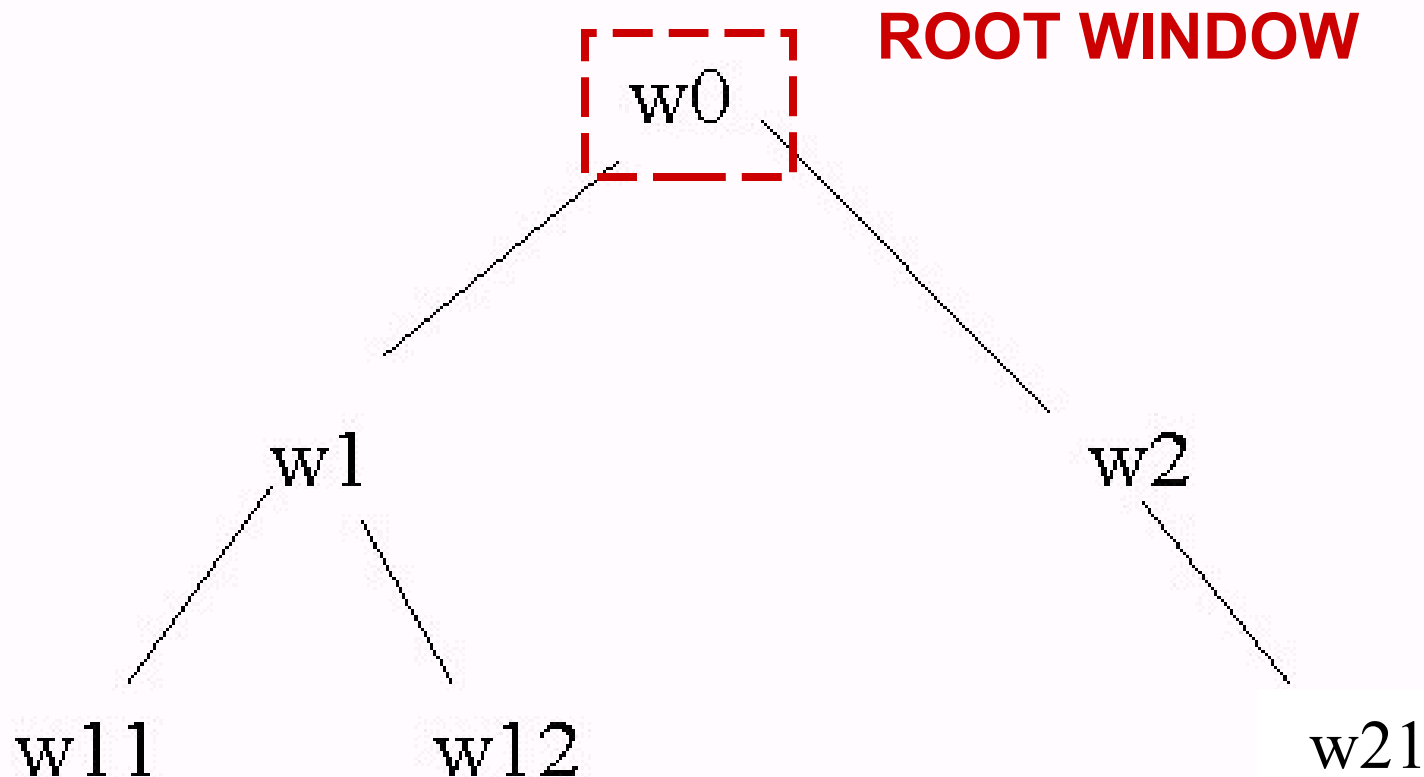
1.1 Grundlagen von Windowsystemen

Jedes Window besitzt eine Position in Bezug auf sein Parent.



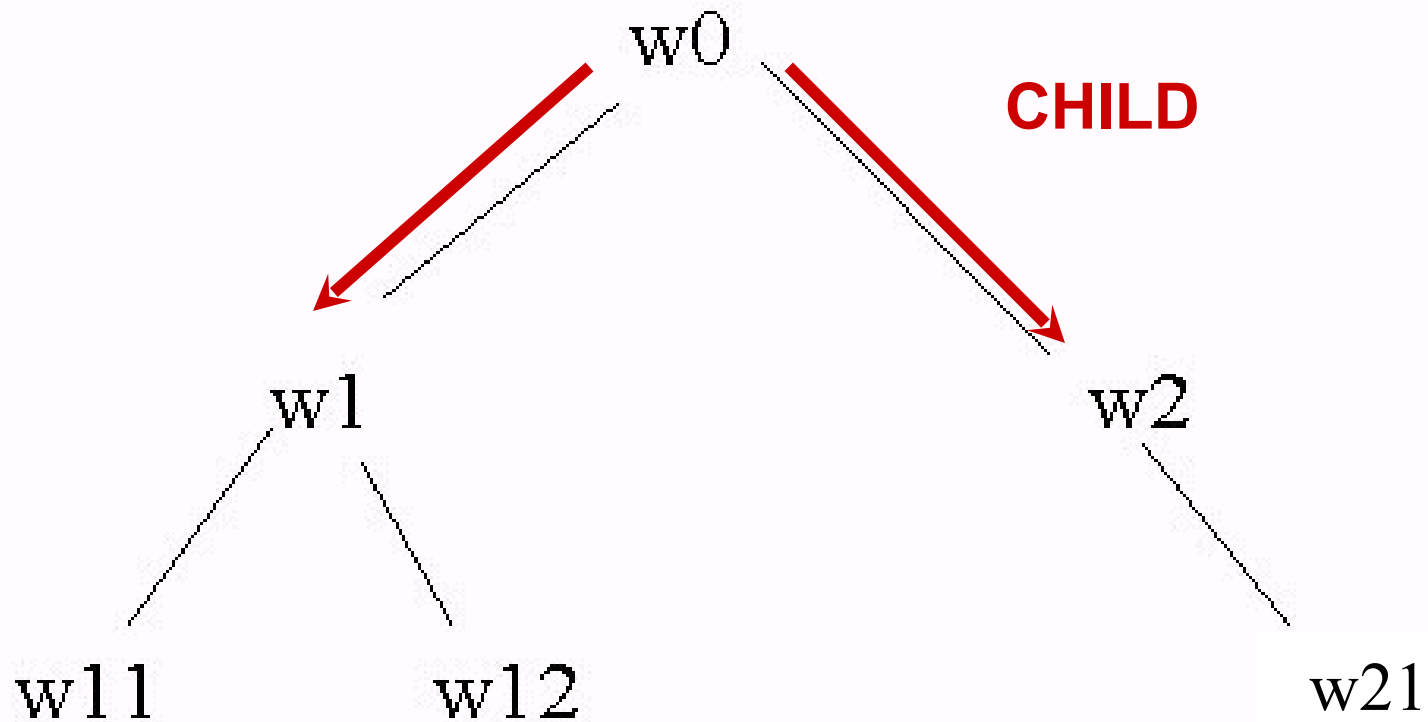
1.1 Grundlagen von Windowsystemen

Die Parent-Child-Beziehungen definieren eine Hierarchie.
Das root window ist das oberste Window in der Hierarchie.



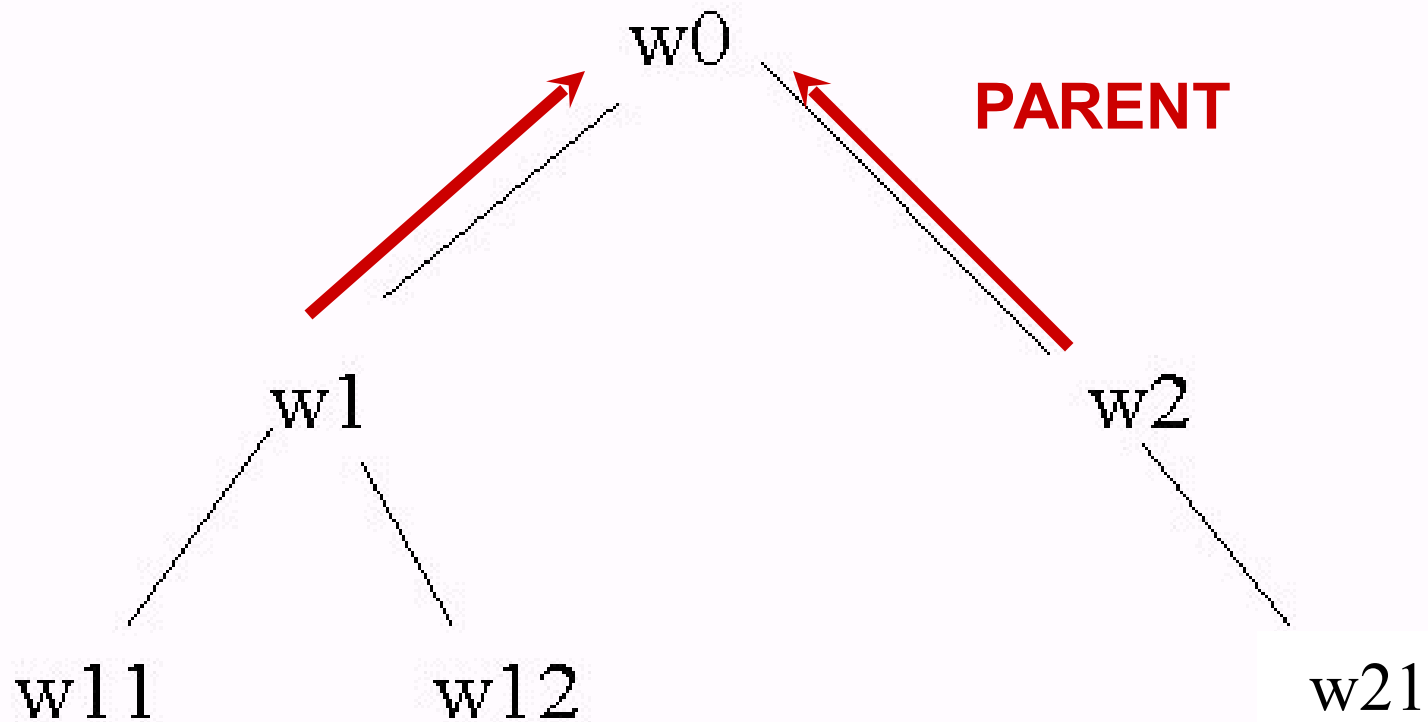
1.1 Grundlagen von Windowsystemen

Child-Relationship



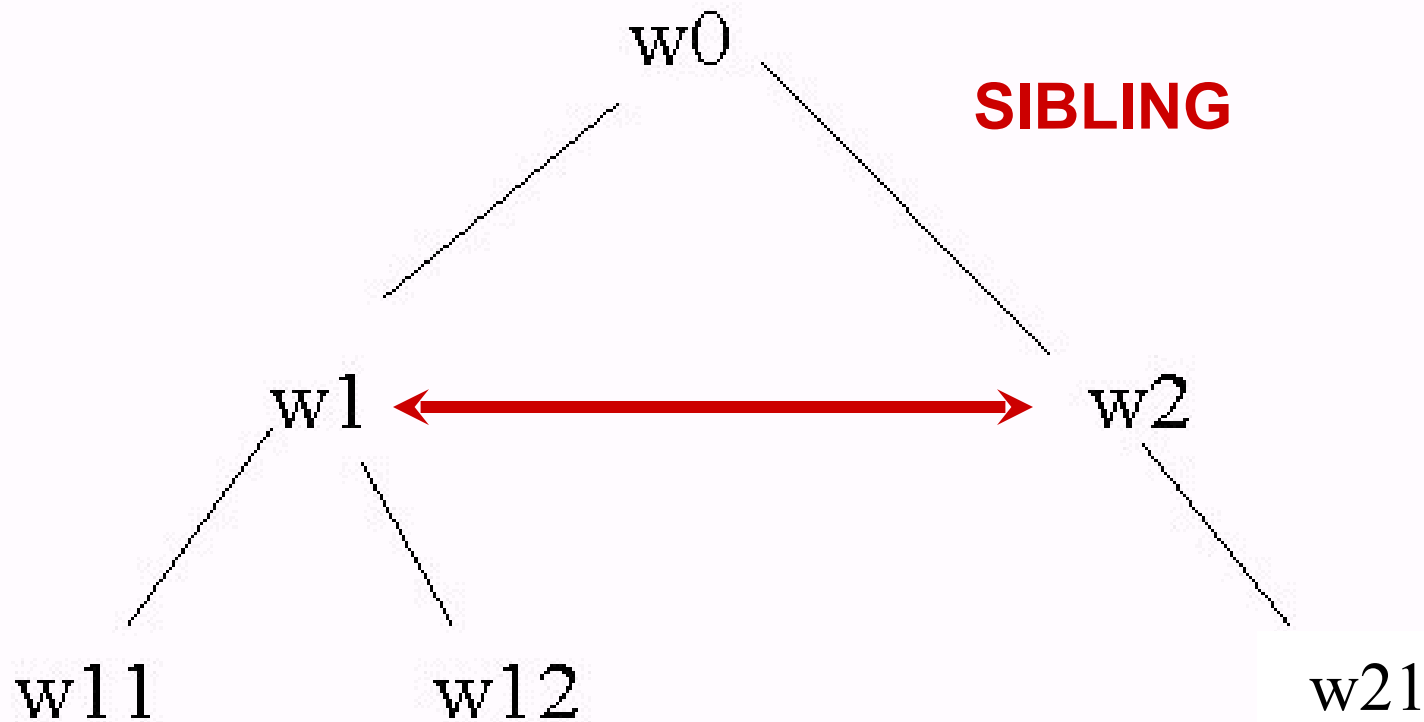
1.1 Grundlagen von Windowsystemen

Parent-Relationship



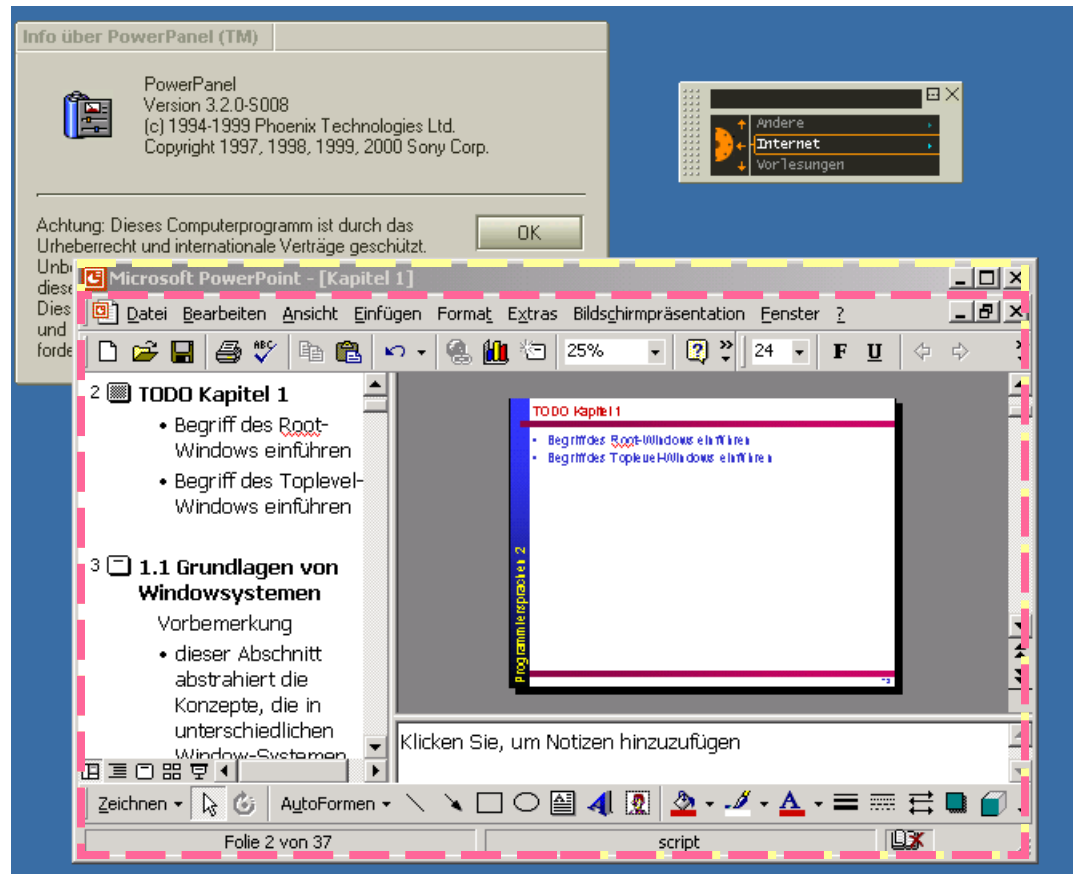
1.1 Grundlagen von Windowsystemen

Sibling-Relationship



1.1 Grundlagen von Windowsystemen

Als **toplevel window** bezeichnet man die obersten Windows einer Anwendung. Diese sind entweder direkte Kinder des root windows oder des **window managers**.



1.2 Window-Operationen

Vorbemerkung

- Alle folgenden Datentypen und Methoden sind in Pseudocode geschrieben
- Sie entsprechen keinem der realen Window-Systeme sondern abstrahieren gemeinsame Eigenschaften

Grundlegende Operationen sind

- Windows erzeugen, löschen
- Window-Hierarchie verändern bzw. abfragen
- Window-Eigenschaften setzen bzw. abfragen
- Window sichtbar bzw. unsichtbar machen
- Graphik-Ausgabe
- Ereignisbehandlung (Graphik-Eingabe)

1.2 Window-Operationen

Erzeugen und Löschen

```
WINDOW create ( WINDOW parent );  
void destroy ( WINDOW window );
```

Hierarchie abfragen

```
WINDOW get_parent ( WINDOW window );  
W_List get_children ( WINDOW window );  
W_List get_siblings ( WINDOW window );
```

Parent verändern

```
void reparent ( WINDOW window,  
               WINDOW newparent );
```


1.2 Window-Operationen

Window-Eigenschaften verändern / abfragen

- Geometrie

```
void move ( WINDOW window, int x, int y );
```

```
void size ( WINDOW window, int width,  
           int height );
```

```
RECTANGLE get_size ( WINDOW window );
```

```
POSITION get_location ( WINDOW window );
```

Bemerkungen

- Die geometrischen Eigenschaften `x`, `y`, `width` und `height` sind integer (Pixel-) Koordinaten
- Die Position ist die im Koordinatensystem des Parents

1.2 Window-Operationen

Window-Eigenschaften verändern / abfragen

- Graphische Attribute

```
void set_attr ( WINDOW window,  
               ATTR attributes );  
ATTR void get_attr ( WINDOW window );
```

Bemerkungen

- Graphische Attribute beinhalten Eigenschaften wie Hintergrundfarbe oder –Pattern, Rahmen, Rahmenfarbe, Transparenz / Opacity, etc. Diese Eigenschaften sind z.T. sehr unterschiedlich in den unterschiedlichen Window-Systemen (Einführung anhand von AWT/Swing später)

1.2 Window-Operationen

Window sichtbar / unsichtbar machen

```
void show ( WINDOW window );  
void hide ( WINDOW window );
```

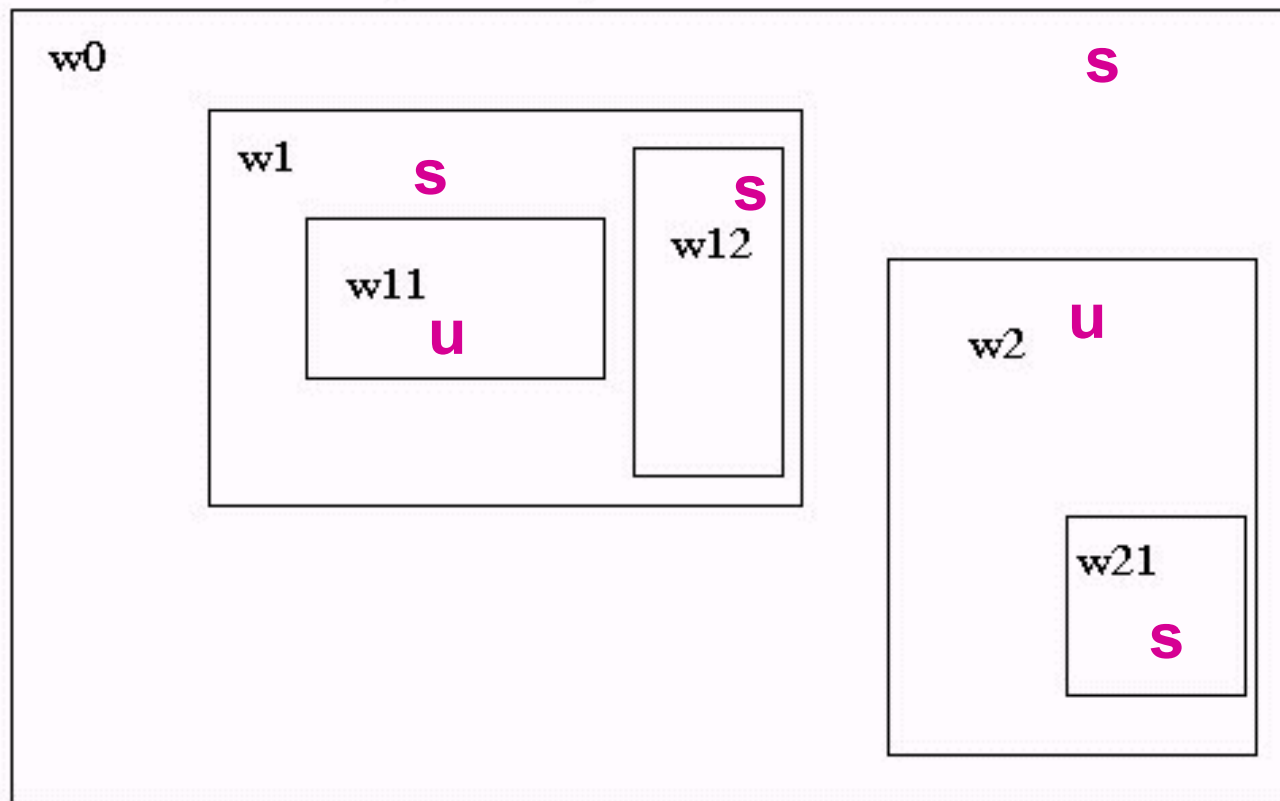
Bemerkungen

- Das Sichtbar-machen eines Windows bedeutet, daß es auf dem Bildschirm erscheint (sofern alle seine Parents sichtbar sind, siehe nächste Folie)
- Das hat nichts damit zu tun, daß ein Window wieder für den Nutzer sichtbar wird, wenn er es hinter einem anderen Toplevel-Window mittels Window-Manager hervor holt (siehe 2 Folien weiter)

1.2 Window-Operationen

s := sichtbar, **u** := unsichtbar

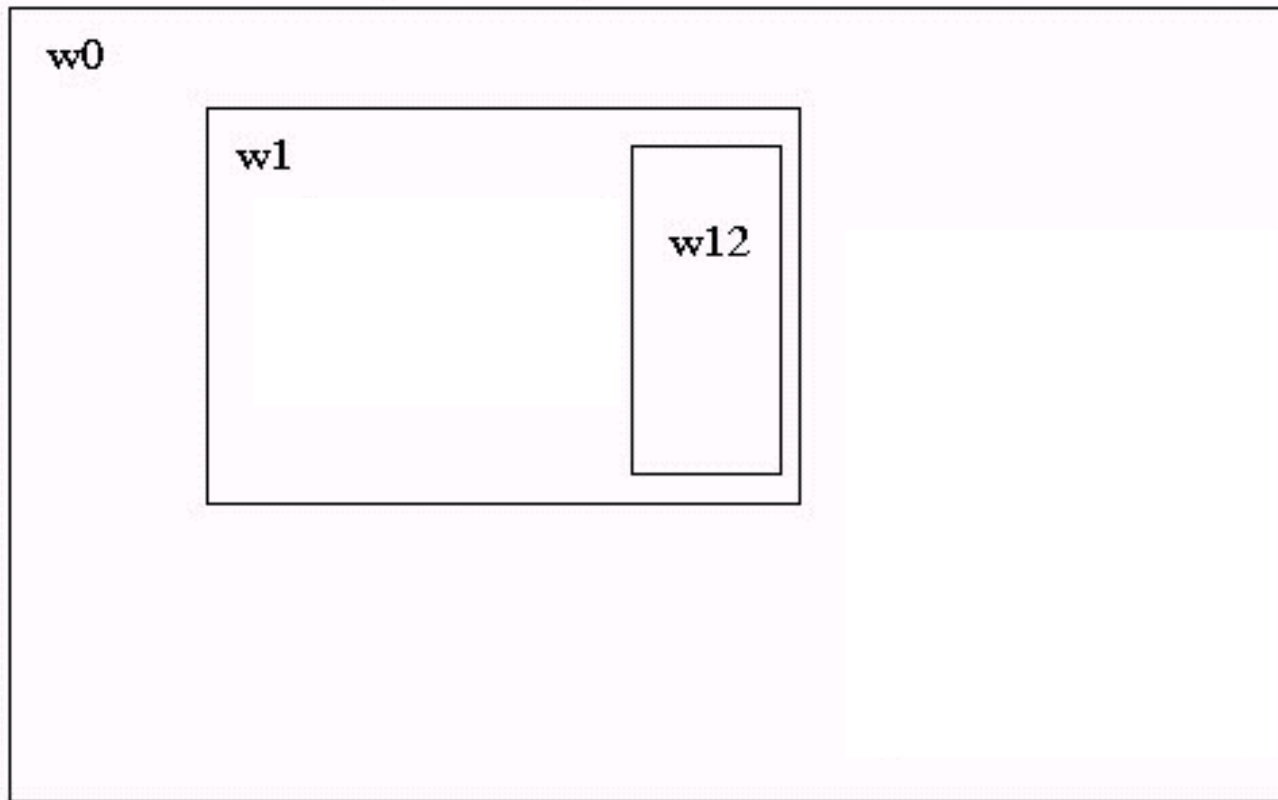
ROOT-WINDOW (SCREEN)



1.2 Window-Operationen

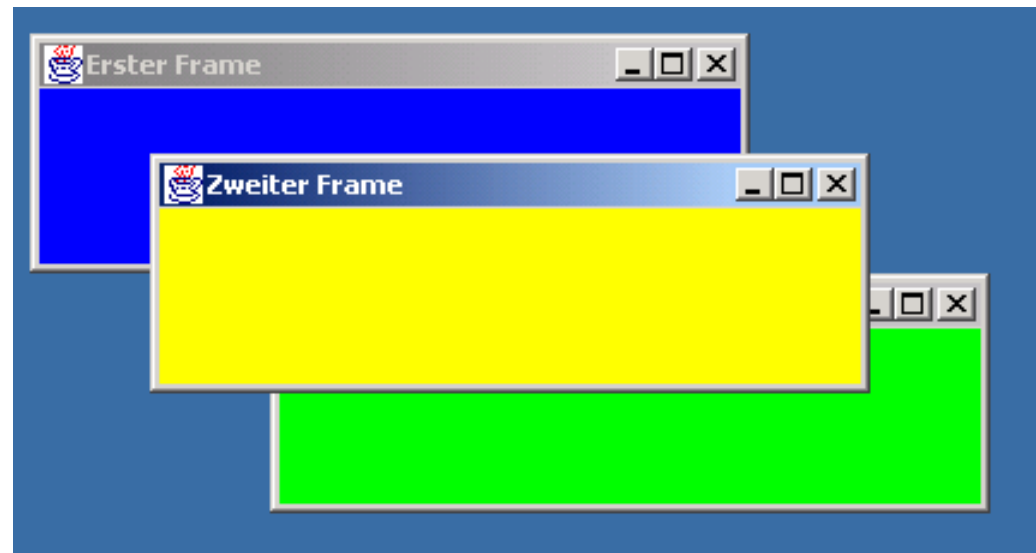
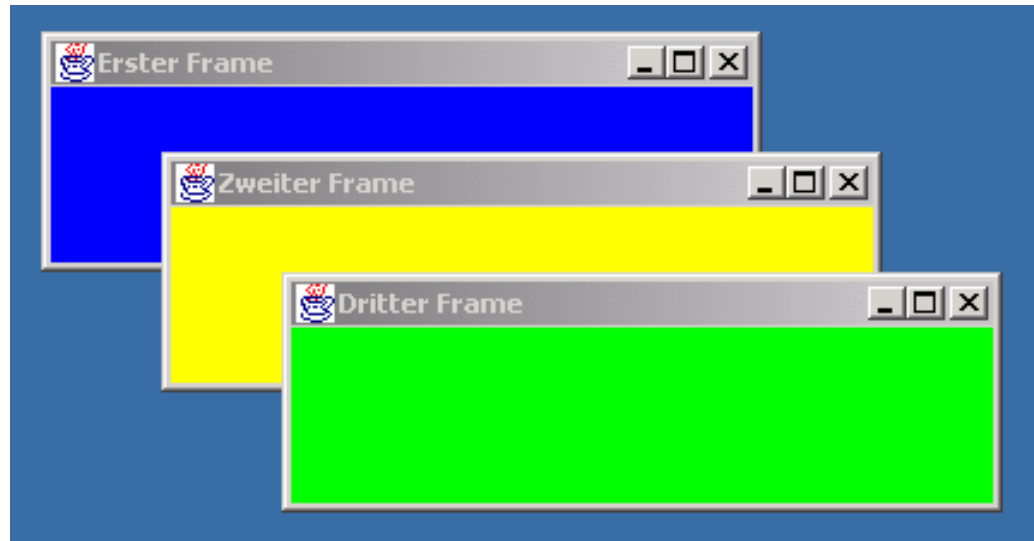
Sichtbarkeit: Ergebnis

ROOT-WINDOW (SCREEN)



1.2 Window-Operationen

Veränderung der
Stacking-Order
durch eine
Window-Manager-
Operation

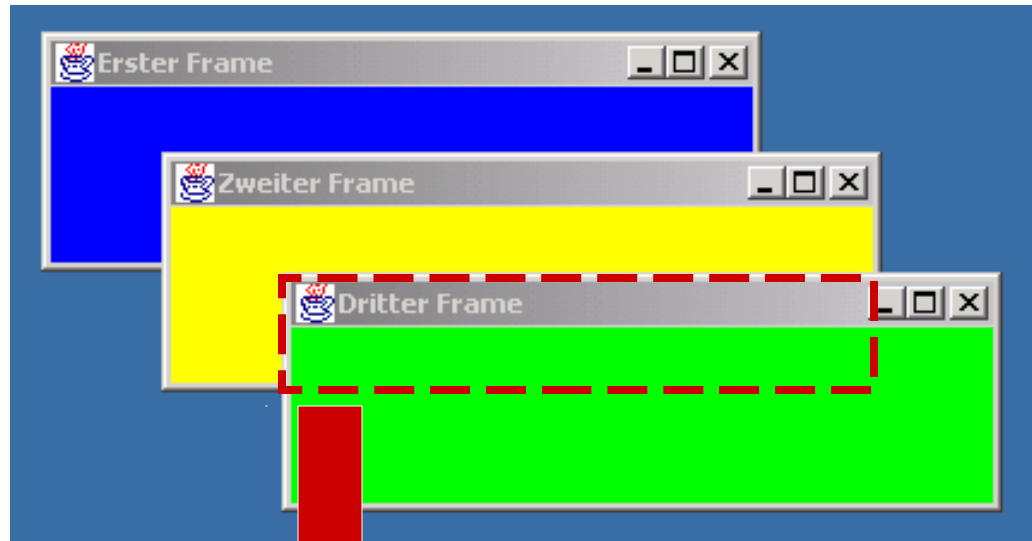


1.2 Window-Operationen

Invalide Bereiche

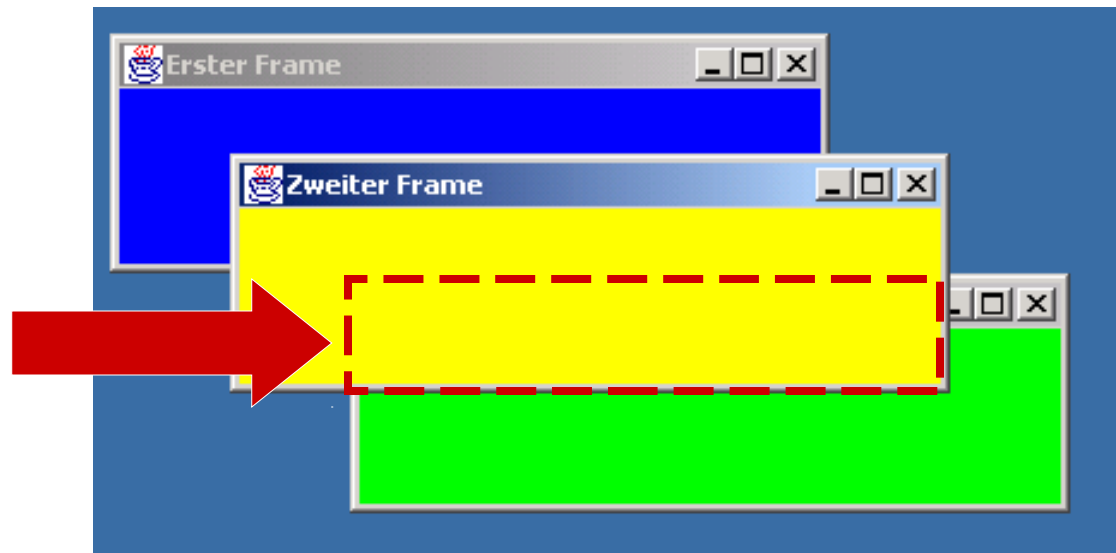
Diese führen zu einem Redraw-Ereignis

Demo: `Frame1Demo2.java`



Aufruf der redraw-Methode des Windows

Reparieren des Ausschnitts (in diesem Fall Füllen mit Hintergrundfarbe)



1.2 Window-Operationen

Graphik-Ausgabe

```
void draw ( WINDOW window,  
           PRIMITIVE primitive );
```

Bemerkungen

- Die Window-Systeme unterscheiden sich stark in den Primitiven, die sie für die Graphik-Ausgabe anbieten. I.d.R. beschränken sie sich jedoch auf **einfache** Primitiven und graphische Attribute (Ausnahme: AWT-2D):
 - Line, Rectangle, Polyline, Arc, Text, Image
 - Outline, Dotted, Dashed, Solid Fill, Pattern Fill
- Grund: Focus auf User-Interface-Umgebung
- Die Graphik-Ausgabe ist immer durch die Größe des Windows beschränkt (Clipping).

1.2 Window-Operationen

Ereignis-Selektion (Graphik-Eingabe)

```
void select ( WINDOW window,  
             EVENT_TYPE event );
```

Bemerkungen

- Will man Graphik-Eingabe nutzen, so muß man auf dem jeweiligen Window die gewünschten Ereignisse auswählen. Im Default sind i.d.R. keine Ereignisse ausgewählt
- Die Auswahl eines Ereignisses auf einem Window bedeutet, daß eine **Meldung** an das auswählende Programm geschickt wird. Diese Meldung beinhaltet Informationen über das Ereignis

1.2 Window-Operationen

Ereignis-Typen

- Auch hier unterscheiden sich die Systeme
- Viele Ereignis-Typen sind gleich bzw. ähnlich, andere Ereignistypen sind sehr spezifisch für ein Window-System
- X-Windows besitzt z.B. über 30 Ereignis-Typen

Beispiele für Ereignistypen

KeyPress

KeyRelease

Redraw/Paint

MousePress

MouseRelease

Configure

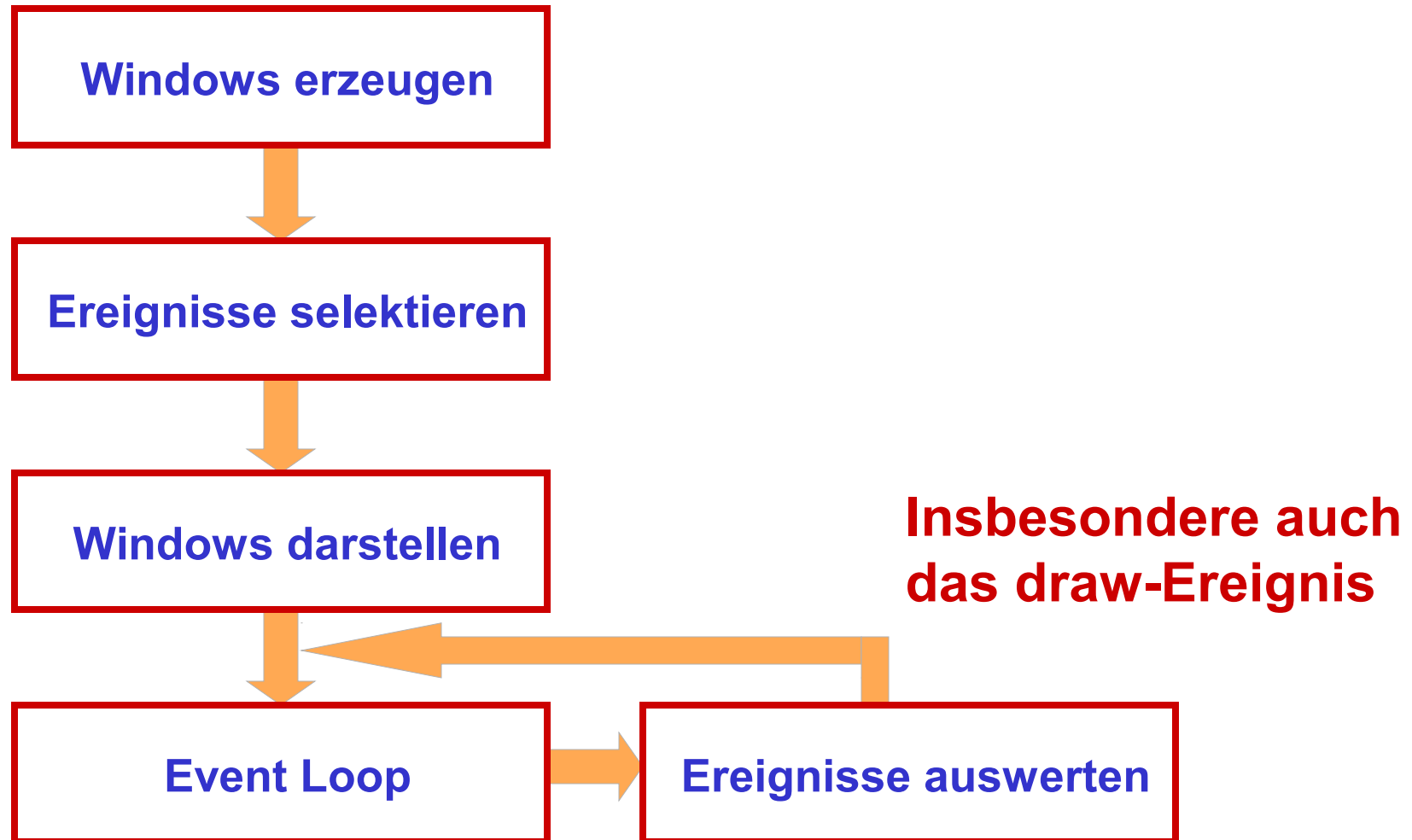
MouseMotion

EnterWindow

LeaveWindow

1.3 Ereignisorientierte Programme

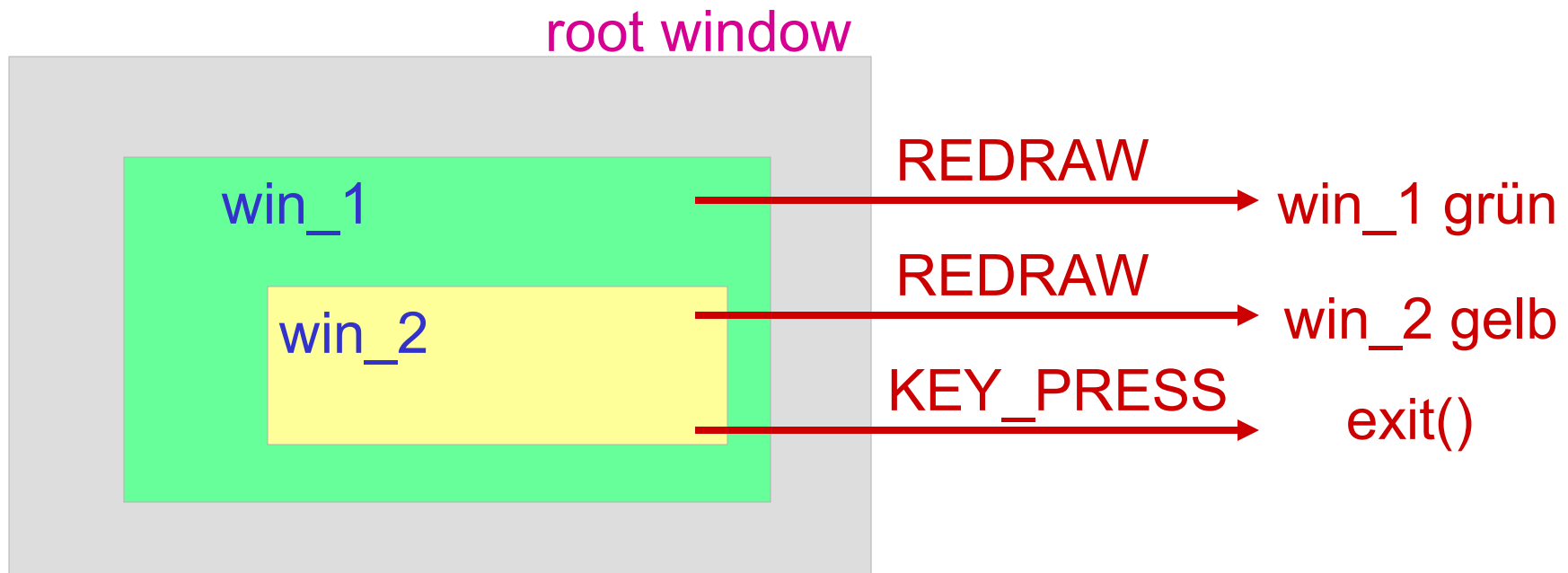
Grundaufbau eines ereignisorientierten Programms



1.3 Ereignisorientierte Programme

Beispiel

- 2 Fenster ineinander
- Beide Fenster sollen gefüllt werden
- Bei einem Tastatur-Ereignis im inneren Fenster soll das Programm beendet werden



1.3 Ereignisorientierte Programme

Programm mit zentraler Event-Loop, Pseudocode

```
win1 = create_window ( root_window );
win2 = create_window ( win1 );
select ( win1, DRAW_EVENT );
select ( win2, DRAW_EVENT | KEYPRESS_EVENT );
show ( win1 );
show ( win2 );    // EVENT: Datentyp bzw. Objektklasse
forever {         // der Ereignisse beschreibt.
    EVENT event = get_next_event ( );
    if ( event.window == win1 && event.type == DRAW )
        draw_window1(); // fülle win_1 grün
    draw analog für win2
    if ( event.type == KEYPRESS ) exit();
}
```

1.3 Ereignisorientierte Programme

Programm mit verborgener Event-Loop, Pseudocode

```
win1 = create_window ( root_window );
win2 = create_window ( win1 );
select ( win1, DRAW_EVENT, draw_window1 );
select ( win2, DRAW_EVENT, draw_window2 )
select ( win2, KEYPRESS_EVENT, key_press )
show ( win1 );
show ( win2 );
//----- end of main -----

void draw_window1()                key_press() {
    { // draw win1                  exit();
    }                                }

draw_window2() analog für win2
```

1.4 XWindows

Eigenschaften

- Basisbibliothek **X-Library** (Xlib): prozedural implementiert, prozedurales API
- Unterhalb der Xlib liegt das **X-Protocol**, dadurch verteilte Anwendbarkeit gegeben
- Austauschbarer Window-Manager (mwm, fwm, ...)
- **XToolkit** beinhaltet ein o-o Konzept (allerdings prozedural implementiert), das Basis für Toolkits ist (Widget-Konzept)
- Grundlage für höherwertige Toolkits, die gleichberechtigt verwendet werden können
- **Toolkits**: Motif, OpenLook, SGI-Toolkit
- Der X-Server verwaltet einen Arbeitsplatz, wobei dieser aus 1 Tastatur, 1 Maus und **mehreren Bildschirmen** bestehen kann

1.4 XWindows

Software-Schichten

Anwendungslogik

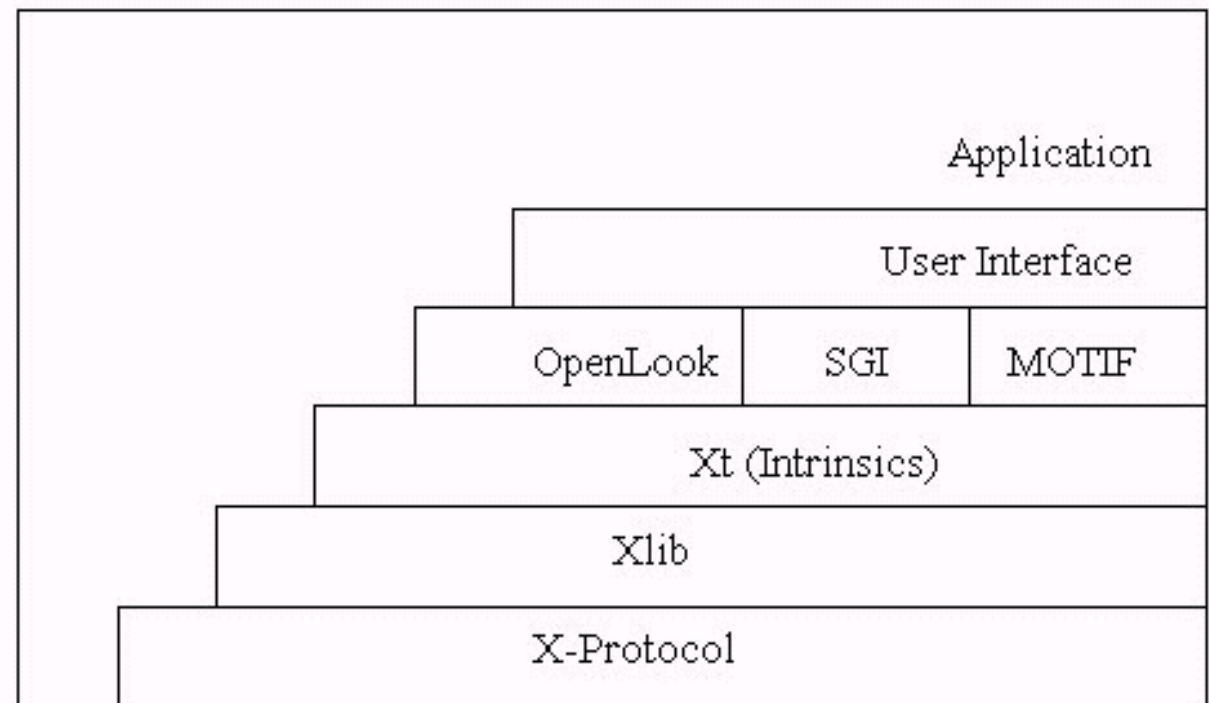
User Interface Logik

GUI Toolkits

O-O GUI-Grundlage

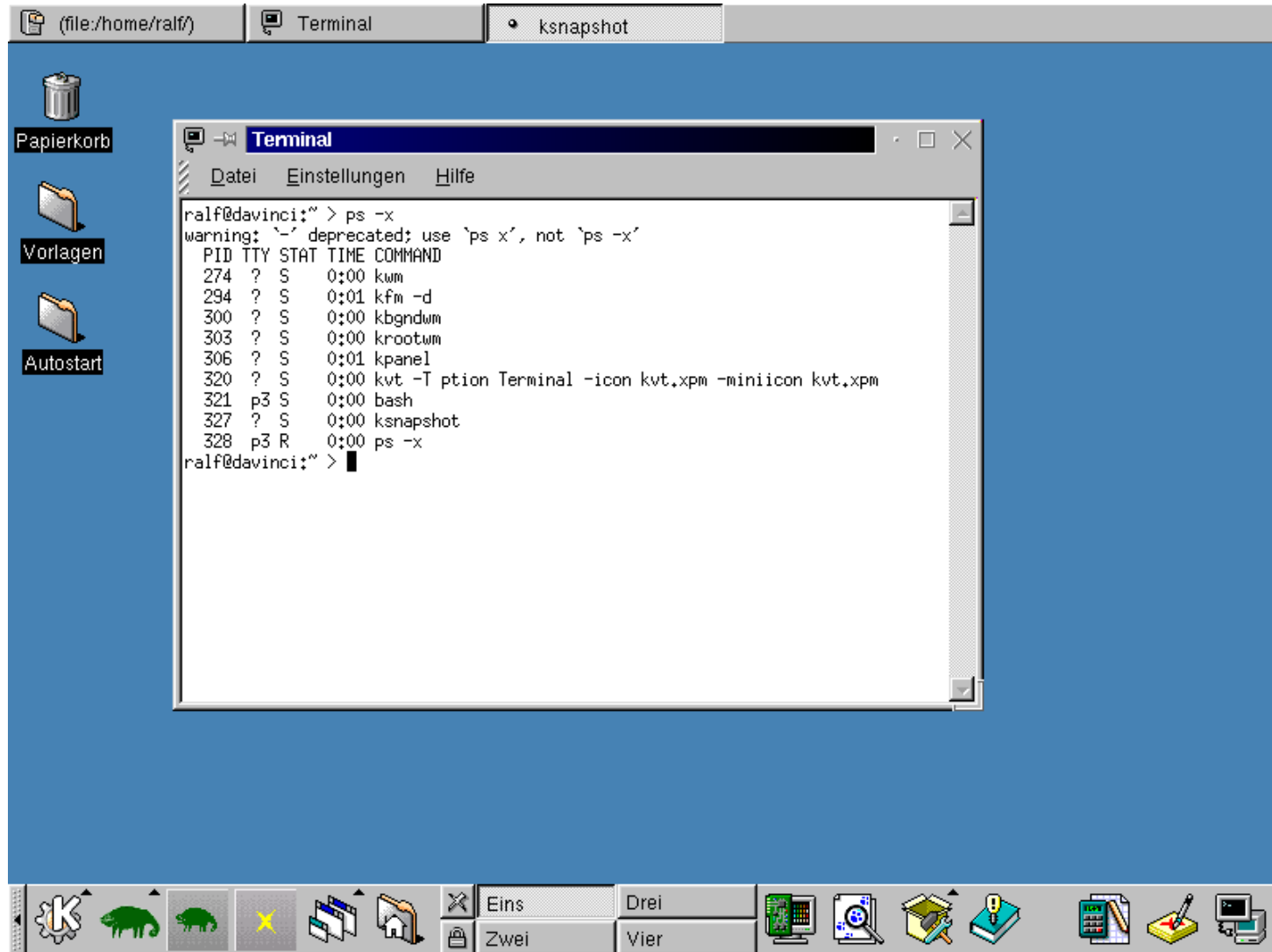
API

Verteilung



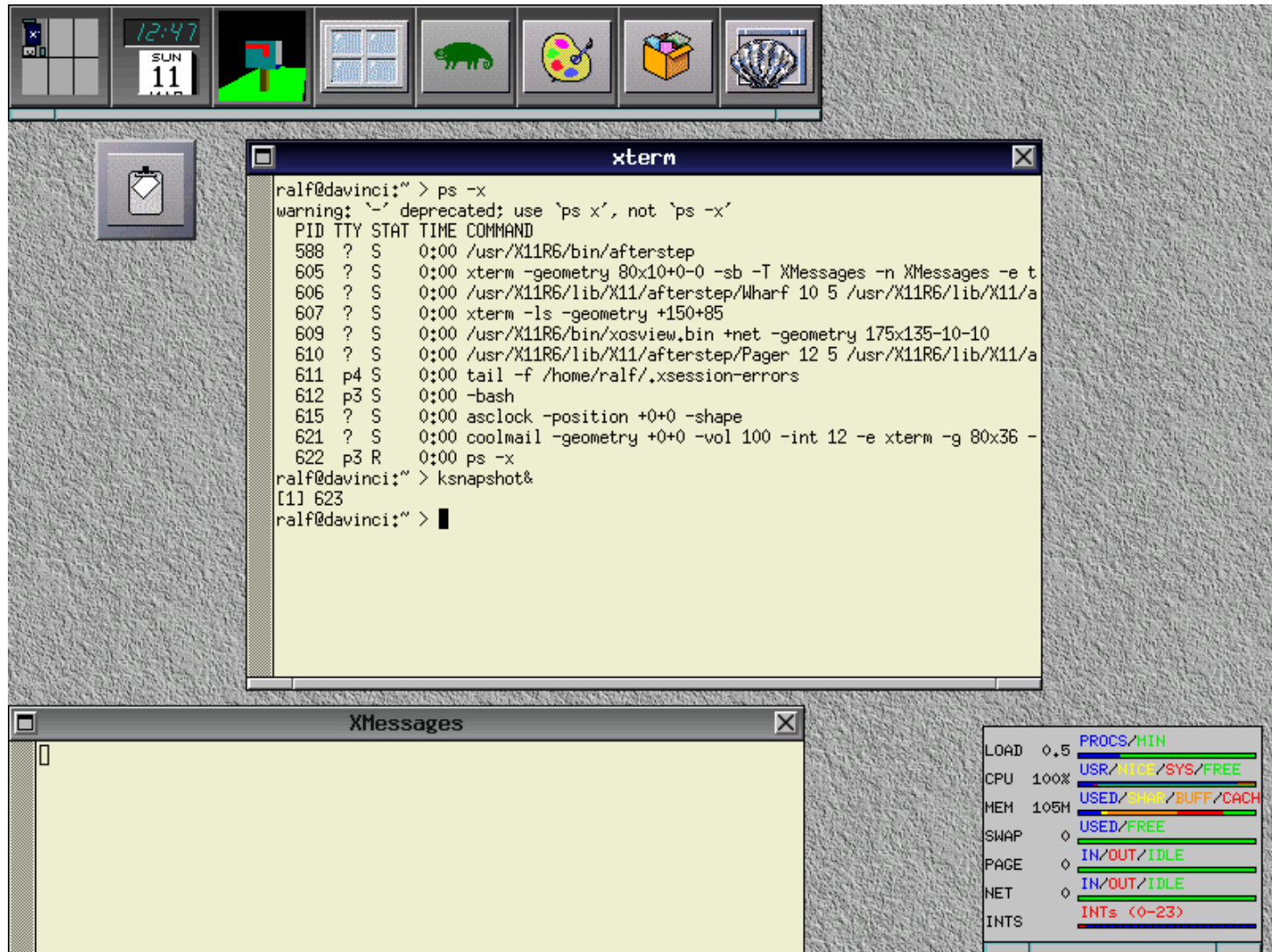
1.4 XWindows

Austauschbarer Window-Manager



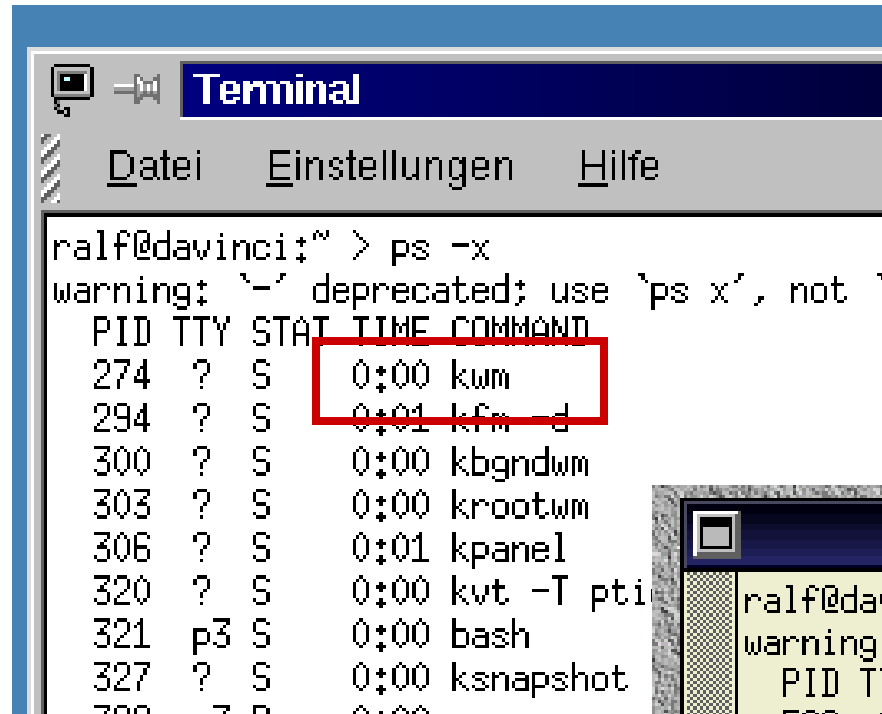
1.4 XWindows

Austauschbarer Window-Manager



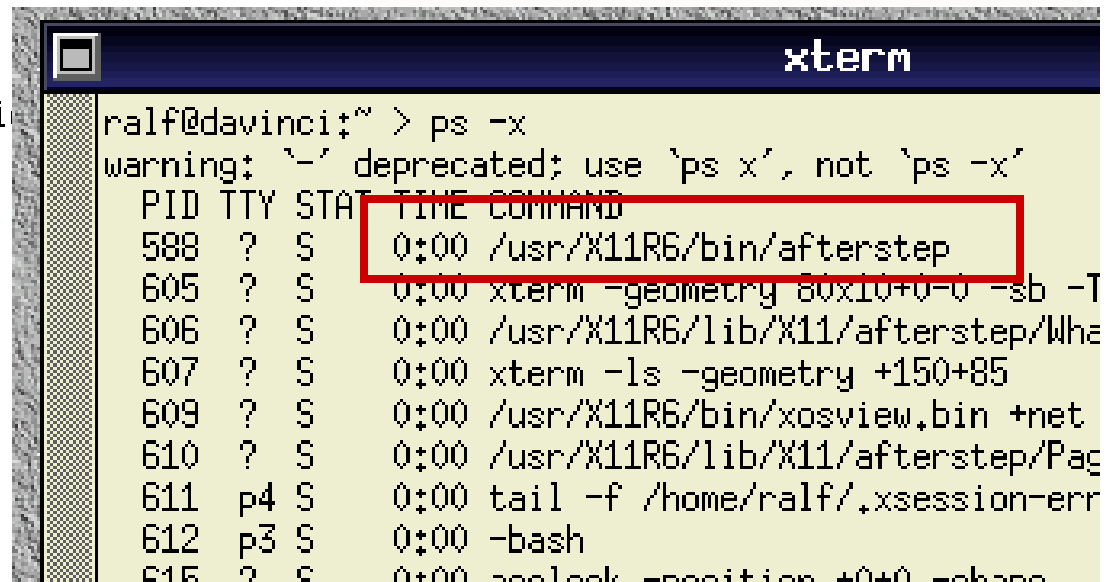
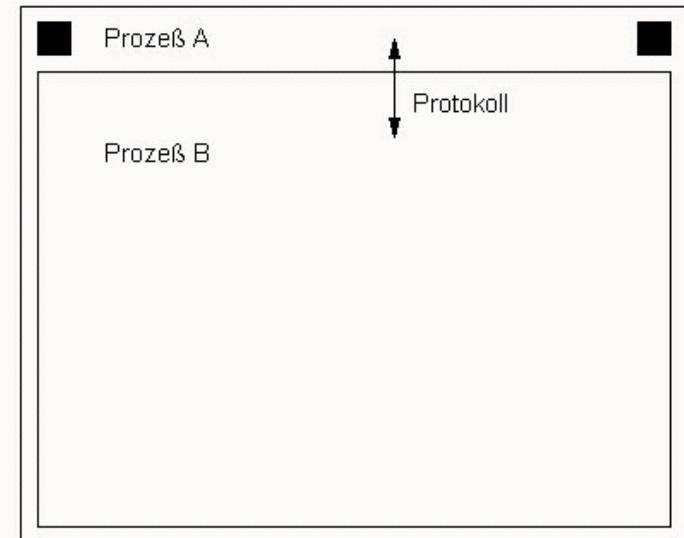
1.4 XWindows

Austauschbarer Window-Manager



A terminal window titled "Terminal" with a menu bar containing "Datei", "Einstellungen", and "Hilfe". The command prompt shows the user 'ralf' at 'davinci' running 'ps -x'. The output is a table of processes. The first two rows are highlighted with a red box.

PID	TTY	STAT	TIME	COMMAND
274	?	S	0:00	kum
294	?	S	0:01	kfm -d
300	?	S	0:00	kbgrndwm
303	?	S	0:00	krootwm
306	?	S	0:01	kpanel
320	?	S	0:00	kvt -T pti
321	p3	S	0:00	bash
327	?	S	0:00	ksnapshot
...



An xterm window titled "xterm" showing the same 'ps -x' command output as the terminal window. The first row of the process list is highlighted with a red box.

PID	TTY	STAT	TIME	COMMAND
588	?	S	0:00	/usr/X11R6/bin/afterstep
605	?	S	0:00	xterm -geometry 80x10+0-0 -sb -T
606	?	S	0:00	/usr/X11R6/lib/X11/afterstep/Whe
607	?	S	0:00	xterm -ls -geometry +150+85
609	?	S	0:00	/usr/X11R6/bin/xosview.bin +net
610	?	S	0:00	/usr/X11R6/lib/X11/afterstep/Pag
611	p4	S	0:00	tail -f /home/ralf/.xsession-err
612	p3	S	0:00	-bash
615	?	S	0:00	ee-lock -position +0+0 -shape

1.4 XWindows

Client-Server-Modell

- Unter der Xlib liegt ein Netzwerk-Protokoll, das sog. X-Protocol
- Der Graphikstrom kann damit auf andere Workstations umgeleitet werden
- Graphik-Ein- und Ausgabe werden dann umgesteuert

```
miro# xhost +
```

```
miro# telnet chagall
```

```
user: snoopy
```

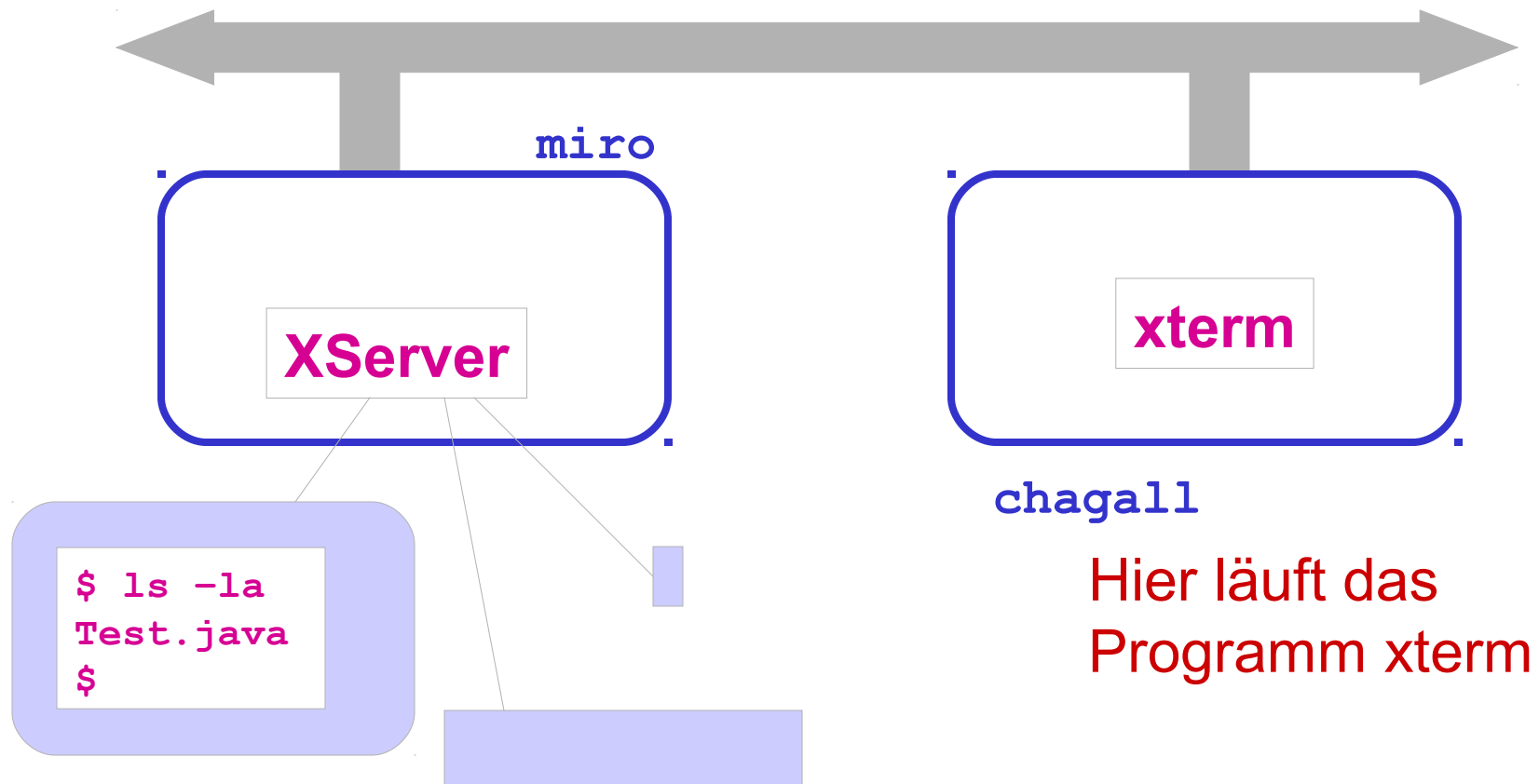
```
password: ****
```

```
chagall# export DISPLAY = miro:0.0
```

```
chagall# xterm&
```

1.4 XWindows

Client-Server-Modell



Hier läuft die Graphik Ein-/Ausgabe von xterm

1.4 XWindows

Programmaufbau

```
int main ( int argc, char* argv[] )
{ int cont=1;

  if ( ( display = XOpenDisplay ( NULL ) ) == NULL ) {
    printf ("Error opening display"); exit(1); }
  screen  = DefaultScreenOfDisplay ( display );

  top = XCreateSimpleWindow ( display,
    DefaultRootWindow(display), 10,10,400,400,0,
    BlackPixelOfScreen(screen),
    WhitePixelOfScreen(screen) );
  XMapWindow ( display, top );

  XSelectInput ( display, top,
    PointerMotionMask|ButtonPressMask );
```

1.4 XWindows

-- FORTSETZUNG --

```
while ( cont==1 ) {
    XNextEvent (display, &event);
    switch (event.type) {
        case MotionNotify : AKTION-1(); /*Programmcode*/
            break;
        case ButtonPress   : AKTION-2(); /*Programmcode*/
            /*z.B cont=0; für Programmende*/
            break;
        case Expose        : NEUZEICHNEN DES/DER WINDOWS
    }
}
}
```

Eigenschaften

- Generisches Toolkit, läuft unter unterschiedlichen Window-Systemen
- Im Prinzip verteilt anwendbar durch Java-Applet-Konzept, allerdings ist die Funktionsweise vollkommen anders
- Im Unterschied zu XWindows wird bei der Verteilung der Programmcode transportiert und nicht der Graphik-Strom
- Voll objekt-orientiert implementiert
- Nutzt den nativen Window-Manager und das native Window-Toolkit
- Höherwertiges GUI-Toolkit nur begrenzt in AWT enthalten, statt dessen wird das seit Java 1.1 verfügbare **Swing-Toolkit** verwendet