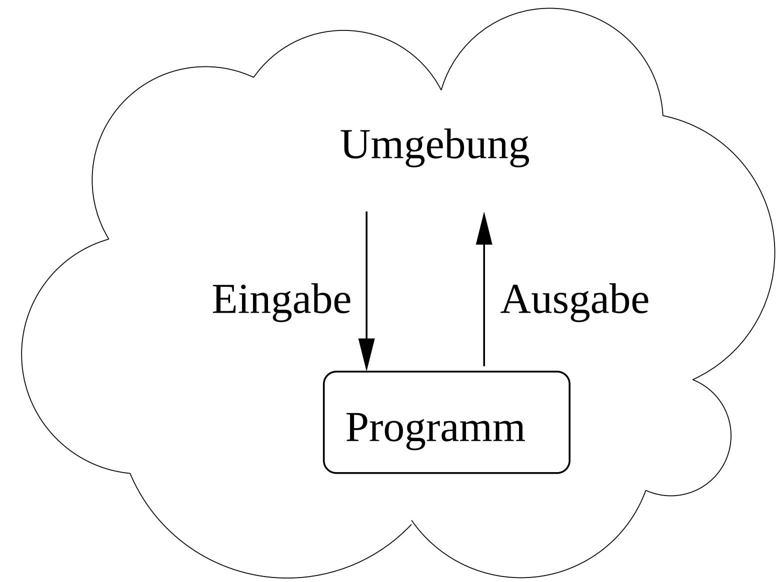
GUIs und reaktive Programmierung

Und heute wieder. . .

- Reaktive Programmierung
  - Abschied vom sequentiellen 1–2–3.
- Programmierung grafischer Benutzerschnittstellen (GUIS) in Haskell:
  - HTk — Tcl/Tk in Haskell
  - wxHaskell — wxWidgets in Haskell

Reaktive Programmierung

Sequentiell



Reaktiv

Beispiele für reaktive Programme: GUIs, Applets, eingebettete Systeme

Reaktive Programme

Herkömmliches sequentielles Programmiermodell basiert auf

- Sequentielle Zustandsübergänge
- Ein/Ausgabe durch Seiteneffekte

Hier unzureichend:

- Grundlegende Operationen: auf Eingabe warten, Ausgabe senden
- Mehrere Eingaben gleichzeitig, Reihenfolge der Ausgabe nicht spezifiert, Nebenläufigkeit

Problem: Realisierung reaktiver Programme in sequentieller Sprache

Realisierung reaktiver Programme

- Eventschleife im Program (X Windows)
- Externe Eventschleife:
  - Callbacks (Tcl/Tk)
  - Signale (Qt, GTK+)
  - Event handlers (Java: Applets, Swing, AWT)
- Komponierbare Events (HTk)

Zentrale Eventschleife: Xlib

```
#include <X11/Xlib.h>                    #include <X11/Xutil.h>
int main(int argc, char **argv) {
 Display *dply; XEvent event;
 /* draw some graphics here */
 for (;;) {   XNextEvent(dply, &event);
   if (event.type == ButtonPress) {
       printf("Button pressed at (%d, %d)\n",
               event.xbutton.x, event.xbutton.y);
     } else if (event.type == KeyPress) {
       KeySym ks;
       ks= XLookupKeysym(&event.xkey, 0);
       printf("Key %04x (%s) pressed.\n", ks, XKeysymToString(ks)); }
   XFlush(dply); }}
```

Zentrale Eventschleife

Vorteile:

- Direkte Kontrolle
- Schnelligkeit

Nachteile:

- umständlich, fehleranfällig
- Nicht kompositional

Eventbehandlung mit Callbacks, Signals etc

- Eine Eventschleife außerhalb des reaktiven Programms
- Programm implementiert Prozeduren, die Events verarbeiten.
- Prozedurparameter: Event-Daten, zum Beispiel gedrückte Taste oder Mausknopf; Zusatzinformationen wie Mausposition.
- Im Beispiel oben:

```
typedef struct {int button; int x_pos; int y_pos} mouse_info;

void handle_buttonPress(mouse_info i);
void handle_keyPress(char key);
```

- Unterschiede: Art der Bindungsbehandlung

Callbacks in Tcl/Tk

- Tcl/Tk: *user interface library and programming language*
- Tcl (tool command language): String-basierte Kommandosprache
- Tk (graphical toolkit): Bibliothek von Funktionen zum Erstellen von GUIs
- Philosophie: Leichtgewichtige Skriptsprache, schnelle Entwicklung von GUIs durch das Zusammenkleben von Applikationen (glueware)
- Plattformunabhängig: X, Windows, MacOS, . . .
- Kein native look&feel

Callbacks in Tcl/Tk

- Bindung von Kommandos an Events durch `bind`:

```
button .b -text Test
bind .b <Enter> {puts "Button Entered!"}
bind .b <ButtonPress-2> { puts "Button pressed at (%x, %y)" }
pack .b
```

- Völlig ungetypt:
  - Funktionen höherer Ordnung durch Quoting
  - Implizite Parameterübergabe über %-Substitution
- Ablauf:
  - Aufbau des Interface
  - Festlegung des dynamischen Verhaltens
  - Interface darstellen und externe Eventschleife aufrufen

Event Handling in Java

- Eventverabeitende Methoden leben in Unterklassen der `Listener`-Klassen:
  - `interface ActionListener` für Aktionen (z.B. Knopfdrücke);
  - `interface MouseListener` für Mouse-Events.
- In `Listener` werden Event-bearbeitende Methoden überladen:

```
interface ActionListener {
 public void actionPerformed(ActionEvent e)
 }
```

- Listener an Event-Quellen binden:

`addActionListener(Object o)`, `addMouseListener(Object o)`

Beispiel: ein einfaches Applet

```
public class Beeper extends JApplet
                    implements ActionListener {
    JButton button;
    public void init() {
        button = new JButton("Click Me");
        getContentPane().add(button, BorderLayout.CENTER);
        button.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e) {
        Toolkit.getDefaultToolkit().beep();
    }
}
```

Zusammenfassung: Callbacks und Event Handling

Vorteile:

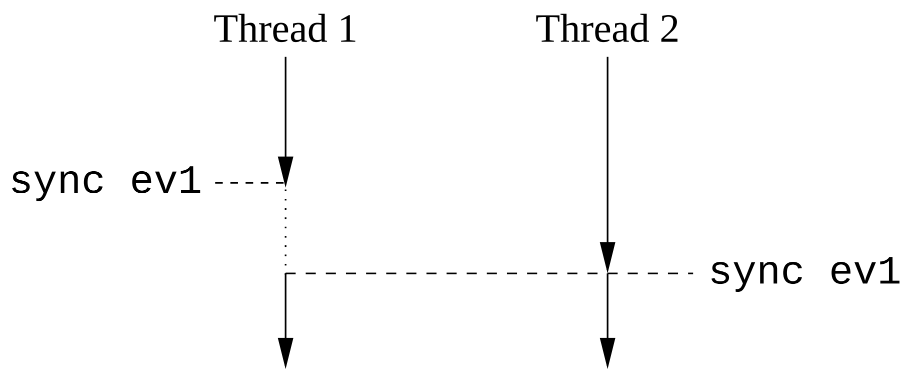
- abstrakter als explizite Eventschleife
- kompositional innerhalb des Toolkits (e.g. zwei Swing-Programme)

Nachteile:

- Mangelnde Modularität und Kompositionalität, weil es immer nur eine Eventschleife geben kann
- Zustandsbasiert

Funktionale reaktive Programmierung: `Events`

- Eine grundlegende Operation: Synchronisation



- Beschreibung der Synchronisationspunkte: `Event a`
- Kombination von Events mit +> (Auswahl)

```
type Event a
sync    :: Event a -> IO a
(+>)    :: Event a -> Event a -> Event a
(>>>=) :: Event a -> (a-> IO b) -> Event b
```

Abgeleitete Events

- Mit `always :: Event a` kann immer synchronisiert werden.
- Events als Monaden:

```
instance Monad Event where
  (>>=) ev1 getEv2 = ev1 >>>= sync . getEv2
  return v         = always (return v)

-- Damit z.B:
forever :: Event a-> Event ()
forever e = do e; forever e

spawnEvent :: Event a-> IO (IO ())
spawnEvent r = do i<- forkIO (sync r); return (killThread i)
```

Beispiel: Events in HTk

```
main :: IO ()
main =
  do htk <- initHTk [text "Main Window"]

     b <- newButton htk [text " Quit ", size(10, 2)]
     pack b []

     press <- clicked b

     spawnEvent (forever (press >>> (putStrLn "Exiting!" >> destroy htk)))

     finishHTk
```

HTk

- `HTk`: graphische Benutzerschnittstellen für Haskell
- Eigenentwicklung Uni Bremen
- Benutzt Tk, aber ersetzt Tcl durch Haskell
- Voll nebenläufig
- Getypte, komponierbare Events als „first-class objects“
- Deckt den gesamten Funktionsumfang von Tk ab:
  widgets, text widgets, canvasses, . . .

Beispiele in `HTk`

- `htkSimple1.hs`
- `htkSimple2.hs`
- `hsMines.hs`
- `hsMines2.hs`

wxHaskell

- Basiert auf wxWidgets
  - "a single, easy-to-use API for writing GUI applications on multiple platforms."
  - Implementiert auf Linux/GTk2, Windows, Mac OS X, . . .
- wxHaskell ist Haskell-Bindung
- Event Handling (externe Eventschleife á la Tcl, Java etc.)
- Vielseitig, aber groß (40 MB zzgl. wxWidgets)

Beispiele in `wxHaskell`

- `HellowWx.hs`
- `BouncingBall.hs`

Zusammenfassung

- Reaktive Programme mit traditionellen sequentiellen Programmiermodell schwer zu fassen — Ansätze:
  - Interne Eventschleifen (Programm durchläuft alle Eingaben)
  - Externe Eventschleifen (Programm als Sammlung von Funktionen, die Eingaben behandeln)
  - Nachteile: Kompositionalität und Modularität
- Funktionale Sprachen erlauben abstraktere Ansätze:
  - Ereignisbasierte Programmierung (HTk)
  - Vorteile HTk: kompositional und modular, nebenläufig
    Nachteile HTk: etwas angestaubt, kein native look&feel
  - Vorteile wxHaskell: größerer Funktionsumfang,
    Nachteil wxHaskell: externe Eventschleife, nicht voll nebenläufig