Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative
Istituto sistemi informativi e networking

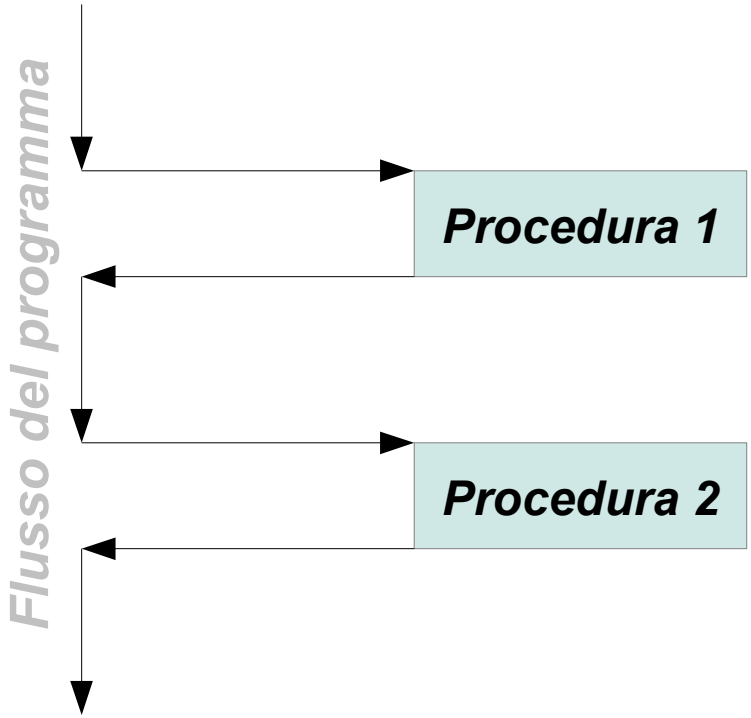
SUPSI

Eventi

Amos Brocco, Ricercatore, DTI / ISIN

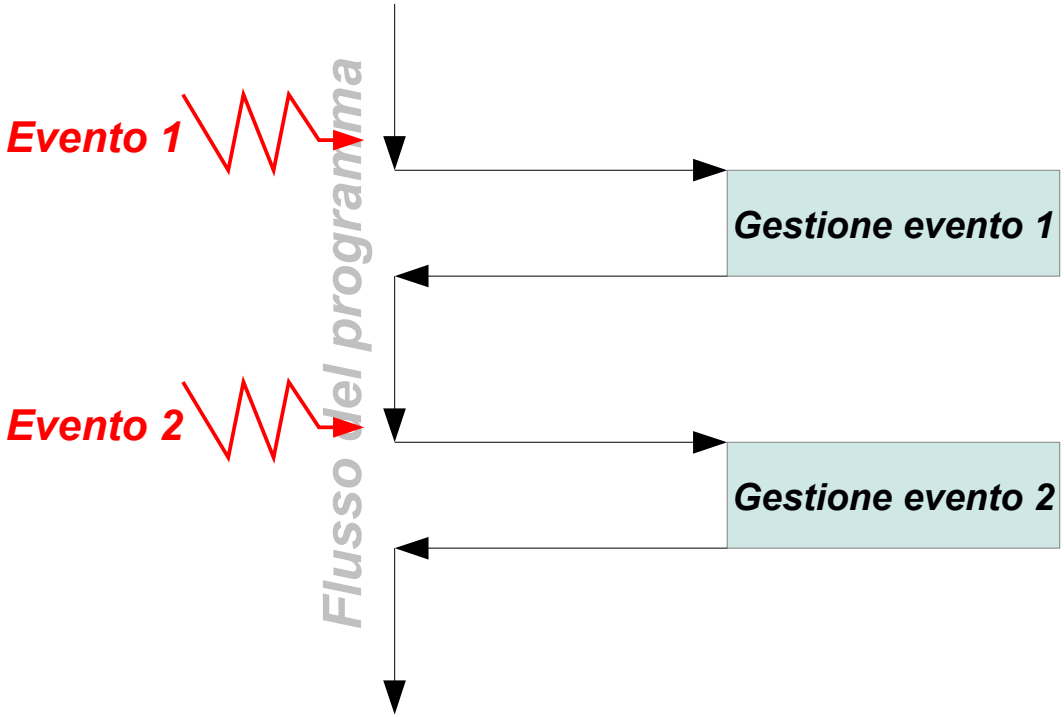
16 febbraio 2012

Programmazione procedurale (imperative / procedural)



Il flusso di esecuzione è determinato dalla sequenza di istruzioni (statements)

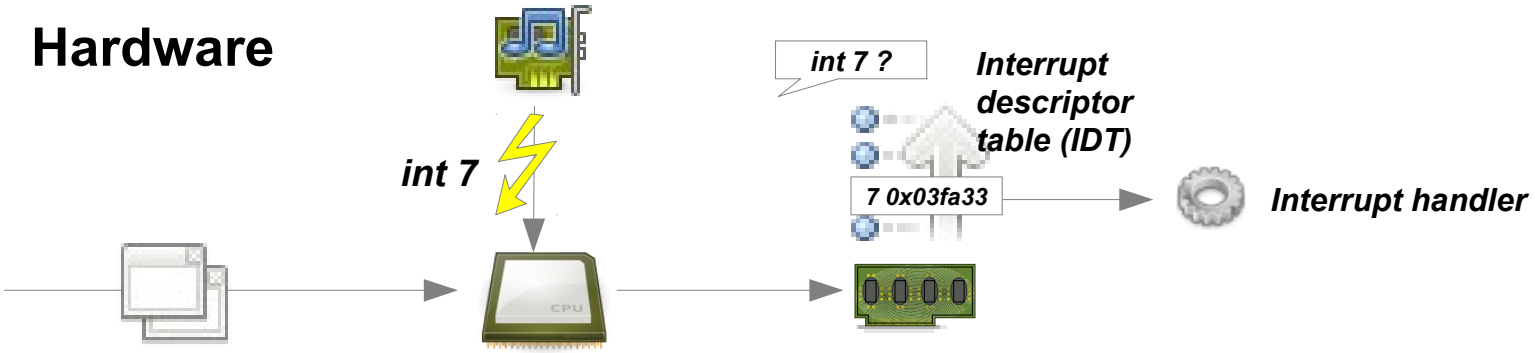
Programmazione a eventi (event driven)



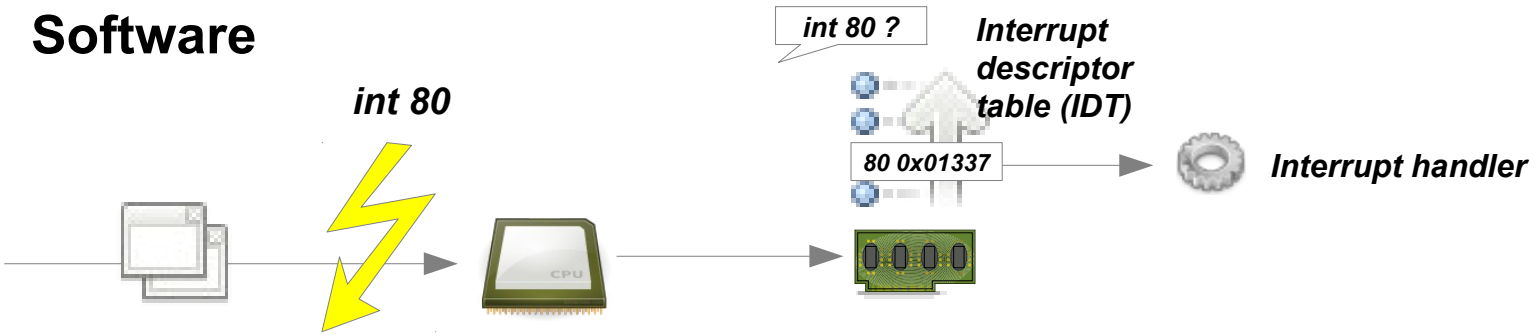
Il flusso di esecuzione è determinato dagli **eventi**, cioè da **informazioni provenienti dall'esterno del programma oppure da componenti del programma stesso**

Eventi a basso livello: esempio di interruzioni (interrupt)

- **Hardware**

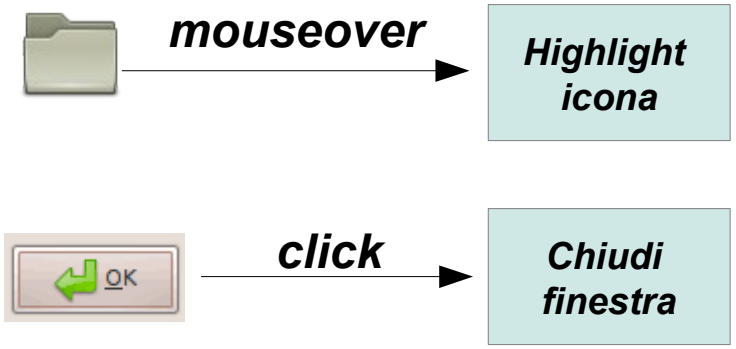


- **Software**

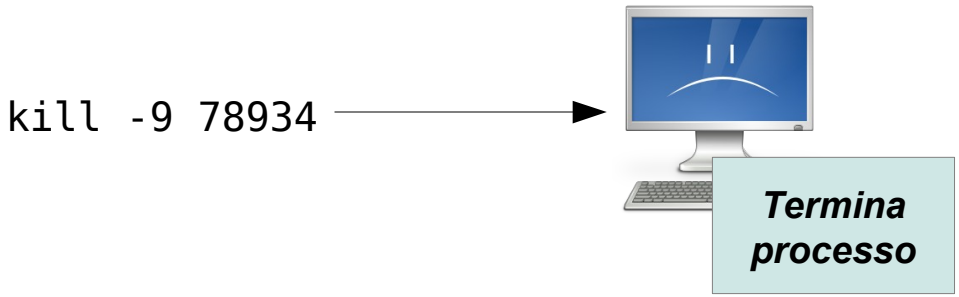


Eventi a alto livello

Interfaccia grafica (GUI)



Segnali UNIX/Linux



Programmazione a eventi

- Gli eventi influenzano l'avanzamento del programma a dipendenza della tecnica di programmazione adottata:
  - **Programmazione sincrona**
    - A ciclo aperto
      - » Bloccante
      - » Non bloccante
    - A ciclo chiuso
  - **Programmazione asincrona**

Programmazione sincrona

- Gli eventi sono analizzati in **punti prestabiliti** del programma e le decisioni su cosa deve fare il programma sono prese in punti prestabiliti.

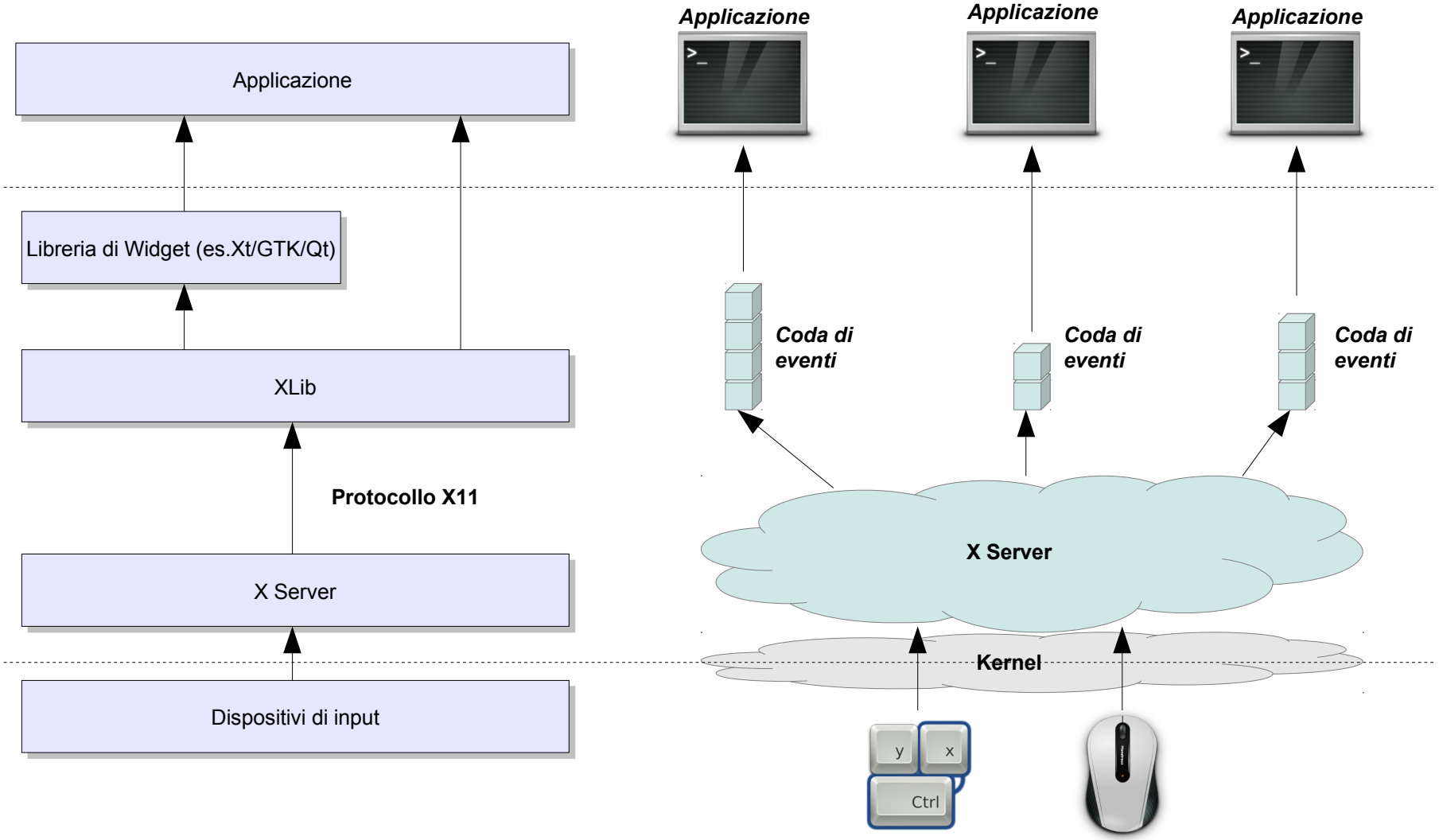
- L'avanzamento è quindi **sincronizzato** con la sequenza di eventi ricevuti: in mancanza di eventi il programma non fa nulla.

  - Esempi: programmi interattivi classici (editor di testo, fogli elettronici, ecc.)

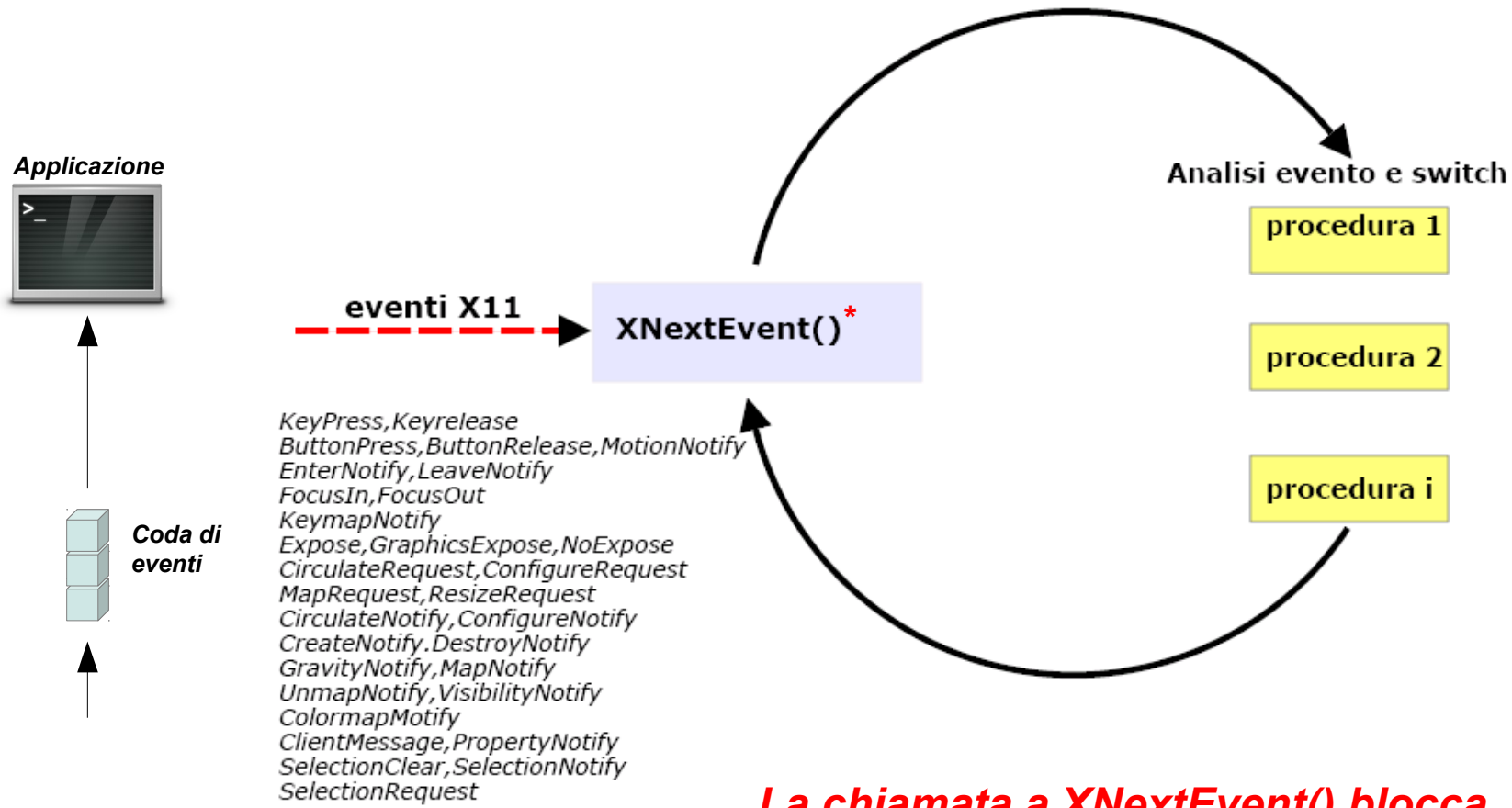
Programmazione asincrona

- Il programma può avanzare anche senza eventi
- Gli eventi possono **interrompere l'attività corrente del programma in qualsiasi punto** e darne inizio a un'altra
- Sono necessarie regole per la gestione della concomitanza di più eventi che arrivano contemporaneamente

Esempio di sistema a eventi sincrono: X Server



Programmazione sincrona: ciclo aperto e bloccante in X11 *



La chiamata a XNextEvent() blocca se non ci sono eventi nella coda

Cos'è un evento X11 ? Un esempio...

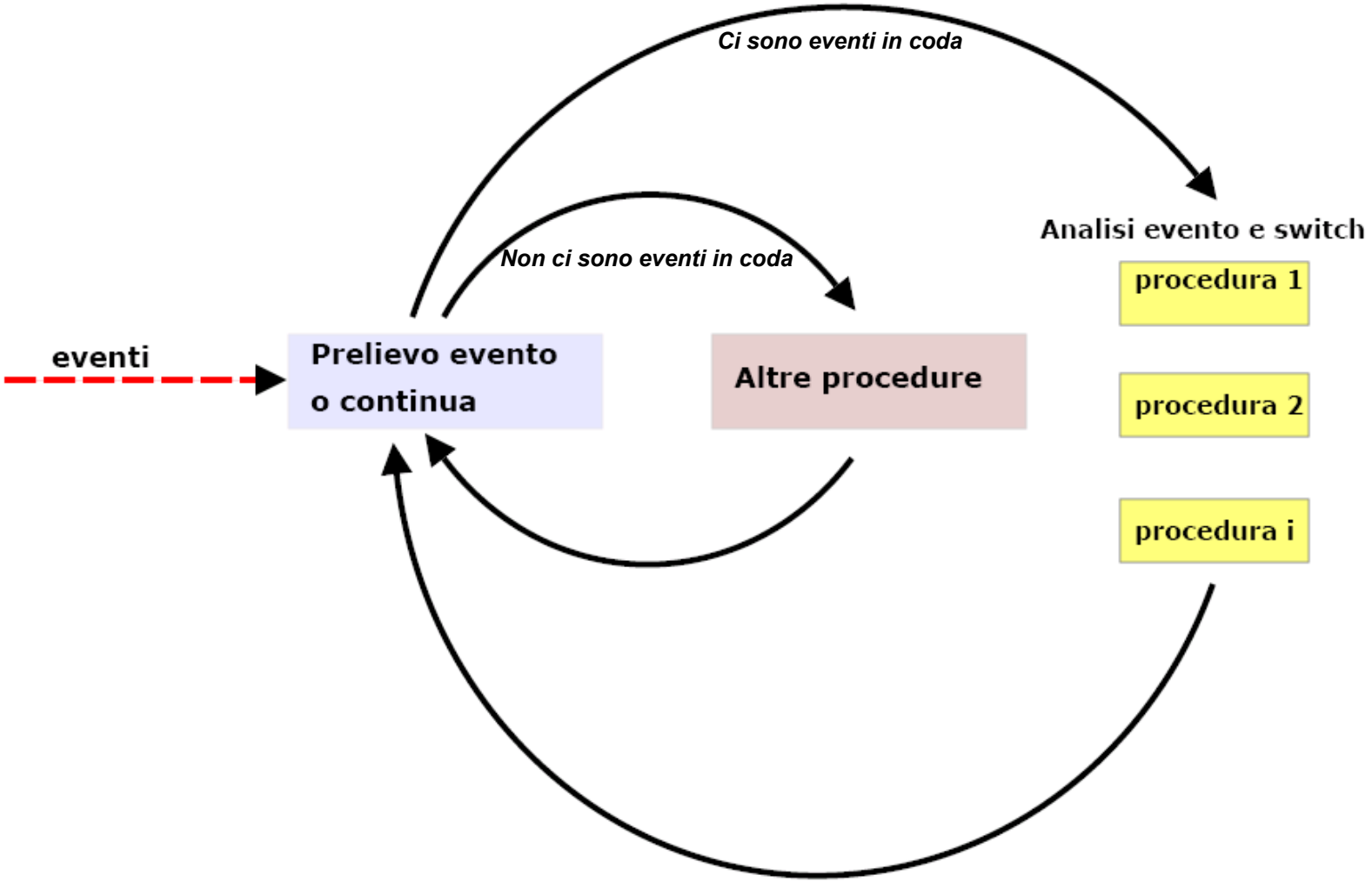
typedef struct{		
int	type;	`ButtonPress` o `ButtonRelease`
unsigned long;	serial	numero dell'ultima richiesta del server
Bool	send_event;	vero se generato da una richiesta di `Send Event`
Display	*display;	`Display` da cui proviene l'evento
Window	window;	finestra che riporta l'evento
Window	root;	finestra radice
Window	subwindow;	finestra che ha catturato l'evento
Time	time;	istante in millisecondi;
int	x;y;	coordinate della freccia
int	x_root;y_root;	coordinate relative alla radice
unsigned int	state;	ShiftMask,ControlMask,LockMask Mod1Mask ... Mod5Mask Button1Mask ... Button5Mask
unsigned int	button;	numero del bottone
Bool	same_screen;	
}XButtonEvent;		

Programmazione sincrona: ciclo aperto e bloccante in X11

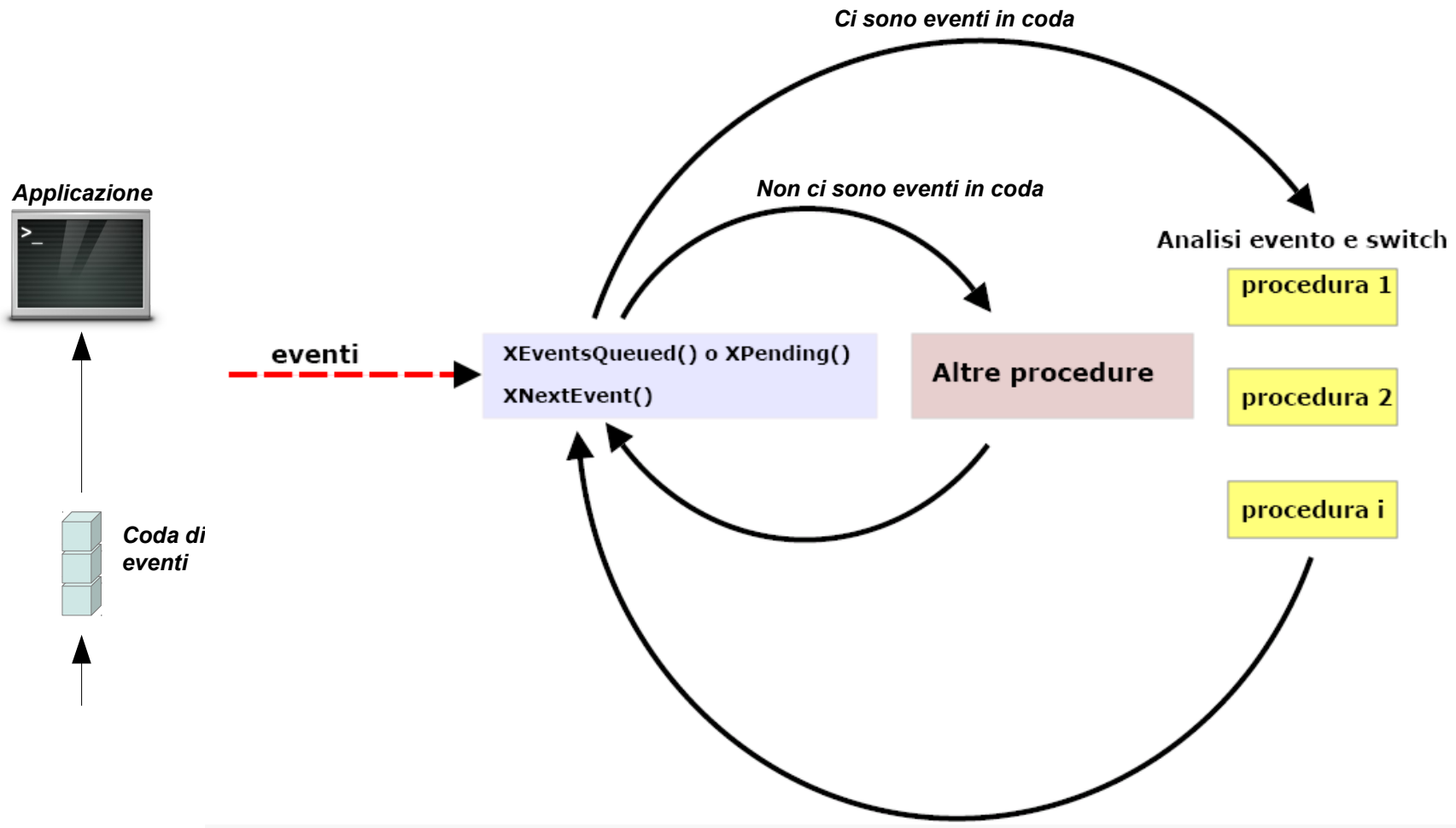
- Gli eventi arrivano al programma attraverso un canale di comunicazione sotto forma di pacchetti di byte.
- La programmazione è detta **bloccante** poiché in mancanza di eventi il programma si blocca e attende.
- La programmazione è detta **a ciclo aperto** poiché il ciclo è visibile al programmatore.

```
while (1) {
    XNextEvent(dpy,&event);
    switch(event.type) {
     case Expose:
       printf("Evento Expose\n");
       break;
     case ButtonPress : {
       printf("Evento ButtonPress\n");
       break;
    }
}
```

Programmazione sincrona: ciclo aperto non bloccante



Programmazione sincrona: ciclo aperto non bloccante in X11



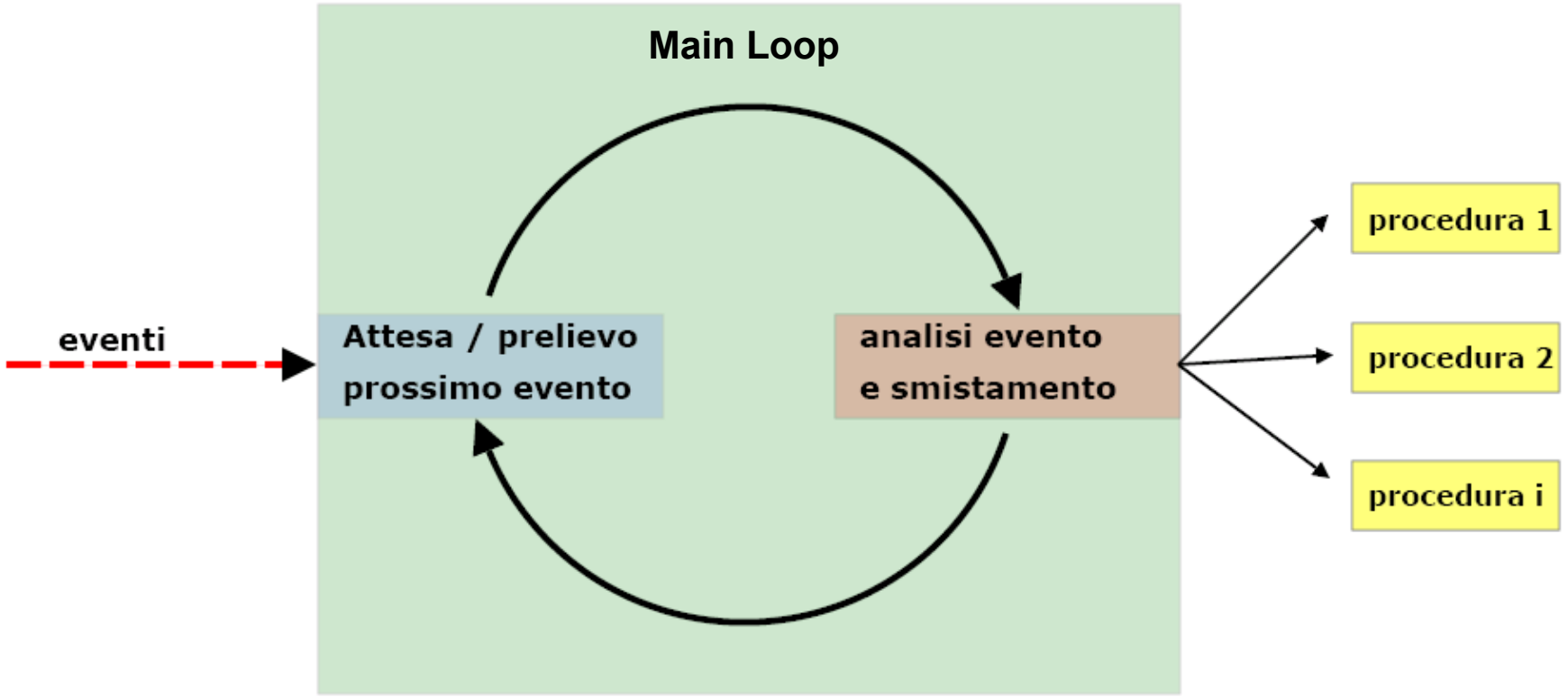
Altre procedure: devono essere brevi per non rallentare la consegna degli eventi

Programmazione sincrona: ciclo aperto non bloccante in X11

- XPending() ritorna il numero di eventi nella coda per il display indicato
- La programmazione è detta **non bloccante** poiché in mancanza di eventi il programma può eseguire altre procedure
- La programmazione è detta **a ciclo aperto** poiché il ciclo è visibile al programmatore.

```
while (1) {
  if (XPending(dpy) > 0) {
   XNextEvent(dpy,&event);
   switch (event.type) {
     case Expose:
      printf("Evento expose\n");
      break;
     case ButtonPress:
      printf("Evento buttonpress\n");
      break;
   }
  } else
    /* altre procedure */
  }
```

Programmazione sincrona: ciclo chiuso e bloccante



Il ciclo viene nascosto all'interno di un **main loop**. Il programma diventa di più facile lettura.

Il main loop preleva un evento per volta dalla coda, lo analizza e richiama una procedura associata dal programma in precedenza all'evento (*event handler*)

Programmazione sincrona: ciclo chiuso e bloccante

- Questa tipologia di programmazione a eventi è utilizzata tipicamente dai toolkit per interfacce grafiche:
  - Xt (X Toolkit)
  - GTK+
  - Qt
  - glut
  - ...
- Altri esempi:
  - Simulatori a eventi discreti
    - OMNET++

Programmazione a eventi sincrona di interfacce grafiche

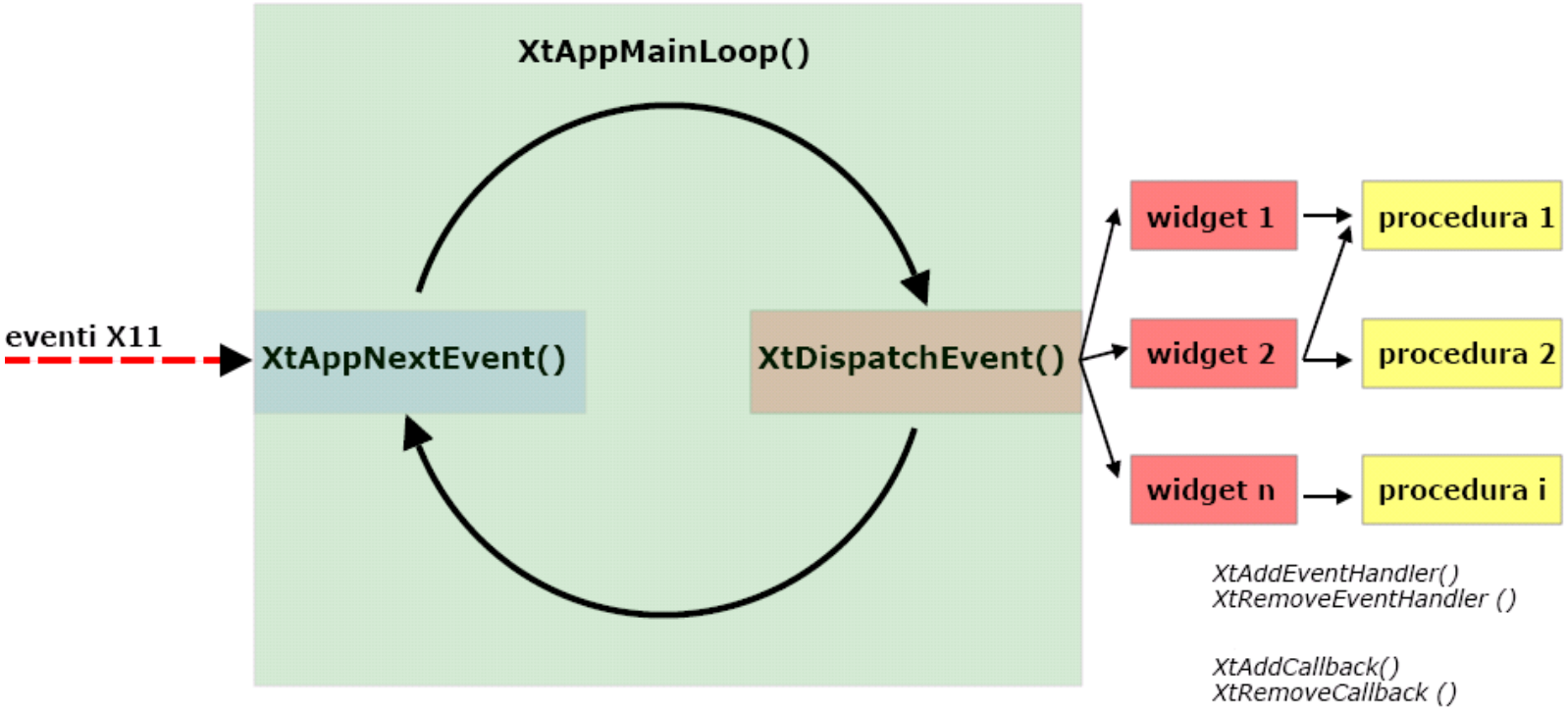
- Ogni ambiente / linguaggio di programmazione definisce gli eventi che sono di suo interesse, eventualmente combinando più eventi elementari per creare eventi complessi, per esempio:
  - un *click* non è un evento elementare di un sistema a finestre, ma una successione di due eventi elementari (tasto del mouse giù + tasto del mouse su)
  - un *doppio* click è una successione di 4 eventi elementari temporalmente vicini.

Programmazione a eventi sincrona di interfacce grafiche

- Gli eventi complessi in un'interfaccia grafica sono tipicamente usati in combinazione con oggetti grafici intelligenti detti **widget** (bottoni, liste, menu, …)

- Il programmatore definisce il comportamento dei widget in risposta agli eventi generati dall'interazione dell'utente con l'interfaccia grafica

Programmazione a eventi sincrona di interfacce grafiche: Xt



Definizione dei gestori di eventi (callback, event handler)

- Esempio:
  - Colleghiamo gli eventi
    - XmNarmCallback alla procedura_1
    - XmNactivateCallback alla procedura_2
  - Contesto è un riferimento alla coda degli eventi

```
XtAddCallback(widget,
     XmNarmCallback,
     procedura_1, NULL);

XtAddCallback(widget,
     XmNactivateCallback,
     procedura_2, NULL);

XtAppMainLoop(contesto);
```

Programmazione a eventi sincrona di interfacce grafiche: Xt

- Tipiche sequenze di programmazione sincrona bloccante:

```
1)XtAppInitialize () ;
2)Creazione e gestione Widget (XtCreateManagedWidget ();
3)XtAddCallback () ;
4)XtRealizeWidget () ;
5)XtAppMainLoop () ;
```

Programmazione a eventi sincrona di interfacce grafiche: GTK+

- Tipiche sequenze di programmazione sincrona bloccante:

```
1)gtk_init () ;
2)gtk_window_new ()
3)gtk_container_add () ;
4)gtk_signal_connect () ;
5)gtk_widget_show () ;
6)gtk_main () ;
```

http://www.gtk.org/

Programmazione a eventi sincrona di interfacce grafiche: Qt

- Tipiche sequenze di programmazione sincrona bloccante:

```
#include <QApplication>
#include <QpushButton>

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    QPushButton hello("Ciao mondo");
    hello.resize(100, 100);
    hello.show();
    return app.exec();  /* qui è nascosto il “Main loop” */
}
```

http://doc.trolltech.com/4.3/tutorial.html

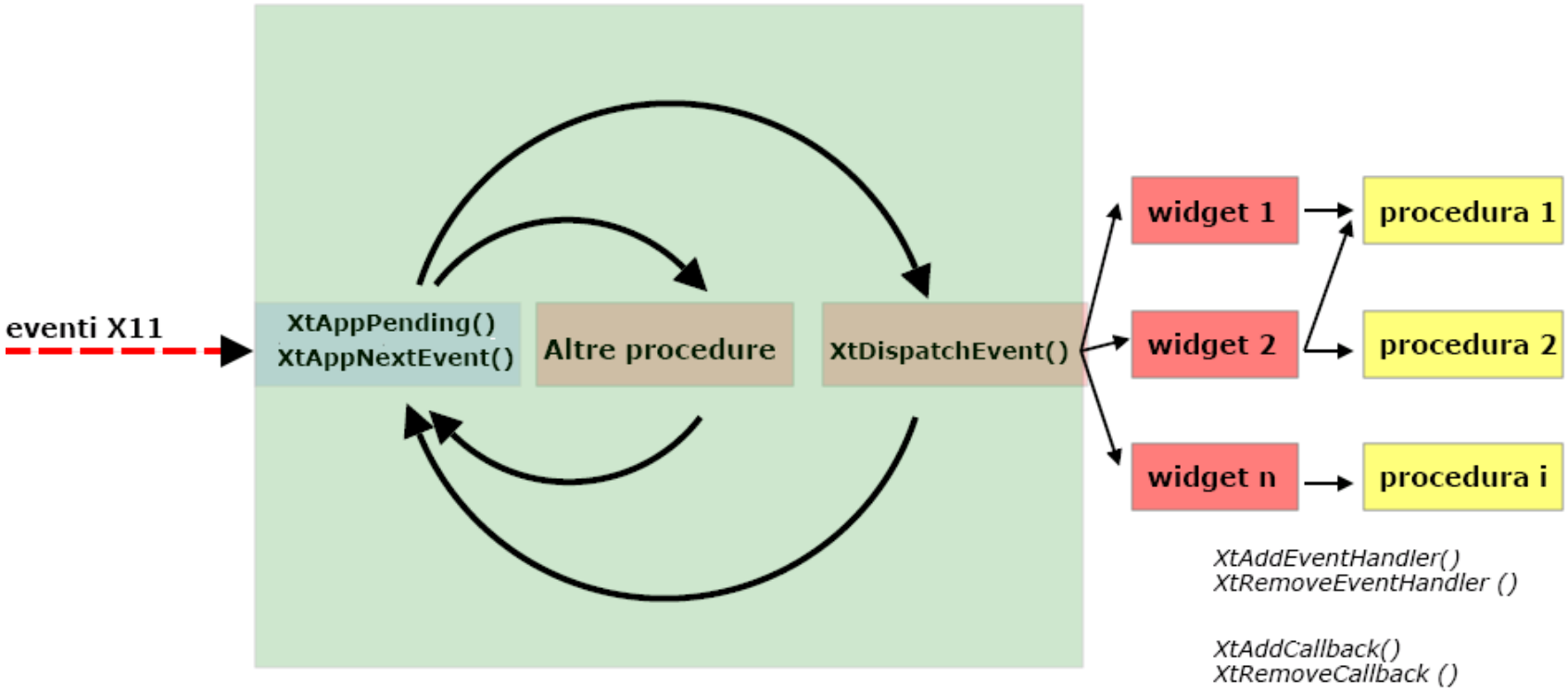
Programmazione a eventi sincrona di interfacce grafiche: glut

- Tipiche sequenze di programmazione sincrona bloccante:

```
1)glutInit () ;
2)gluitInitDisplayMode();
3)glutCreateWindow () ;
4)glutDisplayFunc(); glutReshapeFunc(); glutKeyboardFunc();
5)glutSpecialFunc(); glutMouseFunc(); glutMouseMotionFunc();
6)glutPassiveMotionFunc(); glutEntryFunc(); glutVisibilityFunc();
7)glutMenuStatusFunc();
8)glutIdleFunc();  /* altre attività */
9)glutTimerFunc(); /* la sveglia produce un evento che si
                                    mescola agli altri */
10) glutMainLoop();
```

http://www.opengl.org/code/detail/glut_tutorial/

Programmazione a eventi sincrona: ciclo aperto Xt non bloccante

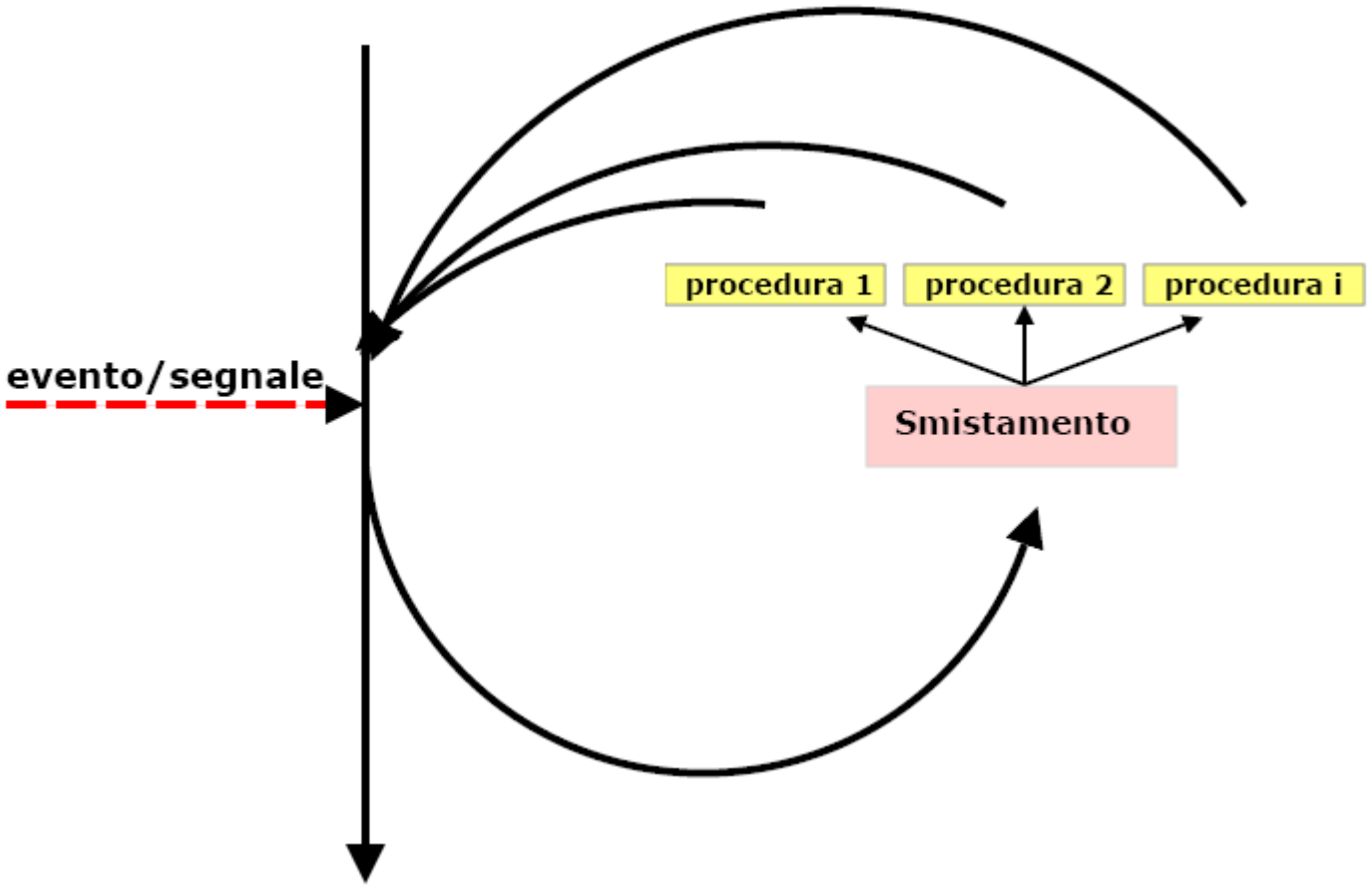


Programmazione a eventi sincrona: ciclo aperto Xt non bloccante

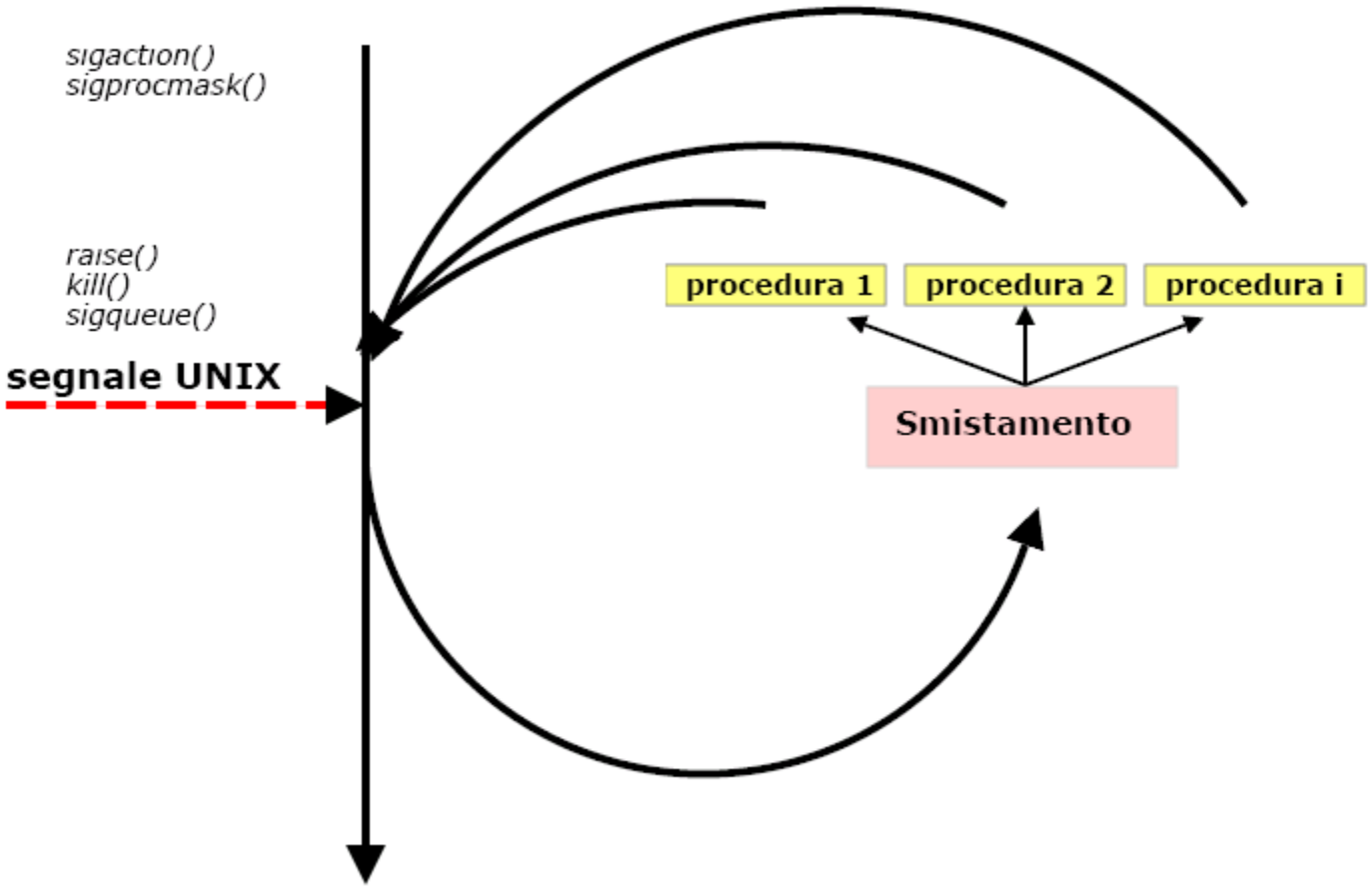
- È possibile replicare il funzionamento di XtAppMainLoop con un ciclo aperto non bloccante
- La funzione XtDispatchEvent() smista l'evento al corretto widget

```
while (1) {
  if (XtAppPending(app_context)) {
    XtAppNextEvent(app_context,
                   &event);
    XtDispatchEvent(&event);
  } else {
    /* altre procedure */
  }
}
```

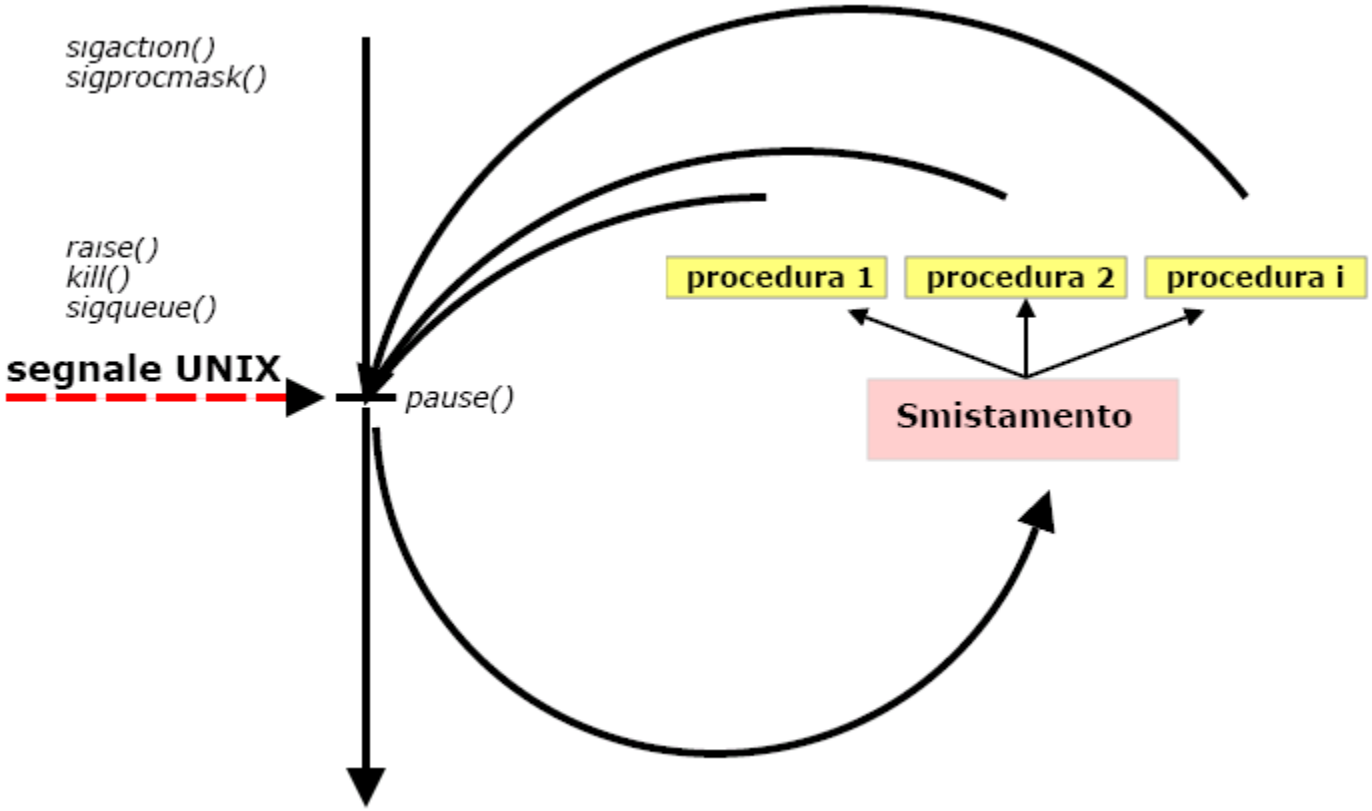
Gestione degli eventi asincrona



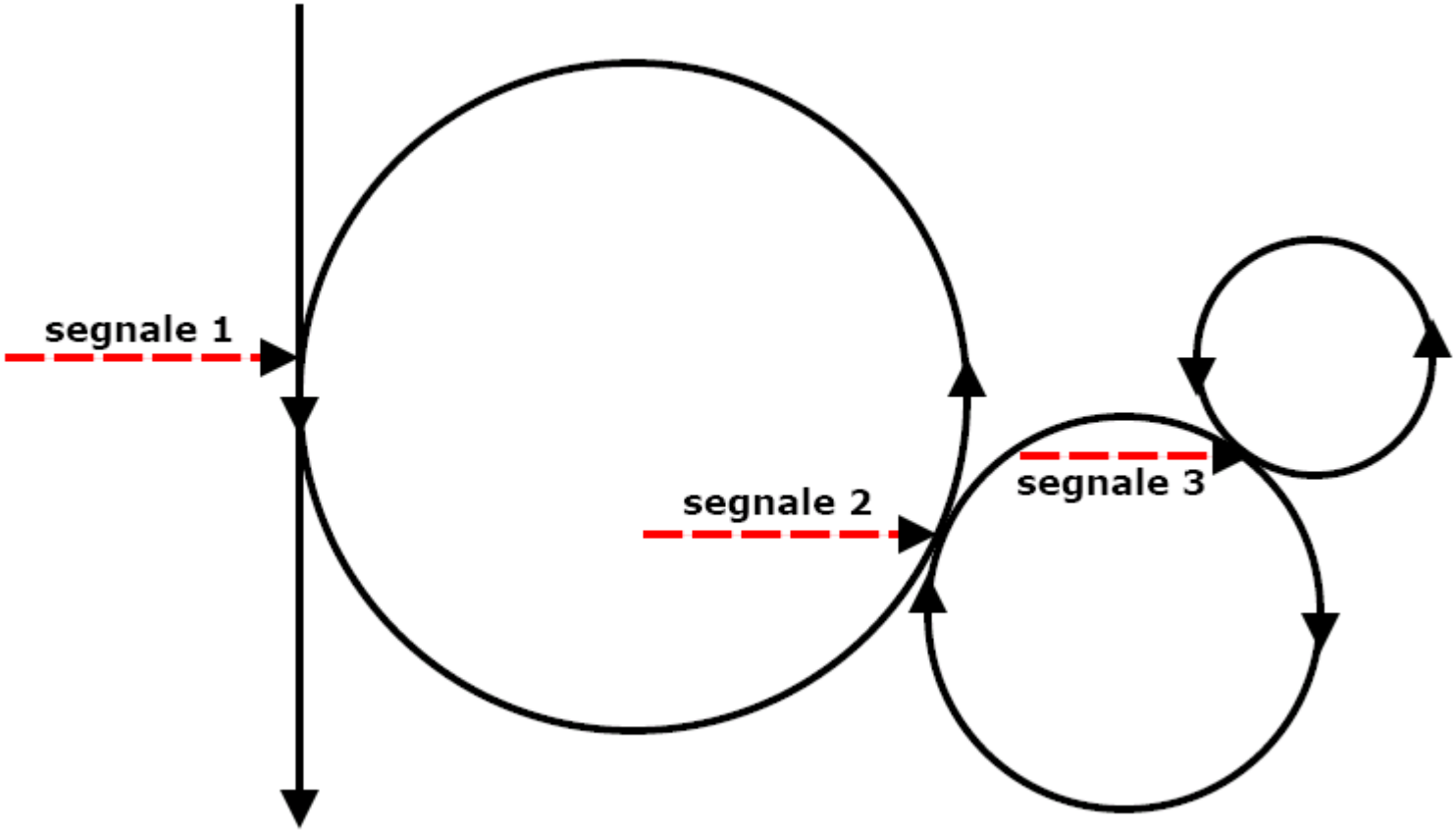
Esempio di programmazione a eventi asincrona: segnali UNIX



Segnali UNIX a un programma in pausa



Interruzioni multiple



Interruzioni multiple

Quando un programma deve gestire più interruzioni concorrenti (eventi/segnali che accadono mentre il programma è all'interno di una routine di gestione) **è necessario un meccanismo di controllo**:

- Mascheramento dei segnali
- Assegnazione di priorità ai segnali
- ...

Controllo tramite priorità

- Esempio: kernel di sistema operativo
  - Interruzioni hardware e software
  - Priorità per gestire i conflitti
    - Una procedura può essere interrotta da un‘altra solo se la sua priorità è maggiore di quella della procedura in corso.
    - La priorità è definita per l‘interruzione.

Mascheramento o bloccaggio

- Esempio: segnali di UNIX
  - Un segnale può essere temporaneamente bloccato:
    - in modo implicito durante l'esecuzione di una procedura provocata da un altro segnale.
    - in modo esplicito.

Esempio di programmazione a eventi mista (sincrona/asincrona): Xt

```
while (TRUE) {
  if(XtAppPending(app_context)) {
    XtAppNextEvent(app_context, &event);
    XtDispatchEvent(&event);
  } else {
    sigprocmask(SIG_UNBLOCK, set, NULL) ; /* sbloccaggio
                        dei segnali definiti in 'set' */
    usleep (mseconds);          /* breve pausa */
    sigprocmask(SIG_BLOCK, set, NULL) ;  /* bloccaggio */
  }
}
```

Esempio di programmazione a eventi mista (sincrona/asincrona): Xt

```
struct itimerval timer ;
struct sigaction action ;

timer.it_interval.tv_sec=0;
timer.it_interval.tv_usec=10000;
timer.it_value.tv_sec=0;
timer.it_value.tv_usec=10000;
action.sa_handler=altre_attività;

sigaction (SIGALRM,&action,NULL); /*associazione segnale-
procedura*/
setitimer (ITIMER_REAL,&timer,NULL); /*montaggio sveglia
periodica*/
```

Elementi di programmazione asincrona/sincrona senza attesa di GTK+

- Timeouts
  - `id=g_timeout_add (intervallo, funzione,dati);`
  - `g_source_remove(id);`
- Monitoring I/O
  - `id=gdk_input_add (fd,GDK_INPUT_READ,funzione,dati);`
  - `id=gdk_input_add (fd,GDK_INPUT_WRITE,funzione,dati);`
  - `gkt_input_remove(id) ;`
- Idle Functions
  - `id=gtk_idle_add (funzione dati) ;`
  - `gtk_idle_remove (id) ;`