Monitor - Introduzione

- **Un monitor è un modulo software che consiste di:**
  - dati locali
  - una sequenza di inizializzazione
  - una o più "procedure"
- **Le caratteristiche principali sono:**
  - i dati locali sono accessibili solo alle procedure del modulo stesso
  - un processo entra in un monitor invocando una delle sue procedure
  - solo un processo alla volta può essere all'interno del monitor; gli altri processi che invocano il monitor sono sospesi, in attesa che il monitor diventi disponibile

Monitor - Sintassi

```
monitor name {
    variable declarations...
    procedure entry type procedurename1(args...) {
      ...
    }
    type procedurename2(args...) {
      ...
    }
    name(args...) {
      ...
    }
}
```

variabili private del monitor

procedure visibili all'esterno

procedure private

inizializzazione

Monitor - Alcuni paragoni

- **Assomiglia ad un "oggetto" nella programmazione o.o.**
  - il codice di inizializzazione corrisponde al costruttore
  - le *procedure entry* sono richiamabili dall'esterno e corrispondono ai metodi pubblici di un oggetto
  - le procedure "normali" corrispondono ai metodi privati
  - le variabili locali corrispondono alle variabili pubbliche
- **Sintassi**
  - originariamente, sarebbe basata su quella del Pascal
    - var, procedure entry, etc.
  - in questi lucidi, utilizziamo una sintassi simile a Java

Monitor - Caratteristiche base

- **Solo un processo alla volta può essere all'interno del monitor**
  - il monitor fornisce un semplice meccanismo di mutua esclusione
  - strutture dati condivise possono essere messe all'interno del monitor
- **Per essere utile per la programmazione concorrente, è necessario un meccanismo di sincronizzazione**
- **Abbiamo necessità di:**
  - poter sospendere i processi in attesi di qualche condizione
  - far uscire i processi dalla mutua esclusione mentre sono in attesa
  - permettergli di rientrare quando la condizione è verificata

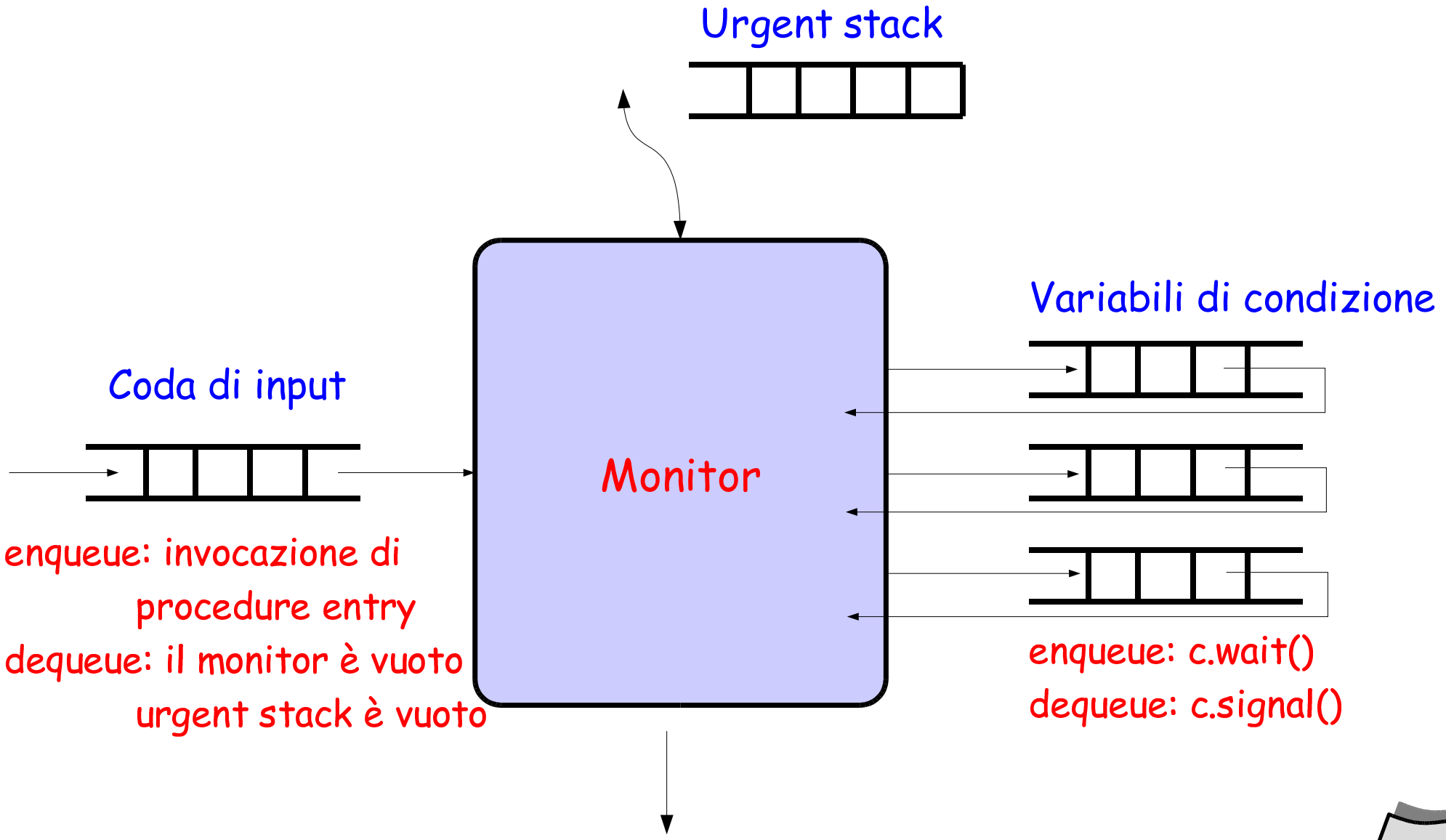
Monitor - Meccanismi di sincronizzazione

- **Dichiarazione di variabili di condizione (CV)**
  - `condition c;`
- **Le operazioni definite sulle CV sono:**
  - `c.wait()`
    attende il verificarsi della condizione
  - `c.signal()`
    segnala che la condizione è vera

Monitor - Politica signal urgent

- `c.wait()`
  - viene rilasciata la mutua esclusione
  - il processo che chiama `c.wait()` viene sospeso in una coda di attesa della condizione `c`
- `c.signal()`
  - causa la riattivazione immediata di un processo (secondo una politica FIFO)
  - il chiamante viene posto in attesa
  - verrà riattivato quando il processo risvegliato avrà rilasciato la mutua esclusione (*urgent stack*)
  - se nessun processo sta attendando `c` la chiamata non avrà nessun effetto

Monitor - Rappresentazione intuitiva



Monitor - wait/signal vs P/V

- **A prima vista:**
  - **`wait`** e **`signal`** potrebbero sembrare simili alle operazioni sui semafori
    P e **V**
- **Non è vero!**
  - **`signal`** non ha alcun effetto se nessun processo sta attendendo la condizione
    V "memorizza" il verificarsi degli eventi
  - **`wait`** è sempre bloccante
    P (se il semaforo ha valore positivo) no
  - il processo risvegliato dalla **`signal`** viene eseguito per primo

Monitor - Politiche di signaling

- **Signal urgent è la politica "classica" di signaling**
  - SU - *signal urgent*
    - proposta da Hoare
- **Ne esistono altre:**
  - SW - *signal wait*
    - no urgent stack, il processo che fa la signal viene messo nella entry queue
  - SR - *signal and return*
    - dopo la **`signal`** si esce subito dal monitor
  - SC - *signal and continue*
    - la **`signal`** segnala solamente che un processo può continuare, il chiamante prosegue l'esecuzione
    - quando lascia il monitor viene riattivato il processo segnalato

Monitor - Implementazione dei semafori

```
monitor Semaphore {
  int value;
  condition c;          /* value > 0 */

  procedure entry void P() {
    value--;
    if (value < 0)
      c.wait();
  }

  procedure entry void V() {
    value++;
    c.signal();
  }

  Semaphore(int init) {
    value = init;
  }
}
```