Linux Kernel

Introduzione

Strutture dati

Servizi di sistema

Operazioni sui processi

Introduzione

- **Alcuni anni fa un sistema operativo era definito come:**

“Il software necessario a controllare l’hardware di un calcolatore”

- **Da allora…**
  - Il panorama informatico è cambiato significativamente
    - Potenza di calcolo
    - Complessità dei sistemi
    - Complessità delle applicazioni
  - Una nuova definizione si rende necessaria
    - Più ampia e completa

Introduzione

- **Un sistema operativo**
  - Separa le applicazioni dall'hardware che queste utilizzano
    - Si tratta di uno strato di software
  - Offre un supporto per ottenere i risultati voluti mediante
    - Gestione dell'hardware
    - Gestione del software
- **Principalmente un sistema operativo è un gestore di risorse**
  - Hardware
    - Processori
    - Memorie
    - Dispositivi di Input/output
    - Dispostivi di Comunicazione
  - Software
    - Applicazioni

Breve storia dei sistemi operativi

- **Anni ‘40**
  - I primi calcolatori non disponevano di sistema operativo
- **Anni ‘50**
  - Eseguivano una applicazione (job) alla volta
  - Semplificavano il passaggio da un job all’altro
  - Sistemi batch per l’esecuzione di un singolo flusso
  - Programmi e dati su nastro
- **Primi anni ’60**
  - Ancora sistemi batch
    - Più job contemporaneamente: multiprogrammazione
    - Un job utilizza il processore, un’altro i dispositivi di I/O
  - Primi sistemi multi-utente

Breve storia dei sistemi operativi

- **Anni ’60**
  - Si consolida il concetto di timesharing
    - Sviluppato per supportare più utenti simultaneamente
    - Il turnaround time passa da minuti a secondi
  - Sistemi real-time
    - Forniscono una risposta entro un tempo prefissato
  - La memoria virtuale
    - Possibilità di indirizzare più memoria di quella disponibile
  - Alcuni sistemi
    - TSS
    - Multics
    - CP/CMS

Breve storia dei sistemi operativi

- **Anni ’70**
  - Sistemi multimodali in timesharing
    - Supporo per elaborazione batch, timesharing e applicazioni real-time
  - Stanno nascendo i primi personal computer
    - Sotto la spinta della nuova tecnologia del microprocessore
  - Il Dipartimento della Difesa americano sviluppa TCP/IP
    - Protocollo di comunicazione standard
    - Inizialmente molto utilizzato nelle università ed in ambito militare
  - Primi problemi di sicurezza
    - Molte informazioni passano su canali vulnerabili

Breve storia dei sistemi operativi

- **Anni ’80**
  - Vedono lo sviluppo di
    - Personal computer
    - Workstation
  - L’elaborazione viene delocalizzata ove rischiesto
  - User friendly
    - Più semplici da utilizzare
    - Nascono le prime interfacce grafiche
  - Comunicazione
    - La trasmissione di dati tra computer remoti diviene più semplice ed economica
  - Nasce il modello client/server
    - I client richiedono vari servizi
    - I server eseguono le operazioni richieste

Breve storia dei sistemi operativi

- **Anni ’90**
  - Le prestazioni dei calcolatori crescono esponenzialmente
    - Potenza di calcolo e capacità di memoria a basso costo
    - Personal computer in grado di eseguire programmi molto complessi
    - Grande sviluppo delle interfacce grafiche
  - Ci si sposta verso il calcolo distribuito
    - Calcolatori economici per applicazioni database e job processing
  - I mainframes iniziano a divenire obsoleti
    - Più computer concorrono allo svolgimento di un singolo task
    - Il supporto alle applicazioni distribuite diviene lo standard
  - Microsoft e Windows divengono dominanti
    - Mutua molti concetti dai primi sistemi operativi Macintosh
    - Semplifica l’uso di più applicazioni concorrenti

Breve storia dei sistemi operativi

- **Anni ’90 - continua**
  - Il paradigma ad oggetti diviene dominante
    - Molte applicazioni sono sviluppat con linguaggi object-oriented
    - C++ or Java
  - Object-oriented operating systems (OOOS)
    - Gli oggetti rappresentano i compoennti del sistema operativo
  - Si sviluppano i concetti di interfaccia ed ereditarietà
    - Utilizzati per sviluppare sistemi operativi più modulari
    - Semplificano la manutenzione e l’aggiornamento
  - Inizia a formarsi l’idea di open-source
    - Programmi e sistemi operativi distribuiti sotto forma di codice sorgente
    - Permette ai singoli programmatori di esaminare e modificare il codice
  - Esempi
    - Linux
    - Apache Web server

Breve storia dei sistemi operativi

- **Anni ’90 - continua**
  - Richard Stallman lancia il progetto GNU
    - Ricreare e estendere i tool per il sistema UNIX di AT&T
  - Nasce Open Source Initiative (OSI)
    - Estendere il concetto e la diffusione dell’open-source
    - Facilita il miglioramento del software
    - Aumenta la probabilità di individuare errori molto rari e nascosti
  - I sistemi operativi diventano sempre più user friendly
    - Migliorano le interfaccie, sotto la spinta di Apple
  - Nasce il concetto di “Plug-and-play”
    - Gli utenti possono aggiungere, togliere o sostituire componenti hardware senza dover riconfigurare manualmente il sistema operativo

Breve storia dei sistemi operativi

- **Anni 2000**
  - Middleware
    - Collega due applicazioni distinte
    - Spesso attraverso una rete e su macchine incompatibili
    - Semplifica la comunicazione tra più architetture
    - Particularmente importante per i Web services
  - Web services
    - Servizi standard che permettono a due macchine di interagire
    - Si tratta di porzioni di software disponibili su Internet
  - E molto altro...

Ambiti applicativi

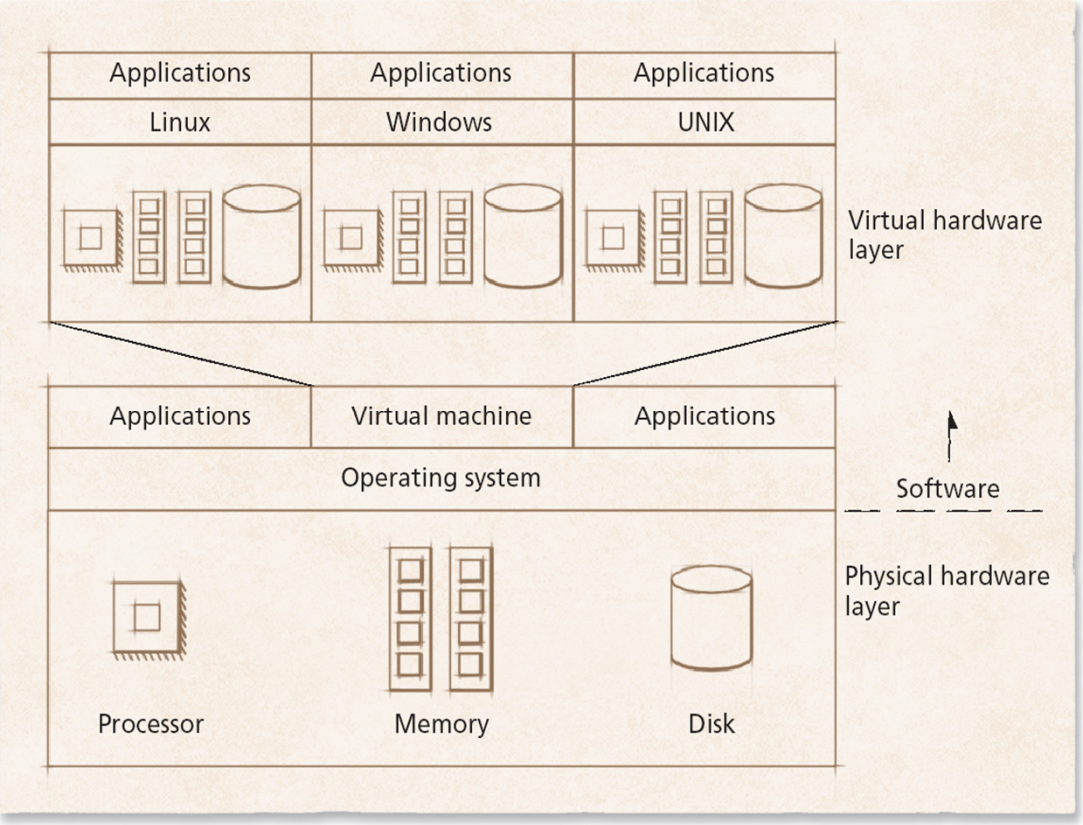
- **Server, server farm**
  - Particolari requisiti
    - Struttura scalabile, numero elevatissimo di processi
  - Supporto hardware
    - Multiprocessore, memoria centrale molto grande, hardware dedicato

- **Embedded systems**
  - Altamente vincolato
    - Tempo di esecuzione, memoria richiesta, dissipazione di potenza
  - Dispositivi molto specifici
    - Difficilmente portabile

- **Real-time systems**
  - Un task deve essere completato entro un periodo di tempo fissato, spesso breve
  - Un task deve reagire ad un evento entro un periodo di tempo, spesso breve

Ambiti applicativi

- **Macchine virtuali**
  - Astrazione software di un computer fisico
  - Il sistema operativo
    - In esecuzione come processo di un sistema operativo nativo
    - Gestisce le risorse della macchina virtuale
  - Applicazioni
    - Più istanze di un sistema operativo in esecuzione
    - Emulazione
    - Portabilità



Componenti di un sistema operativo

- **I principali componenti di ogni sistema operativo sono**
  - Lo scheduler
    - Processi e thread
  - Il gestore della memoria
    - Memoria virtuale, paginazione e segmentazione
  - Il gestore dell'I/O
    - Comunicazione con i dispositivi, mutua esclusione
  - Il gestore dell'IPC (Inter-Process Communication)
    - Comunicazione tra processi
  - Il gestore del file system
    - Accesso uniforme e strutturato ai dati
  - Una shell
    - Interazioen con l'utente

Architetture

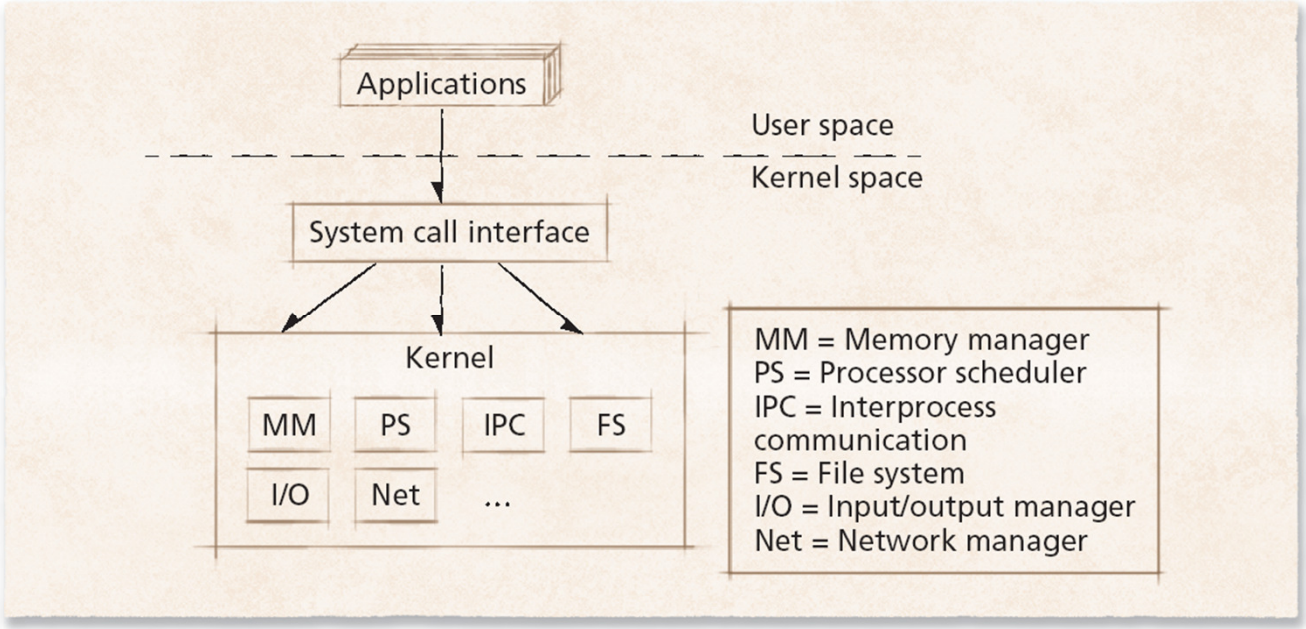
- **I moderni sistemi operativi sono molto complessi**
  - Forniscono moltissimi servizi
  - Supportano una enorme varietà
    - Di componenti hardware
    - Di applicazioni

- **La definizione di una architettura del sistema operativo**
  - Facilita l'organizzazione delle varie componenti del sistema
  - Definisce i privilegi con cui le diverse componenti devono essere eseguite

- **Quattro architetture principali**
  - Monolitica
  - A livelli
  - Microkernel
  - Distribuita

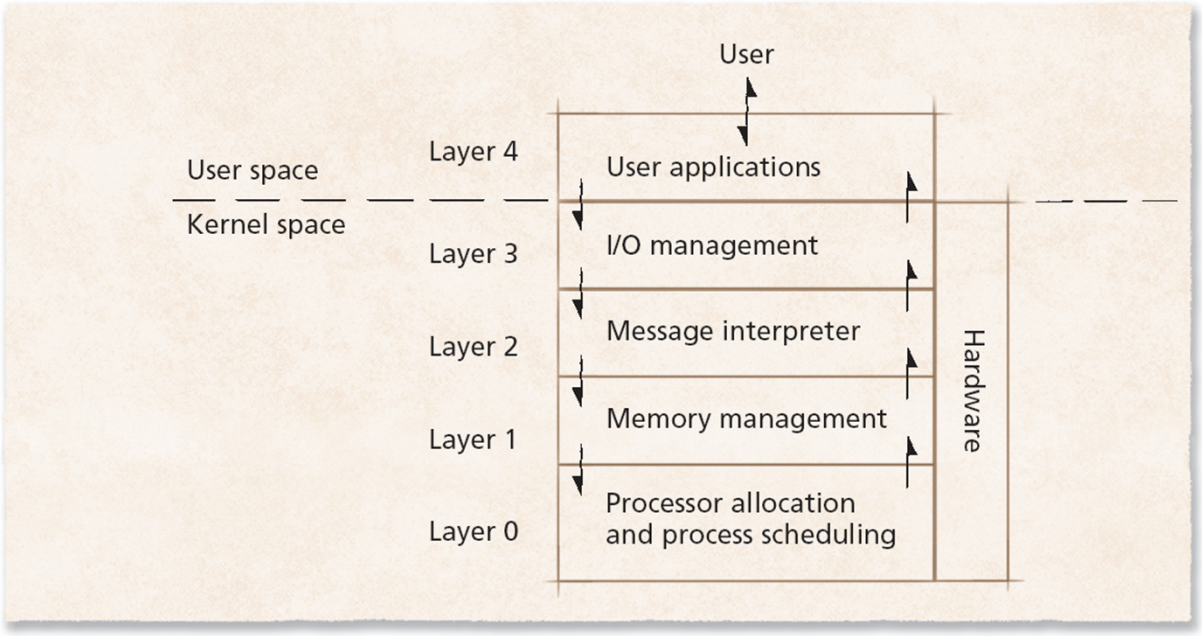
Architettura monolitica

- **Tutti I componenti sono contenuti nel kernel**
  - Ogni componente può comunicare direttamente con ogni altro
    - Molto efficiente
  - Non vi è una marcata separazione tra i componenti
    - Potenzialmente più critico
    - Difficile individuare l'origine di eventuali malfunzionamenti



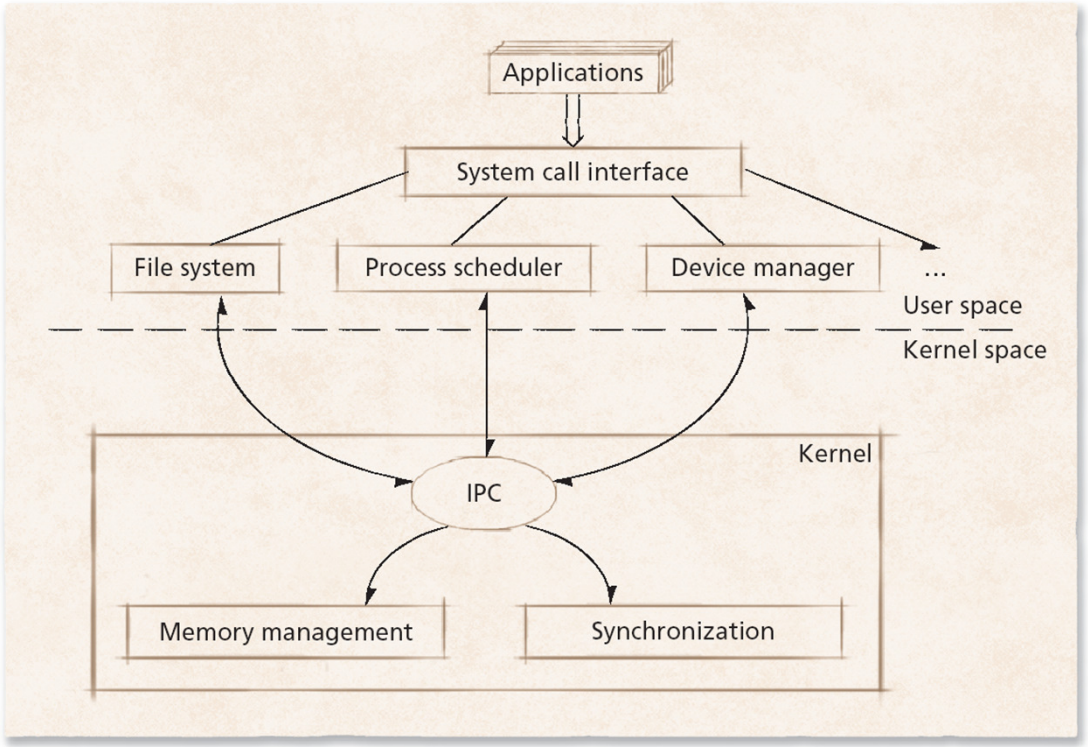
Architettura a livelli o strati

- **Miglioramento rispetto alla soluzione monolitica**
  - Raggrouppa componenti con funzioni simili in un livello
  - Ogni livello comunica solamente con i livelli immediatamente superiore e inferiore
    - Una richiesta può attraversare diversi livelli prima di essere soddisfatta
    - Le prestazioni tendono ad essere peggiori della soluzione monolotoca



Architettura a microkernel

- **Fornisce solo un insieme molto ristretto di servizi**
  - Si vuole mantenere il kernel molto piccolo scalabile
    - Elevata estendibilità, portabilità, scalabilità
  - Richiede maggiore comunicazione tra i moduli
    - Peggioramento delle prestazioni

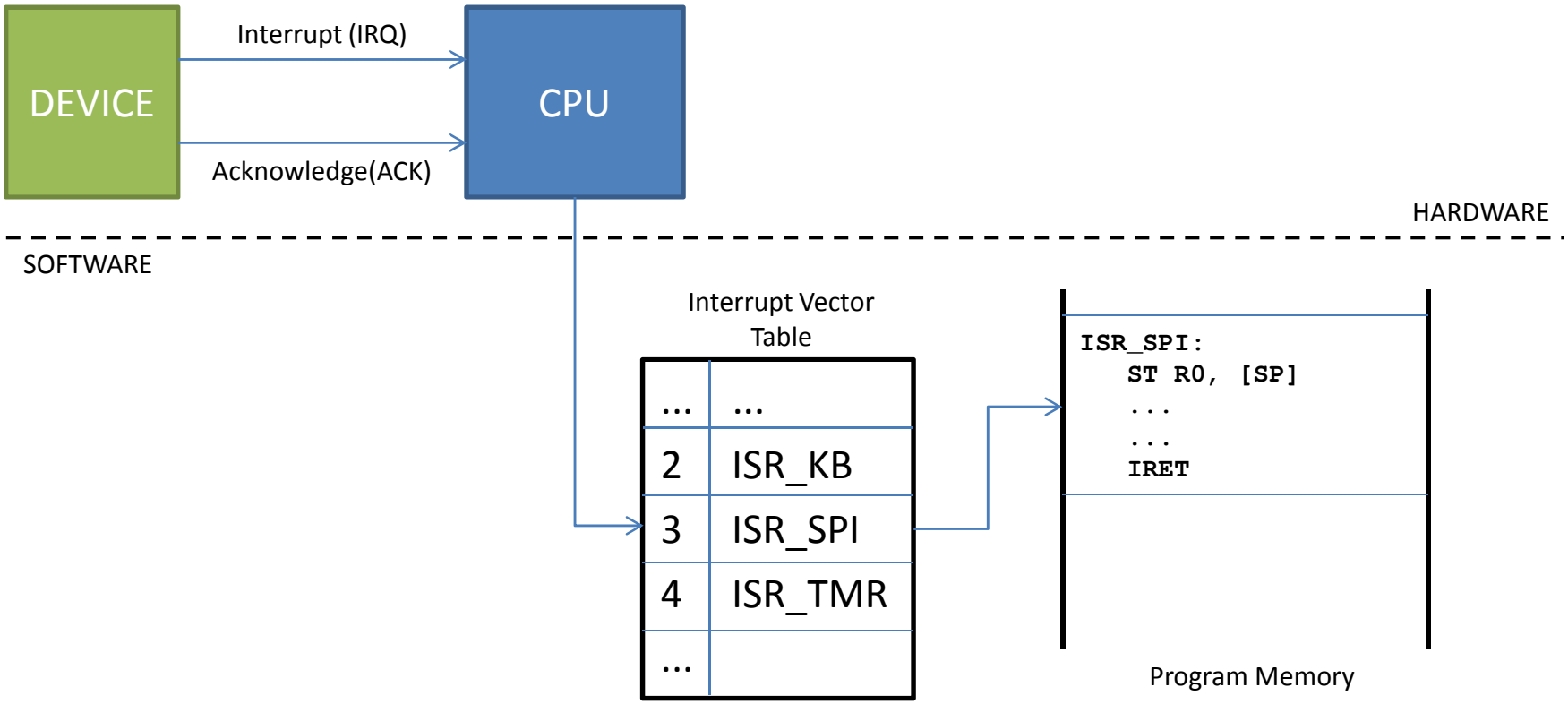


Kernel

- **Il kenel racchiude e raggruppa le funczioni centrali del sistema operativo**
  - Ha una complessità variabile a seconda dell'architettura
- **Nella maggior parte dei casi**
  - La funzione principale del kernel è quella di realizzare la "virtualizzazione dei processi"
- **Ciò comprta diversi aspetti**
  - Virtualizzazione della memoria
  - Virtualizzazione delle periferiche
    - Terminali
    - Filesystem
    - Interfaccia di rete
    - ....
  - Realizzazione del concetto di processo
    - Processi
    - Thread
- **Ciò comporta, al livello più basso**
  - La gestione degli eventi hardware, ovvero gli interrupt

Gestione degli interrrupt

- **Quando si verifica un interrupt**
  - Il processore lo gestisce a livello hardware
  - In assenza di sistema operativo è necessario gestilo in modo diretto



Gestione degli interrrupt

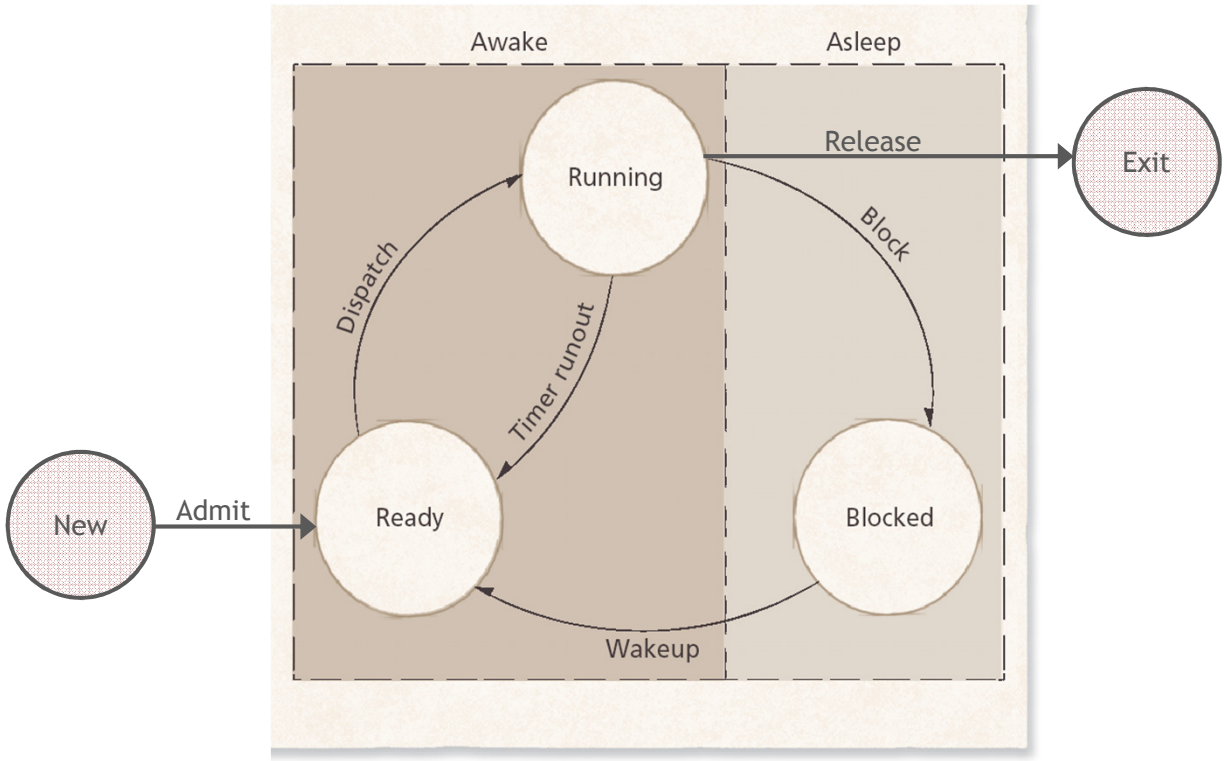
- **Ciò non permette al sistema operativo di avere il controllo sull'hardware**
- **Soluzione alternativa**
  - Il sistema operativo
    - Incorpora tutte le routine di gestione degli interrupt
    - Grazie alla gestione dei processi, "associa" una routine di gestione al processo in esecuzione
  - Dal punto di vista implementativo ciò comporta la centralizzazione degli interrupt
    - Ancora una volta il sistema operativo fornisce una astrazione dell'hardware
  - Gestione degli interrupt
    - In parte a carico direttamente del kernel
    - In parte a carico dei driver dei dispositivi
- **Quando si verifica un interrupt**
  - Il processore lo gestisce a livello hardware
    - Priorità, scheduling, ...
  - In risposta si ha l'esecuzione di una routine di gestione
    - Riconoscimento del tipo di interrupt, esecuzione della funzione richiesta
    - Ciò avviene in parte nel contesto del kernel, in parte in quello del processo

Operazioni del kernel

- **Creazione di un processo**
  - Il SO alloca una zona di memoria dedicaata al processo
  - Il SO copia il codice eseguibile del programma nella memoria allocata
- **Esecuzione di un processo**
  - Lo scheduler sceglie un processo tra quelli pronti e lo pone in esecuzione
- **Richiesta di un servizio**
  - Un processo richiede un servizio di sistema operativo mediante una speciale chiamata
  - Il SO esegue una specifica funzione "per conto" e "nel contesto" del processo
- **Sospensione su un evento**
  - Un servizio richiesto dal processo deve attendere un evento perr poter essere eseguito
  - Il SO pone il processo in stato di attesa
  - Lo scheduler sceglie un nuovo processo da eseguire
- **Sospensione per esaurimento del quanto di tempo**
  - Quando si esaurisce il quanto di tempo il SO sospende il processo in esecuzione
  - Il SO pone il processo sospeso nello stato di pronto
  - Lo scheduler sceglie un nuovo processo da eseguire

Operazioni del kernel

- **A tale scopo è necessario definire il concetto di "stato" di un processo**
  - Lo stato di un processo è impostato dal sistema operativo
  - Le transizioni tra gli stati dei processi sono definite rigidamente da un modello FSM
- **Un mdoello minimale di diagrammd i transizione di stato è il seguente**



Strutture dati

- La struttura principale contiene le informazioni relative ad un processo:

```
typedef struct ProcessDescr {
  ProcessStatus Status;            // The process status
  char*         StackPtr;          // Process system's stack pointer
  char*         BasePtr;           // Process base address
  int           Event;             // Event on which the process is suspended
  int           Files[MAX_FILES]; // Open files table
  ...
};
```

- Lo stato di un processo è descritto da un'enumerazione:

```
typedef enum {
  PS_NEW,
  PS_RUNNING,
  PS_READY,
  PS_BLOCKED,
  PS_TERMINATED,
  ...
} ProcessStatus;
```

Strutture dati

- **Il kernel deve quindi disporre di**
  - Una tabella contenente tutti i descrittori di processo
  - Una variabile che indica il processo corrente, ovvero l'indice del descrittore del processo all'interno della tabella

```
// Maximum number of processes allowed
#define MAX_PROCESS   ...

// Process table
ProcessDescr ProcessTable[MAX_PROCESS];

// Current process
int CurrentProcess;
```

- **Le informazioni relative al processo corrente si ricavano accedendo ai vari campi della struttura:**

```
ProcessTable[CurrentProcess]
```

Servizi di sistema per la gestione dei processi

- Sono funzioni di sistema operativo che manipolano la tabella dei processi
- Sospensione di un processo in attesa di un evento

```
void SleepOn( int event );
```

- Cambiamento del processo corrente o context switch

```
void Change();
```

- Ripresa di un processo sospeso su un dato evento

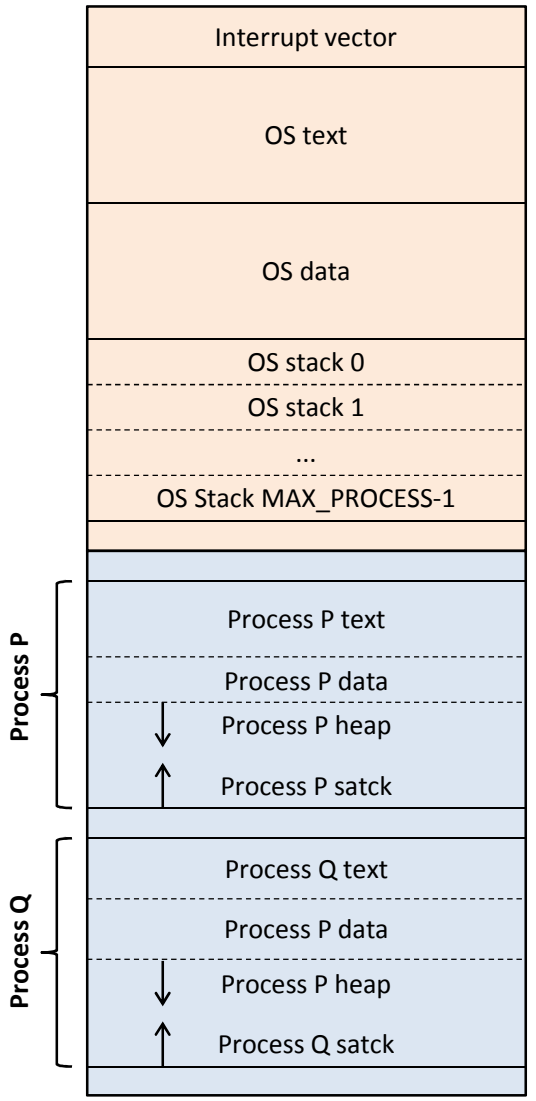
```
void WakeUp( int event );
```

- Sospensione di un processo da parte del sistema operativo a causa dell'esaurimento del suo quanto di tempo

```
void Preempt();
```

Organizzazione della memoria

- **La memoria (virtuale) del sistema è organizzata secondo una ben precisa struttura logica**
- **Vettore delle interruzioni (interrupt vector)**
  - Contiene gli indirizzi delle ISR
- **Memoria di sistema**
  - Codice del sistema operativo (OS text)
    - Servizi di sistema, Driver, ...
  - Strutture dati del sistema operativo (OS data)
    - Process table, current process
  - Stack di sistema (OS stacks)
- **Memoria utente**
  - Codice dei vari processi (Process text)
  - Dati dei vari processi (Process data)
    - Stack dei vari processi (cresce verso il basso)
    - Heap dei vari processi (cresce verso l'alto)

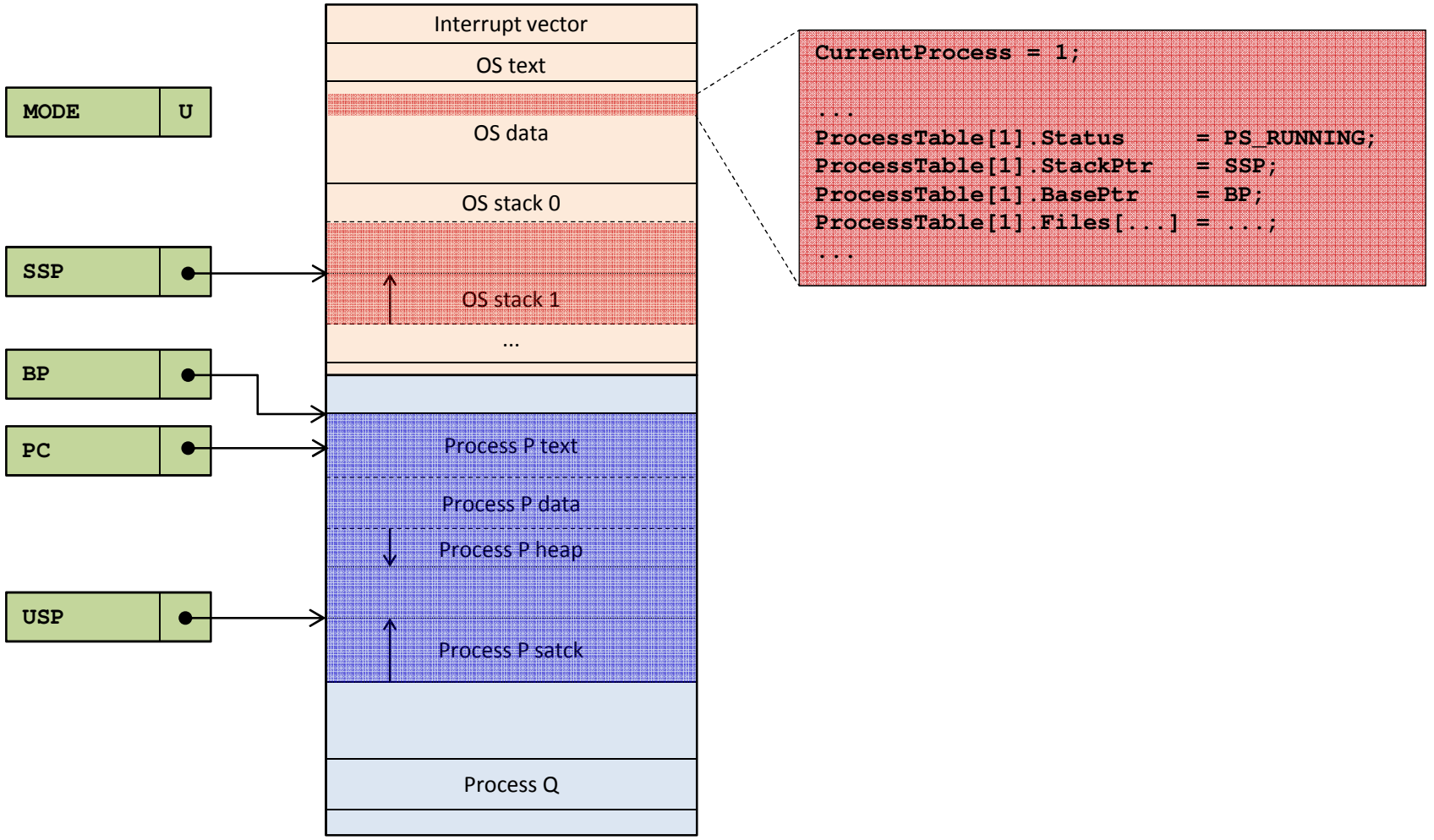


Organizzazione della memoria

- **Per poter gestire i processi**
  - Sono necessarie altre informazioni
  - Tali informazioni sono generalmente memorizzate in opportuni "registri" dedicati
    - Registri speciali della CPU
    - Locazioni di memoria nella zona di sistema operativo
- **Mode: Modo di esecuzione**
  - S: Supervisor
  - U: User
- **USP: User Stack Pointer**
  - Punta alla cima dello stack di un processo in memoria user
- **SSP: System Stack Pointer**
  - Punta alla cima dello stack di un processo nella memoria di sistema
- **BP: Base Pointer**
  - Punta alla base dell'area di memoria di un processo
- **PC: Program Counter**
  - Punta alla prossima istruzione da eseguire

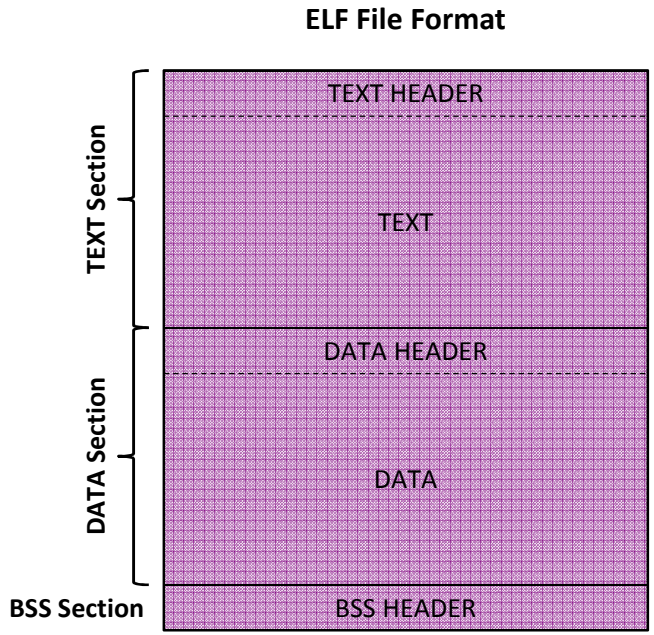
Organizzazione della memoria

- Un processo P è completamente descritto dalle informazioni mostrate



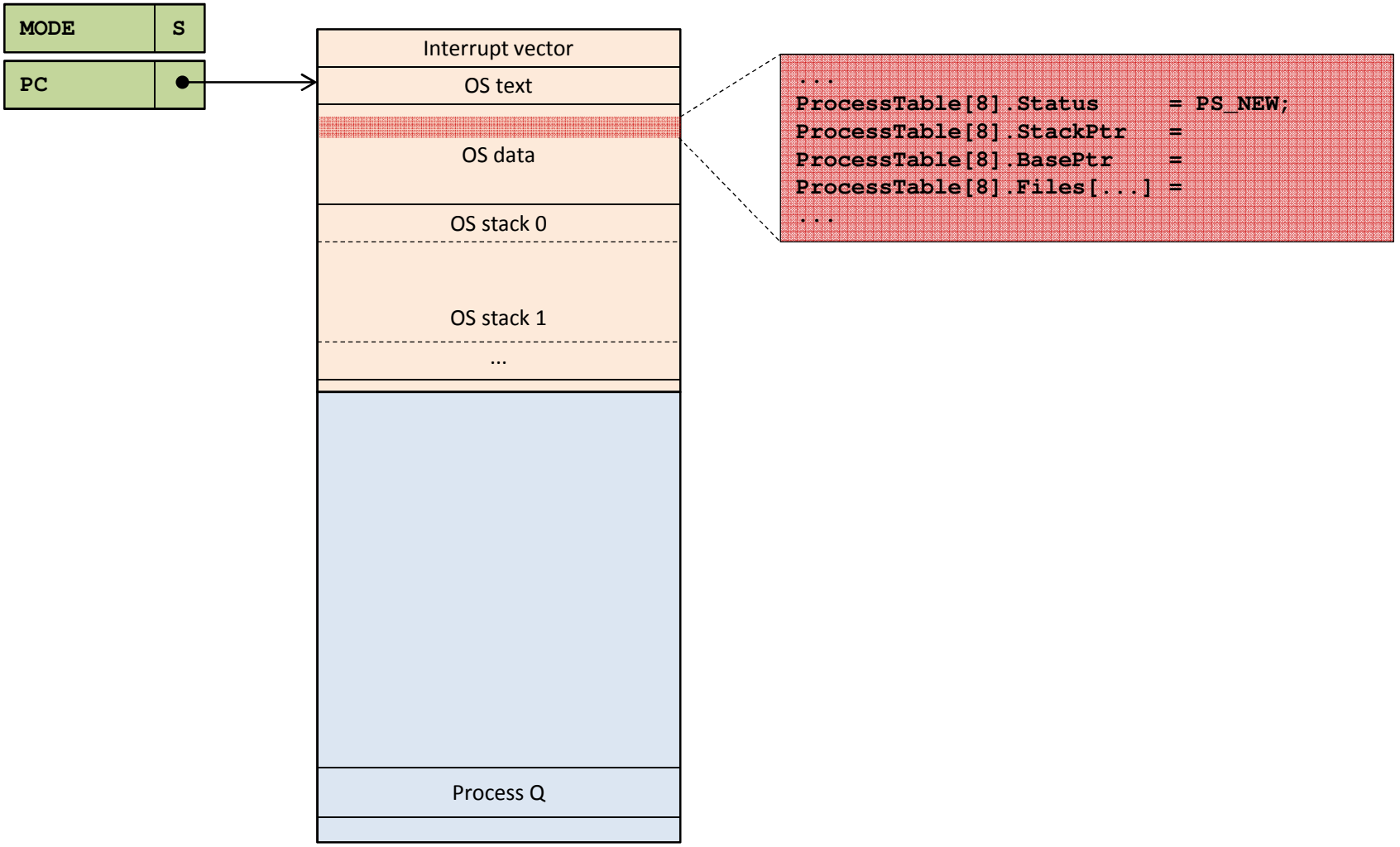
Creazione di un processo

- **La compilazione di un programma produce come risultato un file binario**
  - Nei sistemi Linux tale formato prende il nome di ELF: Executable and Linkable Format
    - Executable: Il file contiene (anche) codice eseguibile
    - Linkable: Il file contiene tutte le informazioni necessarie al linker / loader
- **Un file ELF è diviso in "sezioni"**
  - Sono molte e variabili da caso a caso
  - Contengono informazioni di natura diversa
- **Le sezioni principali sono tre**
  - TEXT (.text)
    - Contiene il codice macchina del programma
  - DATA (.data)
    - Contiene i dati del programma inizializzati
      - (variabili globali, costanti, ...)
  - BSS (.bss)
    - E' una sezione vuota
      - Ciò che conta è solo la sua dimensione
    - Indica la dimensione dei dati non inizializzati



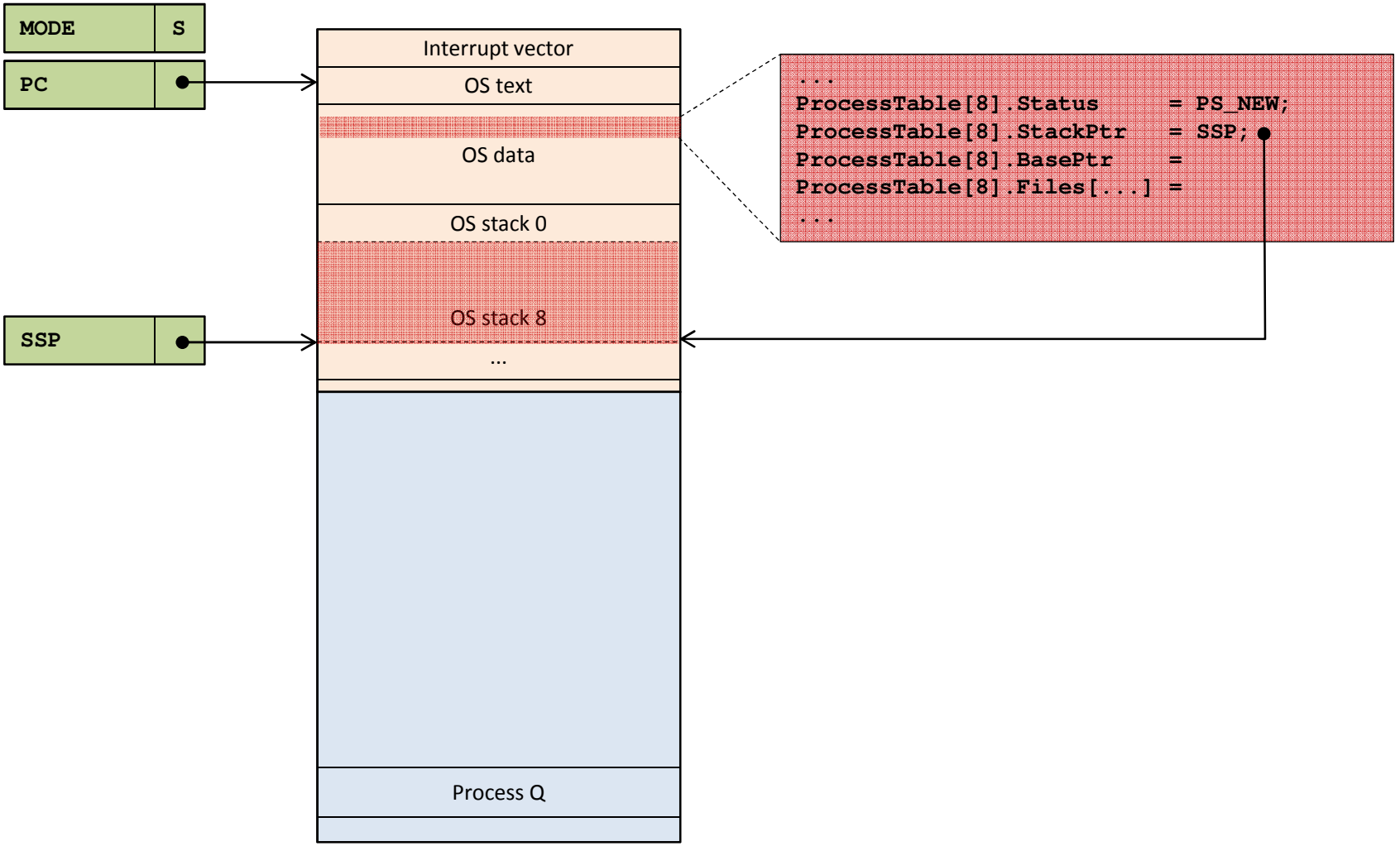
Creazione di un processo

1. Il SO crea una nuova entry nella Process Table e pone lo stato a NEW



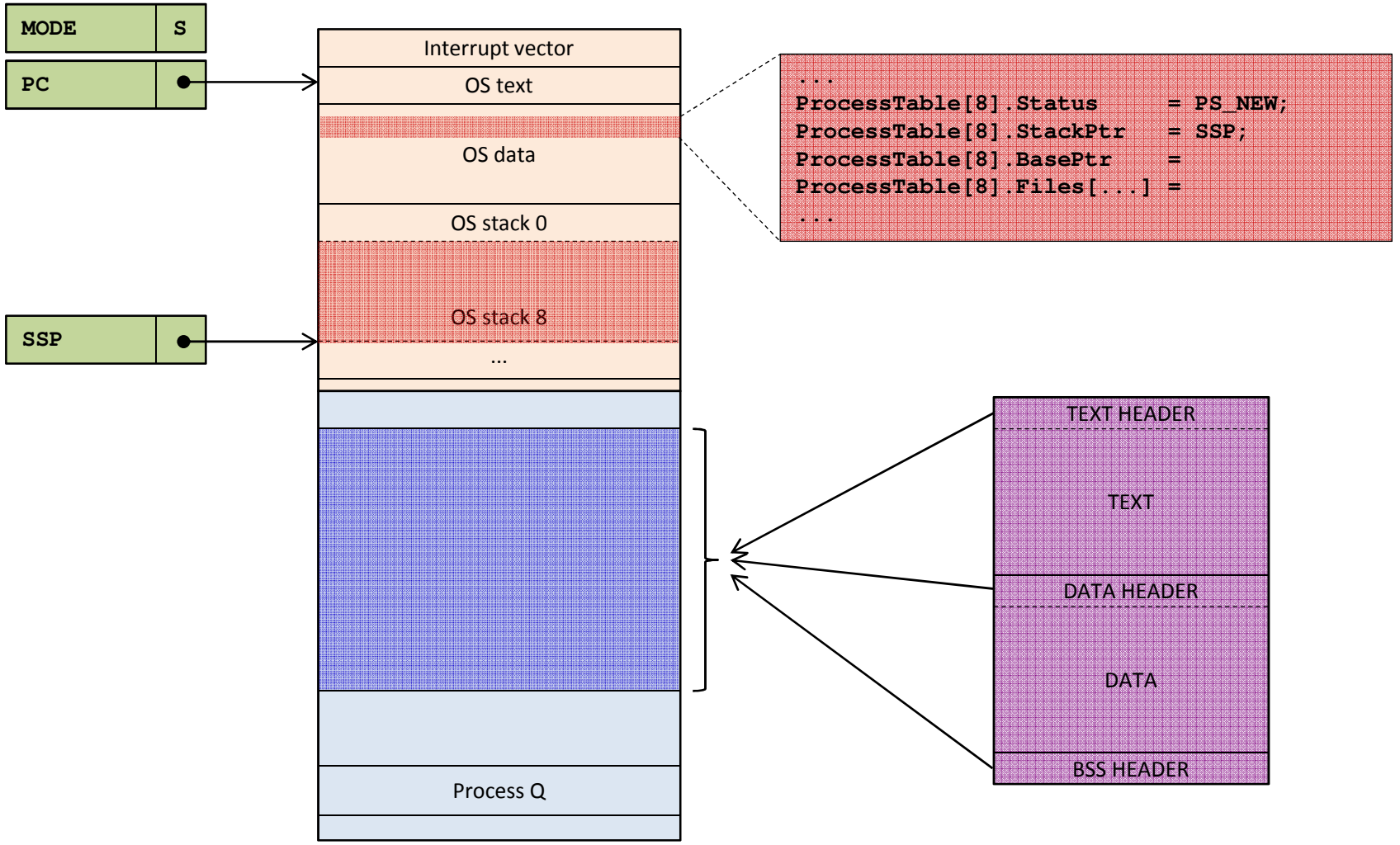
Creazione di un processo

2. Alloca lo stack di sistema ed inizializza il puntatore nella Process Table



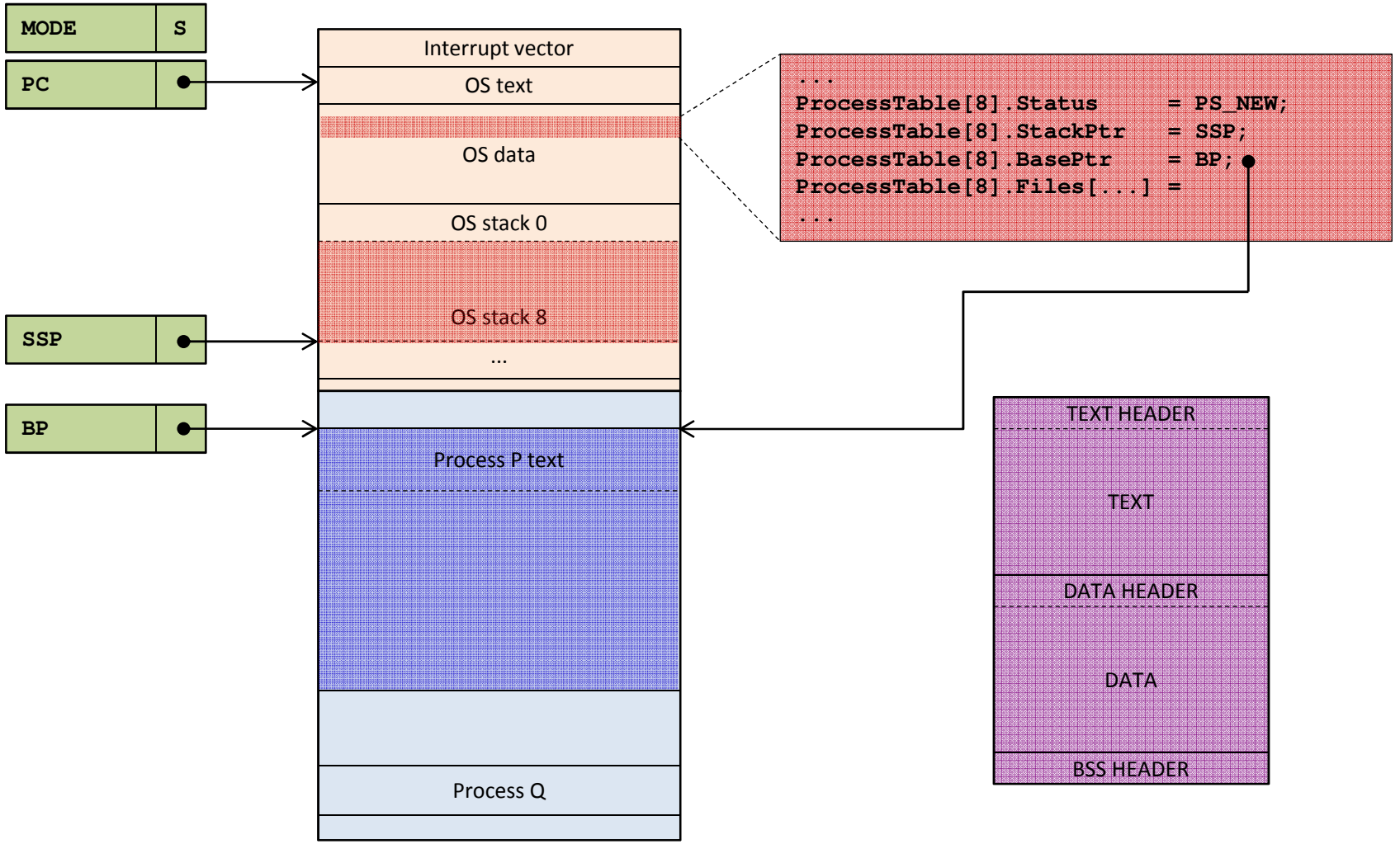
Creazione di un processo

3. Alloca la memoria necessaria nella zona User in base alle informazioni nell'ELF



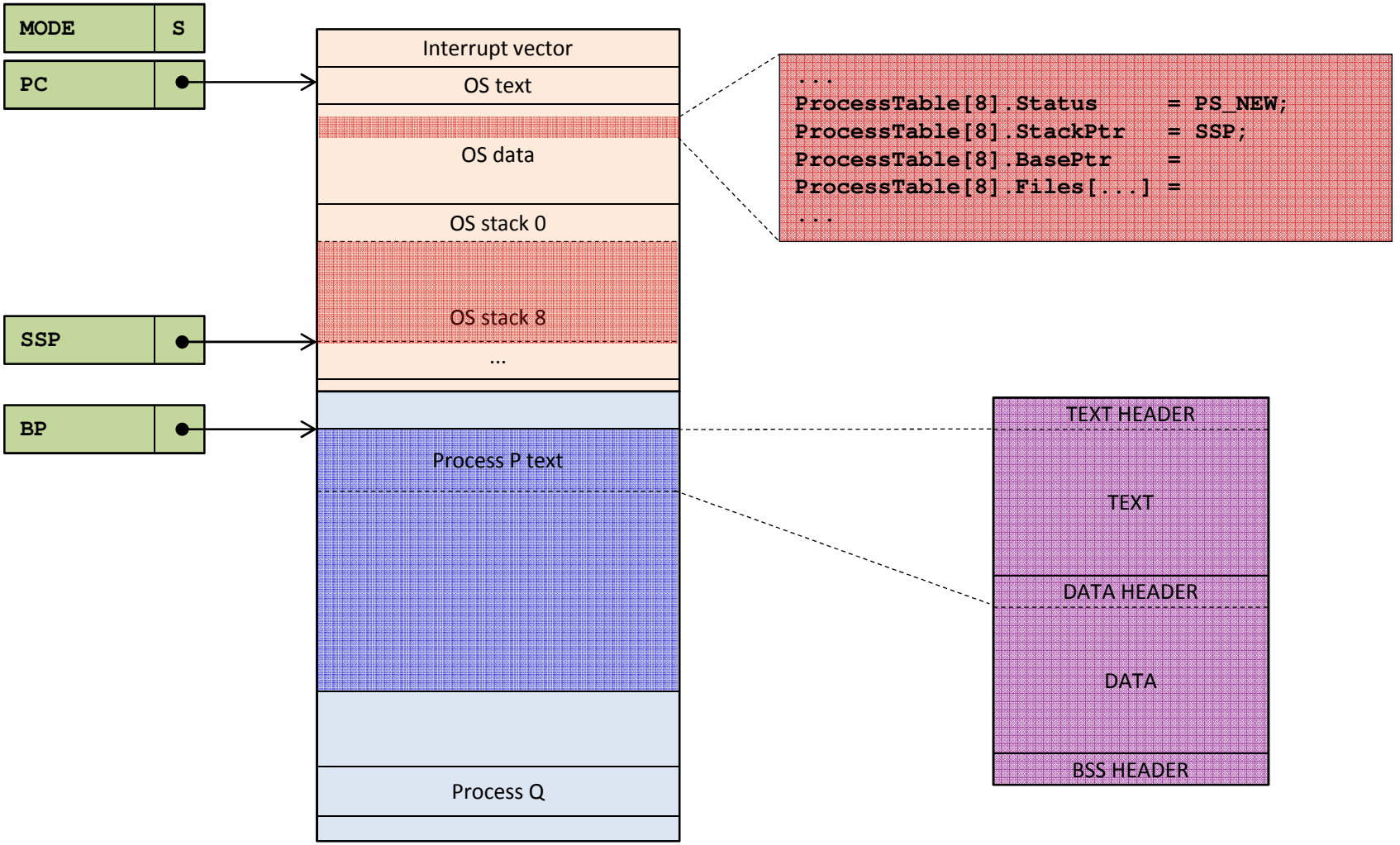
Creazione di un processo

4. Imposta il Base Pointer all'inizio della memoria riservata per il processo



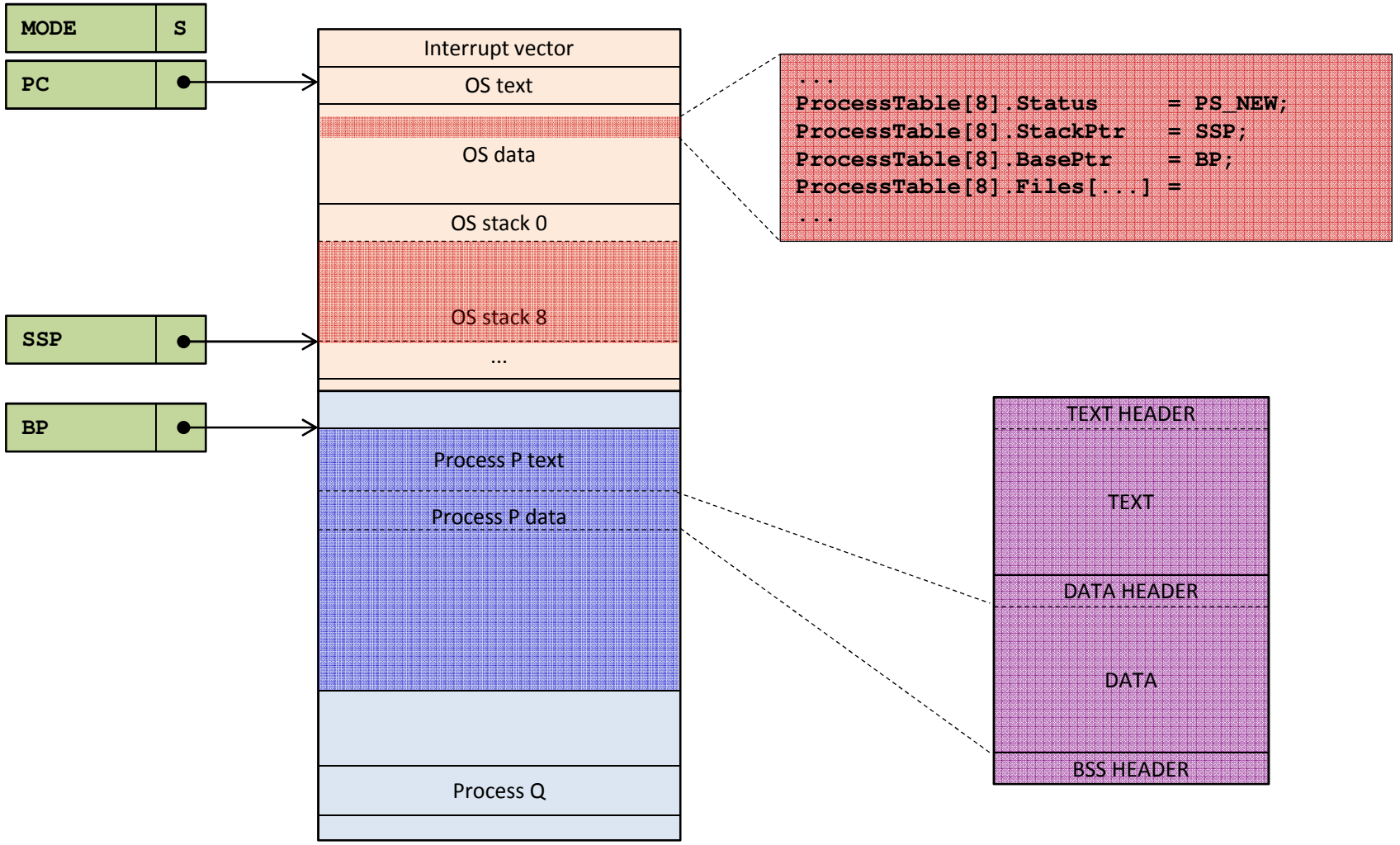
Creazione di un processo

5. Copia la sezione TEXT del file ELF nella memoria appena allocata



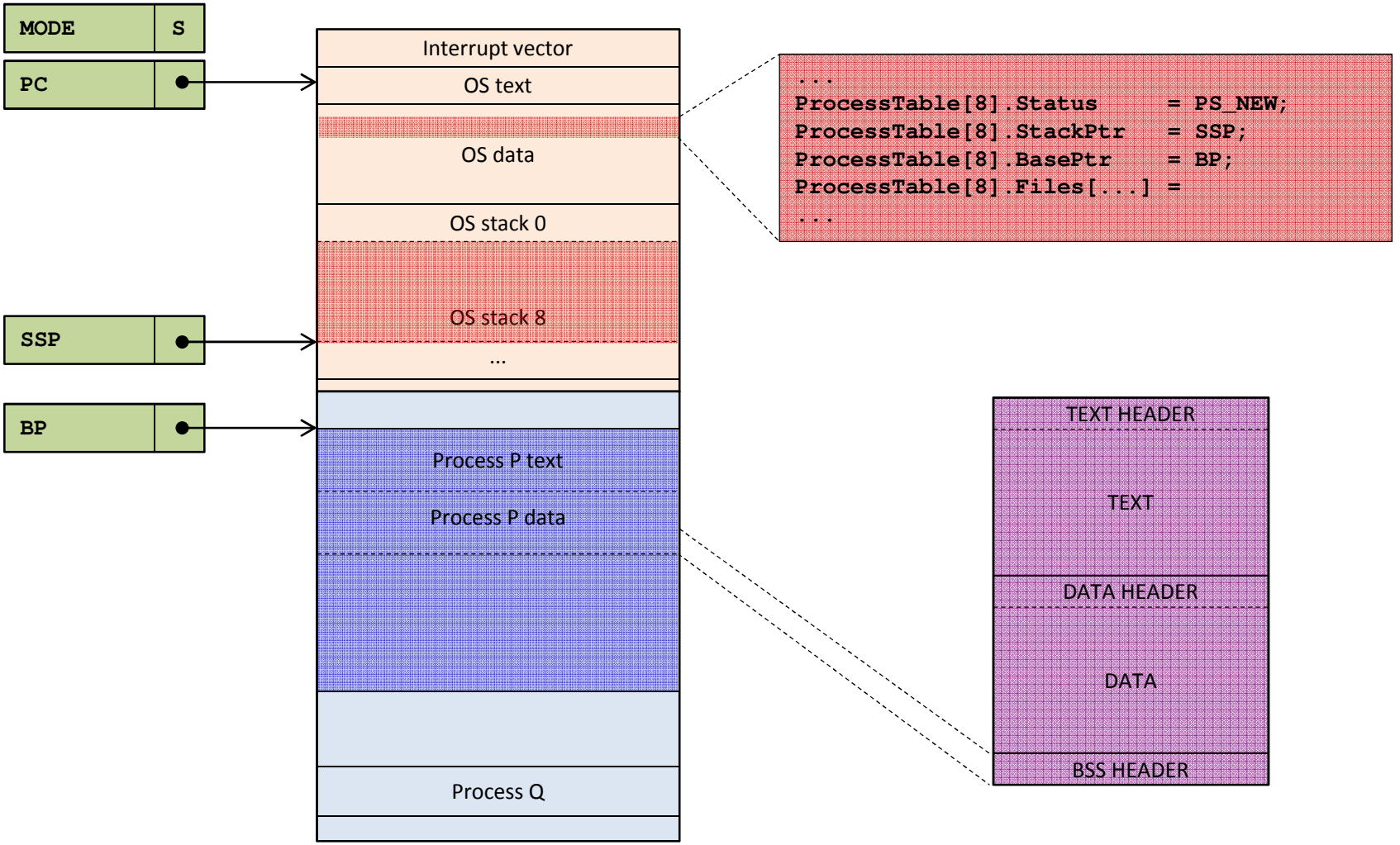
Creazione di un processo

6. Copia la sezione DATA del file ELF nella memoria appena allocata



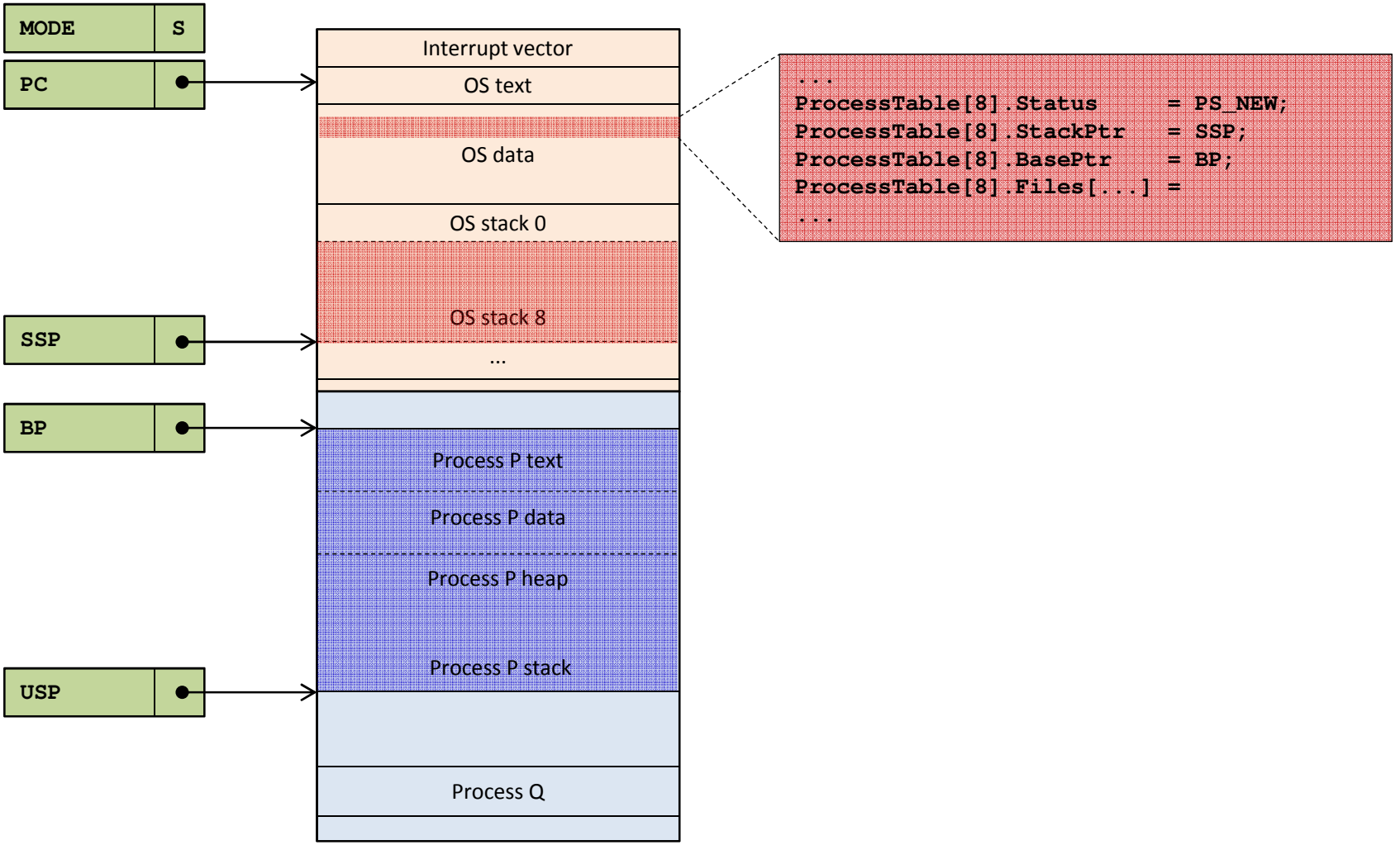
Creazione di un processo

7. Alloca la memoria richiesta dalla sezione BSS e la inizializza a 0



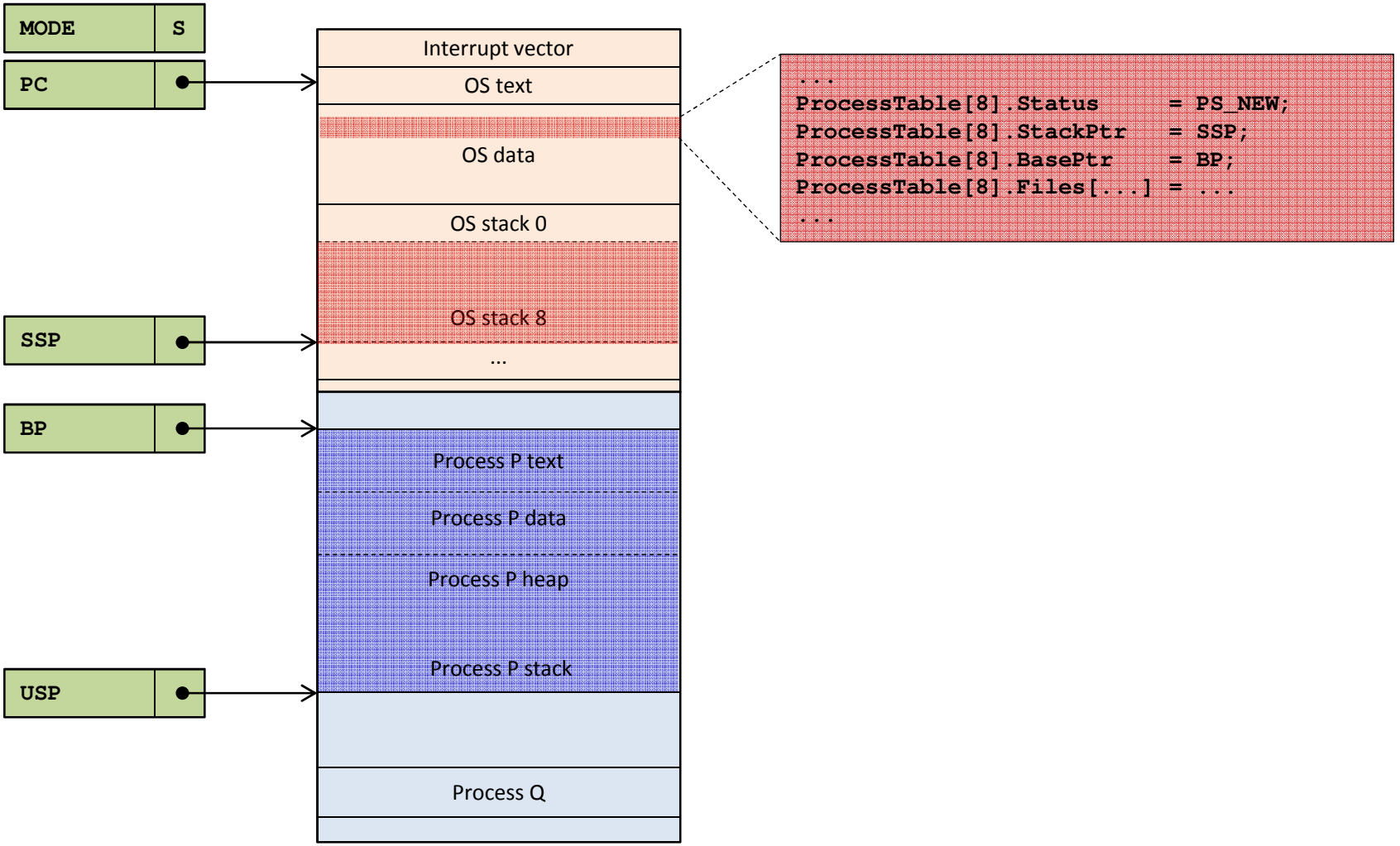
Creazione di un processo

8. Inizializza lo User Stack Pointer alla basse dello stack



Creazione di un processo

9. Inizializza tutti i campi rimanenti del process descriptor (tabella dei file, ...)



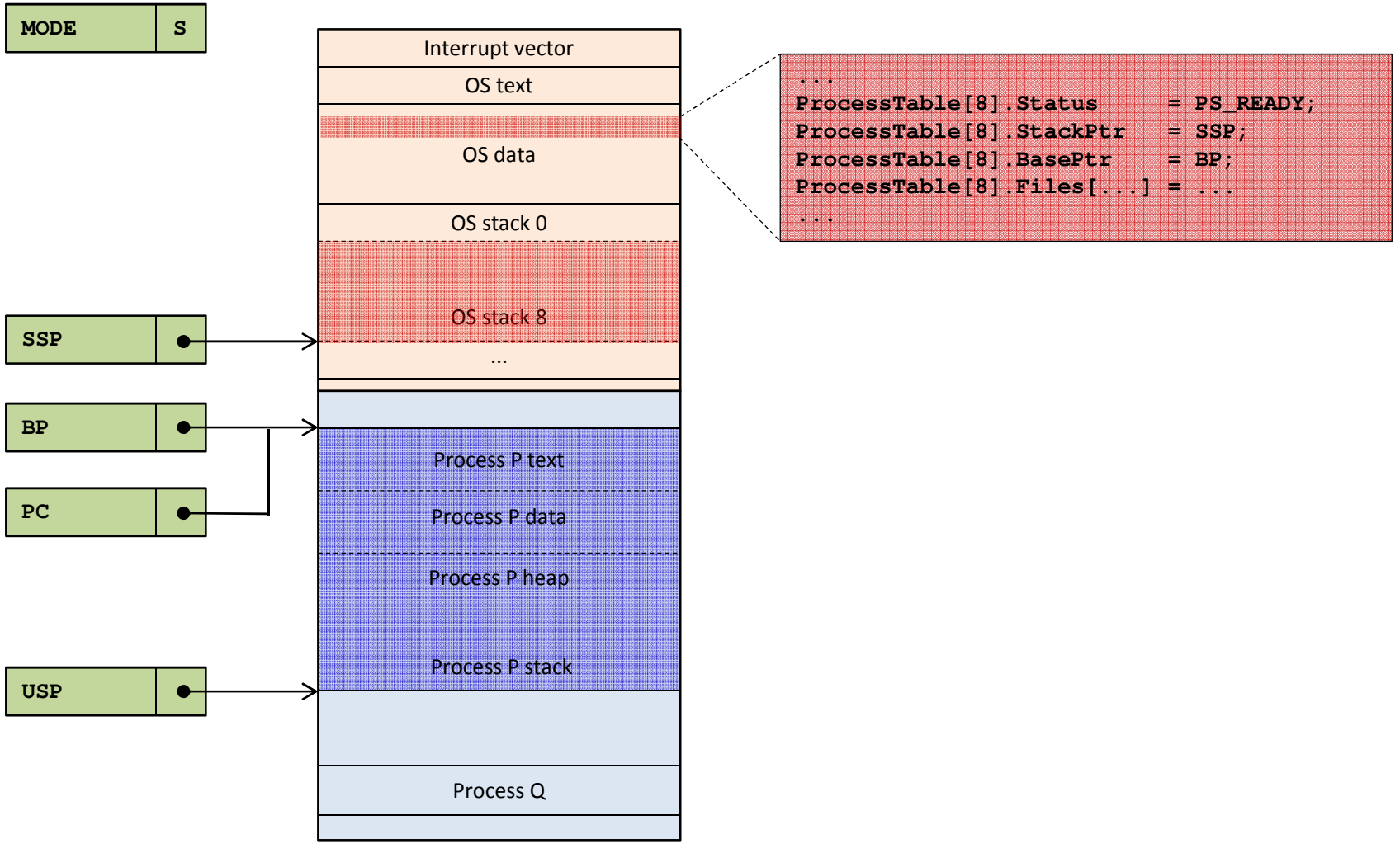
Esecuzione di un processo

- **L'esecuzione di un processo può avvenire**
  - Su richiesta di un processo
    - Mediante chiamate di sistema fork() / exec()
  - Su richiesta dell'utente
    - Di fatto si tratta ancora di chiamate di sistema fork() / exec()
    - Le chiamate sono eseguite dalla shell

- **L'esecuzione avviene mediante due passaggi di stato**
  - Da PS_NEW a PS_READY
    - A carico dello scheduler di lungo periodo
    - Il nuovo processo entra nella coda dei processi gestiti dallo scheduler di breve termine
  - Da PS_READY a PS_RUNNING
    - A carico dello scheduler di breve periodo

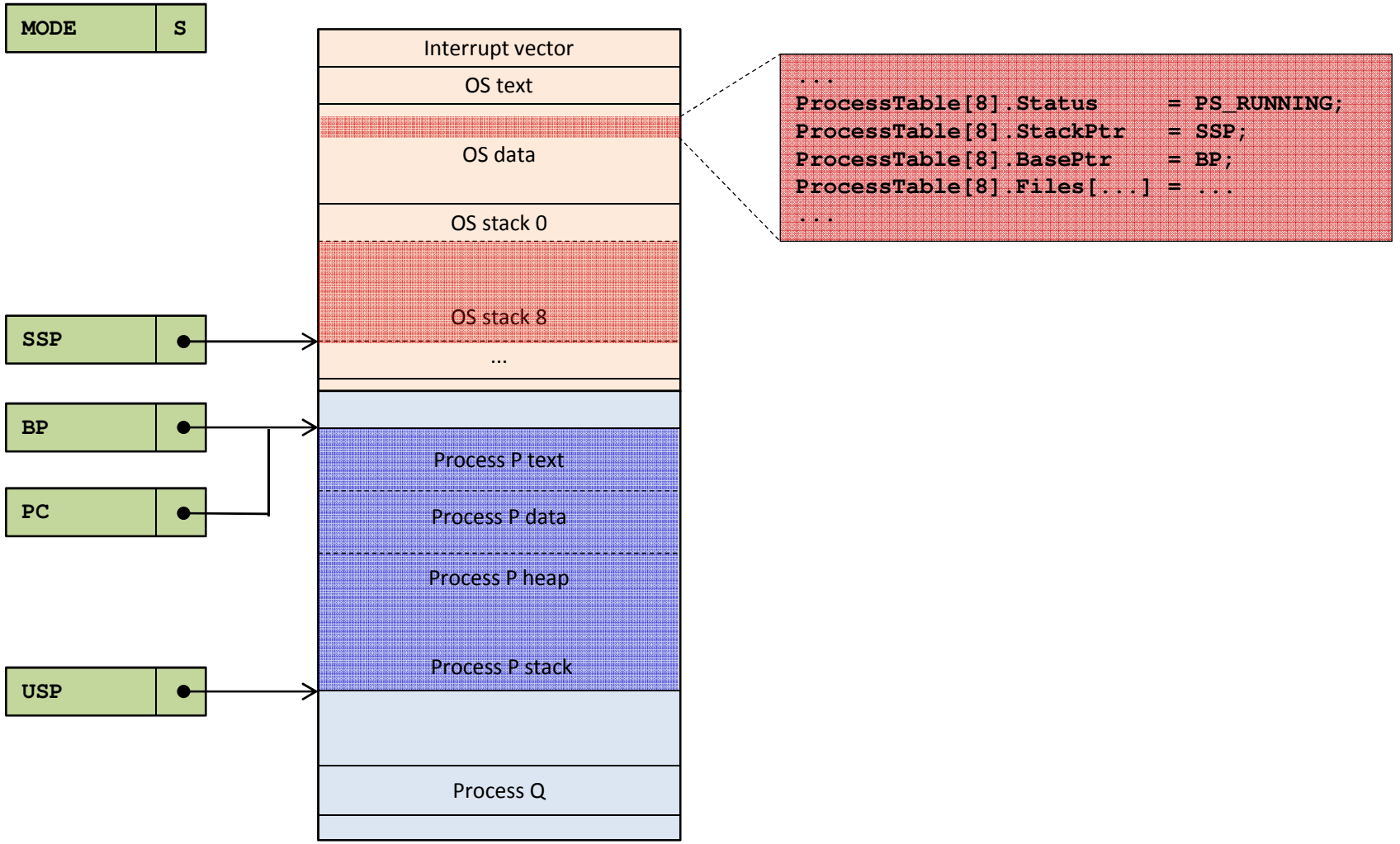
Esecuzione di un processo

1. Lo scheduler di lungo periodo pone lo stato a PS_READY e imposta il PC



Esecuzione di un processo

2. In un secondo momento lo scheduler pone lo stato a PS_RUNNING



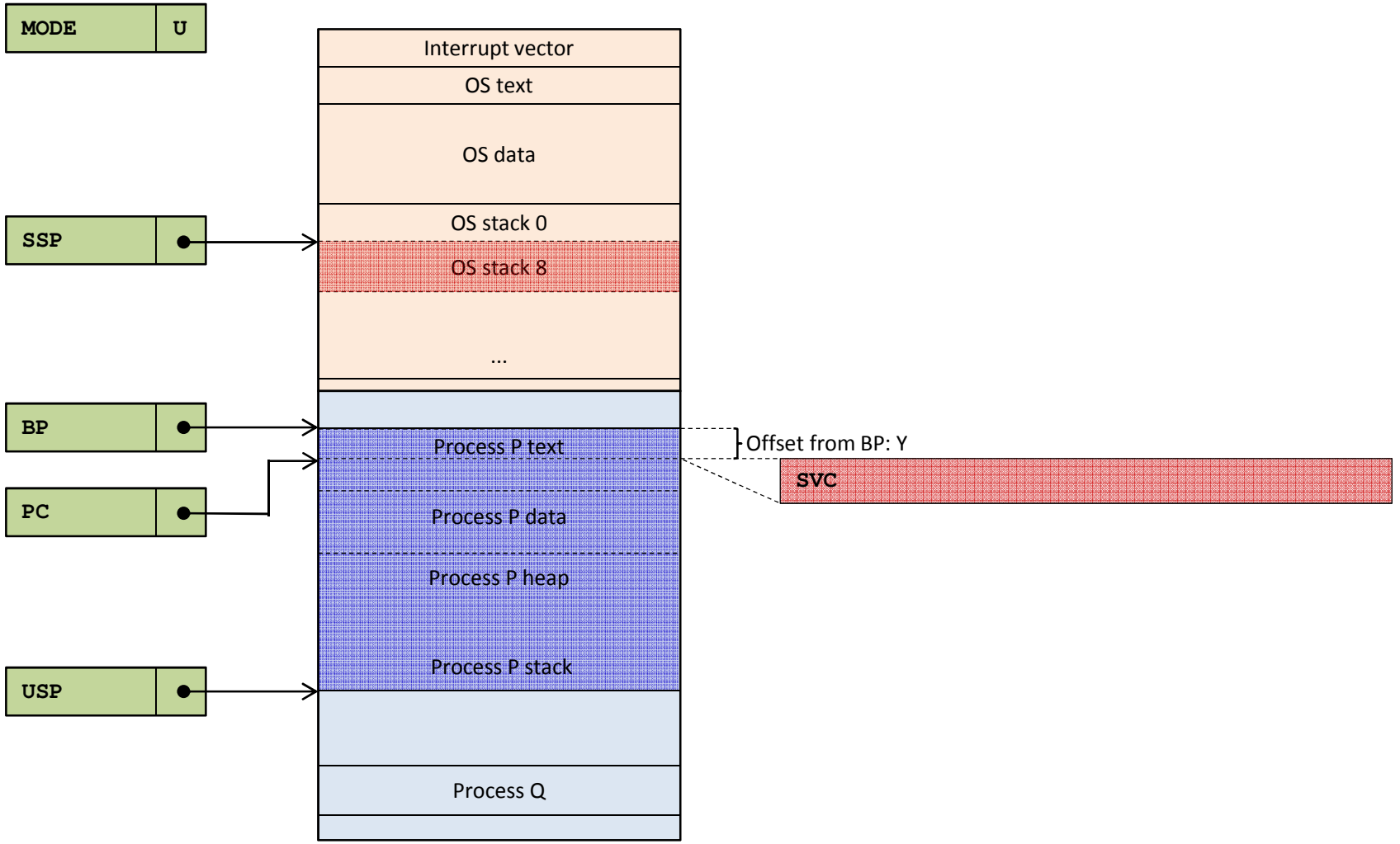
Richiesta di un servizio di sistema

- **A livello di codice sorgente la richiesta di un servizio di sistema**
  - Avviene mediante la chiamata di una funzione di libreria
    - Standard C Library
    - Standard C++
    - Standard Template Library
    - POSIX Library
    - ...
- **Una tipica funzione di libreria ha la struttura seguente**

```
int read( int fd, char* buffer, size_t nbytes )
{
  // Prolog code. Executed in User Mode
  // Checks parameters, prepares buffers, ...
  ...
  // System Call. Executed in Supervisor Mode
  ...
  // Epilog code. Executed in User Mode
  // Checks return codes, frees memory, ...
  return nread;
}
```

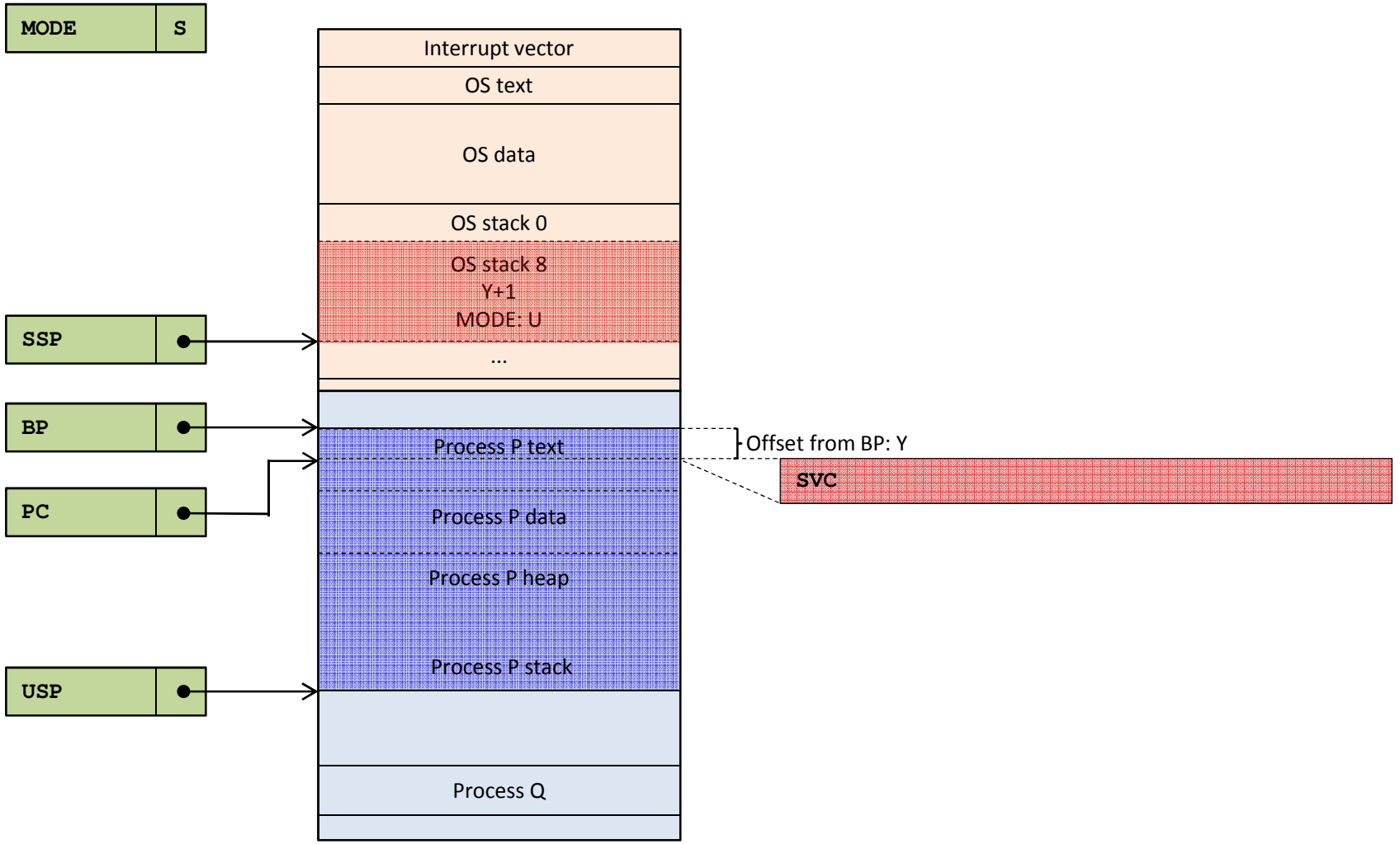
Richiesta di un servizio di sistema

1. Il processo richiede un servizio mediante l'istruzione SVC all'indirizzo BP + Y



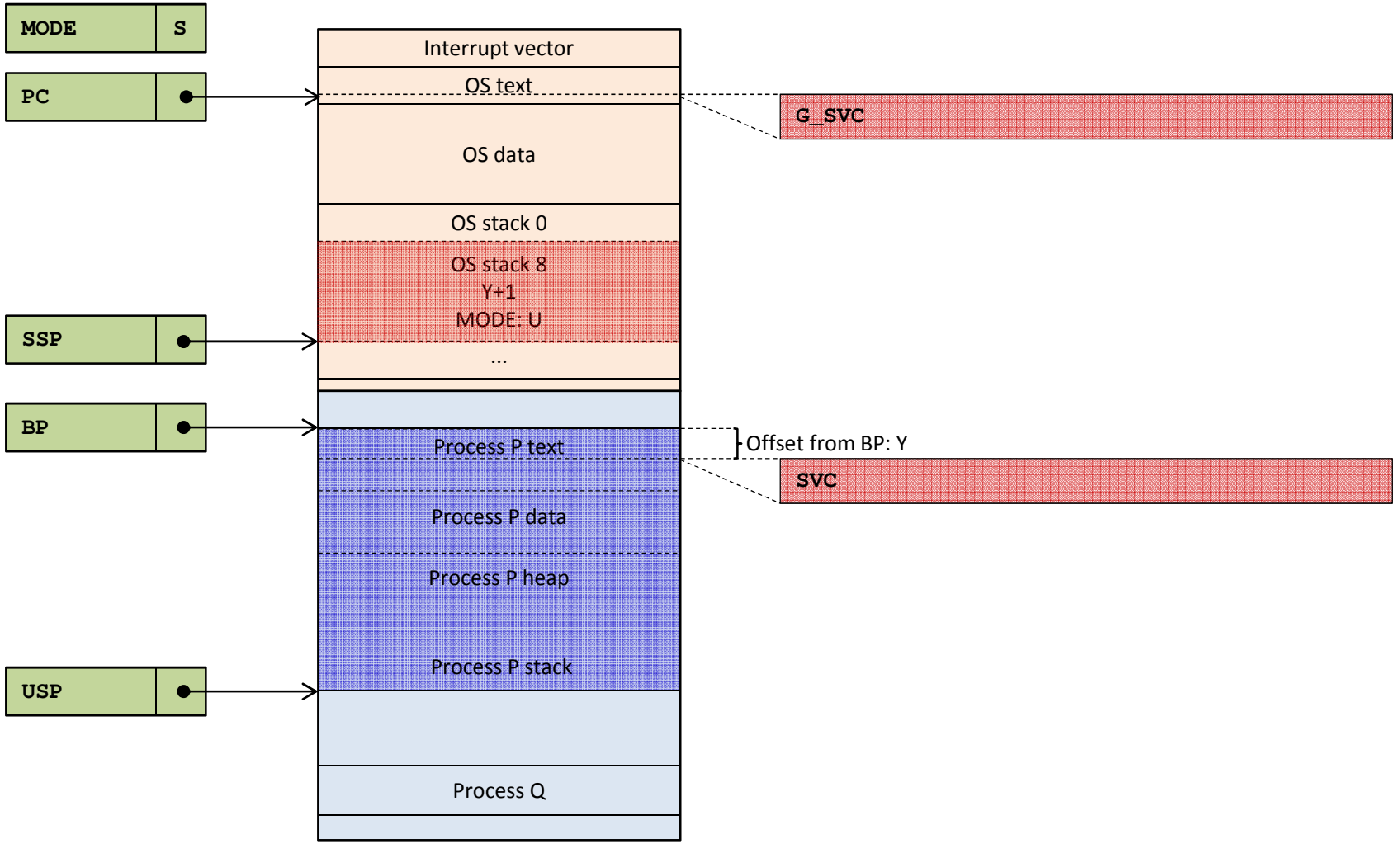
Richiesta di un servizio di sistema

2. Il sistema entra in modo S e salva indirizzo e modo di ritorno sul system stack



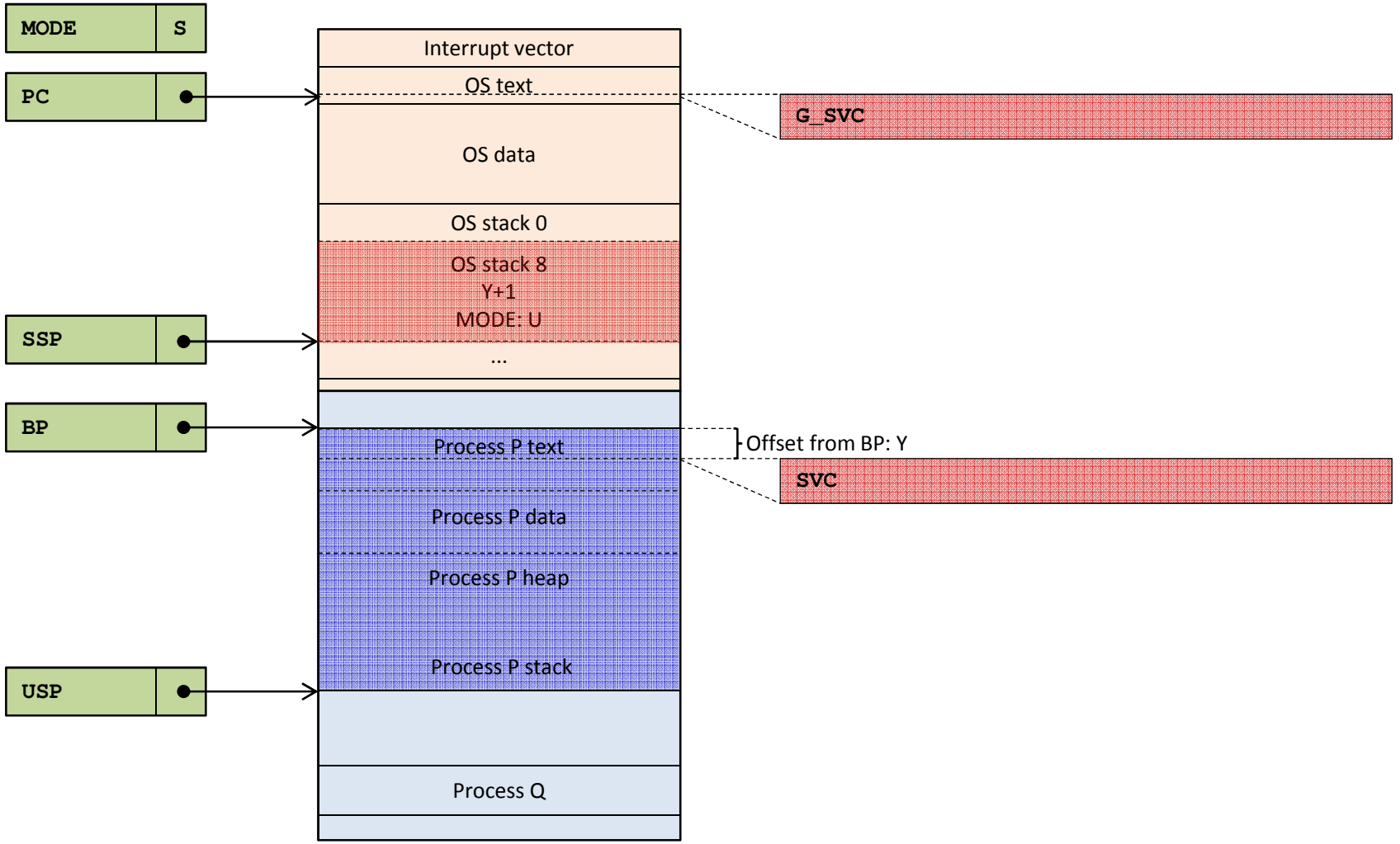
Richiesta di un servizio di sistema

3. Il program counter punta al codice del gestore dei servizi



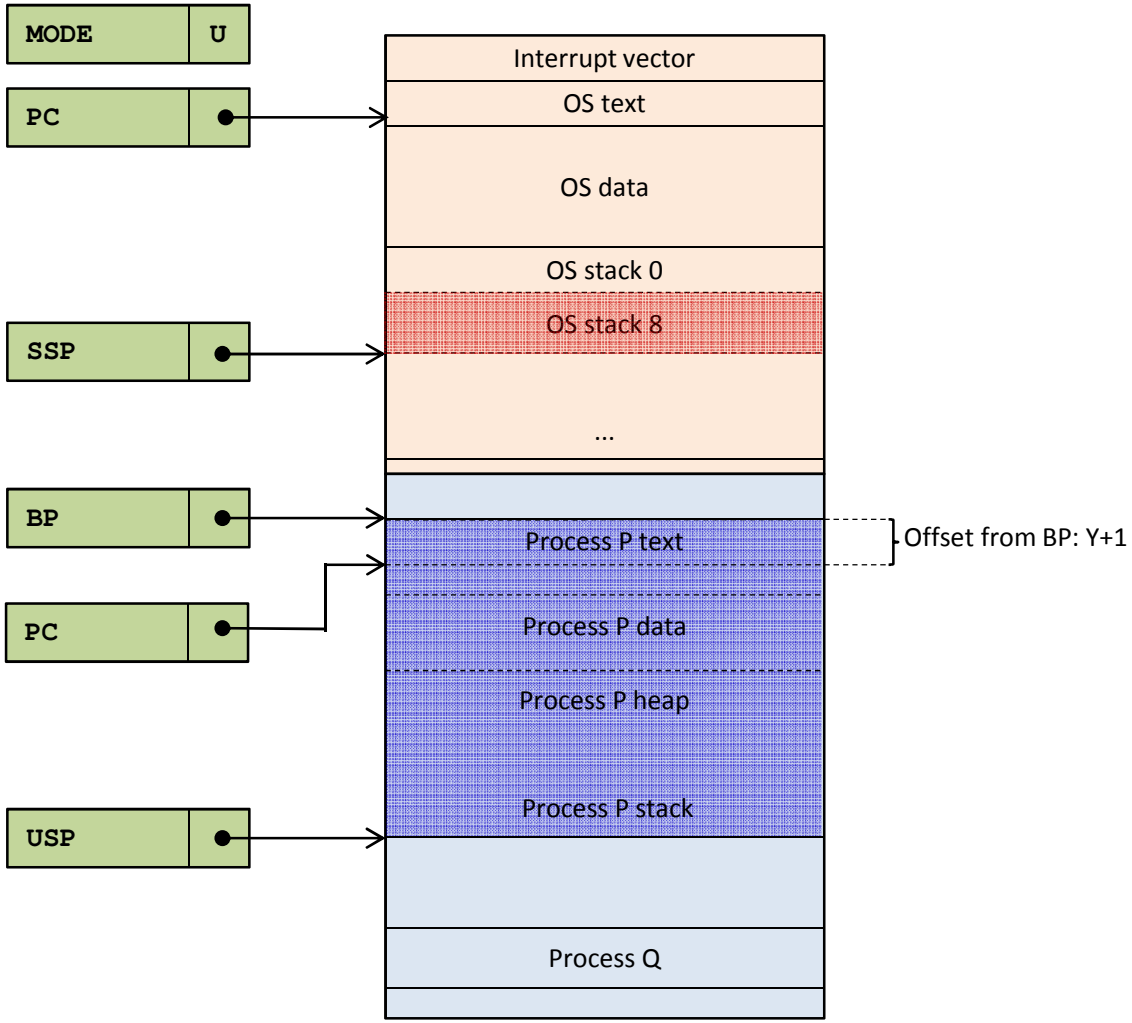
Richiesta di un servizio di sistema

4. Il sistema esegue il sevizio in modo S nel contesto del processo



Richiesta di un servizio di sistema

5. Al termine del sevizio il sistema ripristina il Program Counter e il modo salvati

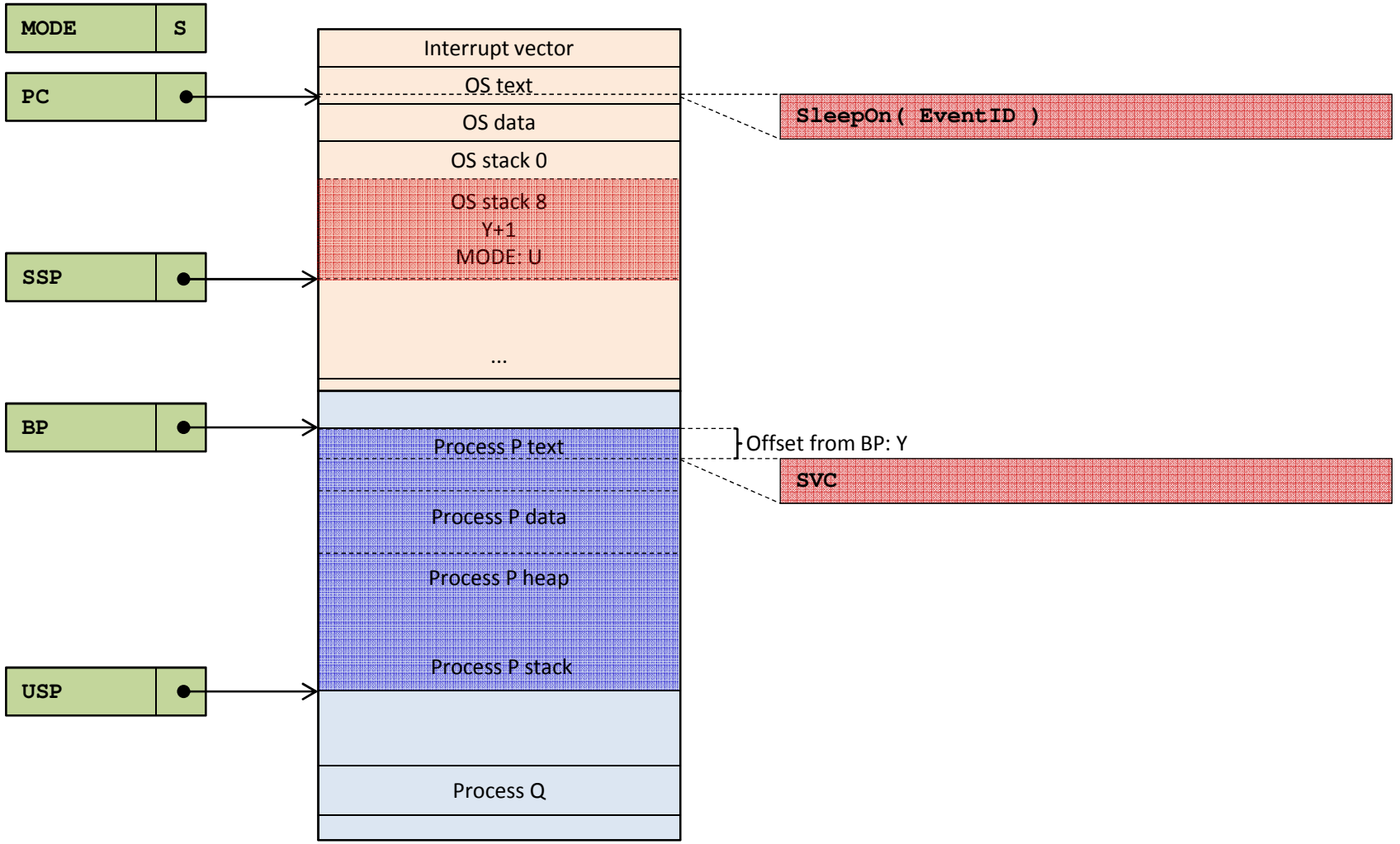


Richiesta di un servizio di sistema

- **A volte l'esecuzione di un servizio non può essere completata poiché il sistema deve attendere uno specifico evento**
  - Tipicamente un interrupt
  - Disponibilità dei dati
  - Rilascio di un lock
  - ...
- **In tal caso il servizio procede come segue**
  - Dapprima sospende il processo in esecuzione
  - Quindi passa il controllo ad un nuovo processo
- **Nella descrizione riprendiamo dal punto (4)**
  - Il sistema sta eseguendo un dato servizio
  - Il servizio è eseguito nel contesto del processo

Sospensione di un processo

1. Durante l'esecuzione del servizio il processo si sospende in attesa di un evento



Sospensione di un processo

- **La funzione SleepOn()**
  - Salva tutti i registri del processore sullo stack di sistema
    - General purpose registers, Status register, Base pointer, User stack pointer
  - Salva l'ID dell'evento nel descrittore del processo
  - Cambia lo stato del processo corrente da PS_RUNNING a PS_BLOCKED
  - Invoca la funzione Change() per eseguire il cambio di contesto

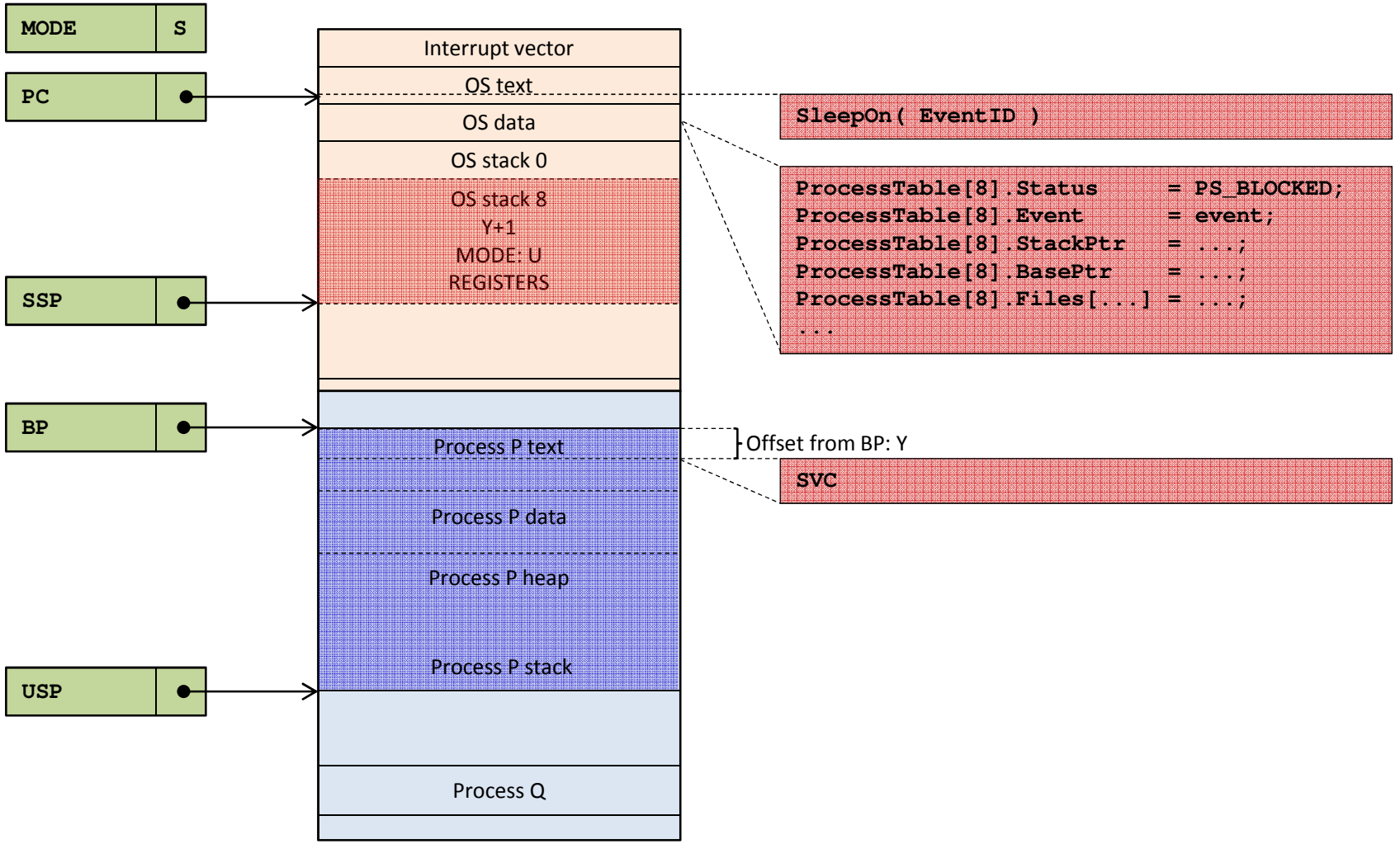
```
void SleepOn( int event ) {
  // Before context switch: save context
    // Save registers on SSP
    ProcessTable[CurrentProcess].Event  = event;
    ProcessTable[CurrentProcess].Status = PS_BLOCKED;

  // Context switch
    Change();

  // After context switch: restore context and returns
    // Restores registers
    return;
}
```

Sospensione di un processo

2. Prima di invocare la funzione Change() lo stato è il seguente



Sospensione di un processo

- **La funzione Change()**
  - Completa il salvataggio del contesto
    - Salva il valore corrente dello stack pointer di sistema nel descrittore del processo corrente
  - Invoca lo scheduler per scegliere il nuovo processo (Q, con PID=9)
  - Recupera dal descrittore del nuovo processo il base pointer e lo stack pointer di sistema
    - In questo modo l'esecuzione avviene ora nel contesto del processo Q

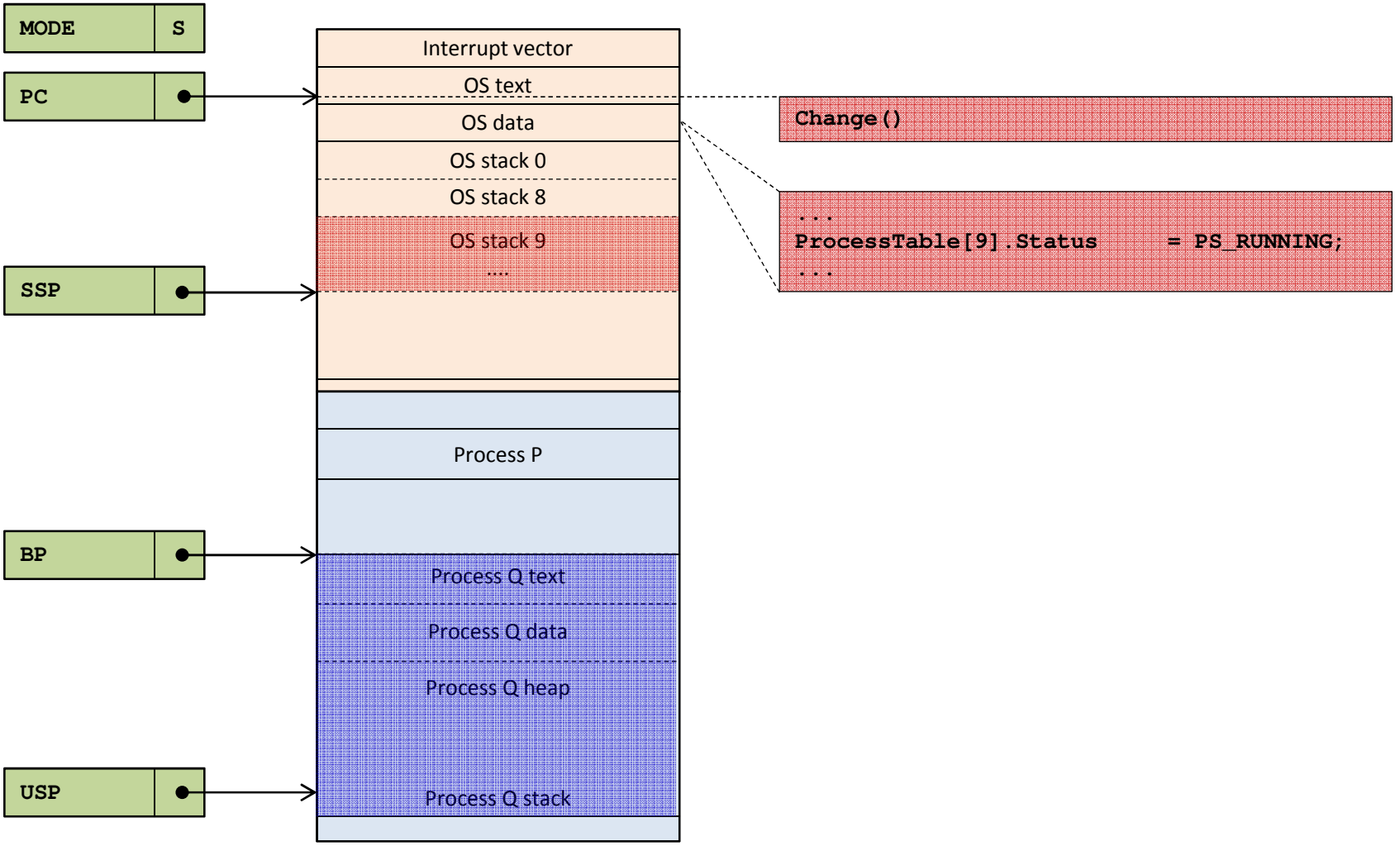
```
void Change() {
  // Completes context saving
  ProcessTable[CurrentProcess].StackPtr = SSP;

  // Selects the new process to execute
  CurrentProcess = Scheduler();

  // Starts the new process
  SSP = ProcessTable[CurrentProcess].StackPtr;
  BP  = ProcessTable[CurrentProcess].BasePtr;
  ProcessTable[CurrentProcess].Status = PS_RUNNING;
  return;
}
```

Sospensione di un processo

3. Prima dell'istruzione di ritorno dalla funzione Change() lo stato è il seguente

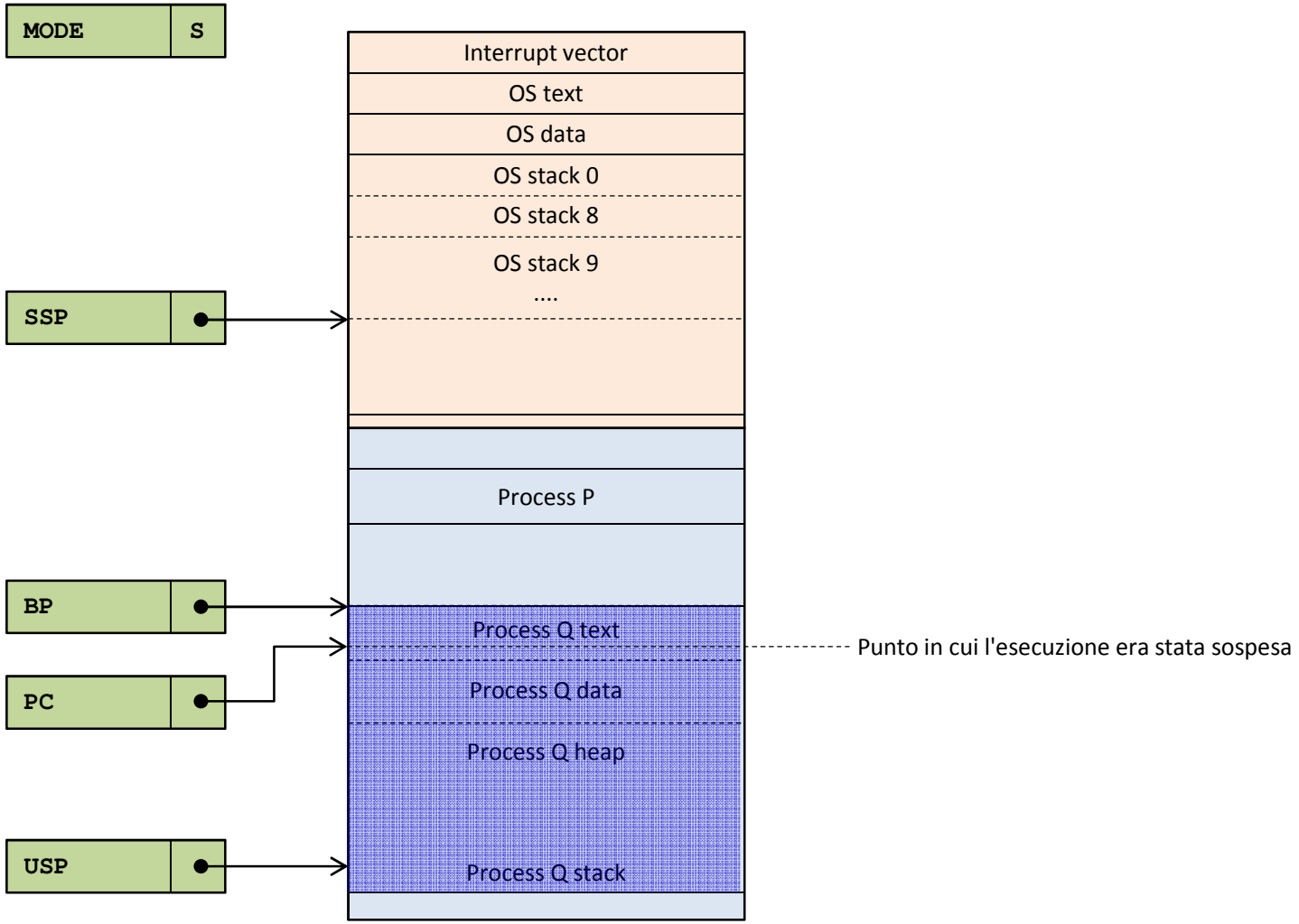


Sospensione di un processo

- **L'istruzione di ritorno della funzione Change() deve causare la ripresa dell'esecuzione del processo Q, nel punto in cui era stato sospeso**
- **L'istruzione di ritorno infatti preleva l'indirizzo di ritorno**
  - Dallo stack di sistema, poichè il processore è in modo supervisor
  - Lo stack di sistema è quello relativo al processo Q, poichè SSP punta a tale stack
- **Ne consegue che**
  - Il ritorno avviene a quella funzione che aveva invocato Change()
  - Tale invocazione era avvenuta in corrispondenza dell'ultima volta che l'esecuzione del processo Q era stata sospesa
- **Quindi il processo Q riprende l'esecuzione**
  - Come se non fosse avvenuto nulla dall'ultima volta in cui era stato sospeso
  - L'esecuzione di altri processi, avvenuta nel frattempo, non lo ha influenzato.

Sospensione di un processo

4. Prima dell'istruzione di ritorno dalla funzione Change() lo stato è il seguente



Ripresa di un processo sospeso

- **Supponiamo che durante l'esecuzione di Q si verifichi l'evento in attesa del quale il processo P era stato sospeso**
- **Gli eventi possono essere di natura**
  - Hardware: interrupt di periferiche o timer
  - Software: eventi del sistema operativo
- **In generale ad ogni evento corrisponde una routine di servizio**
- **Nel nostro esempio, mentre Q è in esecuzione**
  - Si verifica l'evento
  - Viene invocata la routine di servizio, diciamo, per esempio una ISR
  - La ISR invoca il servizio WakeUp()
    - Passando a questa funzione l'identificativo dell'evento
  - La funzione WakeUp()
    - Risveglia tutti i processi in bloccati
    - Restituisce il controllo alla ISR
  - La ISR termina e l'esecuzione di Q continua normalmente

Ripresa di un processo sospeso

- Lo pseudocodice della funzione WakeUp() ne chiarisce il funzionamento

```
void WakeUp( int event ) {
  int id;

  for( id = 0; id < MAX_PROCESS; id++ )
  {
    if( ProcessTable[id].Staus == PS_BLOCKED &&
        ProcessTable[id].Event == event )
    {
      ProcessTable[id].Staus = PS_READY;
      ProcessTable[id].Event = EV_NONE;
    }
  }
  return;
}
```

Esaurimento del quanto di tempo

- **Ad un certo punto il processo Q esaurirà il suo quanto di tempo**
  - Ad ogni processo è associato un quanto di tempo
    - Può essere uguale per tutti i processi oppure specifico per ogni processo
- **Il tempo viene misurato da una particolare ISR associata al "system clock"**
  - Il system clock è un timer hardware programmato per generare interrupt periodici
    - 100us – 1ms per i sistemi real-time
    - 10ms – 100ms per i sistemi di uso comune
  - Ogni volta che viene sollevato un interrupt da system clock
    - Si dice che vi è stato un "system tick" o "tick"
  - Una variabile globale mantiene il tempo corrente
    - Tale variabile viene incrementata ad ogni tick dalla ISR del system clock
- **Periodicamente il sistema operativo invoca la funzione Preempt() che**
  - Verifica se il quanto di tempo del processo corrente è esaurito
  - Nel caso sia esaurito provvede al cambiamento del contesto
    - Mediante la funzione Change()
    - In modo molto simile a quanto visto per la funzione SleepOn()

Esaurimento del quanto di tempo

- Lo pseudocodice della funzione Preempt() è simile a quello di SleepOn()

```
void Preempt() {
  if( /** Time slot exipred **/ ) {
    // Before context switch: save context
      // Save registers on SSP
      ProcessTable[CurrentProcess].Status = PS_READY;

    // Context switch
      Change();

    // After context switch: restore context and returns
      // Restores registers
  }
  return;
}
```

Struttura del kernel