Politiche a Stack per il Page Replacement

(Corso di Calcolo Parallelo)

Francesco Versaci

(versacif@dei.unipd.it)

DEI – Università degli Studi di Padova

19 marzo 2010

Università degli Studi di Padova

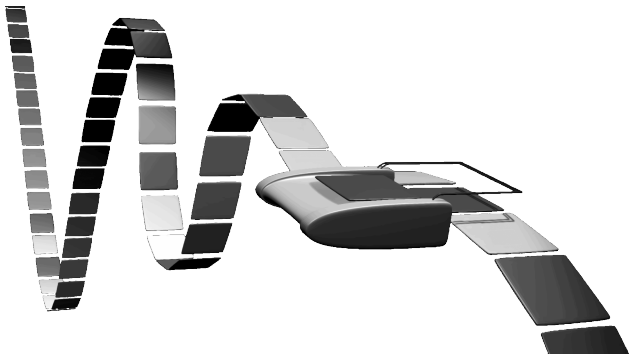
DEPARTMENT OF INFORMATION ENGINEERING

UNIVERSITY OF PADOVA

Sommario

Limiti Fisici e Calcolo

- La densità spaziale di informazione è fisicamente limitata...
- ...e quindi i dati che vengono processati da un calcolatore sono distribuiti spazialmente.
- Alcuni dati sono piú vicini al processore (bassa latenza)...
- ...e altri piú distanti (alta latenza)

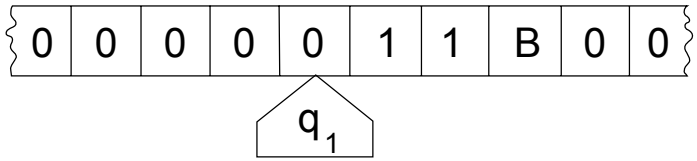


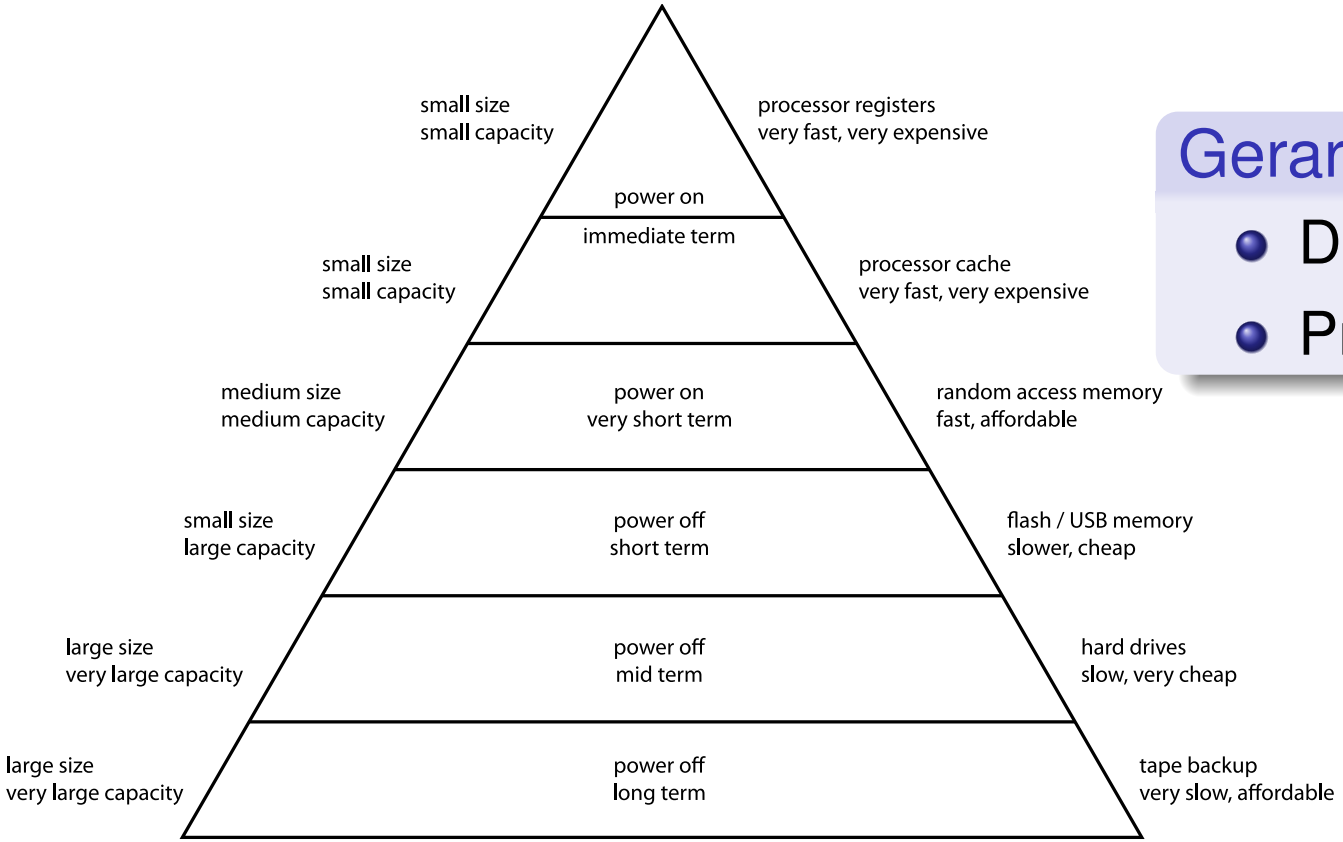
Una macchina di Turing.

Le Memorie Gerarchiche

Macchina di Turing

- Dati fermi
- Processore mobile





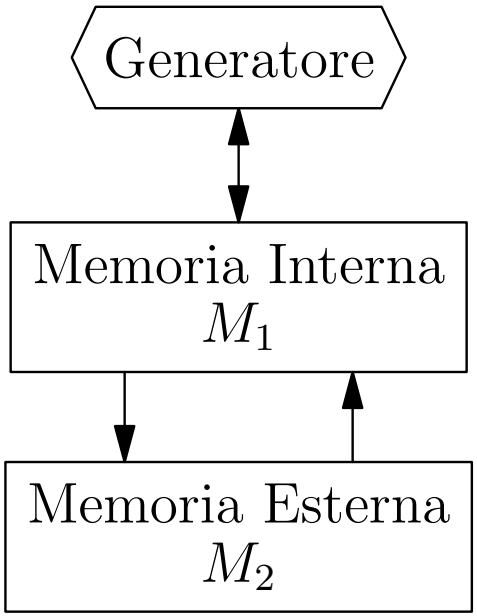
Gerarchia di Memorie

- Dati mobili
- Processore fermo

Gerarchia a Due Livelli

Caratteristiche

- Generatore (tipicamente la CPU) che produce riferimenti a dati in memoria
- Memoria di buffer M_1 (piccola e veloce)
- Memoria esterna M_2 (grande e lenta)
- I dati per essere usati devono essere nella memoria piú interna M_1
- Se M_1 è piena si deve scartare qualche dato per introdurne di nuovi



Il Tempo di Accesso

Traccia dei Riferimenti

$$X = [x_1, x_2, \ldots, x_L]$$

Frequenze d'Accesso

$$F_1 = \frac{N_1}{L} \qquad F_2 = \frac{N_2}{L}$$

Tempo Medio d'Accesso

$$\bar{T} = F_1 T_1 + F_2 T_2$$

Dove...

- $L =$ Tempo totale
- $T_1 =$ Tempo di accesso a M_1
- $T_2 =$ Tempo di accesso a M_2
- $N_1 =$ Numero di accessi a M_1 (hit)
- $N_2 =$ Numero di accessi a M_2 (miss)

Costi

- Tipicamente si ha $T_2 \gg T_1$...
- ...quindi il costo temporale è approssimato dal numero di miss...
- ...o dal miss-rate per tracce di lunghezza infinita

Notazione

Contenuto del Buffer

- $C =$ Capacità del buffer
- $t \in \{1, \ldots, L\} =$ Istante temporale
- $B_t(C) =$ Insieme di elementi nel buffer M_1 di taglia C al tempo t
- Il buffer è inizialmente vuoto:

$$B_0(C) = \emptyset$$

Aggiornamento del Buffer

Possibili Casi:

1. Se il dato referenziato è nel buffer (hit), questo non viene modificato:

$$x_t \in B_{t-1}(C) \quad \Rightarrow \quad B_t(C) = B_{t-1}(C)$$

2. Se il dato referenziato non è nel buffer e il buffer non è pieno (miss con buffer disponibile), il dato viene aggiunto al buffer:

$$x_t \notin B_{t-1}(C) \ \wedge \ |B_{t-1}(C)| < C \quad \Rightarrow \quad B_t(C) = B_{t-1}(C) \cup \{x_t\}$$

3. Se il dato referenziato non è nel buffer e il buffer è pieno (miss con buffer pieno), un elemento è scartato dal buffer (eviction) e quello nuovo viene aggiunto:

$$x_t \notin B_{t-1}(C) \wedge |B_{t-1}(C)| = C \quad \Rightarrow \quad B_t(C) = B_{t-1}(C) \setminus \{y_t\} \cup \{x_t\}$$

Scelta delle Politiche di Eviction

Off-Line

- La traccia è nota in anticipo
- Esiste una politica ottima di eviction

On-Line

- La traccia non è nota inizialmente
- Si utilizzano spesso delle euristiche (LRU, MRU, FIFO, . . .)

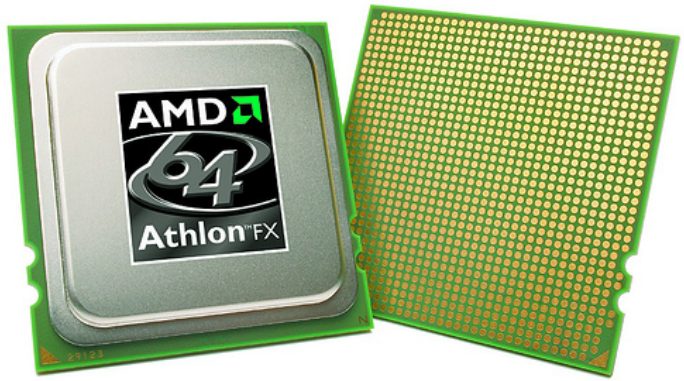
Modelli Stocastici

- La traccia è un processo stocastico conosciuto
- Si può trovare una politica ottima

Sommario

Località Temporale

- Molti algoritmi presentano località temporale...
- ...cioè tendono a riutilizzare dati usati di recente, per "sfruttare al massimo" i dati dopo averli prelevati



Politica **Least Recently Used**

Quando il buffer è pieno scarta l'elemento usato meno recentemente

Proprietà di Inclusione

Dati Referenziati

- $\Gamma_t =$ Insieme di elementi *distinti* cha hanno subito un accesso entro il t-esimo passo
- $\gamma_t = |\Gamma_t| =$ Numero di elementi referenziati

I buffer, quando viene applicata LRU come politica di eviction, soddisfano questa proprietà di inclusione:

$$B_t(1) \subset B_t(2) \subset \cdots \subset B_t(\gamma_t) = B_t(\gamma_t + 1) = \ldots$$

Dove

- $|B_t(C)| = C, \quad \gamma_t > C$
- $|B_t(C)| = \gamma_t, \quad \gamma_t \leq C$

Lo Stack LRU

Usando la proprietà di inclusione possiamo ordinare Γ_t in una lista S_t:

- $S_t = [s_t(1), s_t(2), \ldots, s_t(\gamma_t)]$
- $s_t(i) = B_t(i) \setminus B_t(i-1), \quad 1 \leq i \leq \gamma_t$

Quindi abbiamo

$$B_t(C) = \begin{cases} \{s_t(1), s_t(2), \ldots, s_t(C)\}, & \gamma_t > C \\ \{s_t(1), s_t(2), \ldots, s_t(\gamma_t)\}, & \gamma_t \leq C \end{cases}$$

Distanza di Stack

La distanza (o profondità) di stack Δ_t è la posizione dell'elemento x_t nello stack S_{t-1}

$$x_t = s_{t-1}(\Delta_t)$$

Proprietà di Δ_t

- La distanza di stack Δ_t è la minima capacità richiesta per avere uno hit al tempo t:

$$\Delta_t = \min \{C : x_t \in B_{t-1}(C)\}$$

- Usando la proprietà di inclusione sappiamo che

$$\forall C \geq \Delta_t \quad x_t \in B_{t-1}(C)$$

Funzione di Successo

Contare gli Hit

- Definiamo $n(\Delta)$ come il numero di volte che un elemento a profondità Δ viene acceduto quando si processa una certa traccia
- Il numero di hit $N(C)$ si può calcolare come:

$$N(C) = \sum_{\Delta=1}^{C} n(\Delta)$$

- La funzione di successo è definita come:

$$F(C) = \frac{N(C)}{L}$$

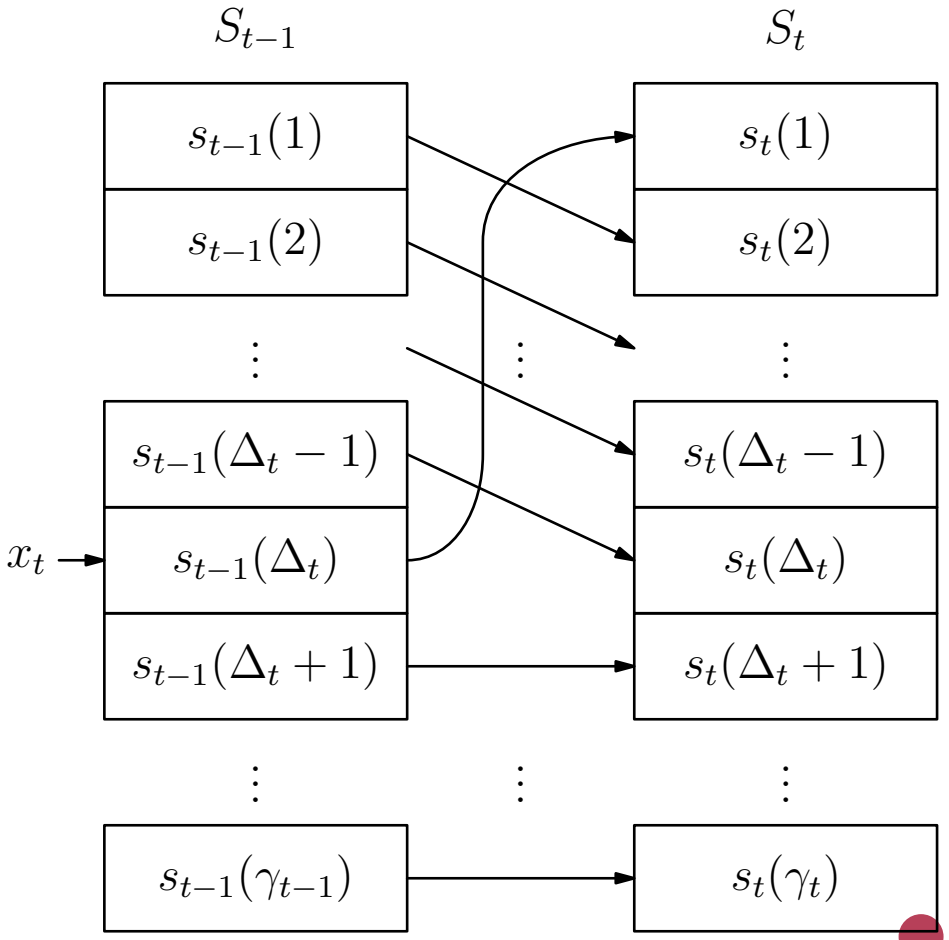
- $F(C)$ è monotona non decrescente
- $F(C) \leq \frac{L - \gamma_L}{L}$

Costruzione Iterativa dello Stack LRU – 1/2

Aggiornamento dello Stack

Se x_t è stata precedentemente referenziata, allora il nuovo stack si ottiene dal precedente tramite rotazione ciclica delle prime Δ_t posizioni, cioè:

- Gli elementi nelle prime $\Delta_t - 1$ posizioni scendono di una posizione
- x_t va in testa allo stack
- Il resto non cambia



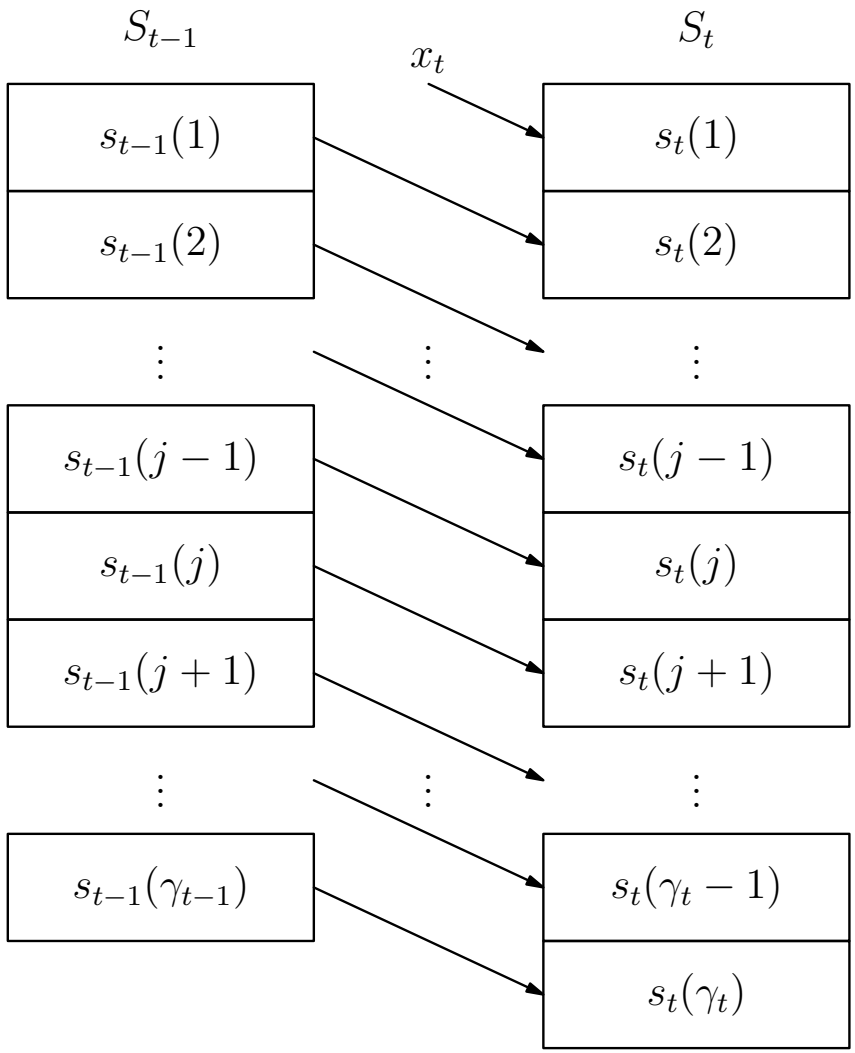
$x_t \in S_{t-1}$

Iterative Construction of LRU Stack – 2/2

Aggiornamento dello stack

Se x_t non è stato precedentemente referenziato il nuovo stack si ottiene:

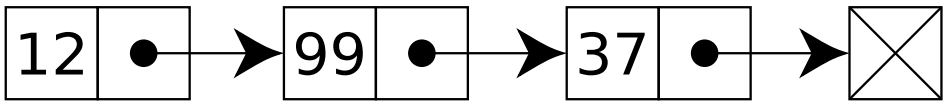
- Facendo scendere di una posizione tutti gli elementi
- Mettendo x_t in testa allo stack

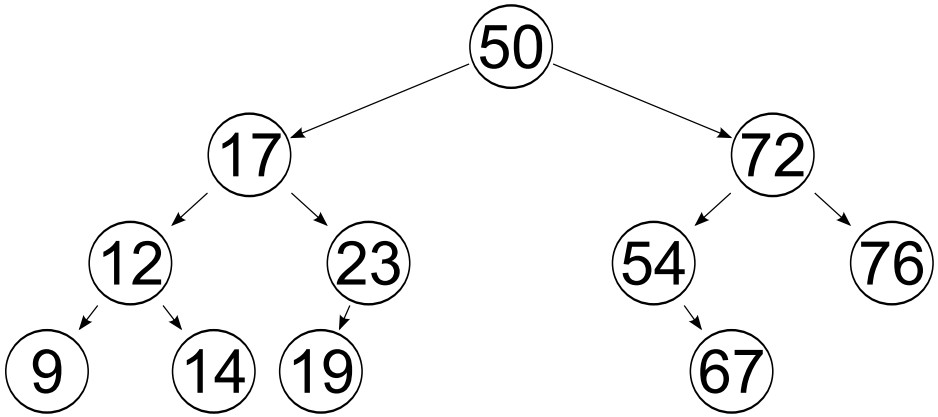


$x_t \notin S_{t-1}$

Calcolo del Miss-Rate

- Mantenendo lo stack LRU è possibile calcolare il miss-rate di una traccia per tutte le possibili taglie C in parallelo
- Memorizzando lo stack LRU come linked-list il tempo per il calcolo dei miss-rate è $O(LV)$, con V taglia della memoria esterna M_2
- Se manteniamo lo stack tramite un albero bilanciato abbassiamo il tempo di calcolo a $O(L \log V)$





Sommario

Politiche a Stack

Politiche Generali

Una generica politica di eviction è una mappa

$$R_C : \ (t, B_{t-1}(C)) \mapsto y_t(C) \qquad \text{con} \quad y_t(C) \in B_{t-1}(C)$$

Politiche a Stack

- Si chiamano politiche a stack quelle che soddisfano la proprietà di inclusione:

$$B_t(1) \subset B_t(2) \subset \cdots \subset B_t(\gamma_t) = B_t(\gamma_t + 1) = \ldots$$

- Questa condizione induce la seguente restrizione sulla mappa precedente:

$$y_t(C+1) \in \{y_t(C), s_{t-1}(C+1)\}$$

Politica a Lista di Priorità – 1/2

Lista di Priorità

- Una lista di priorità è una permutazione dei riferimenti degli elementi acceduti precedentemente:

$$P_t = [p_t(1), p_t(2), \ldots, p_t(\gamma_{t-1})]$$

- I dati vengono espulsi in base alla loro priorità: la politica di eviction sceglie l'elemento in $B_{t-1}(C)$ a priorità minore
- Le priorità sono in generale tempo-varianti

Le Politiche a Lista di Priorità sono anche a Stack

- Dato un insieme $A \subseteq \Gamma_{t-1}$ di riferimenti definiamo $\min(A)$ l'elemento con priorità minore nella lista P_t
- L'elemento scelto per l'eviction da $B_{t-1}(C)$ al tempo t è

$$y_t(C) = \min\left(B_{t-1}(C)\right)$$

- L'elemento scelto per l'eviction da $B_{t-1}(C+1)$ è

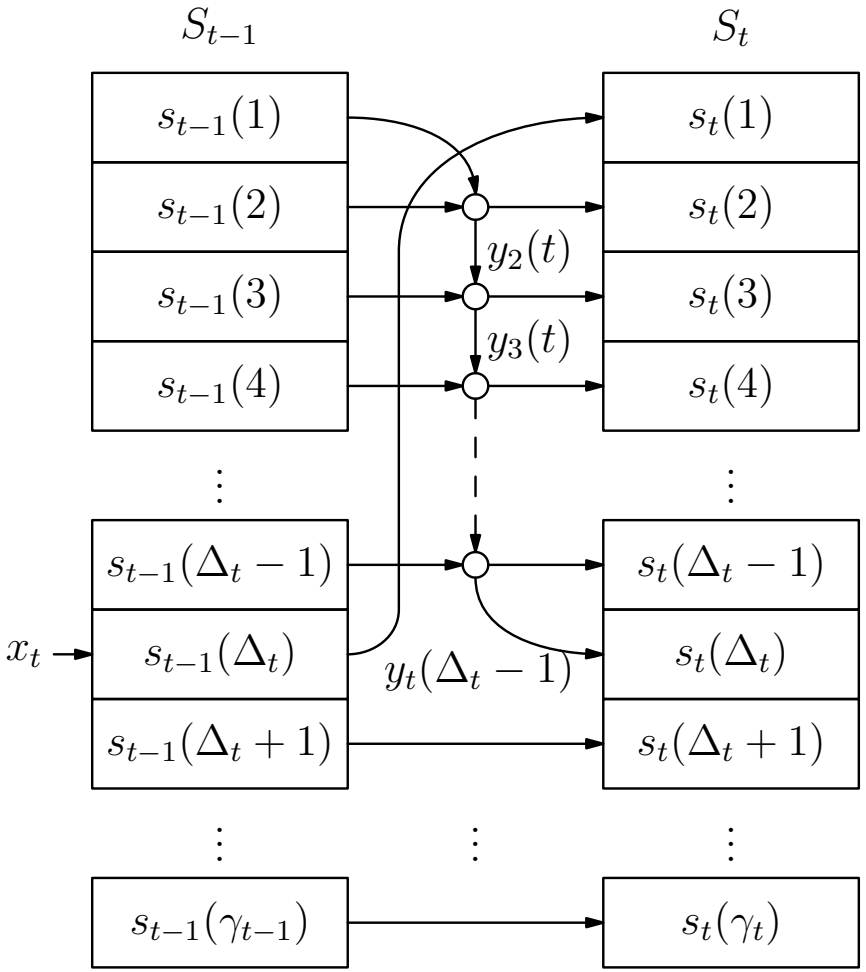
$$y_t(C+1) = \min\left(B_{t-1}(C+1)\right) = \min\left(y_t(C), s_{t-1}(C+1)\right)$$

- . . . e quindi le politiche a lista di priorità hanno la proprietà di inclusione

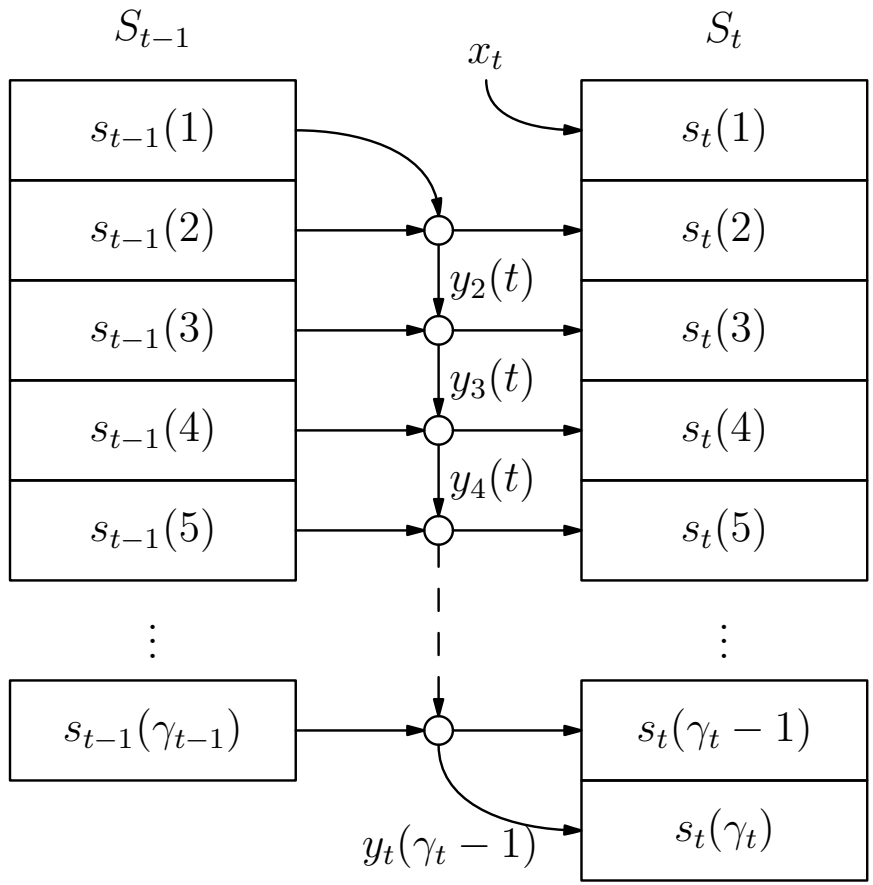
Le Politiche a Stack sono Prioritarie

È vero anche il viceversa. La prova è lasciata come esercizio.

Aggiornamento Stack



$x_t \in S_{t-1}$



$x_t \notin S_{t-1}$

Sommario

Politica OPT (Furthest First)

Definizione

- La politica OPT elimina l'elemento nel buffer che ricomparirà più tardi in futuro
- È conosciuta anche come politica di Belady, anche se in realtà è lievemente differente

Proprietà

- È una politica off-line, cioè pressupone la conoscenza del futuro
- È la politica ottima
- Viene usata quando la traccia è nota in anticipo (ad es. nella register allocation)...
- ...oppure come metro di paragone per le politiche on-line

OPT come Politica Prioritaria

- Dato un elemento x si consideri la priorità

$$d_t(x) = -\text{tempo fino al prossimo accesso a } x$$

- OPT è una politica indotta dalla priorità d_t e quindi
  - soddisfa la proprietà di inclusione
  - ha uno hit-rate monotono non-decrescente
- Tenere aggiornato il tempo del prossimo accesso può essere un po' complicato, quindi si preferisce usare un'altra priorità...

Distanza in Avanti (Forward Distance)

Definizione

- Definiamo la distanza in avanti $w_t(a)$, per un elemento a al tempo t, il numero di dati elementi distinti acceduti dal tempo $t+1$ fino al prossimo riferimento ad a:

$$w_t(a) = \#\{x_{t+1} \neq a, \ldots, x_{t+w-1} \neq a, x_{t+w} = a\}$$

- Immaginiamo che $w_t = w_t(x_t)$ sia noto per qualunque t, allora possiamo ricavare $w_t(a)$ per qualunque $a \neq x_t$:

$$w_t(a) = \begin{cases} w_{t-1}(a) - 1, & \text{per } w_{t-1}(a) \leq w_t \ \wedge \ w_{t-1}(a) \neq \infty \\ w_{t-1}(a), & \text{per } w_{t-1}(a) > w_t \ \vee \ w_{t-1}(a) = \infty \end{cases}$$

Legami fra OPT ed LRU

- Sia X^R la traccia inversa di X:

$$X^R = [x_L, x_{L-1}, \ldots, x_1]$$

- La sequenza delle profondità di stack LRU per X^R è l'inverso della sequenza di distanze in avanti di OPT per X:

$$[\Delta_L, \Delta_{L-1}, \ldots, \Delta_1] = [w_1, w_2, \ldots, w_L]$$

- Possiamo ottenere la funzione di successo per OPT tramite una doppia scansione indietro-avanti della traccia
- Possiamo anche usare una scansione avanti-indietro, visto che

$$F_{\text{LRU}}(C, X) = F_{\text{LRU}}(C, X^R) \qquad F_{\text{OPT}}(C, X) = F_{\text{OPT}}(C, X^R)$$

Bibliografia


BELADY, L. A.
A study of replacement algorithms for virtual-storage computer.
IBM Systems Journal 5, 2 (1966), 78–101.


MATTSON, R. L., GECSEI, J., SLUTZ, D. R., AND TRAIGER, I. L.
Evaluation techniques for storage hierarchies.
IBM Systems Journal 9, 2 (1970), 78–117.


ALMASI, G.S., CASCAVAL, C., PADUA, D.A.
Calculating stack distances efficiently.
MSP/ISMM, (2002), 37–43.