6.2 List Scheduling

6.12 List Scheduling Problem
Input: a list of *n* processes $P_1 \ldots P_n$ with execution times $p_j > 0$, $1 \leq j \leq n$, *m* processors $M_1, \ldots, M_m$.
Output: Assignment of the n processes to the processors: Each process needs an uninterrupted execution time of p_j on one of the *m* processors. Each processor can handle at most one process at a time.

Algorithmus 6.13 List Scheduling

```
WHILE L≠∅
  P= first(L)
  Wait until a processor M becomes free
  Assign P to processor M
END WHILE
```

Theorem 6.14:
The list scheduling algorithm 6.13 is (2-1/m)-competitive.

Proof:
Let s_j and e_j be the start and end time of process *j* in the order produced by algorithm 6.13.
Let P_k be the process that ends last, i.e., $e_k = \max\{e_1, \ldots, e_n\}$.
$\Longrightarrow$ No processor is free before s_k (otherwise P_k would have been assigned to that processor before s_k)

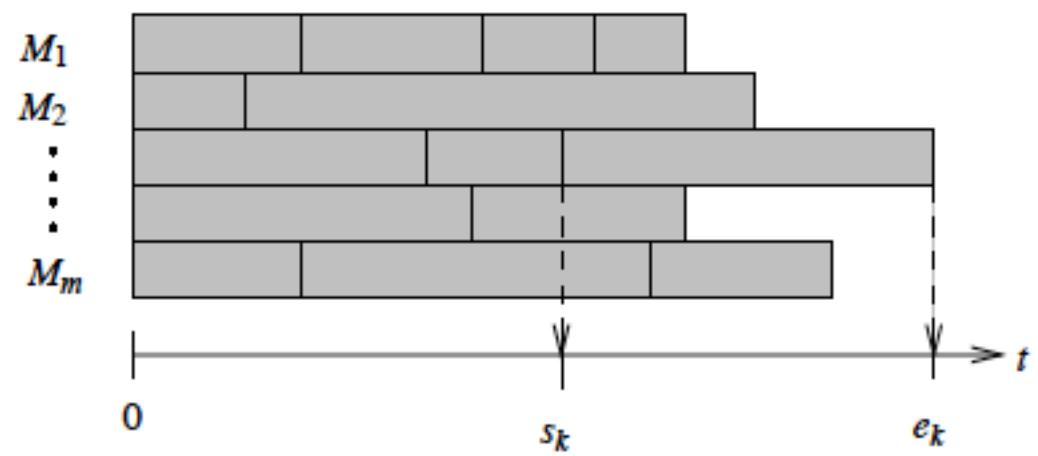

Abbildung 8.1: Die Analyse von Grahams Scheduling Algorithmus.

Let C_{LS} be the time used to process all processes by algorithm 6.13, and C_{OPT} the optimal time.

$\Longrightarrow$ 1. $C_{OPT} \geq p_k$

2. $C_{OPT} \geq \frac{1}{m} \sum_{j=1}^{n} p_j$ (lower bound for best possible situation of all processes running in parallel until the end.

$\Longrightarrow$

$$C_{LS} = e_k = s_k + p_k \leq \frac{1}{m} \sum_{j \neq k} p_j + p_k = \frac{1}{m} \sum j = 1^n p_j + (1 - \frac{1}{m}) p_k$$
$$\leq C_{OPT} + (1 - \frac{1}{m}) C_{OPT} = (2 - \frac{1}{m}) C_{OPT}$$

Algorithm 6.13 is a greedy algorithm.
Greedy does not always lead to a good result:
Consider the ski rental problem (rental fee 50€, price 500€).
If we'd know beforehand that we'll ski at most 9 times, we'll rent.
For more ski trips, we would buy skies.
Assume you rent (m-1) times, and buy for the m-th skiing trip.
$\Longrightarrow$ we pay (m-1)*50 + 500€
If we know how many skiing trips we make, we pay at most min{m*50, 500}€
The ratio has the minimum at m=10, with a competitive ratio of 1.9
$\Longrightarrow$ There is no online algorithm with a competitive ratio better than 1.9
The trivial algorithm of renting 9 times and buying for the 10th trip achieves this ratio
The greedy strategy would rent skies every time
$\Longrightarrow$ The greedy strategy would lead to an arbitrarily bad competitive ratio.

6.3 Randomized Online Algorithms

We considered deterministic online algorithms so far.
Disadvantage: for, e.g., paging the adversary can determine the page order a priori, such that the online algorithm will occur a page fault at every request.
If the algorithm can hide its inner state from the adversary, it would not be possible to create such worst-case requests.
One option to do so are randomised algorithm (access to a random number source/throwing an imaginary coin).
Then the cost of the randomised algorithm depends on the random numbers

$\Longrightarrow$ We consider the expected value of the cost to measure the algorithm

$\Longrightarrow$ A randomised algorithm is called c-competitive if the expected cost is at most c-times higher than the cost of the adversary

We need to distinguish different adversaries:

- **Oblivious adversary**: Does not know about the random decisions of the algorithm.
  - A randomised online algorithm A is c-competitive against an oblivious adversary G, if $E[C_A(\sigma)] \le c\, C_{OPT}(\sigma) + \alpha$ (expected value over all random decisions of A; the oblivious adversary must choose σ in the beginning, hence, no expected value on the right hand side)
- **Adaptive adversary**: All random decisions that the algorithms performs after requests are told to the adversary.
  - The request σ_t depends on the answers given by the online algorithm so far
  - ➡ We need another definition for competitiveness
  - ➡ $\sigma=\sigma(A,G)$ with A-online algorithm, G-adversary
  - ➡ Request order σ is a random variable
  - ➡A randomised online algorithm is c-competitive against an **adaptive online adversary G**, if $E[C_a(\sigma(A,G))] \le c\, E[C_{A'}(\sigma(A,G))] + \alpha$. For all adversaries G, where G may only use an online algorithm A' to answer $\sigma(A,G)$
  - ➡A randomised online algorithm is c-competitive against an **adaptive offline adversary G**, if $E[C_A(\sigma(A,G))] \le c\, E[C_{OPT}(\sigma(A,G))] + \alpha$. - here the adversary can wait until all of $\sigma(A,G)$ is created and then answer it with OPT.

We consider an oblivious adversary.

Algorithm 6.15 Marking
Input: a page request σ_i
Output: an evicted page
IF $\sigma_i \notin$ cache C **THEN**
IF C is not full
THEN load σ_i to C
ELSE IF all pages are marked
THEN delete all markings
Choose a random unmarked page s_j (uniformly distributed)
Delete sj and load σ_i
Mark σ_i

Algorithm 6.15 follows the general scheme for exchanging pages, the important step is choosing a random page from the unmarked pages.

Without proof: The optimal offline strategy MIN replaces the page that was not used for the longest time (also greedy).

Algorithm 6.15 Marking
Input: a page request σ_i
Output: an evicted page
IF $\sigma_i \notin$ cache C **THEN**
  IF C is not full
    THEN load σ_i to C
    ELSE IF all pages are marked
      THEN delete all markings
      Choose a random unmarked page s_j (uniformly distributed)
      Delete sj and load σ_i
Mark σ_i

Theorem 6.16 [A. Fiat, R. Karp, M. Luby, L. McGeoch, D. Sleator, N. Young: Competitive paging algorithms. Journal of Algorithms 12, 1991, 685 - 699**]:**
Algorithm 6.15 is $2H_k$-competitive against every oblivious adversary.

H_k: k-th harmonic number, $H_k=1+1/2+1/3+\ldots+1/k < 1 + \ln(k)$

Proof: We denote the cost of algorithm 6.15 on request sequence σ by $C_M(\sigma)$.
We need to show that for all request sequences σ we have: $E[C_M(\sigma)] \leq 2H_kC_{MIN}(\sigma)$

To simplify the proof, we assume: Marking and MIN start both with empty cache. (Otherwise we would need to add k on the right hand side.)

Strategy:
1. Upper bound for cost of algorithm 6.15
2. Lower bound for cost of MIN.

Algorithm 6.15 Marking
Input: a page request σ_i
Output: an evicted page
IF $\sigma_i \notin$ cache C **THEN**
  IF C is not full
    THEN load σ_i to C
    ELSE IF all pages are marked
      THEN delete all markings
      Choose a random unmarked page s_j (uniformly distributed)
      Delete sj and load σ_i
Mark σ_i

1. Upper bound for cost of algorithm 6.15:

We split σ into phases (again):

- Phase 0 starts with the first page request
- Phase *i* starts after phase *i*-1 and ends before the request of the (k-1)st page in phase i
- k pages of phase m are denoted by P_m

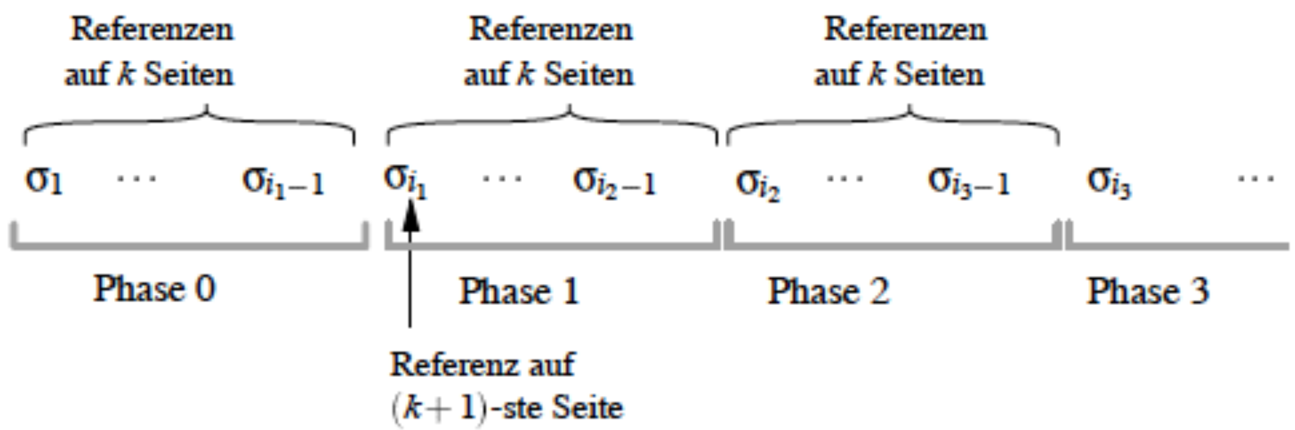

Abbildung 8.7: Die Einteilung von σ in Phasen.

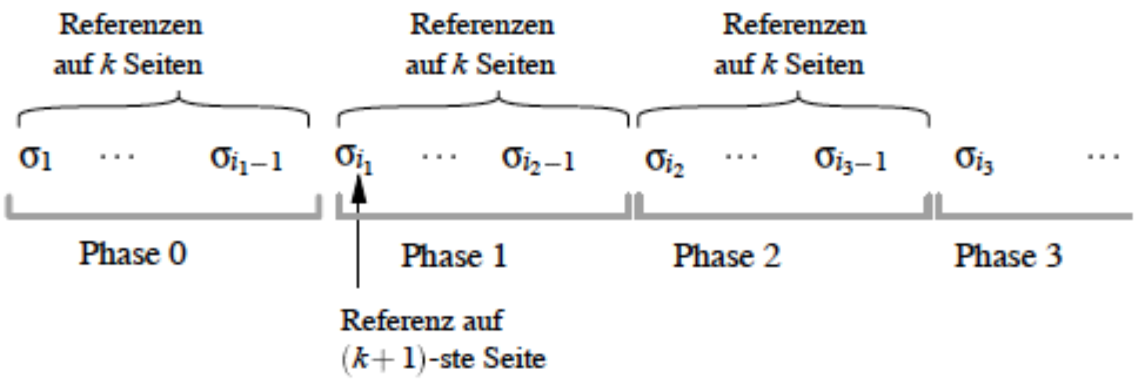

Abbildung 8.7: Die Einteilung von σ in Phasen.

Algorithm 6.15 Marking
Input: a page request σ_i
Output: an evicted page
IF $\sigma_i \notin$ cache C **THEN**
  IF C is not full
    THEN load σ_i to C
    ELSE IF all pages are marked
      THEN delete all markings
      Choose a random unmarked page s_j (uniformly distributed)
      Delete sj and load σ_i
Mark σ_i

1. Upper bound for cost of algorithm 6.15:

Observation 1: The split of σ into phases depends only on σ and not on the algorithm we consider.
Observation 2: At the end of each phase m, all pages are marked, and there are exactly the k pages requested in phase m in the cache.
Proof: by induction.
Obviously holds for phase 0, as exactly k pages are loaded into the cache.
Assume that also at the end of phase i-1 all pages are marked, and exactly the pages requested in phase i-1 are in the cache.
$\Longrightarrow$ In step 5 of algorithm 6.15 all markings are deleted by requesting page (k-1)
In phase i one after another k pages are marked
$\Longrightarrow$ Shortly before the end of the phase all pages are marked again, and exactly those pages requested in phase i are in the cache.

Observation 3: At most the first page request for a page in a phase results in a page fault.
(After the first request the page does not get deleted in that phase.)

$\Longrightarrow$ We can restrict to consider one phase m for algorithm 6.15.

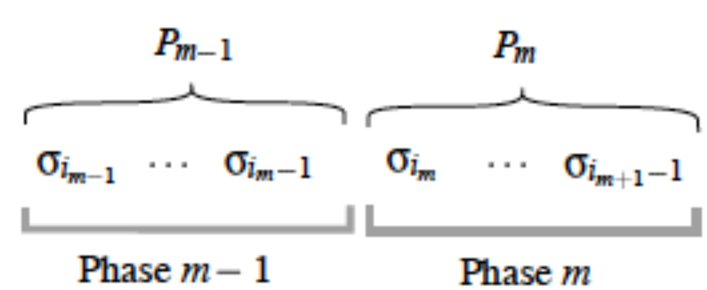

Abbildung 8.8: Phasen $m-1$ und m von Marking.

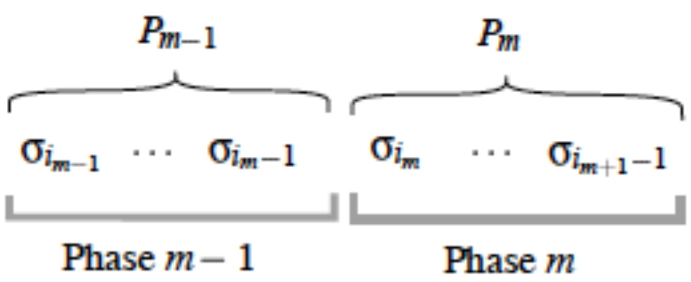

Abbildung 8.8: Phasen $m-1$ und m von Marking.

Algorithm 6.15 Marking
Input: a page request σ_i
Output: an evicted page
IF $\sigma_i \notin$ cache C **THEN**
  IF C is not full
    THEN load σ_i to C
    ELSE IF all pages are marked
      THEN delete all markings
      Choose a random unmarked page s_j (uniformly distributed)
      Delete sj and load σ_i
Mark σ_i

1. Upper bound for cost of algorithm 6.15:

We consider the k different pages $s_1 \ldots s_k$ requested in phase m.

Observation 3 $\Longrightarrow$ each of these pages results in a page fault at most at the first request in phase m

$\Longrightarrow$We only need to consider the page requests that request a page for the first time

Let σ_t be a page reference in P_m that requests a page from $s_1 \ldots s_k$ for the first time.

We distinguish two categories of requests:

1. σ_t is an *old request*, if σ_t was requested also in P_{m-1}
2. σ_t is a *fresh request,* if it was not requested in P_{m-1}

Obviously, all fresh requests result in a page fault.

$\Longrightarrow$ If σ_t is a fresh request: $E[C_M(\sigma_t)]=1$

(Holds for all marking page exchange algorithms, as the cache is filled with old pages at the end of each phase.)

$\Longrightarrow$ Only interested in the expected cost of an old request

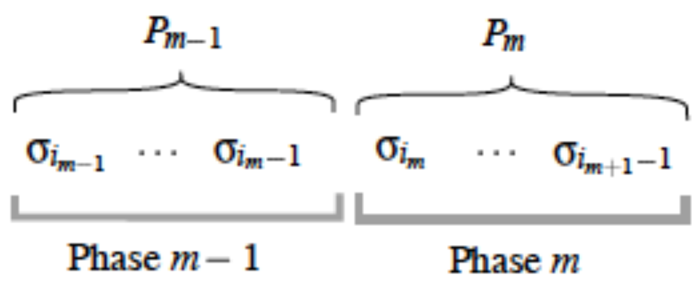

Abbildung 8.8: Phasen $m-1$ und m von Marking.

Algorithm 6.15 Marking
Input: a page request σ_i
Output: an evicted page
IF $\sigma_i \notin$ cache C **THEN**
  IF C is not full
    THEN load σ_i to C
    ELSE IF all pages are marked
      THEN delete all markings
      Choose a random unmarked page s_j (uniformly distributed)
      Delete sj and load σ_i
Mark σ_i

1. Upper bound for cost of algorithm 6.15:

expected cost of an old request

Let σ_t be an old reference, and assume before σ_t there were *f* fresh and *v* old requests, S_t the cache state at time t

$\Longrightarrow E[C_M(\sigma_t)] = 0 * Pr(\sigma_t \in S_t) + 1 * Pr(\sigma_t \notin S_t)$
$= Pr(\sigma_t \notin S_t)$
$= 1 - Pr(\sigma_t \in S_t)$

$\Longrightarrow$ We need to determine the probability that σ_t was in the cache at time t

Page σ_t was in the cache at the start of phase m, as it is an old request

$Pr(\sigma_t \in S_t)$ is the ratio between the number of cache states that contain σ_t, and the number of all possible cache states:

$Pr(\sigma_t \in S_t) = \#(S_t \text{ with } \sigma_t \in S_t) / \#(S_t)$

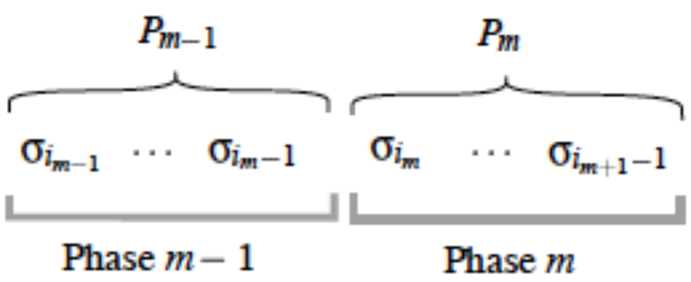

Abbildung 8.8: Phasen $m-1$ und m von Marking.

Algorithm 6.15 Marking
Input: a page request σ_i
Output: an evicted page
IF $\sigma_i \notin$ cache C **THEN**
  IF C is not full
    THEN load σ_i to C
    ELSE IF all pages are marked
      THEN delete all markings
      Choose a random unmarked page s_j (uniformly distributed)
      Delete sj and load σ_i
Mark σ_i

1. Upper bound for cost of algorithm 6.15:

We consider the following figure to determine the number of possible cache states:

f+v pages were requested and marked before t in phase m

$\Longrightarrow$ k-(f+v) free for storing pages

In those spaces we can have all k pages that were in the cache at the start of phase m, except for the v already requested pages.

$\Longrightarrow$ There are as many cache states S_t as there exist possibilities to distribute the not yet *k-v* referenced pages from phase m-1 to the *k-(f+v)*

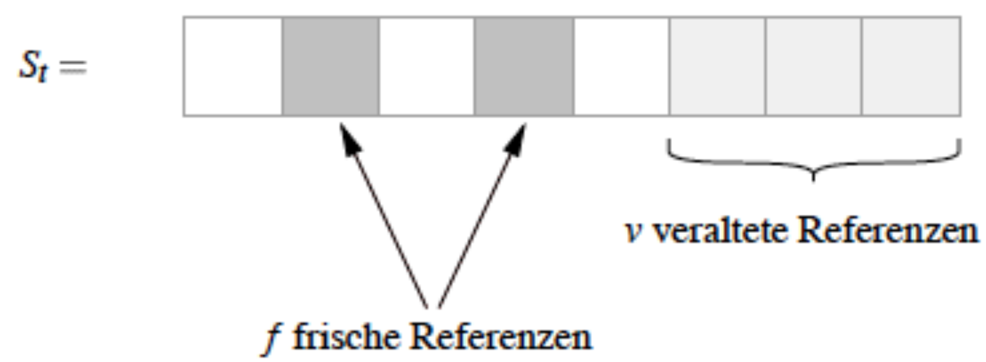

Abbildung 8.9: Die Belegung des Speichers zum Zeitpunkt *t*.

$\Longrightarrow$

$$\#(S_t) = \binom{k-v}{k-f-v}$$

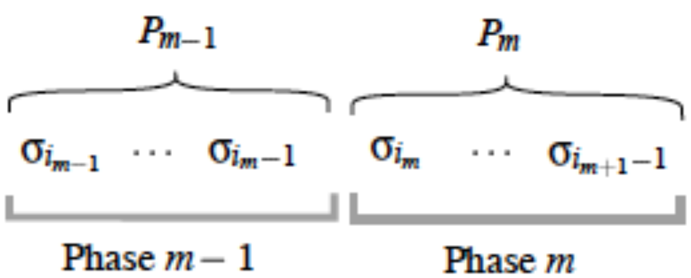

Abbildung 8.8: Phasen $m-1$ und m von Marking.

Algorithm 6.15 Marking
Input: a page request σ_i
Output: an evicted page
IF $\sigma_i \notin$ cache C **THEN**
  IF C is not full
    THEN load σ_i to C
    ELSE IF all pages are marked
      THEN delete all markings
      Choose a random unmarked page s_j (uniformly distributed)
      Delete sj and load σ_i
Mark σ_i

1. Upper bound for cost of algorithm 6.15:

We consider the following figure to determine the number of possible cache states for which σ_t is in S_t:
σ_t can be considered as an old request, and we obtain

$$\#(S_t \text{ mit } \sigma_t \in S_t) = \binom{k-v-1}{k-f-v-1}$$

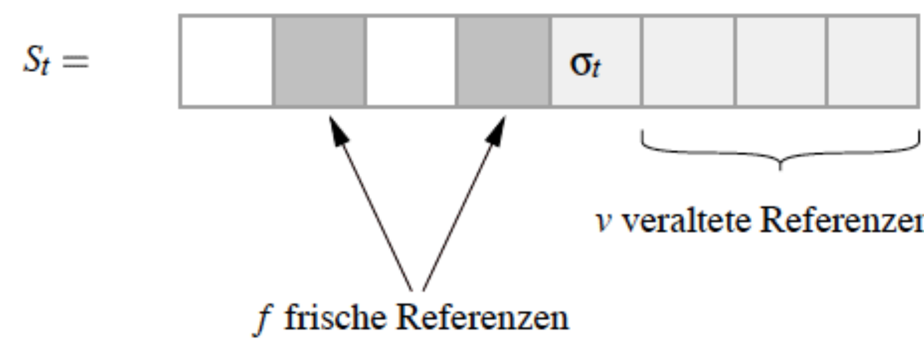

Abbildung 8.10: Die Speicherzustände, falls σ_t in S_t enthalten ist

$$\begin{aligned} E[C_M(\sigma_t)] &= 1 - \frac{\#(S_t \text{ mit } \sigma_t \in S_t)}{\#(S_t)} \\ &= 1 - \frac{\binom{k-v-1}{k-f-v-1}}{\binom{k-v}{k-f-v}} \\ &= 1 - \frac{(k-v-1)!}{(k-f-v-1)!\,f!} \cdot \frac{(k-f-v)!\,f!}{(k-v)!} \\ &= 1 - \frac{k-f-v}{k-v} \\ &= \frac{f}{k-v}. \end{aligned}$$

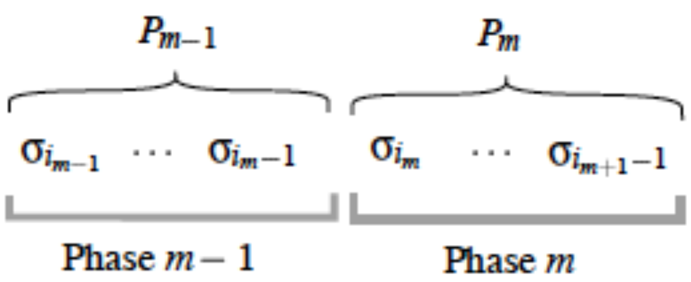

Abbildung 8.8: Phasen $m-1$ und m von Marking.

Algorithm 6.15 Marking
Input: a page request σ_i
Output: an evicted page
IF $\sigma_i \notin$ cache C **THEN**
  IF C is not full
    THEN load σ_i to C
    ELSE IF all pages are marked
      THEN delete all markings
      Choose a random unmarked page s_j (uniformly distributed)
      Delete sj and load σ_i
Mark σ_i

1. Upper bound for cost of algorithm 6.15:

⟹The expected cost for an old request σ_t is higher for more fresh references before σ_t.

Let f_i be the number of fresh requests in phase i

⟹Expected cost for the k-f_i old requests in phase i:

$$V_i = \frac{f_i}{k} + \frac{f_i}{k-1} + \cdots + \frac{f_i}{k-(k-f_i-1)}.$$

⟹ total cost for algorithm 6.15 in phase i for f_i fresh and k-f_i old requests:

$$f_i + V_i = f_i\left(1 + \frac{1}{f_i+1} + \cdots + \frac{1}{k}\right) \leq f_i H_k.$$

⟹ Summing over all phases of σ:

$$E[C_M(\sigma)] \leq H_k \sum_{i=1}^{n} f_i.$$

Proof:

2. Lower bound for cost of MIN.

Let Δ_i be the number of pages at the end of phase i-1 that are in the cache of MIN, but not in the cache of alg. 6.15.

We consider MIN at the begin of phase i, i.e., before the first page request:

Assume, the number f_i of fresh requests in phase i is larger than Δ_i.

As the fresh requests were not in the cache of alg. 6.15 at the begin of phase i, they are not part of the k-Δ_i pages of MIN.

$\Longrightarrow$ Each of the additional fresh requests results in a page fault

$\Longrightarrow$ $C_{MIN}(\text{Phase i}) \geq f_i - \Delta_i$

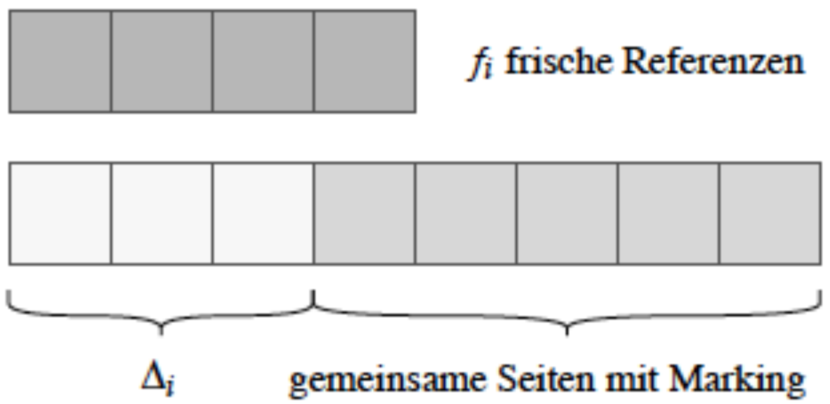

Abbildung 8.11: Die Situation von MIN zu Beginn der Phase i.

Proof:

2. Lower bound for cost of MIN.

Consider MIN at the end of phase i:

In phase i k different pages are requested, all of which are in the Cache of alg. 6.15 by observation 2.

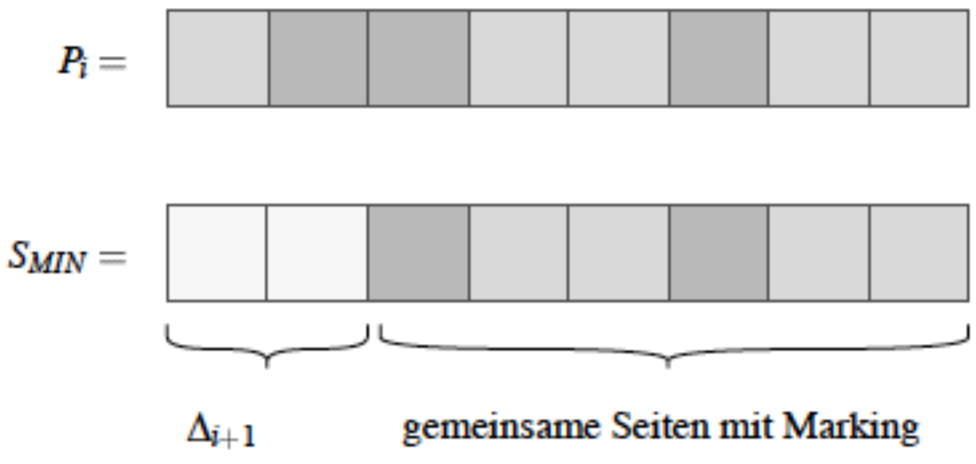

Abbildung 8.12: Die Situation von MIN am Ende der Phase *i*.

$\Longrightarrow$ Number of different pages in MIN's cache at some time during phase i is at least $k+\Delta_{i+1}$

$\Longrightarrow$ As at most k pages can be the cache at one time:

$C_{MIN}(\text{Phase } i) \geq \Delta_{i+1}$

$\Longrightarrow$. $C_{MIN}(\text{Phase } i) \geq \max(f_i-\Delta_i, \Delta_{i+1}) \geq 1/2\ (f_i-\Delta_i + \Delta_{i+1})$

$\Longrightarrow$ Summing over all phases (most Δ_i cancel out, and $\Delta_1=0$, $\Delta_{n+1}>0$):

$$\begin{aligned} C_{MIN}(\sigma) &\geq \frac{1}{2}(f_1 - \Delta_1 + \Delta_2 + f_2 - \Delta_2 + \Delta_3 + \cdots + \Delta_{n+1}) \\ &\geq \frac{1}{2}\left(\sum_{i=1}^{n} f_i - \Delta_1 + \Delta_{n+1}\right) \\ &\geq \frac{1}{2}\sum_{i=1}^{n} f_i, \end{aligned}$$

$$\Longrightarrow \quad E[C_M(\sigma)] \leq H_k \sum_{i=1}^{n} f_i = 2H_k \left(\frac{1}{2}\sum_{i=1}^{n} f_i\right) \leq 2H_k C_{MIN}(\sigma).$$

Algorithm 6.15 Marking
Input: a page request σ_i
Output: an evicted page
IF $\sigma_i \notin$ cache C **THEN**
    IF C is not full
        THEN load σ_i to C
        ELSE IF all pages are marked
            THEN delete all markings
            Choose a random unmarked page s_j (uniformly distributed)
            Delete sj and load σ_i
Mark σ_i

Theorem 6.17:
Let A be a randomised paging algorithm. There exists an arbitrary long page sequence σ such that $C_A(\sigma) \geq H_k C_{MIN}(\sigma)$.

That is, apart for the factor of two, algorithm 6.15 is optimal.

6.3 Online Search