Paging

Ground Rules:

N Pages of memory $1, 2, \cdots, N$
k Pages can be in fast memory (or cache) $k < N$
initially Pages $1, 2, 3 \cdots k$ in Cache

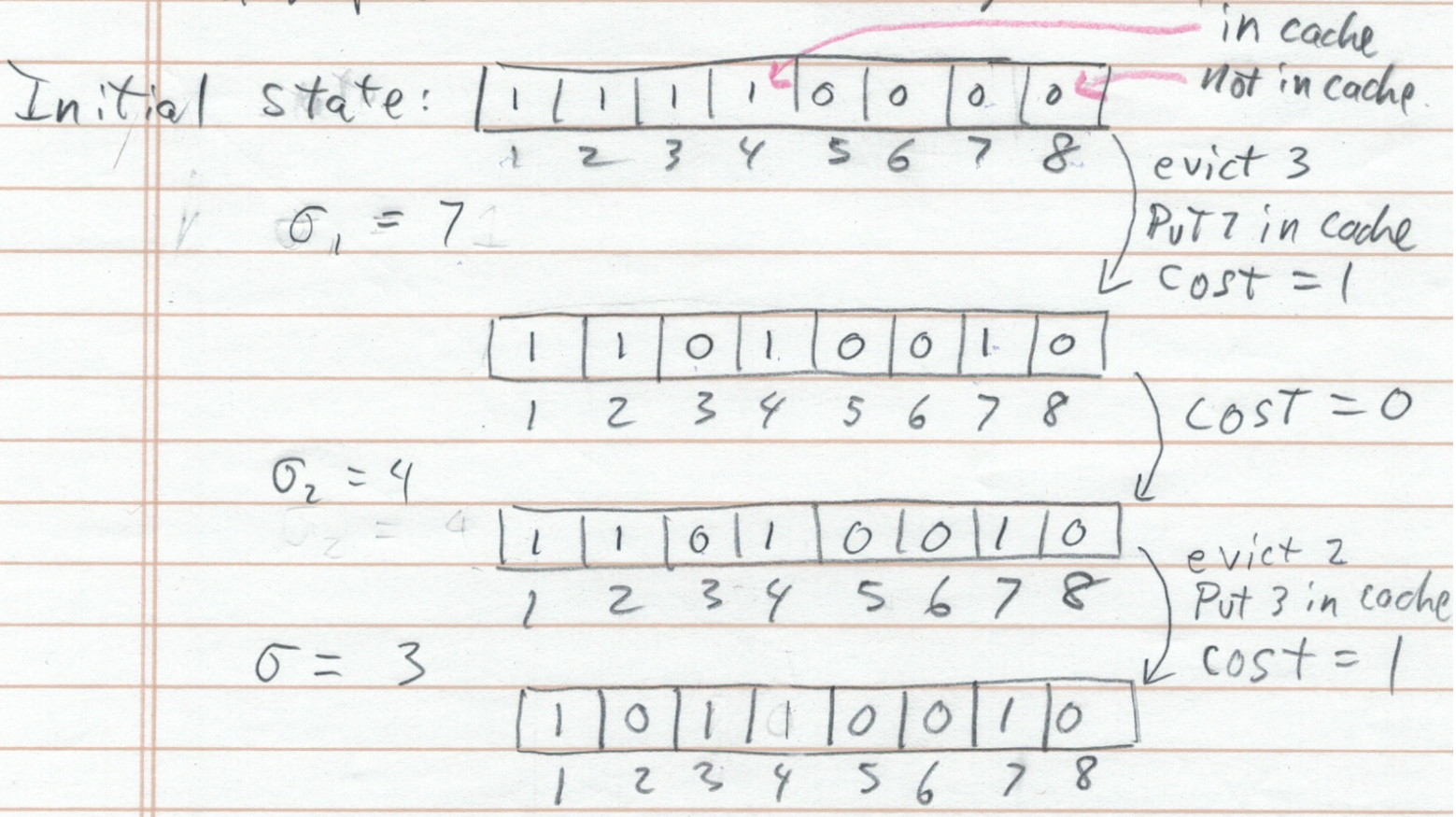
σ = Sequence of requests for pages.

if σ_i is in cache then cost = 0

if σ_i is not, then a page in cache is evicted and σ_i replaces it in cache.
cost = 1. called a page fault

Our goal: a competitive on-line algorithm (with small competitive factor) for the paging problem.

Example with $N = 8, K = 4$

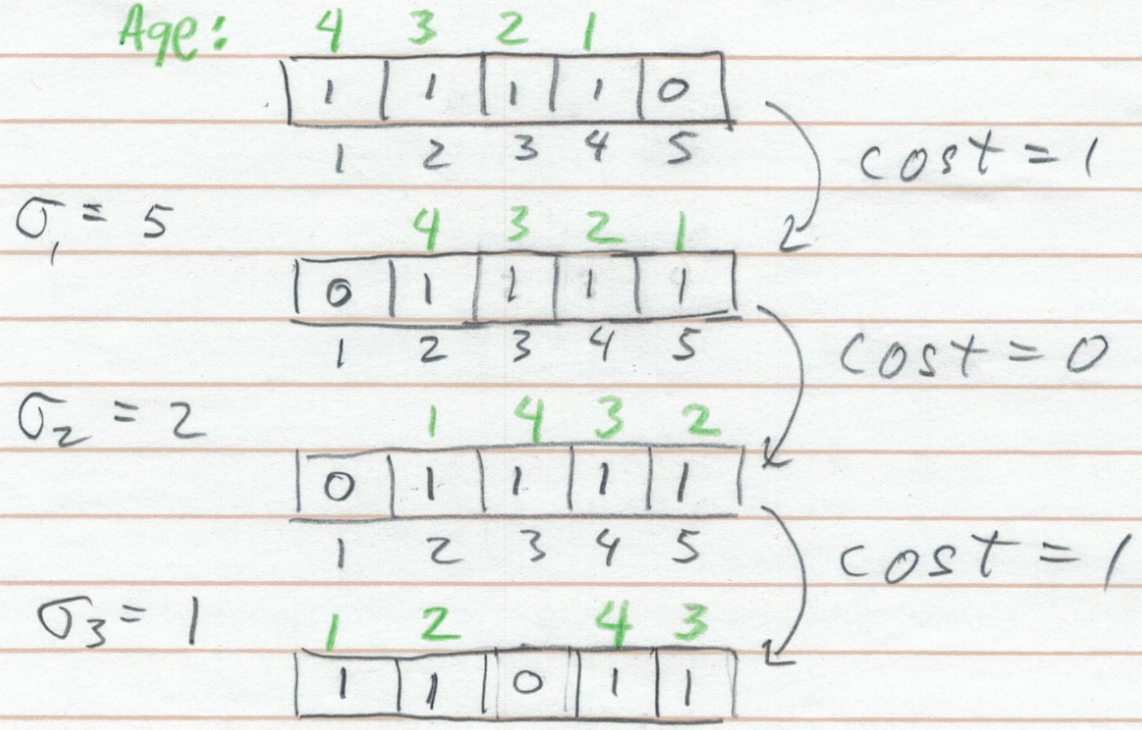


Classic Deterministic Algorithm for Paging: LRU

LRU: Evict the Least Recently Used Page

(Initially 1 is oldest, 2 is second, etc)

Example of LRU with $N=5$, $K=4$



The optimal offline algorithm:
Longest Forward Distance (LFD)

It is: On a page fault, evict the page that is next used later than all others in the cache.

Theorem: LFD is optimal.

No proof given today, but it's not that hard.

What about our goal? A competitive on-line algorithm.

Unfortunately....

theorem: No on-line deterministic algorithm can have a competitive factor $< k$

Proof: We'll prove it for $N = k+1$ (but proof holds for $N > k+1$ too.)

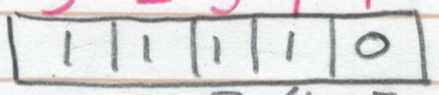
Let A be any on-line algorithm for the paging problem.

let σ = the sequence that causes A to page fault every time. Using only pages $1 \cdots N$.

We will compare this with how LFD performs.

Take this example for $k=4, N=5$

future time: 3 2 5 4 1



As LFD's state

The future times of cache pages are all ≥ 2 and all different.

$\Rightarrow$ there exists one with a future time $\geq$ current time $+k$

e.g. 5 in this case.

So $\sigma_1 = 5$ and LFD evicts page 3. Now the next 3 (i.e. $k-1$) accesses are free for LFD.

Summarize

$$\sigma = \sigma_1, \sigma_2 \sigma_3 \cdots \sigma_k \sigma_{k+1}$$

σ_1: Page fault for LFD

$\sigma_2 \cdots \sigma_k$: No LFD page faults

σ_{k+1}: Possible next LFD page fault.

$\Rightarrow$ for every k accesses LFD has ≤ 1 fault while A has k. ∎

Randomization to the Rescue!

Marking Algorithm:

- Initial state: $\{1, \dots, k\}$ in cache
  pages $1 \dots k$ marked

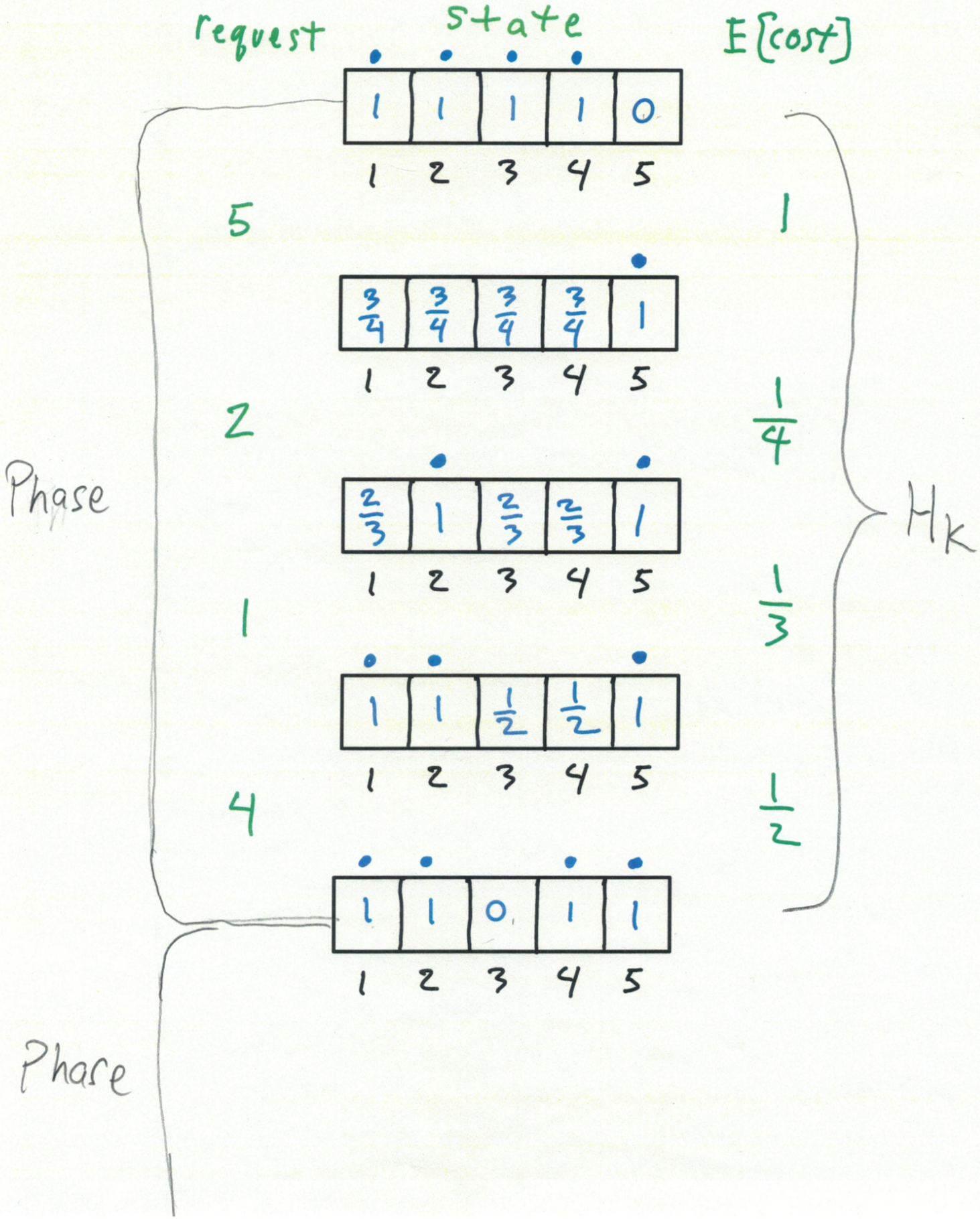
- When a page is requested:

  If in cache, mark it.
  else:

  ① If all pages in cache are marked then unmark all

  ② Pick a random unmarked page in cache and evict. Put the request in cache, mark it.

[Example from lecture notes]


request
state
E[cost]
1 1 1 1 0
1 2 3 4 5
5
1
3/4 3/4 3/4 3/4 1
1 2 3 4 5
2
1/4
Phase
2/3 1 2/3 2/3 1
1 2 3 4 5
1
1/3
1 1 1/2 1/2 1
1 2 3 4 5
4
1/2
1 1 0 1 1
1 2 3 4 5
Hk
Phase

Properties of Marking:

- The marked pages are exactly those with probability 1
- The state at the start of a phase is deterministic (Probs are all 0 or 1)
- During a phase k distinct pages* are requested, including the one not in the cache at the start of phase
- The state at the start and end of a phase differ.

* Ignoring requests to pages that are probability 1.

Theorem:

$$\forall \sigma \quad \frac{E[\text{Marking}(\sigma)]}{OPT(\sigma)} \leq \begin{cases} H_k & \text{if } N = k+1 \\ 2H_k & \text{if } N > k+1 \end{cases}$$

$$\left(H_k = 1 + \frac{1}{2} + \cdots + \frac{1}{k} \leq 1 + \ln k \right)$$

Proof (Just $N = k+1$ case):

The expected cost of a phase for Marking is H_k

Note: we can ignore all requests to marked pages.

(This does not affect Marking cost, and may improve OPT.)

$\Rightarrow$ WLOG the first request in a phase is to the unmarked page.

Lemma: If m phases begin, then the cost to OPT is at least m.

Proof of Lemma:

If OPT and Marking start and end a phase in the same state, then cost to OPT ≥ 1.

Because: the first request costs OPT 1.

In general:

	start phase	end phase	cost to OPT $\geq$
✓	=	=	1
✓	=	$\neq$	2
✓	$\neq$	=	0
	$\neq$	$\neq$	1

Verify all cases.

the result follows from this table.