

Online algorithms

Lecture 1

Probability

Events, probabilities,
conditional probabilities,
union bound

Random variables,
expected value,
linearity of expectation
Markov's inequality

Algorithms

Approx. 3-SAT,

Big idea: we only want the average

Linearity of Expectation
 $E[X+Y] = E[X] + E[Y]$ (stochastically additive)
"average sum = sum of averages"

let $Y_i = \begin{cases} 1 & \text{if } i\text{th clause satisfied} \\ 0 & \text{otherwise} \end{cases}$ ($i=1, \dots, m$)

$E[Y_i] = 7/8$ (1 clause case)

$E[\text{\# clauses satisfied}] = E[\sum_{i=1}^m Y_i] = \sum_{i=1}^m E[Y_i] = 7/8 m$

let $Y_i = \begin{cases} 1 & \text{if } i\text{th clause satisfied} \\ 0 & \text{otherwise} \end{cases}$

$Z = \sum_{i=1}^m Y_i = \text{\# clauses satisfied}$

$E[Z] = \frac{1}{2} E[Z | x_1 = \text{true}] + \frac{1}{2} E[Z | x_1 = \text{false}]$

$\Rightarrow$ either $E[Z | x_1 = \text{true}]$ or $E[Z | x_1 = \text{false}]$ is $\geq E[Z]$!

QuickSort,

Random divide and conquer

① pick random 'pivot'

② Divide A into two parts



Expected running time of QuickSort

For each $i, j \in [n]$ w/ $i < j$,

$X_{ij} = \begin{cases} 1 & \text{if } i\text{th smallest element is compared to } j\text{th smallest} \\ 0 & \text{otherwise} \end{cases}$

$\sum_{i < j} X_{ij}$ = total # comparisons made by the algorithm
 $\approx$ total running time

$E[\sum_{i < j} X_{ij}]$ = average running time

$E[\sum_{i < j} X_{ij}] = \sum_{i < j} E[X_{ij}]$ (linearity of expectation)

now analyze $E[X_{ij}]$ in isolation

$E[X_{ij}] = \Pr[X_{ij} = 1]$
 $= \Pr[i\text{th element is compared to } j\text{th}]$

i th and j th elem selected before any element in between happens w/ probability $\frac{2}{j-i+1}$

$E[X_{ij}] = \frac{2}{j-i+1}$

$E[\sum_{i < j} X_{ij}] = \sum_{i < j} \frac{2}{j-i+1}$ (by linearity of expectation)

$\leq 2n \sum_{k=1}^n \frac{1}{k} \approx 2n \ln n$
 $\leq O(n \log n)$ with harmonic series

Quickselect (HW)

Lecture 2

Probability

Hash Functions,

Universal Hash Functions

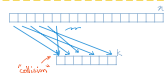
$h: [n] \rightarrow k$ s.t. for all pairs $i, j \in [n]$,
 $\Pr[h(i) = h(j)] \leq \frac{1}{k}$ (much weaker than ideal hash)

e.g. $h(x) = (ax + b) \bmod p \bmod k$

where • p prime, $> k$

• a in $\{1, \dots, p-1\}$ unif. random

• b in $\{0, \dots, p-1\}$ unif. random



Markov's Inequality

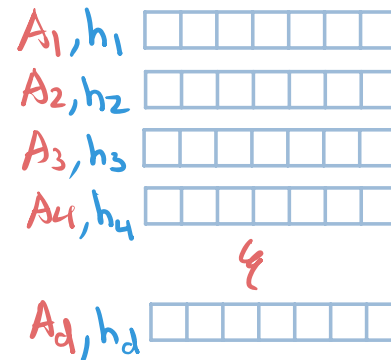
Markov's Inequality (special case)

Let $X \geq 0$ be nonnegative RV

$$\Pr[X \geq 2E[X]] \leq \frac{1}{2}$$

Algorithms

Frequency Estimation, Heavy Hitters (in streams)



estimate f_i w/
 $\min_{j=1, \dots, d} A_j[h_j(i)]$

Lecture 3

Probability

k-wise independence,
higher order moments
(generalizing universal hash
functions and Markov's inequality)

Algorithms

Hash tables w/
chaining +
linear probing

k-wise independent hash functions

$$h: [U] \rightarrow [m]$$

s.t. for any k keys $x_1, x_2, \dots, x_k \in [U]$,
 k values $y_1, y_2, \dots, y_k \in [m]$,

$$P[h(x_1)=y_1, h(x_2)=y_2, \dots, h(x_k)=y_k] = \frac{1}{m^k}$$

eg. $h(x) = a_0 + a_1x + a_2x^2 + \dots + a_{k-1}x^{k-1} \pmod{p}$

where $p \geq U$ prime,
 $a_0, a_1, \dots, a_{k-1} \in [p]$ indep, uniformly random

Important lemma

let $X_1, \dots, X_n \in \{0, 1\}$ be 4-wise independent
and $\mu = E[X_1 + \dots + X_n]$. then for $\beta > 0$,

$$P[X_1 + \dots + X_n \geq \mu + \beta] \leq \frac{4 \max\{\mu, \mu^3\}}{\beta^4}$$

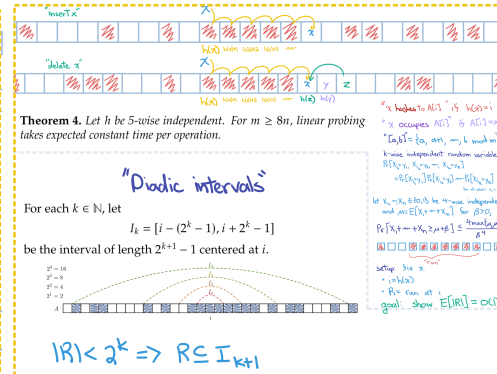
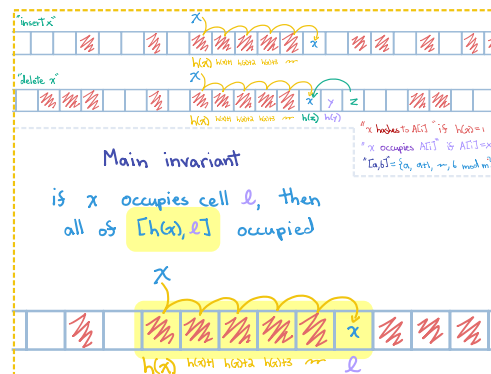
k-wise independent random variables
 $P[X_1=y_1, X_2=y_2, \dots, X_k=y_k]$

$$= P[X_1=y_1] P[X_2=y_2] \dots P[X_k=y_k]$$

for all values $y_1, \dots, y_k$

Main invariant

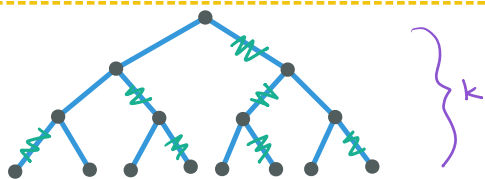
if x occupies cell i , then
all of $[h(x), i]$ occupied



Lecture 4

Probability

Galton-Watson processes



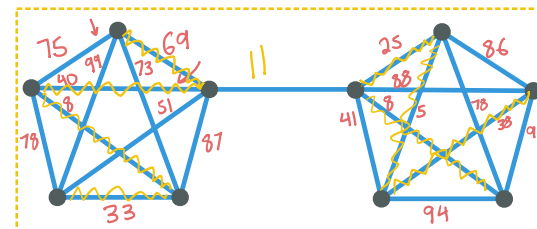
let T = complete binary tree of height k ,
w/ every edge deleted w/ prob $\leq 1/2$

Fact: $P[\text{some leaf connected to root}] \geq \frac{1}{k}$

"Galton-Watson process"

Algorithms

Random Contractions for min cut



Randomized contractions

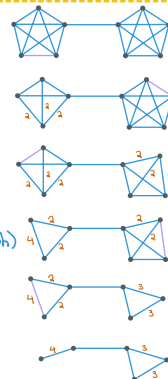
Input: $G=(V,E)$, weights $c \in \mathbb{R}_{>0}^E$
(unweighted)

① while $|E| > 1$

 Ⓐ sample $e \in E$ in proportion to c

 Ⓑ "contract" e

② return set of edges (in original graph)
 contracted to remaining edge



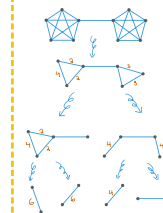
Amplification by branching

Instead of restarting at the beginning, restart partway through

High-level algorithm

① randomly contract until prob.
of ruining the min-cut is $\sim 1/2$

② Branch:
make two copies of current graph
recurse on both copies



Lecture 5

Probability

Multiplicative Chernoff Bound

Multiplicative Chernoff bound

let $X_1, \dots, X_n \in [0, 1]$ be independent

$$\mu = E[X_1 + \dots + X_n]$$

$$\epsilon \in (0, 1)$$

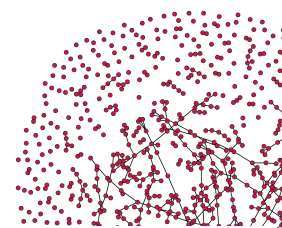
- $P[X_1 + \dots + X_n \geq (1 + \epsilon)\mu] \leq e^{-\epsilon^2 \mu / 3}$
- $P[X_1 + \dots + X_n \leq (1 - \epsilon)\mu] \leq e^{-\epsilon^2 \mu / 2}$

Algorithms (/ results)

Random graphs & the giant component

Erdős-Rényi graphs $G(n, p)$

- n vertices
- each edge sampled independently w/ probability p



Theorem 5.3. Consider a random graph $G \sim \mathcal{G}(n, p)$ for $p = c/n$, where c is a constant.

- If $c > 1$, then with high probability, there is exactly one connected component of G with $\Omega(n)$ vertices, and all other components have size $\leq O(\log n)$.
- For $c < 1$, then with high probability, all connected components of G has size $< O(\log n)$.

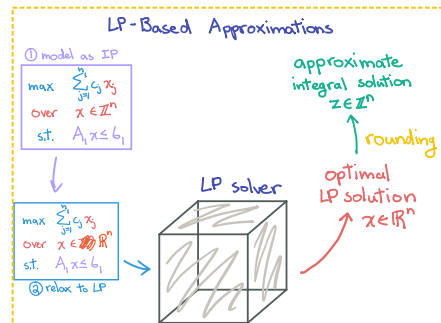
Lecture 6

Probability (& techniques)

Approximations via linear programming & randomized rounding

Linear programs (LPs)
Similar to IPs, except no integrality constraints

min/max $\sum_{j=1}^n c_j x_j$ linear objective
over $x \in \mathbb{R}^n$ linear integral variables
st. $A_1 x \leq b_1$ linear constraints
 $A_2 x = b_2$
 $A_3 x \geq b_3$



Continuous optimization vs Discrete optimization

randomization

Algorithms

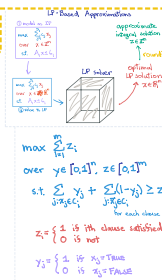
Max Sat Set Cover Covering Integer Programs

For each variable x_j , independently, randomly set $x_j = 1$ with probability p_j , and $x_j = 0$ otherwise.

$$E[\# \text{ sat. clauses}] \geq (1 - 1/e) \text{OPT}$$

k	oblivious	LP	average
1	1/2	$z_i \geq (3/4)z_i$	$\geq (3/4)z_i$
2	3/4	$(3/4)z_i \geq (3/4)z_i$	$\geq (3/4)z_i$
3	7/8	$(1 - 1/e)z_i \geq (3/4)z_i$	$\geq (3/4)z_i$
4	15/16	$(1 - 1/e)z_i \geq (3/4)z_i$	$\geq (3/4)z_i$
5	31/32	$(1 - 1/e)z_i \geq (3/4)z_i$	$\geq (3/4)z_i$
...	...	...	...
k	$1 - 1/2^k$	$(1 - 1/e)z_i \geq (3/4)z_i$	$\geq (3/4)z_i$

Can we get $\frac{3}{4}$ -APX?



Rounding LP solution
 $x \in [0, 1]^n \rightsquigarrow F \subseteq \{S_1, \dots, S_n\}$

For each set S_j , independently, take S_j with probability x_j .

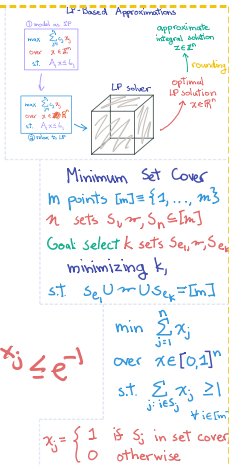
$$E[|F|] = \sum x_j \leq \text{OPT}$$

Fix a point i .

$$E[\# \text{ sets in } F \text{ covering } i] = \sum_{j \in S_i} x_j \geq 1$$

$$P[F \text{ not cover } i] = \prod_{j \in S_i} (1 - x_j) \leq e^{-\sum_{j \in S_i} x_j} \leq e^{-1}$$

$$P[F \text{ cover } i] \geq 1 - e^{-1}$$



Today

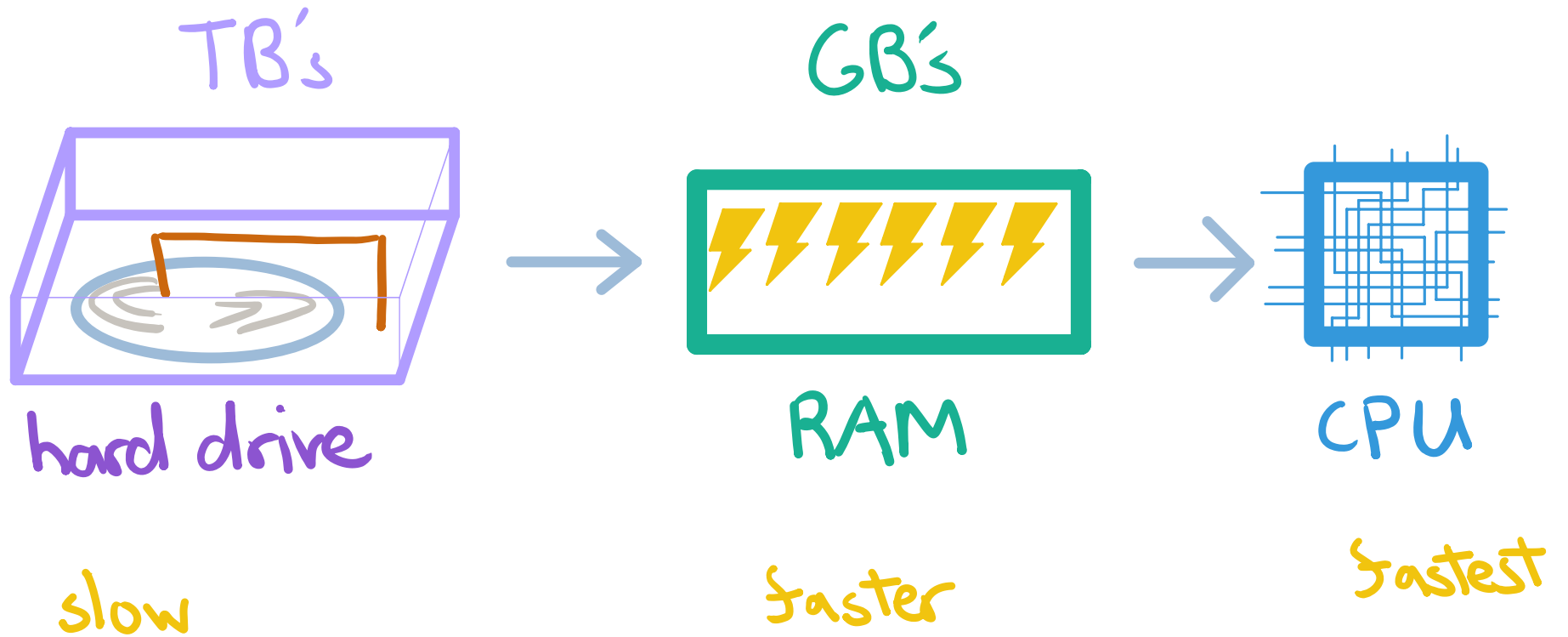
Probability
(and techniques)

Algorithms
(models and applications)

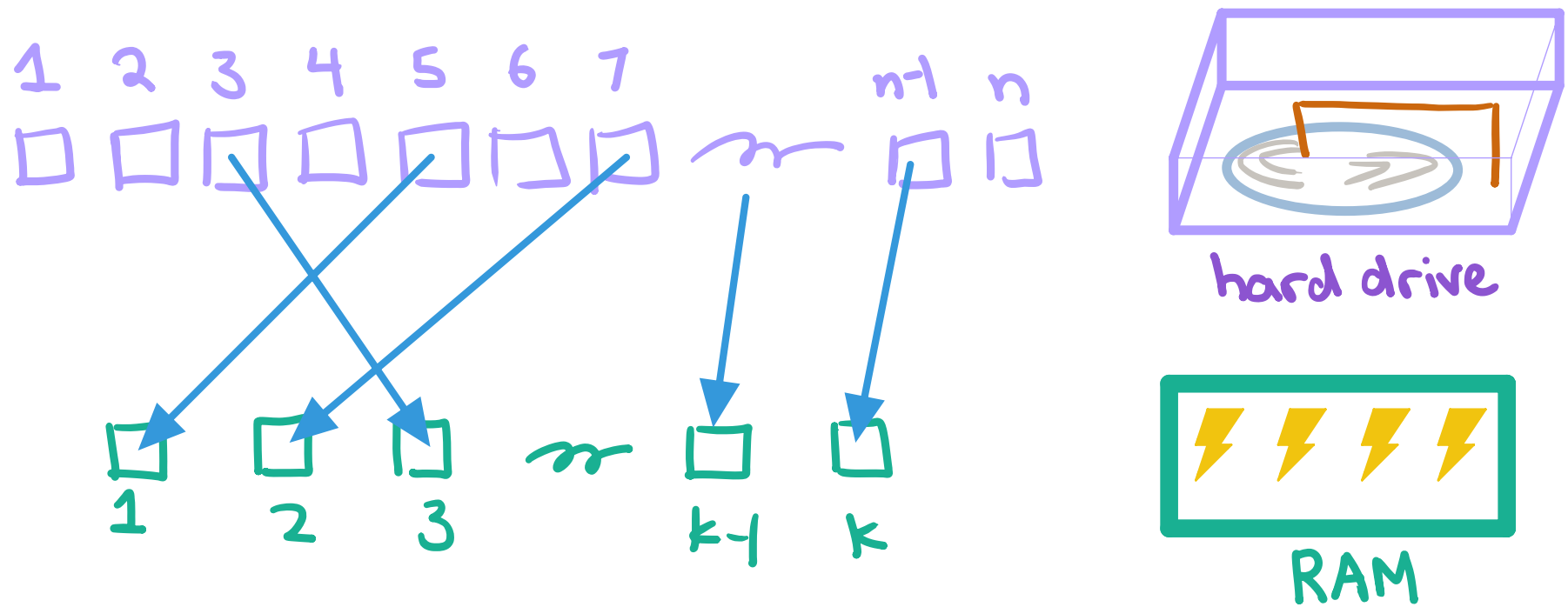
Caching

Buy-or-rent

Caching



Model: n items, k cache slots, $n \gg k$

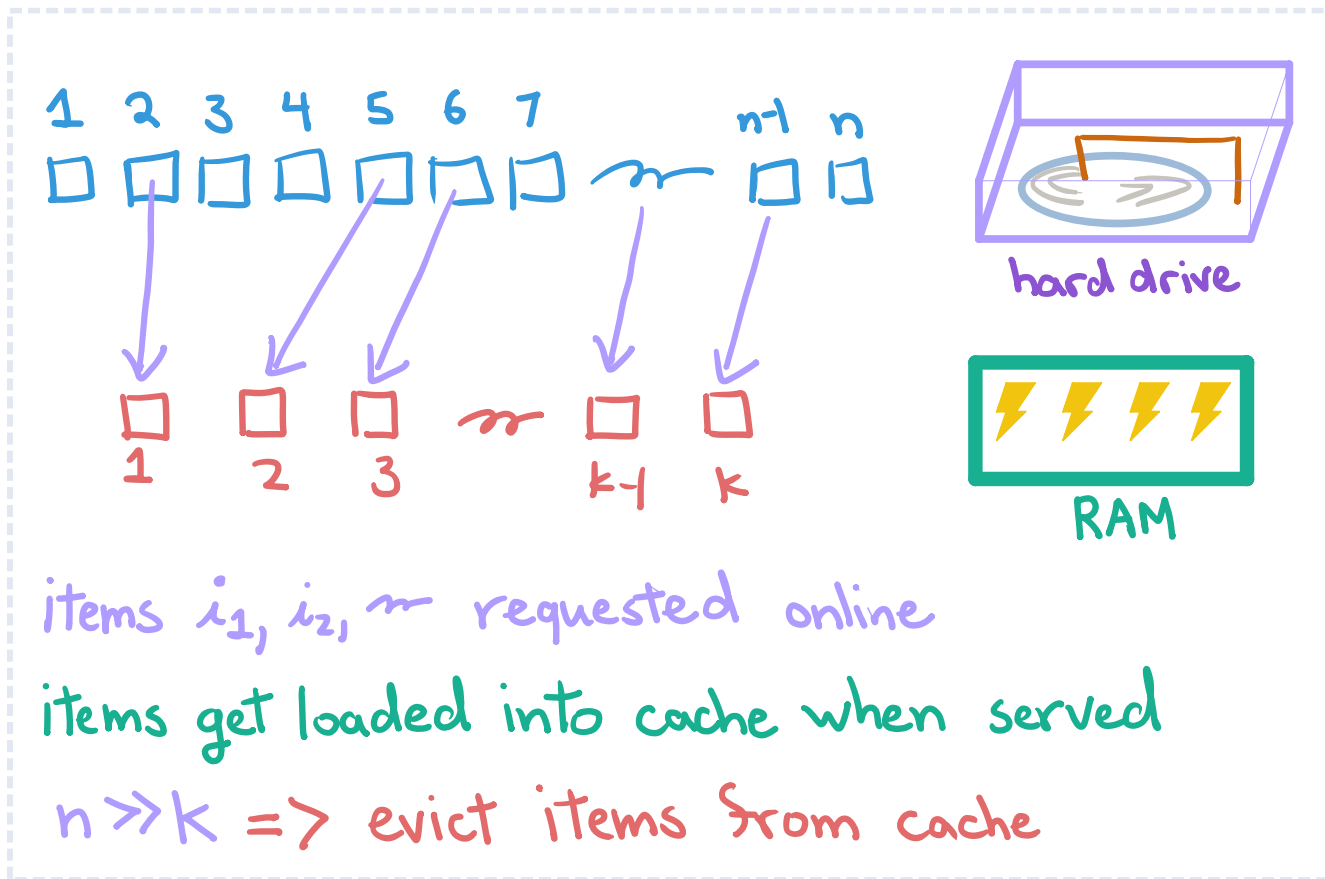


items $i_1, i_2, \dots$ requested online

items get loaded into cache when served

$n \gg k \Rightarrow$ evict items from cache

cache strategies: who to evict?



good: item already in cache when requested

bad: have to load item into cache when requested, "cache miss"

Requests

1

2

1

4

3

5

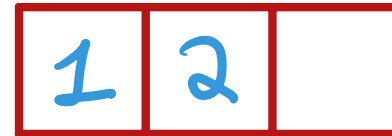
2

1

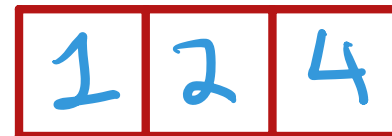
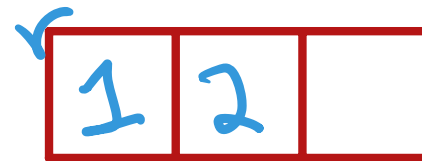
Cache (k=3)



+1



+2



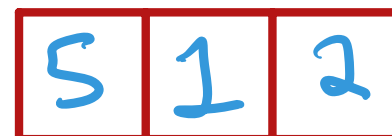
+1



+1



+2



+1

Requests Cache ($k=3$)

1

1		
---	--	--

2

1	2	
---	---	--

1

1	2	
---	---	--

4

1	2	4
---	---	---

Requests Cache ($k=3$)

3

1	3	4
---	---	---

5

5	3	4
---	---	---

2

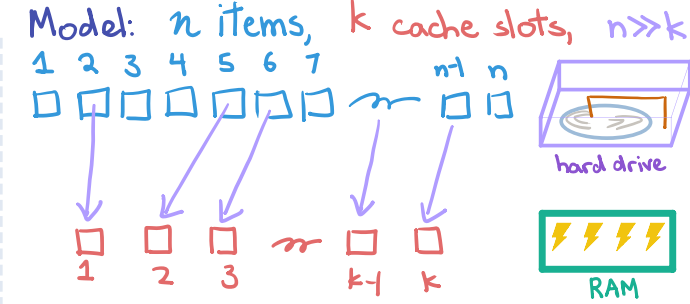
5	3	2
---	---	---

1

5	1	2
---	---	---

Can we come up w/ smart caching strategies and improve performance?

Caching Strategies



items $i_1, i_2, \dots$ requested online

items get loaded into cache when served

$n \gg k \Rightarrow$ also evict items from cache

Goal: minimize cache misses

Caching Strategies

Least frequently used (LFU)

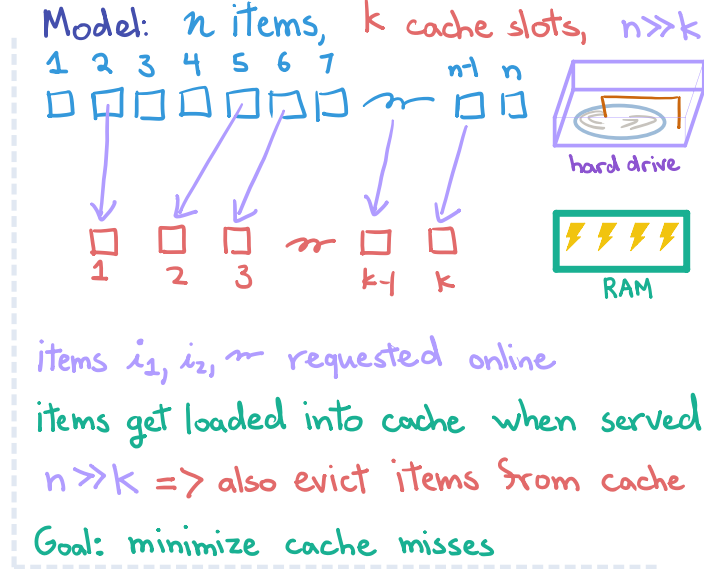
Least recently used (LRU)

Not recently used (NRU, aka 1-Bit LRU)

"mark" requested items

evict unmarked items

if all cache items marked, then
remove all marks and try again

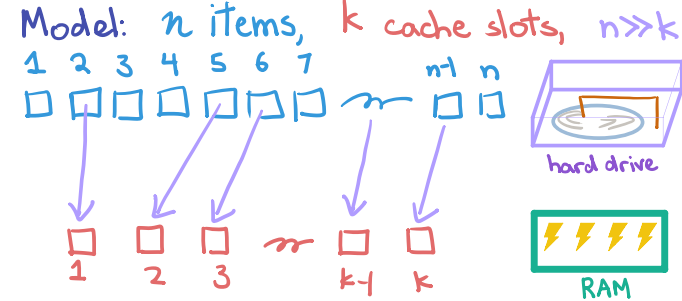


Worst Case Analysis

count # cache misses
try to minimize

adversary can always request
new item and force cache miss

cache misses unbounded ;)



items $i_1, i_2, \dots$ requested online

items get loaded into cache when served

$n \gg k \Rightarrow$ also evict items from cache

Goal: minimize cache misses

Caching Strategies

Least frequently used (LFU)

Least recently used (LRU)

Not recently used (NRU, aka 1-Bit LRU)

"mark" requested items

evict unmarked items

if all cache items marked, then
remove all marks and try again

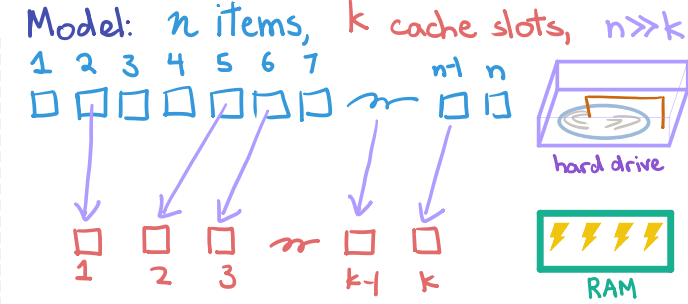
~~Worst Case Analysis~~

Reframe analysis (competitive)

if there is a good cache strategy, then we want to find a comparably good one

"competitive analysis"

minimize ratio $\frac{\text{User misses}}{\text{OPT misses}}$
(in hindsight)



items $i_1, i_2, \dots$ requested online

items get loaded into cache when served

$n \gg k \Rightarrow$ also evict items from cache

Goal: minimize cache misses

Caching Strategies

Least frequently used (LFU)

Least recently used (LRU)

Not recently used (NRU, aka 1-Bit LRU)

"mark" requested items

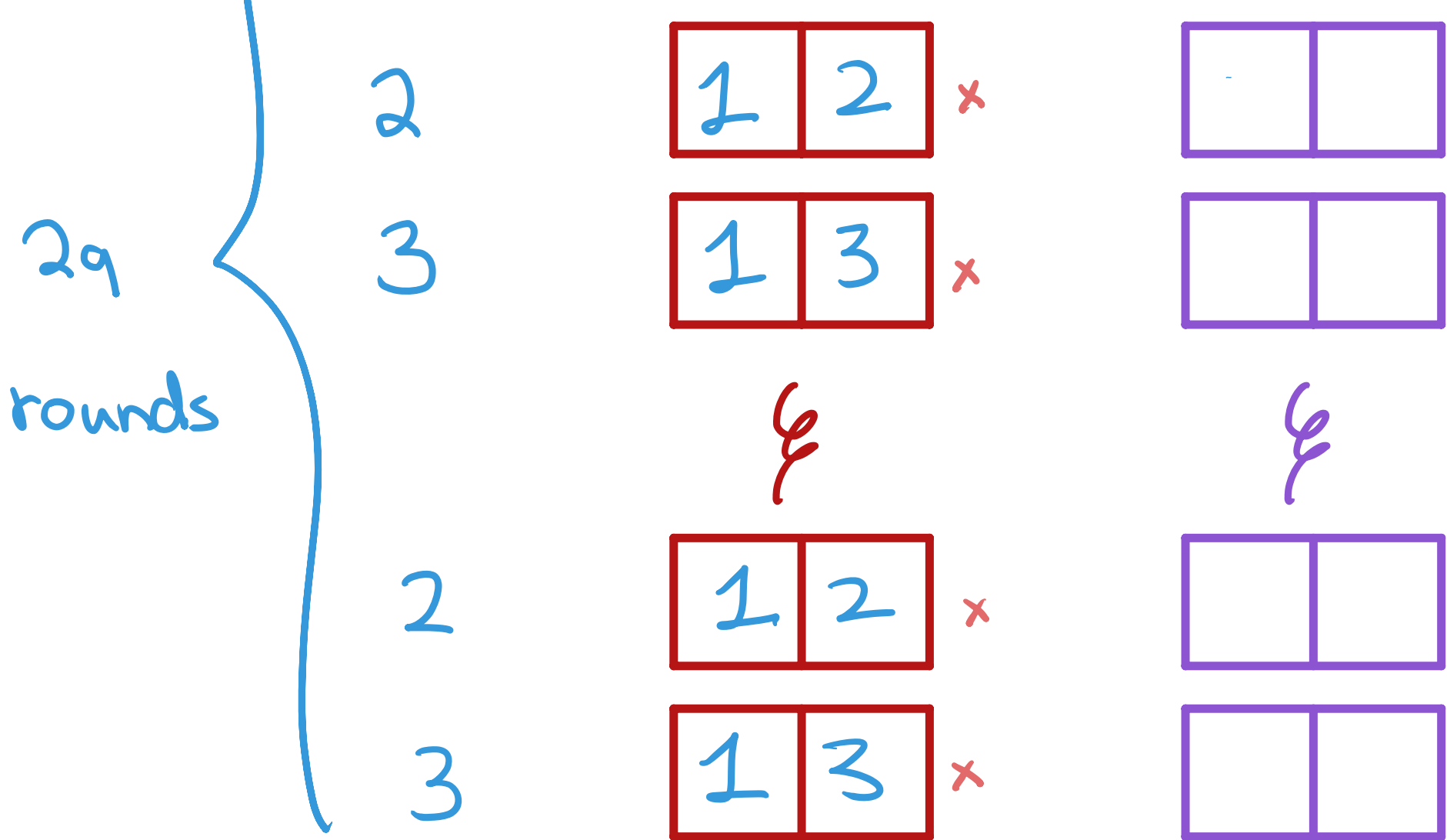
evict unmarked items

if all cache items marked, then remove all marks and try again

Competitive analysis of LFU

is it good? bad? why?

Request		LFU	Optimal
q rounds {	1	1 x	1 x
	1	1	1
	ℓ	ℓ	ℓ
	1	1	1 2
{	2	1 2 x	1 2 x
	3	1 3 x	3 2 x



Request		LFU	Optimal
q rounds {	1	x	x
	1	1	1
	ℓ	ℓ	ℓ
	1	1	1
{	2	1 x	1 x
	3	1 2 x	1 2 x

29
rounds

2

1	3	x
---	---	---

2	3
---	---

3

1	2	x
---	---	---

2	3
---	---

ℓ

2

1	3	x
---	---	---

2	3
---	---

3

1	2	x
---	---	---

2	3
---	---

cont'd

	Request	LFU	Optimal
q rounds	1	x	x
	1	1	1
	∅	∅	∅
	1	1	1

	Request	LFU	Optimal
2q rounds	2	1 x	1 x
	3	1 2 x	1 2 x
	2	1 3 x	2 3
	3	1 2 x	2 3
	∅	∅	∅
	2	1 3 x	2 3
	3	1 2 x	2 3

Competitive analysis of LFU (least frequently used)

3 items {1, 2, 3}

cache size $k=2$

To fool LFU:

load up on first item for q rounds

alternate other items for $2q$ rounds

	Request	LFU	Optimal
q rounds	1		×
	1	1	1
	⋮	⋮	⋮
	1	1	1
$2q$ rounds	2	1 ×	1 ×
	3	1 2 ×	1 2 ×
	2	1 3 ×	2 3
	3	1 2 ×	2 3
	⋮	⋮	⋮
	2	1 3 ×	2 3
	3	1 2 ×	2 3
	⋮	⋮	⋮

OPT has 3 cache misses

LFU has $2q+1$ misses

$$\text{Competitive ratio} = \frac{2q+1}{3} \xrightarrow{q \rightarrow \infty} +\infty$$

Theorem

LFU has unbounded competitive ratio

	Request	LFU		Optimal
q rounds	1		x	x
	1	1		1
	$\emptyset$	$\emptyset$		$\emptyset$
	1	1		1
2q rounds	2	1	x	1 x
	3	1 2	x	1 2 x
	2	1 3	x	2 3
	3	1 2	x	2 3
	$\emptyset$	$\emptyset$		$\emptyset$
	2	1 3	x	2 3
	3	1 2	x	2 3

Competitive Analysis of NRU ^(not recently used)

Recall

NRU (1-Bit LRU)

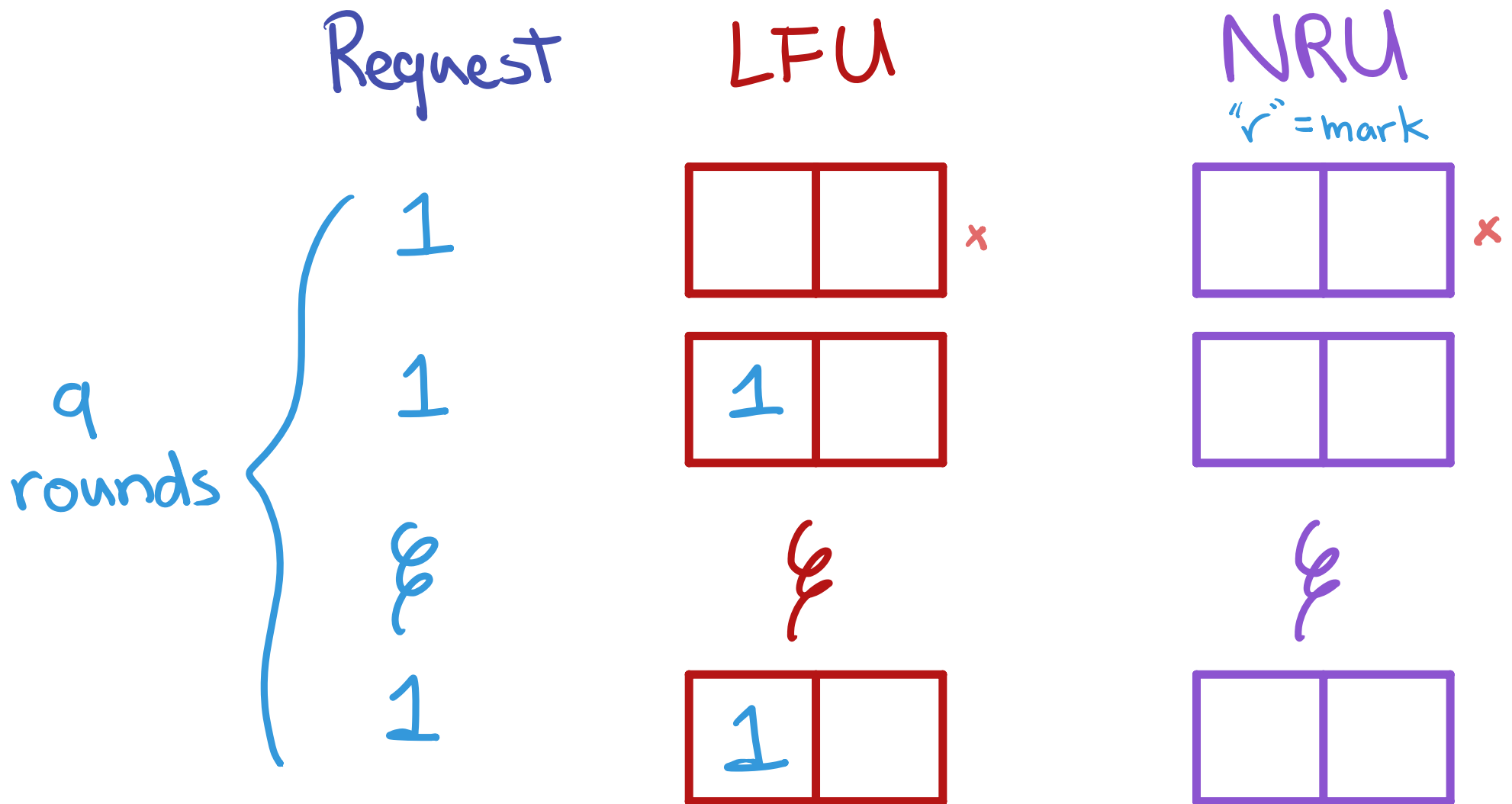
"mark" requested items

evict unmarked items

if all cache items marked, then
remove all marks and try again

Claim: NRU has competitive ratio $\leq k$

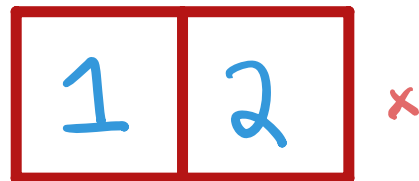
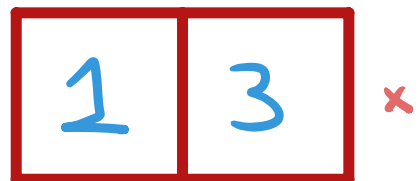
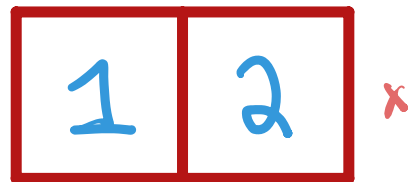
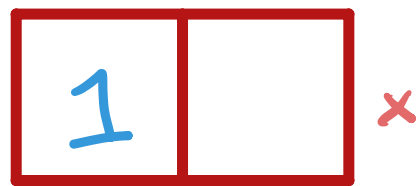
$$\frac{\text{\#NRU misses}}{\text{\#OPT misses}} \leq k$$



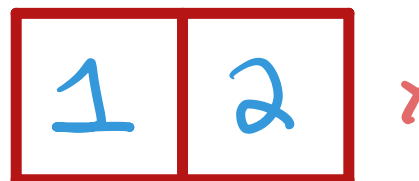
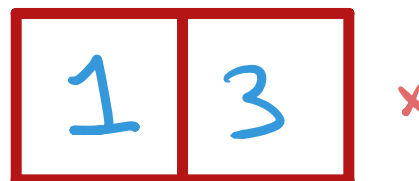
29
rounds

2
3
2
3

2
3



6



6



Claim: $\frac{\# \text{NRU misses}}{\# \text{OPT misses}} \leq k$

NRU (1-Bit LRU)

"mark" requested items

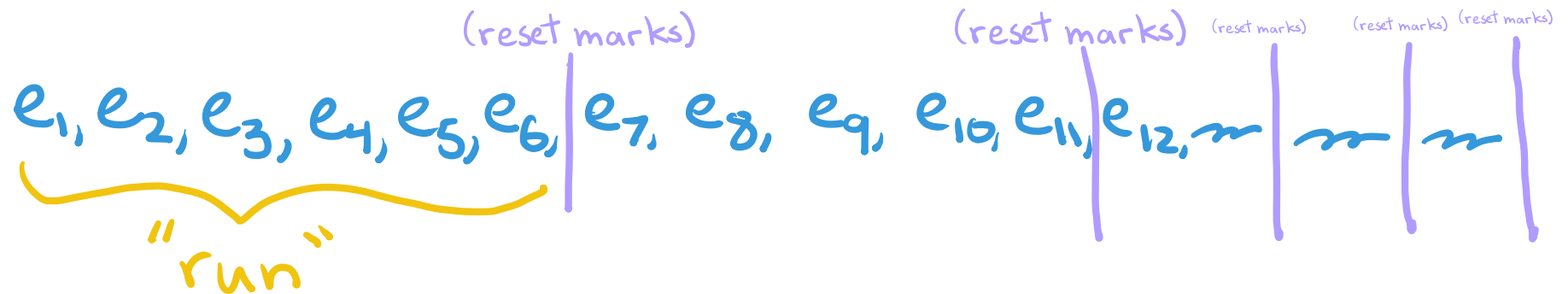
evict unmarked items

if all cache items marked, then
remove all marks and try again

Proof: fix sequence of requests

$e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, \dots$

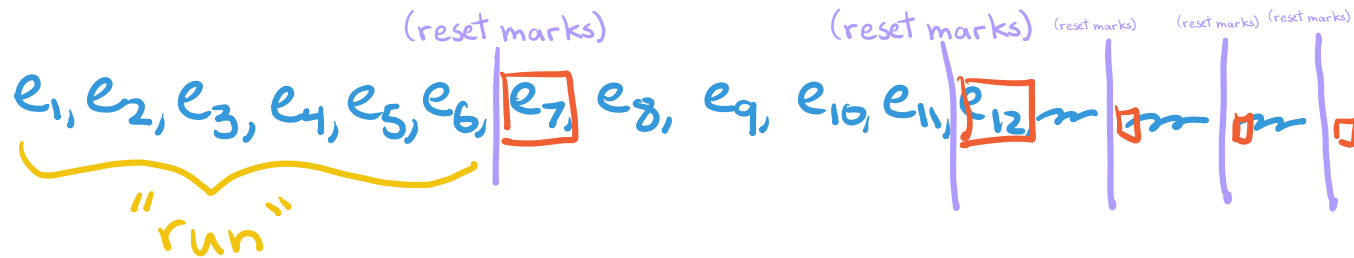
split into "runs" whenever we reset marks



each run (between resets) has k distinct items

NRU has $\leq k$ misses per run

each **run** (between resets) has k distinct items



First item after **run** is distinct from the k distinct items in the run

Any cache must miss either:

- One of the last $k-1$ items in run
- 1st item in the next run

$\Rightarrow$ OPT has ≥ 1 miss per run

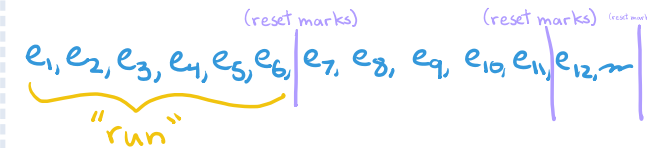
NRU (1-Bit LRU)

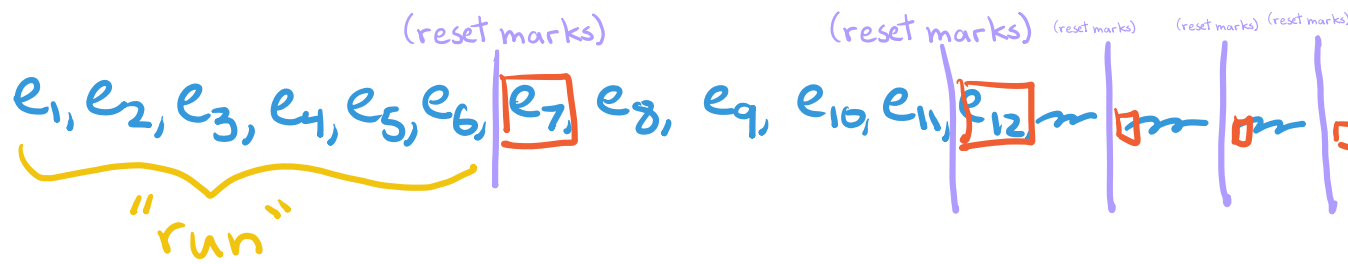
"mark" requested items

evict unmarked items

if all cache items marked, then remove all marks and try again

split into "runs" whenever we reset marks





NRU (1-Bit LRU)

"mark" requested items

evict unmarked items

if all cache items marked, then
remove all marks and try again

each run (between resets) has k distinct items

NRU has $\leq k$ misses per run

first item after run is distinct from
the k distinct items in the run

Any cache must miss either:

- one of the last $k-1$ items in run
- 1st item in the next run

$\Rightarrow$ OPT has ≥ 1 miss per run

$$\frac{\# \text{ NRU misses}}{\# \text{ OPT misses}} \leq \frac{k (\# \text{ runs})}{(\# \text{ runs})} = k$$

Competitive Analysis of NRU^(not recently used)

Recall

NRU (1-Bit LRU)

"mark" requested items

evict unmarked items randomly

if all cache items marked, then
remove all marks and try again

Theorem: NRU^{*} has competitive ratio $\leq k$

$$\frac{\text{\#NRU misses}}{\text{\#OPT misses}} \leq k$$

^{*} LRU is a special case of NRU

better?

Randomized NRU (1-Bit LRU)

"mark" requested items

evict uniformly random unmarked items

if all cache items marked, then
remove all marks and try again

claim: randomized NRU is

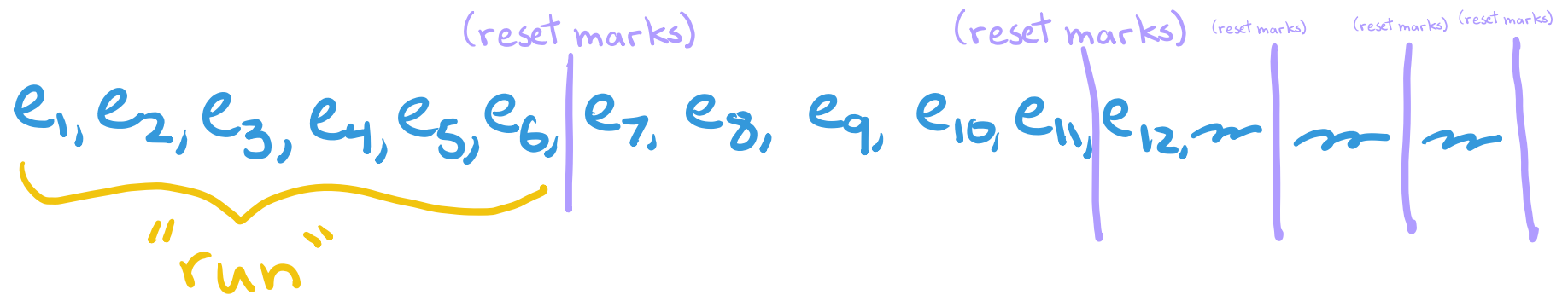
$O(\log k)$ -competitive in expectation

claim: randomized NRU is
 $O(\log k)$ -competitive in expectation

Randomized NRU (1-Bit LRU)
"mark" requested items
evict uniformly random unmarked items
if all cache items marked, then
remove all marks and try again

proof.

split into "runs" whenever we reset marks



each run (between resets) has k distinct items, divided into:

- "new" items that are not in previous run
- "repeat" items also in previous run

potential

$\Phi = \# \text{ items both } \begin{cases} \text{in OPT's cache} \\ \text{not in NRU cache} \end{cases}$

consider a single run $\overset{\text{(reset marks)}}{e_7, e_8, e_9, e_{10}, e_{11}} \mid \overset{\text{(reset marks)}}{e_{12}, \dots}$

let $\Phi_{\text{in}} = \Phi$ before run, $\Phi_{\text{out}} = \Phi$ after run

$\ell = \# \text{ new items in run}$ ($k - \ell$ repeats)

claim: OPT misses $\geq \ell - \Phi_{\text{in}}$

why?

Randomized NRU (1-Bit LRU)

"mark" requested items

evict uniformly random unmarked items

if all cache items marked, then
remove all marks and try again

claim: randomized NRU is

$O(\log k)$ -competitive in expectation

split into "runs" whenever we reset marks

$\overset{\text{(reset marks)}}{e_1, e_2, e_3, e_4, e_5, e_6} \mid \overset{\text{(reset marks)}}{e_7, e_8, e_9, e_{10}, e_{11}} \mid \overset{\text{(reset marks)}}{e_{12}, \dots}$
"run"

each run (between resets) has k distinct items:

- "new" items not in previous run
- "repeat" items also in previous run

$\Phi = \# \text{ items both } \begin{cases} \text{in OPT's cache} \\ \text{not in NRU cache} \end{cases}$

Randomized NRU (1-Bit LRU)

"mark" requested items
evict uniformly random unmarked items
if all cache items marked, then
remove all marks and try again

consider a single run $\begin{matrix} \text{(reset marks)} & & \text{(reset marks)} \\ |e_7, e_8, e_9, e_{10}, e_{11}| \end{matrix}$

let $\Phi_{\text{in}} = \Phi$ before run, $\Phi_{\text{out}} = \Phi$ after run
 $\ell = \# \text{ new items in run}$ ($k - \ell$ repeats)

✓ OPT misses $\geq \ell - \Phi_{\text{IN}}$

claim: OPT misses $\geq \Phi_{\text{OUT}}$

why?

claim: randomized NRU is
 $O(\log k)$ -competitive in expectation

split into "runs" whenever we reset marks

$\begin{matrix} \text{(reset marks)} & & \text{(reset marks)} & \text{(reset marks)} \\ e_1, e_2, e_3, e_4, e_5, e_6 & | & e_7, e_8, e_9, e_{10}, e_{11} & | & e_{12}, \dots \end{matrix}$
"run"

each run (between resets) has k distinct items:

- "new" items not in previous run
- "repeat" items also in previous run

$\Phi = \# \text{ items both } \begin{cases} \text{in OPT's cache} \\ \text{not in NRU cache} \end{cases}$

Randomized NRU (1-Bit LRU)

"mark" requested items
evict uniformly random unmarked items
if all cache items marked, then
remove all marks and try again

consider a single run $\begin{matrix} \text{(reset marks)} & & \text{(reset marks)} \\ |e_7, e_8, e_9, e_{10}, e_{11}| \end{matrix}$

let $\Phi_{in} = \Phi$ before run, $\Phi_{out} = \Phi$ after run
 $\ell = \# \text{ new items in run } (k-\ell \text{ repeats})$

✓ OPT misses $\geq \ell - \Phi_{in}$

✓ OPT misses $\geq \Phi_{out}$

$$\Rightarrow \text{OPT misses} \geq \frac{\ell + \Phi_{out} - \Phi_{in}}{2}$$

claim: randomized NRU is
 $O(\log k)$ -competitive in expectation

split into "runs" whenever we reset marks

$\begin{matrix} \text{(reset marks)} & & \text{(reset marks)} & \text{(reset marks)} \\ e_1, e_2, e_3, e_4, e_5, e_6, | e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, \dots \end{matrix}$
"run" (between resets)

each run (between resets) has k distinct items:

- "new" items not in previous run
- "repeat" items also in previous run

$\Phi = \# \text{ items both } \begin{cases} \text{in OPT's cache} \\ \text{not in NRU cache} \end{cases}$

Randomized NRU (1-Bit LRU)

"mark" requested items
evict uniformly random unmarked items
if all cache items marked, then
remove all marks and try again

consider a single run $|e_7, e_8, e_9, e_{10}, e_{11}|$ (reset marks)

let $\Phi_{in} = \Phi$ before run, $\Phi_{out} = \Phi$ after run
 $\ell = \# \text{ new items in run } (k-\ell \text{ repeats})$ (reset marks)

claim: randomized NRU is
 $O(\log k)$ -competitive in expectation

split into "runs" whenever we reset marks

$e_1, e_2, e_3, e_4, e_5, e_6 | e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, \dots$
"run" (reset marks) (reset marks) (reset marks)

each run (between resets) has k distinct items:

- "new" items not in previous run
- "repeat" items also in previous run

NRU: miss all ℓ new items

$$P[\text{NRU misses 1st repeat}] \leq \frac{\ell}{k}$$

$$\text{OPT misses} \geq \frac{\ell + \Phi_{out} - \Phi_{in}}{2}$$

$\Phi = \# \text{ items both } \begin{cases} \text{in OPT's cache} \\ \text{not in NRU cache} \end{cases}$

Randomized NRU (1-Bit LRU)

"mark" requested items
evict uniformly random unmarked items
if all cache items marked, then
remove all marks and try again

consider a single run $|e_7, e_8, e_9, e_{10}, e_{11}|$

(reset marks)

(reset marks)

let $\Phi_{in} = \Phi$ before run, $\Phi_{out} = \Phi$ after run

$\ell = \#$ new items in run ($k - \ell$ repeats)

claim: randomized NRU is

$O(\log k)$ -competitive in expectation

split into "runs" whenever we reset marks

$e_1, e_2, e_3, e_4, e_5, e_6 | e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, \dots$

(reset marks)

(reset marks) (reset marks)

"run"

each run (between resets) has k distinct items:

- "new" items not in previous run
- "repeat" items also in previous run

NRU: miss all ℓ new items

$P[\text{NRU misses 2nd repeat}] \leq$

$$\text{OPT misses} \geq \frac{\ell + \Phi_{out} - \Phi_{in}}{2}$$

$\Phi = \# \text{ items both } \begin{cases} \text{in OPT's cache} \\ \text{not in NRU cache} \end{cases}$

Randomized NRU (1-Bit LRU)

"mark" requested items
evict uniformly random unmarked items
if all cache items marked, then
remove all marks and try again

consider a single run $|e_7, e_8, e_9, e_{10}, e_{11}|$ (reset marks) (reset marks)

let $\Phi_{in} = \Phi$ before run, $\Phi_{out} = \Phi$ after run
 $\ell = \# \text{ new items in run } (k-\ell \text{ repeats})$

claim: randomized NRU is
 $O(\log k)$ -competitive in expectation

split into "runs" whenever we reset marks

$e_1, e_2, e_3, e_4, e_5, e_6 | e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, \dots$
(reset marks) (reset marks) (reset marks)
"run"

each run (between resets) has k distinct items:

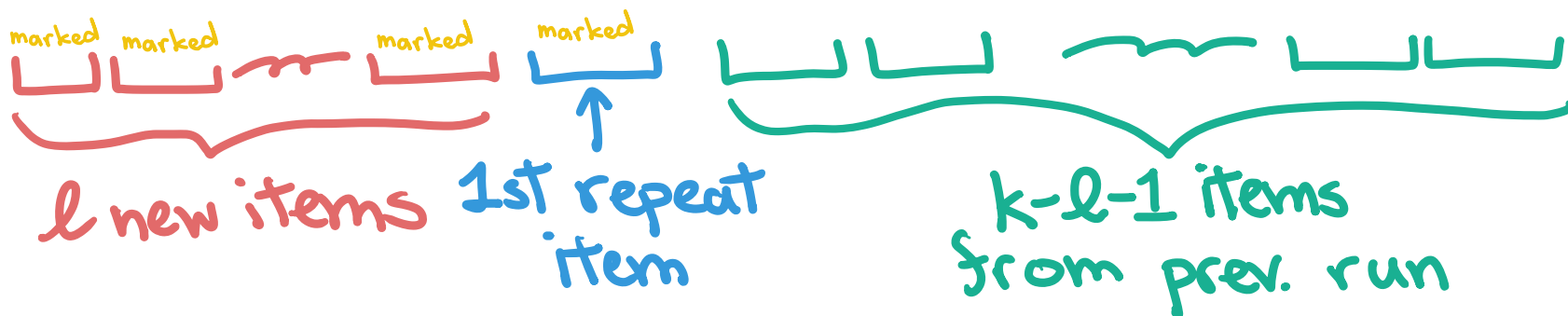
- "new" items not in previous run
- "repeat" items also in previous run

NRU: miss all ℓ new items

$$P[\text{NRU misses 2nd repeat}] \leq \frac{\ell}{k-1}$$

$$\text{OPT misses} \geq \frac{\ell + \Phi_{out} - \Phi_{in}}{2}$$

After 1st repeat, cache has



$\Phi = \# \text{ items both } \begin{cases} \text{in OPT's cache} \\ \text{not in NRU cache} \end{cases}$

Randomized NRU (1-Bit LRU)

"mark" requested items
evict uniformly random unmarked items
if all cache items marked, then
remove all marks and try again

consider a single run $\begin{matrix} \text{(reset marks)} & \text{(reset marks)} \\ |e_7, e_8, e_9, e_{10}, e_{11}| \end{matrix}$

let $\Phi_{in} = \Phi$ before run, $\Phi_{out} = \Phi$ after run
 $\ell = \# \text{ new items in run } (k-\ell \text{ repeats})$

claim: randomized NRU is
 $O(\log k)$ -competitive in expectation

split into "runs" whenever we reset marks

$\begin{matrix} \text{(reset marks)} & \text{(reset marks)} & \text{(reset marks)} \\ e_1, e_2, e_3, e_4, e_5, e_6, | e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, \dots \end{matrix}$
"run"

each run (between resets) has k distinct items:

- "new" items not in previous run
- "repeat" items also in previous run

NRU: miss all ℓ new items

$P[\text{NRU misses } i\text{th repeat}] \leq$

$$\text{OPT misses} \geq \frac{\ell + \Phi_{out} - \Phi_{in}}{2}$$

$\Phi = \# \text{ items both } \begin{cases} \text{in OPT's cache} \\ \text{not in NRU cache} \end{cases}$

Randomized NRU (1-Bit LRU)

"mark" requested items
evict uniformly random unmarked items
if all cache items marked, then
remove all marks and try again

consider a single run $|e_7, e_8, e_9, e_{10}, e_{11}|$ (reset marks) (reset marks)

let $\Phi_{in} = \Phi$ before run, $\Phi_{out} = \Phi$ after run

$\ell = \#$ new items in run ($k - \ell$ repeats)

claim: randomized NRU is

$O(\log k)$ -competitive in expectation

split into "runs" whenever we reset marks

$e_1, e_2, e_3, e_4, e_5, e_6 | e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, \dots$ (reset marks) (reset marks) (reset marks)

"run"

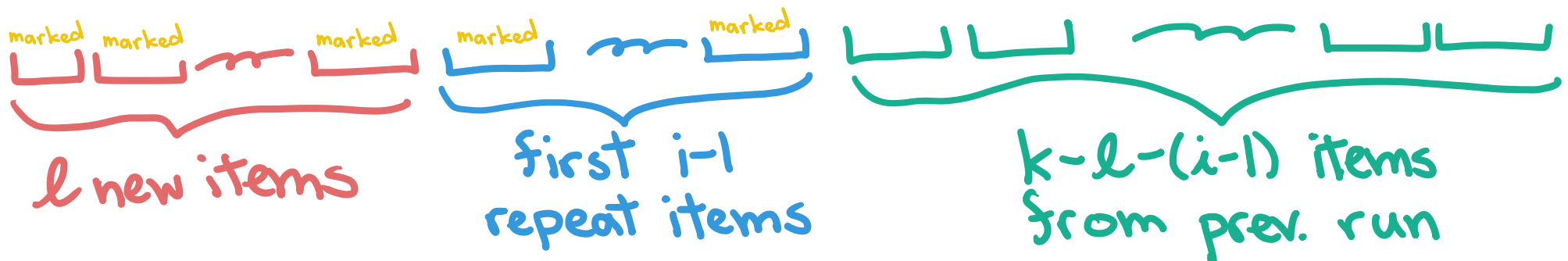
each run (between resets) has k distinct items:

- "new" items not in previous run
- "repeat" items also in previous run

NRU: miss all ℓ new items

$P[\text{NRU misses } i\text{th repeat}] \leq$

$$\text{OPT misses} \geq \frac{\ell + \Phi_{out} - \Phi_{in}}{2}$$



$\Phi = \# \text{ items both } \begin{cases} \text{in OPT's cache} \\ \text{not in NRU cache} \end{cases}$

Randomized NRU (1-Bit LRU)

"mark" requested items
evict uniformly random unmarked items
if all cache items marked, then
remove all marks and try again

consider a single run $\begin{matrix} \text{(reset marks)} & & \text{(reset marks)} \\ |e_7, e_8, e_9, e_{10}, e_{11}| \end{matrix}$

let $\Phi_{in} = \Phi$ before run, $\Phi_{out} = \Phi$ after run
 $\ell = \# \text{ new items in run } (k-\ell \text{ repeats})$

claim: randomized NRU is
 $O(\log k)$ -competitive in expectation

split into "runs" whenever we reset marks

$\begin{matrix} \text{(reset marks)} & & \text{(reset marks)} & \text{(reset marks)} \\ e_1, e_2, e_3, e_4, e_5, e_6 | e_7, e_8, e_9, e_{10}, e_{11} | e_{12}, \dots \end{matrix}$
"run"

each run (between resets) has k distinct items:

- "new" items not in previous run
- "repeat" items also in previous run

NRU: miss all ℓ new items

$$P[\text{NRU misses 1st repeat}] \leq \frac{\ell}{k}$$

$$P[\text{NRU misses 2nd repeat}] \leq \frac{\ell}{k-1}$$

$\vdots$

$$P[\text{NRU misses } i\text{th repeat}] \leq \frac{\ell}{k-(i-1)}$$

$$\text{OPT misses} \geq \frac{\ell + \Phi_{out} - \Phi_{in}}{2}$$

$\Phi = \# \text{ items both } \begin{cases} \text{in OPT's cache} \\ \text{not in NRU cache} \end{cases}$

Randomized NRU (1-Bit LRU)

"mark" requested items
evict uniformly random unmarked items
if all cache items marked, then
remove all marks and try again

consider a single run $\begin{matrix} \text{(reset marks)} & & \text{(reset marks)} \\ |e_7, e_8, e_9, e_{10}, e_{11}| \end{matrix}$

let $\Phi_{in} = \Phi$ before run, $\Phi_{out} = \Phi$ after run
 $\ell = \# \text{ new items in run } (k-\ell \text{ repeats})$

claim: randomized NRU is
 $O(\log k)$ -competitive in expectation

split into "runs" whenever we reset marks

$\begin{matrix} \text{(reset marks)} & & \text{(reset marks)} & \text{(reset marks)} \\ e_1, e_2, e_3, e_4, e_5, e_6, | e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, \dots \end{matrix}$
"run"

each run (between resets) has k distinct items:

- "new" items not in previous run
- "repeat" items also in previous run

NRU: miss all ℓ new items

$$\checkmark P[\text{NRU misses } i\text{th repeat}] \leq \frac{\ell}{k+1-i}$$

$$E[\# \text{ misses on repeats}] \leq \sum_{i=1}^{k-\ell} \frac{\ell}{k+1-i} \leq \ell \ln(k)$$

$$\text{OPT misses} \geq \frac{\ell + \Phi_{out} - \Phi_{in}}{2}$$

$\Phi = \# \text{ items both } \begin{cases} \text{in OPT's cache} \\ \text{not in NRU cache} \end{cases}$

Randomized NRU (1-Bit LRU)

"mark" requested items
evict uniformly random unmarked items
if all cache items marked, then
remove all marks and try again

claim: randomized NRU is

$O(\log k)$ -competitive in expectation

split into "runs" whenever we reset marks

$e_1, e_2, e_3, e_4, e_5, e_6 \mid e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, \dots$
 (reset marks) (reset marks) (reset marks)
 "run"

each run (between resets) has k distinct items:

- "new" items not in previous run
- "repeat" items also in previous run

ℓ_i new items in run i

$\Phi_i = \Phi$ after run i

consider a single run $|e_7, e_8, e_9, e_{10}, e_{11}|$
 (reset marks) (reset marks)
 let $\Phi_{in} = \Phi$ before run, $\Phi_{out} = \Phi$ after run
 $\ell = \#$ new items in run ($k - \ell$ repeats)

$$\# \text{ OPT misses} \geq \frac{\ell + \Phi_{out} - \Phi_{in}}{2} \quad (\text{per run})$$

$$E[\# \text{ NRU misses}] \leq \ell + \ell \ln k$$

let $L = \text{total } \# \text{ new items over all runs}$

$$(\text{total}) \# \text{ OPT misses} \geq \sum_i \frac{\ell_i + \Phi_i - \Phi_{i-1}}{2} = \frac{L}{2} + \frac{\Phi_N^x - \Phi_0}{2}$$

$\geq 0 \quad = 0$

$$E[\# \text{ NRU misses}] \leq \sum_i \ell_i + \ell_i \ln(k) = L + L \ln(k)$$

Probability (and techniques)

Algorithms (models and applications)

✓ Caching

Buy-or-rent

Skiing seems fun
but you don't own skis
rent or buy?

depends on
how many
times you end
up skiing





or take →
everywhere

Uber

lyft

Keep hard drive
spinning when
not in use?

(even more relevant
in trend towards
heterogeneous systems,
emphasis on power consumption)



model

k dollars to buy
(integer)

1 dollar per day to rent

$l = \#$ days we ski (in hindsight)

tricky: don't know l

each day we decide to buy/rent, online



k to buy, 1 per rent, $l = \#$ ski days

If we knew l :

① if $l \leq k$, rent every day

② if $l \geq k$, buy on first day

$$\text{OPT}_{\text{(in hindsight)}} = \min\{k, l\}$$



k to buy, 1 per rent, $l = \#$ ski days

$$\text{OPT} = \min \{k, l\}$$

(in hindsight)

goal: online strategy
competitive w/ OPT

Question:

2-competitive algorithm?



k to buy, 1 per rent, $l = \#$ ski days

$$OPT = \min \{k, l\}$$

(in hindsight)

goal: online strategy competitive w/ OPT

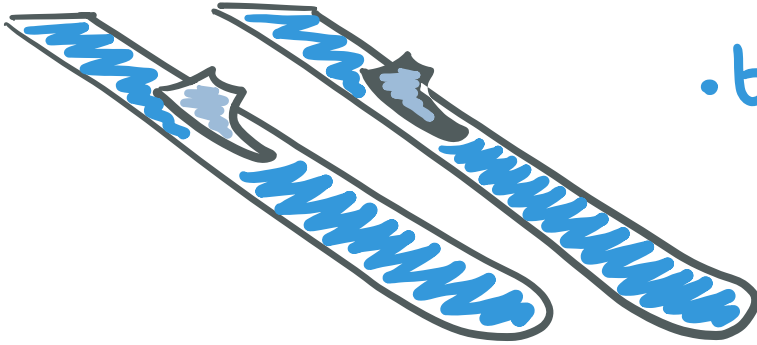
2-competitive algorithm

- First $k-1$ days: rent
- k th day: buy

$$\uparrow \text{cost} < 2 \min \{k, l\} = 2OPT$$



"Half-skis"



- Suppose you can
- buy 1 ski for $k/2$ dollars
 - rent 1 ski for $1/2$ dollars

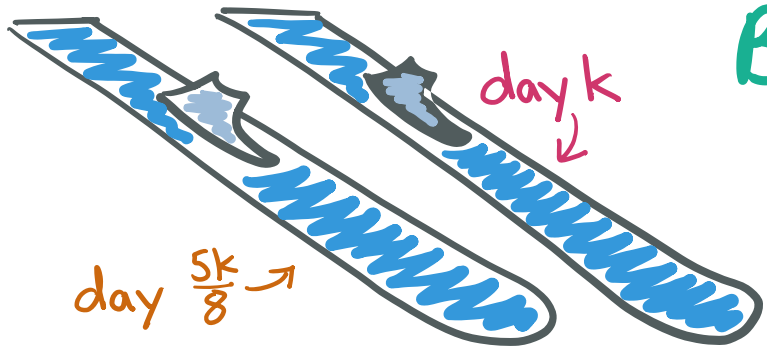
(still need 2 skis each day)

Suppose you can

- buy 1 ski for $k/2$ dollars
- rent 1 ski for $1/2$ dollars
(still need 2 skis each day)

OPT (in hindsight): $\min\{k, \ell\}$ (same as before!)

- $\ell \leq k \Rightarrow$ rent (both skis) each day
- $\ell \geq k \Rightarrow$ buy (both skis) each day



Better-than-2-competitive:

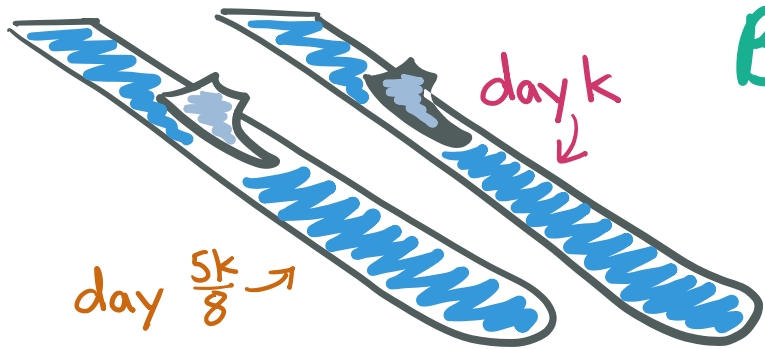
-
-

Suppose you can

- buy 1 ski for $k/2$ dollars
- rent 1 ski for $1/2$ dollars
(still need 2 skis each day)

OPT (in hindsight): $\min\{k, l\}$ (same as before!)

- $l \leq k \Rightarrow$ rent (both skis) each day
- $l \geq k \Rightarrow$ buy (both skis) each day



Better-than-2-competitive:

- Day $\frac{5k}{8}$: buy one ski
- Day k : buy 2nd ski

for $l = 5k/8$, pay $\left(\frac{5k}{8} + \frac{k}{2} \right) / \frac{5k}{8} = \frac{9}{5}$

(rent both) (buy 1)

for l in between, better than

for $\underline{l = k}$, pay $\frac{5k}{8} + \frac{3k}{16} + k = \frac{29}{16}$

(rent both) (rent 1) (buy both)

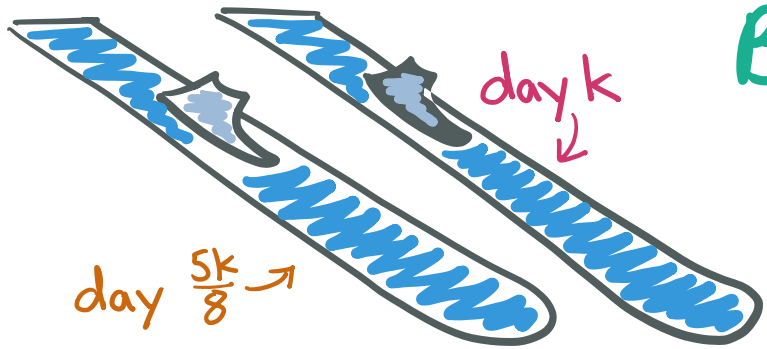


Suppose you can

- buy 1 ski for $k/2$ dollars
- rent 1 ski for $1/2$ dollars
(still need 2 skis each day)

OPT (in hindsight): $\min\{k, \ell\}$ (same as before!)

- $\ell \leq k \Rightarrow$ rent (both skis) each day
- $\ell \geq k \Rightarrow$ buy (both skis) each day



Better-than-2-competitive:

- Day $\frac{5k}{8}$: buy one ski
- Day k : buy 2nd ski

for $\ell = \frac{5k}{8}$, pay $\frac{5k}{8} + \frac{k}{2} = \frac{9k}{8} = \frac{9}{5} \text{OPT}$

(rent both) (buy 1)

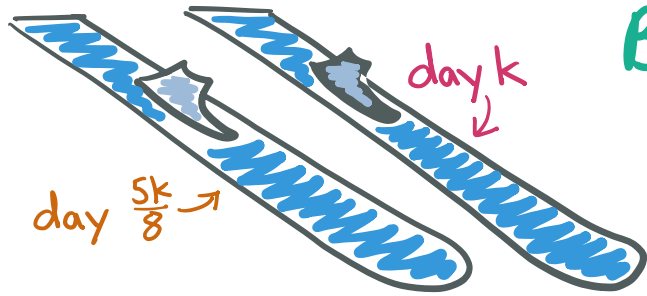
for ℓ in between, better than

for $\ell = k$, pay $\frac{5k}{8} + \frac{1}{2} \cdot \frac{3k}{8} + k = \frac{29}{16} (k = \text{OPT})$

(rent both) (rent 1) (buy both)

"Liquid skis"

- buy 1 ski for $k/2$
- rent 1 ski for $1/2$
- $\text{OPT} = \min\{k, \ell\}$



Better-than-2-competitive:

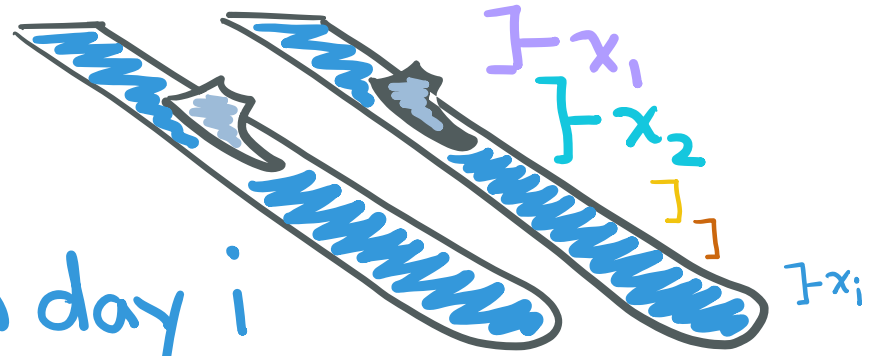
- Day $\frac{5k}{8}$: buy one ski
- Day k: buy 2nd ski

x_1 = fractional skis bought on day 1

x_2 = fractional skis bought
on day 2

x_i = fractional skis bought on day i

y_i = fractional skis rented on day i



$$\min k(x_1 + \dots + x_e) + y_1 + \dots + y_e$$

$$\text{s.t. } x_1 + \dots + x_i + y_i \geq 1 \text{ for each day } i$$

x_i = fractional skis bought on day i

y_i = fractional skis rented on day i

$$\min k(x_1 + r + x_e) + y_1 + r + y_e$$

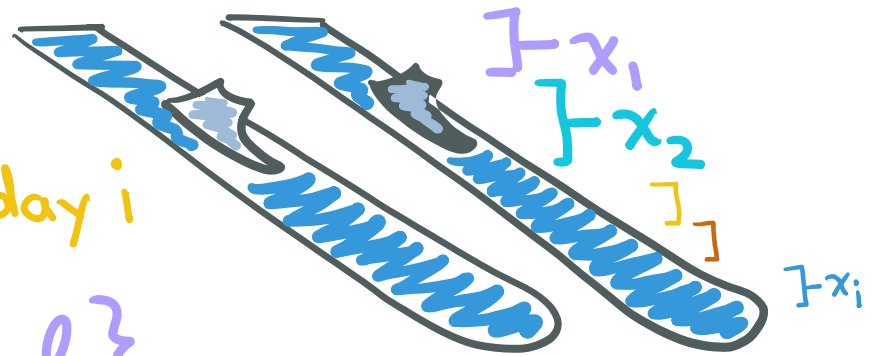
$$\text{s.t. } x_1 + r + x_i + y_i \geq 1 \text{ for each day } i$$

$$\text{OPT (in hindsight)} = \min\{k, l\}$$

- if $l \leq k$, set $x_1 = r = x_e = 0$ and $y_1 = r = y_e = 1$
- if $l \geq k$, set $x_1 = 1$ and $x_2 = r = x_e = y_1 = r = y_e = 0$

$$kx_1 + (1 - x_1) = 1 + \delta$$

$$x_1 = \frac{\delta}{k-1}$$

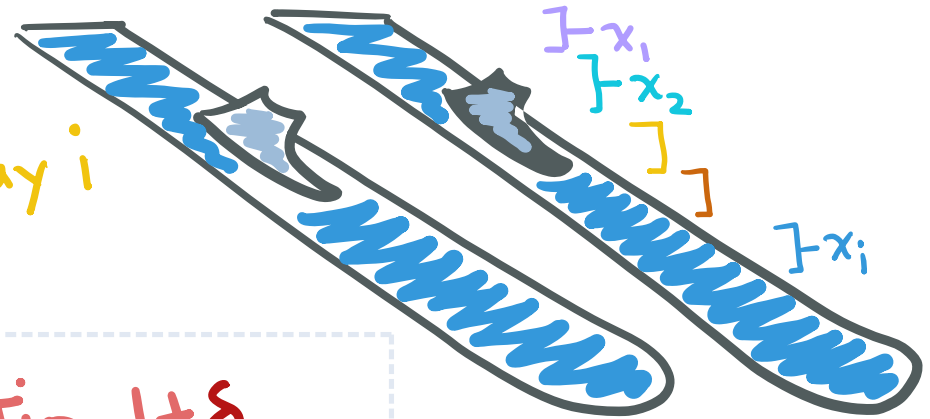


x_i = fractional skis bought on day i y_i = fractional skis rented on day i

$$\min k(x_1 + \dots + x_l) + y_1 + \dots + y_l$$

$$\text{s.t. } x_1 + \dots + x_i + y_i \geq 1 \text{ for each day } i$$

$$\text{OPT (in hindsight)} = \min\{k, l\}$$



Idea: target competitive ratio $1+\delta$
 $\delta > 0$ TBD

Pay $1+\delta$ every day until we completely buy on day k

$$\text{day 1: } 1+\delta = kx_1 + (1-x_1) \Rightarrow x_1 = \frac{\delta}{k-1}$$

$$(k-1)x_2 = \delta + x_1$$

$$\text{day 2: } 1+\delta = kx_2 + (1-x_1-x_2) \Rightarrow x_2 =$$

$$x_2 = \frac{\delta}{(k-1)} + \frac{\delta}{(k-1)^2}$$

⋮

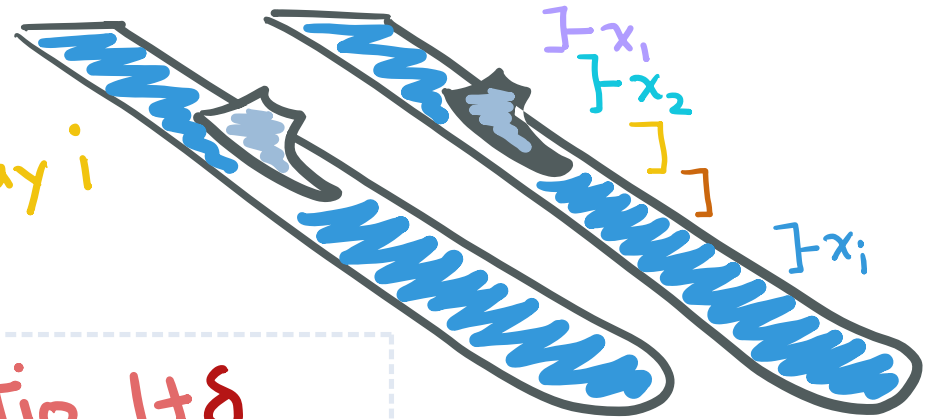
$$\text{day } i: 1+\delta = kx_i + (1-x_1 - \dots - x_{i-1}) \Rightarrow x_i = \frac{\delta}{(k-1)} + \frac{\delta}{(k-1)^2} + \dots + \frac{\delta}{(k-1)^{i-1}}$$

x_i = fractional skis bought on day i y_i = fractional skis rented on day i

$$\min k(x_1 + \dots + x_\ell) + y_1 + \dots + y_\ell$$

$$\text{s.t. } x_1 + \dots + x_i + y_i \geq 1 \text{ for each day } i$$

$$\text{OPT (in hindsight)} = \min\{k, \ell\}$$



Idea: target competitive ratio $1+\delta$
 $\delta > 0$ TBD

Pay $1+\delta$ every day until we completely buy on day k

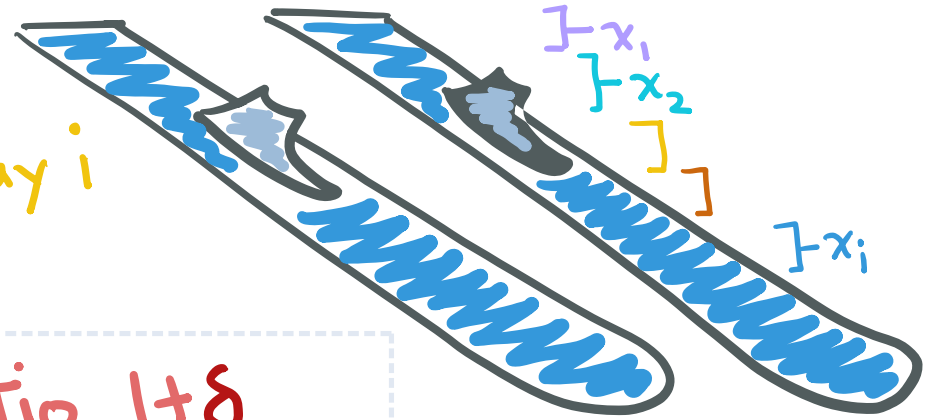
$$x_i =$$

x_i = fractional skis bought on day i y_i = fractional skis rented on day i

$$\min k(x_1 + \dots + x_\ell) + y_1 + \dots + y_\ell$$

$$\text{s.t. } x_1 + \dots + x_i + y_i \geq 1 \text{ for each day } i$$

$$\text{OPT (in hindsight)} = \min\{k, \ell\}$$



Idea: target competitive ratio $1+\delta$
 $\delta > 0$ TBD

Pay $1+\delta$ every day until we completely buy on day k

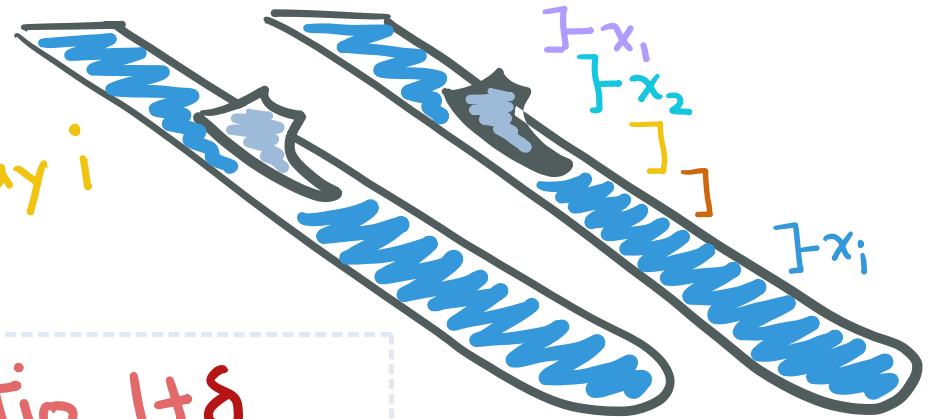
$$x_i = \frac{1}{k-1}(\delta + x_1 + \dots + x_{i-1})$$

x_i = fractional skis bought on day i y_i = fractional skis rented on day i

$$\min k(x_1 + \dots + x_\ell) + y_1 + \dots + y_\ell$$

$$\text{s.t. } x_1 + \dots + x_i + y_i \geq 1 \text{ for each day } i$$

$$\text{OPT (in hindsight)} = \min\{k, \ell\}$$



Idea: target competitive ratio $1+\delta$
 $\delta > 0$ TBD

Pay $1+\delta$ every day until we completely buy on day k

$$x_i = \frac{1}{k-1}(\delta + x_1 + \dots + x_{i-1})$$

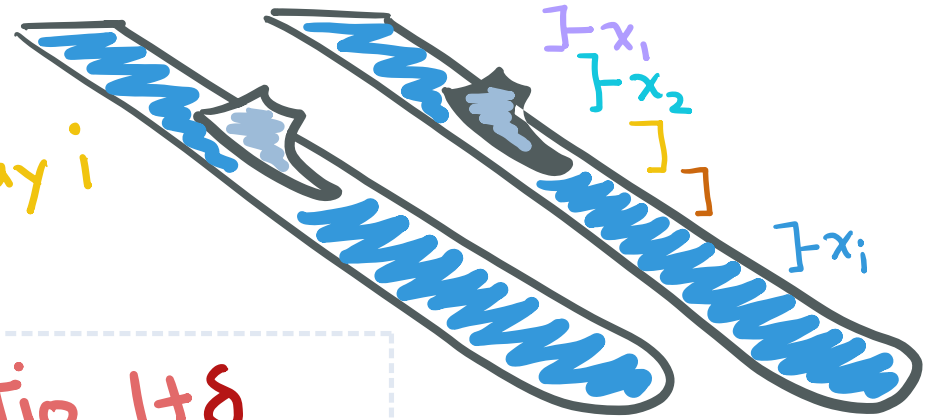
$$1 = x_1 + \dots + x_k =$$

x_i = fractional skis bought on day i y_i = fractional skis rented on day i

$$\min k(x_1 + \dots + x_\ell) + y_1 + \dots + y_\ell$$

$$\text{s.t. } x_1 + \dots + x_i + y_i \geq 1 \text{ for each day } i$$

$$\text{OPT (in hindsight)} = \min\{k, \ell\}$$



Idea: target competitive ratio $1+\delta$
 $\delta > 0$ TBD

Pay $1+\delta$ every day until we completely buy on day k

$$x_i = \frac{1}{k-1} (\delta + x_1 + \dots + x_{i-1})$$

$$1 = x_1 + \dots + x_k = \frac{\delta}{k-1} + \frac{k}{k-1} (x_1 + \dots + x_{k-1}) = \dots$$

$$= \frac{\delta}{k-1} \left(1 + \frac{k}{k-1} + \left(\frac{k}{k-1} \right)^2 + \dots + \left(\frac{k}{k-1} \right)^{k-1} \right)$$

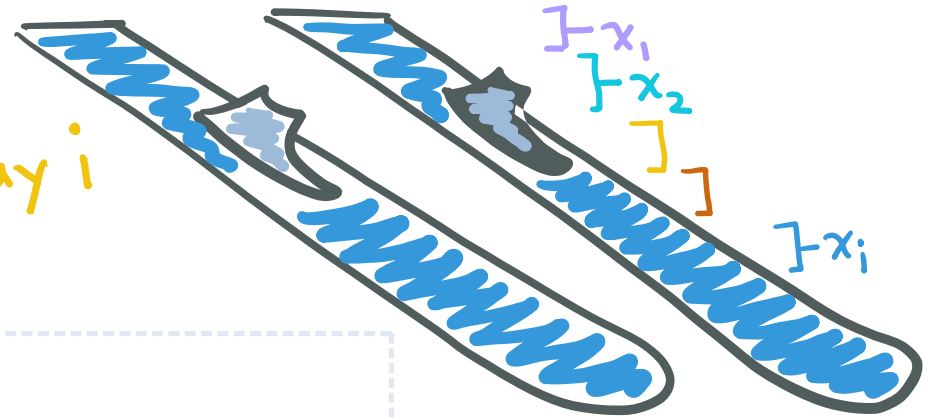
$$= \delta \left(\left(\frac{k}{k-1} \right)^k - \left(\frac{k}{k-1} \right) \right) \leftarrow$$

x_i = fractional skis bought on day i y_i = fractional skis rented on day i

$$\min k(x_1 + \dots + x_\ell) + y_1 + \dots + y_\ell$$

$$\text{s.t. } x_1 + \dots + x_i + y_i \geq 1 \text{ for each day } i$$

$$\text{OPT (in hindsight)} = \min\{k, \ell\}$$



Idea: let $\delta > 0$ TBD.

Pay $1 + \delta$ every day until we completely buy on day k

(then competitive ratio = $1 + \delta$)

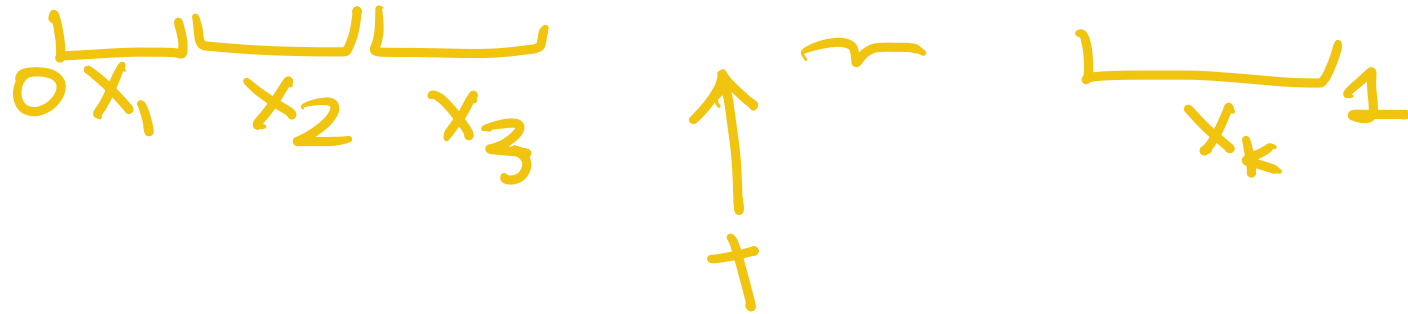
$$x_i = \frac{1}{k-1} (\delta + x_1 + \dots + x_{i-1})$$

$$1 = \dots = \delta \left(\left(\frac{k}{k-1} \right)^k - \left(\frac{k}{k-1} \right) \right)$$

$$\delta = \frac{1}{\left(\frac{k}{k-1} \right)^k - \frac{k}{k-1}} \xrightarrow{k \rightarrow \infty} \frac{1}{e-1}$$

Competitive ratio

$$\xrightarrow{k \rightarrow \infty} \frac{e}{e-1} \approx 1.582$$



$$E[\$ \text{ buy}] = k(x_1 + \dots + x_k)$$

$$E[\text{rent } \$ \text{ day}] = 1 - x_1 - \dots - x_k = Y_j$$

update rule

$$x_i \leftarrow \frac{1}{k}(x_i + r + x_{i-1}) + \frac{\delta}{k} \quad (\delta \text{ TBD})$$

$$y_i \leftarrow 1 - (x_i + r + x_i)$$

Ownership is exponentially increasing

$$\underbrace{x_i + r + x_i}_{\text{total ownership after } i \text{ days}} = (1 + \frac{1}{k}) \underbrace{(x_i + r + x_{i-1})}_{\text{total ownership after } i-1 \text{ days}} + \frac{\delta}{k}$$

cost per day

$$kx_i + y_i = x_i + r + x_{i-1} + \delta + 1 - x_i - r - x_i$$
$$\begin{matrix} x_i \leftarrow \frac{1}{k}(x_i + r + x_{i-1}) + \frac{\delta}{k} \\ y_i \leftarrow 1 - (x_i + r + x_i) \end{matrix} \leq 1 + \delta$$

meanwhile

OPT pays 1 per day for first k days

