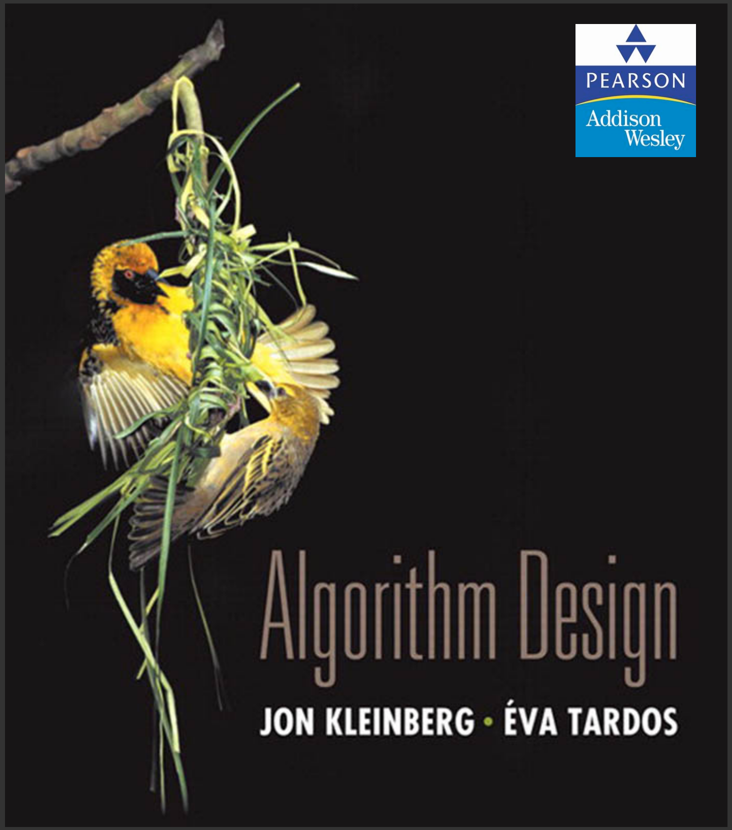


4. GREEDY ALGORITHMS I

- *coin changing*
- *interval scheduling*
- *interval partitioning*
- *scheduling to minimize lateness*
- *optimal caching*
- *stable matching*

Last updated on 2021/12/1 下午12:29

Overview of greedy algorithm design paradigm

Greedy paradigm. Construct a solution iteratively, via a sequence of myopic decisions, and hope that everything works out in the end.

Features.

- Easy to come up with one or more greedy algorithms.
- Easy to analyze the running time.
- Hard to establish correctness.

Warning. Most greedy algorithms are not always correct.

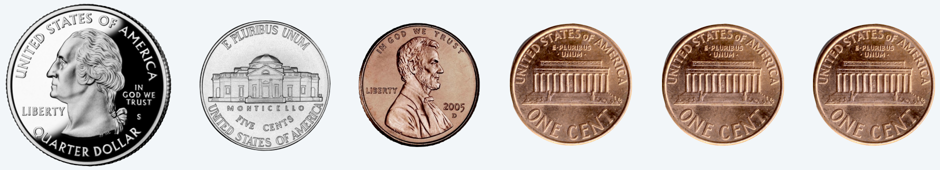
4. Greedy Algorithms I

- *coin changing*
- *interval scheduling*
- *interval partitioning*
- *scheduling to minimize lateness*
- *optimal caching*
- *stable matching*

Coin changing

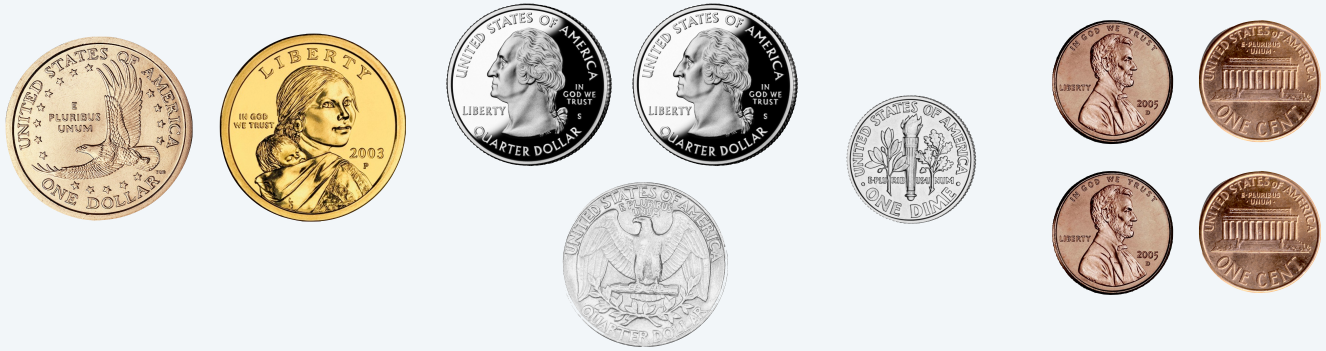
Goal. Given U. S. currency denominations { 1, 5, 10, 25, 100 }, devise a method to pay amount to customer using fewest coins.

Ex. 34¢.



Cashier′s algorithm. At each iteration, add coin of the largest value that does not take us past the amount to be paid.

Ex. $2.89.



Cashier's algorithm

At each iteration, add coin of the largest value that does not take us past the amount to be paid.

```
CASHIERS-ALGORITHM $(x, c_1, c_2, \ldots, c_n)$

SORT $n$ coin denominations so that $0 < c_1 < c_2 < \ldots < c_n$.
$S \leftarrow \varnothing$.   ← multiset of coins selected
WHILE $(x > 0)$
    $k \leftarrow$ largest coin denomination $c_k$ such that $c_k \leq x$.
    IF (no such $k$)
        RETURN "no solution."
    ELSE
        $x \leftarrow x - c_k$.
        $S \leftarrow S \cup \{ k \}$.
RETURN $S$.
```

Greedy algorithms I: quiz 1

Is the cashier's algorithm optimal?

A. Yes, greedy algorithms are always optimal.

B. Yes, for any set of coin denominations $c_1 < c_2 < \ldots < c_n$ provided $c_1 = 1$.

C. Yes, because of special properties of U.S. coin denominations.

D. No.

Cashier's algorithm (for arbitrary coin denominations)

Q. Is cashier's algorithm optimal for any set of denominations?

A. No. Consider U.S. postage: 1, 10, 21, 34, 70, 100, 350, 1225, 1500.

- Cashier's algorithm: 140¢ = 100 + 34 + 1 + 1 + 1 + 1 + 1 + 1.
- Optimal: 140¢ = 70 + 70.

A. No. It may not even lead to a feasible solution if $c_1 > 1$: 7, 8, 9.

- Cashier's algorithm: 15¢ = 9 + ?.
- Optimal: 15¢ = 7 + 8.

Properties of any optimal solution (for U.S. coin denominations)

Property. Number of pennies ≤ 4.

Pf. Replace 5 pennies with 1 nickel.

Property. Number of nickels ≤ 1.

Property. Number of quarters ≤ 3.

Property. Number of nickels + number of dimes ≤ 2.

Pf.

- Recall: ≤ 1nickel.
- Replace 3 dimes and 0 nickels with 1 quarter and 1 nickel;
- Replace 2 dimes and 1 nickel with 1 quarter.

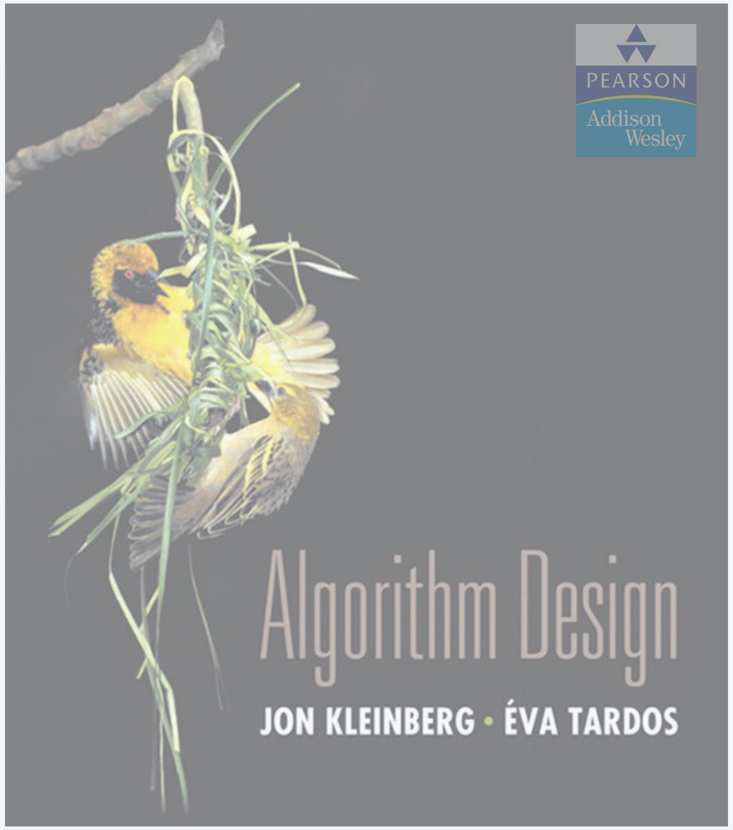
Optimality of cashier's algorithm (for U.S. coin denominations)

Theorem. Cashier's algorithm is optimal for U.S. coins { 1, 5, 10, 25, 100 }.

Pf. [by induction on amount to be paid x]

- Consider optimal way to change $c_k \leq x < c_{k+1}$: greedy takes coin k.
- We claim that any optimal solution must take coin k.
  - if not, it needs enough coins of type $c_1, \ldots, c_{k-1}$ to add up to x
  - table below indicates no optimal solution can do this
- Problem reduces to coin-changing $x - c_k$ cents, which, by induction, is optimally solved by cashier's algorithm. ▪

k	c_k	all optimal solutions must satisfy	max value of coin denominations $c_1, c_2, \ldots, c_{k-1}$ in any optimal solution
1	1	$P \leq 4$	–
2	5	$N \leq 1$	4
3	10	$N + D \leq 2$	4 + 5 = 9
4	25	$Q \leq 3$	20 + 4 = 24
5	100	*no limit*	75 + 24 = 99



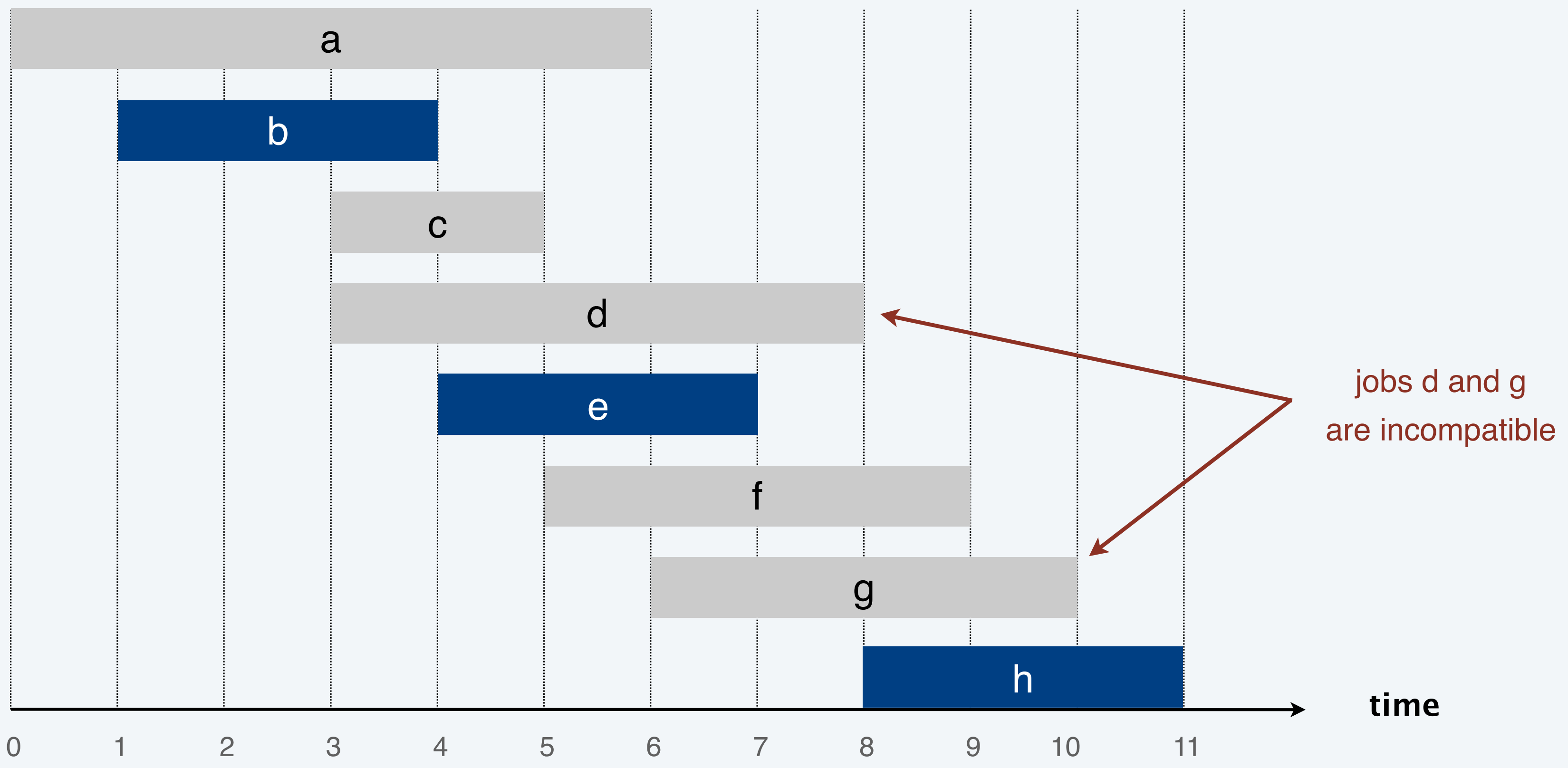
SECTION 4.1

4. GREEDY ALGORITHMS I

- *coin changing*
- *interval scheduling*
- *interval partitioning*
- *scheduling to minimize lateness*
- *optimal caching*
- *stable matching*

Interval scheduling

- Job j starts at s_j and finishes at f_j.
- Two jobs are compatible if they don't overlap.
- Goal: find maximum subset of mutually compatible jobs.



Greedy algorithms I: quiz 2

Consider jobs in some order, taking each job provided it's compatible with the ones already taken. Which rule is optimal?

A. [Earliest start time] Consider jobs in ascending order of s_j.

B. [Earliest finish time] Consider jobs in ascending order of f_j.

C. [Shortest interval] Consider jobs in ascending order of $f_j - s_j$.

D. None of the above.

Interval scheduling: earliest-finish-time-first algorithm

```
EARLIEST-FINISH-TIME-FIRST ($n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n$)

SORT jobs by finish times and renumber so that $f_1 \le f_2 \le \ldots \le f_n$.
$S \leftarrow \varnothing$.   ⟵ set of jobs selected
FOR $j = 1$ TO $n$
    IF (job $j$ is compatible with $S$)
        $S \leftarrow S \cup \{ j \}$.
RETURN $S$.
```

Proposition. Can implement earliest-finish-time first in $O(n \log n)$ time.

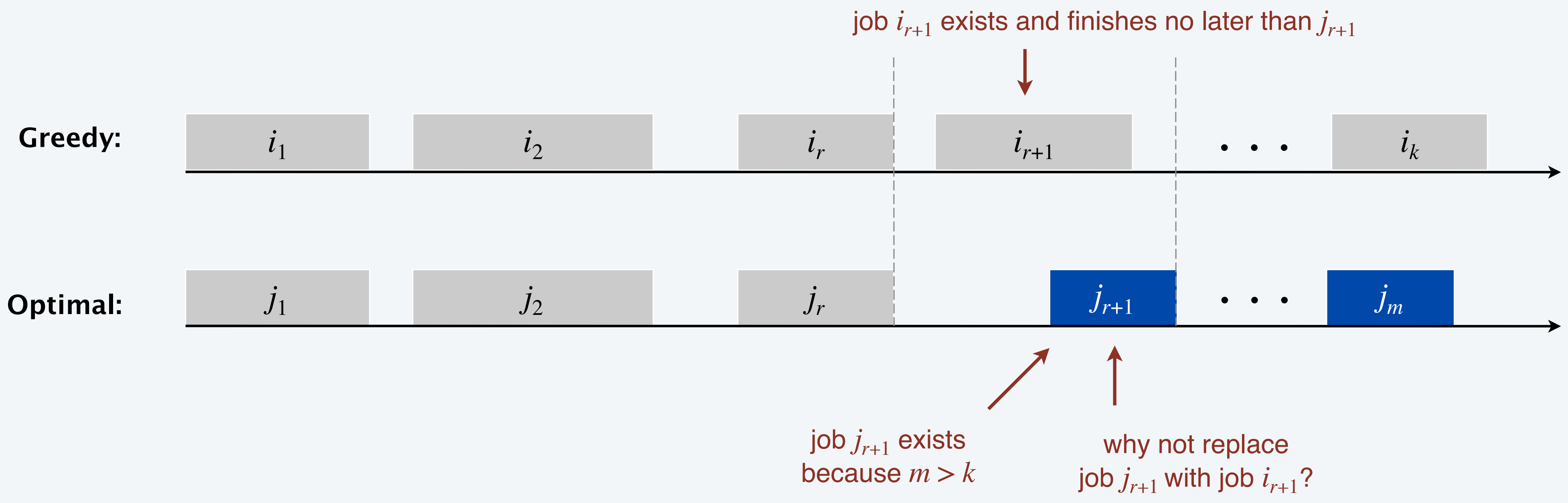
- Keep track of job j^* that was added last to S.
- Job j is compatible with S iff $s_j \ge f_{j^*}$.
- Sorting by finish times takes $O(n \log n)$ time.

Interval scheduling: analysis of earliest-finish-time-first algorithm

Theorem. The earliest-finish-time-first algorithm is optimal.

Pf. [by contradiction]

- Assume greedy is not optimal, and let's see what happens.
- Let $i_1, i_2, \ldots i_k$ denote set of jobs selected by greedy.
- Let $j_1, j_2, \ldots j_m$ denote set of jobs in an optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of r.

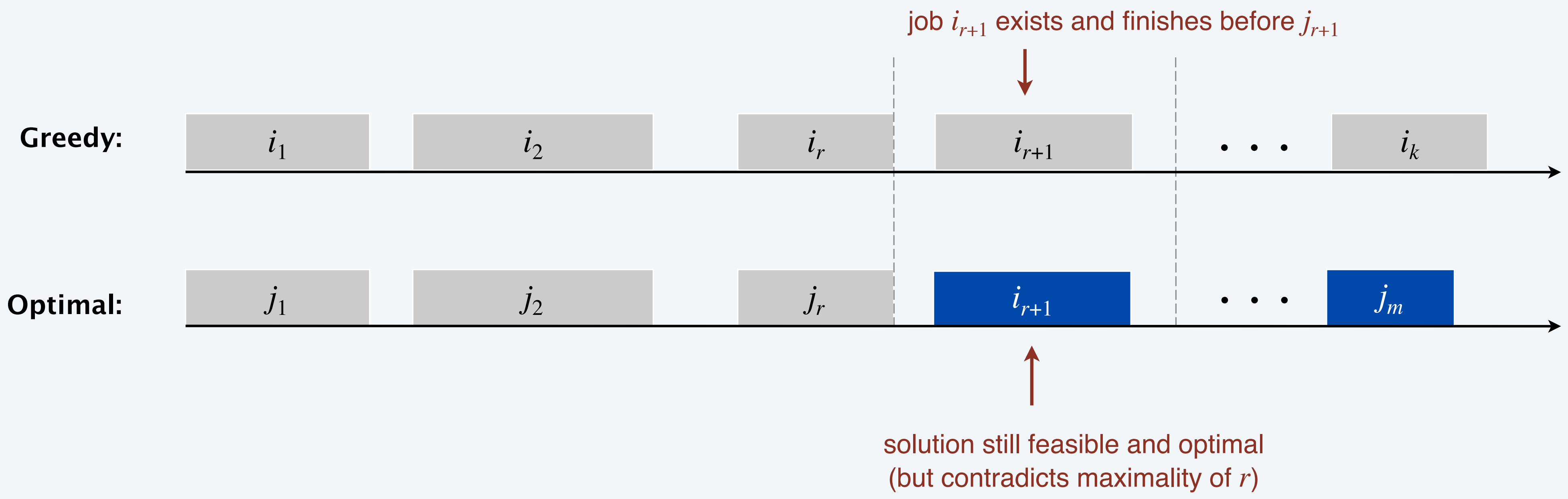

job i_{r+1} exists and finishes no later than j_{r+1}
Greedy:
i_1
i_2
i_r
i_{r+1}
$\cdots$
i_k
Optimal:
j_1
j_2
j_r
j_{r+1}
$\cdots$
j_m
job j_{r+1} exists because $m > k$
why not replace job j_{r+1} with job i_{r+1}?

Interval scheduling: analysis of earliest-finish-time-first algorithm

Theorem. The earliest-finish-time-first algorithm is optimal.

Pf. [by contradiction]

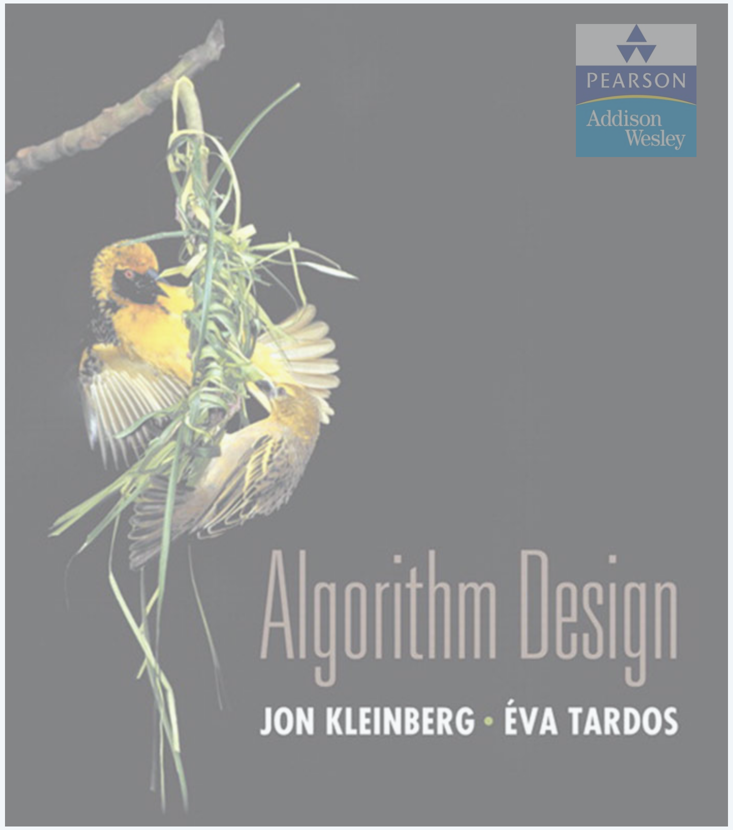
- Assume greedy is not optimal, and let's see what happens.
- Let $i_1, i_2, \ldots\, i_k$ denote set of jobs selected by greedy.
- Let $j_1, j_2, \ldots\, j_m$ denote set of jobs in an optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of r.


job i_{r+1} exists and finishes before j_{r+1}
Greedy:
i_1
i_2
i_r
i_{r+1}
. . .
i_k
Optimal:
j_1
j_2
j_r
i_{r+1}
. . .
j_m
solution still feasible and optimal
(but contradicts maximality of r)

Greedy algorithms I: quiz 3

Suppose that each job also has a positive weight and the goal is to find a maximum weight subset of mutually compatible intervals. Is the earliest-finish-time-first algorithm still optimal?

- **A.** Yes, because greedy algorithms are always optimal.
- **B.** Yes, because the same proof of correctness is valid.
- **C.** No, because the same proof of correctness is no longer valid.
- **D.** No, because you could assign a huge weight to a job that overlaps the job with the earliest finish time.



SECTION 4.1

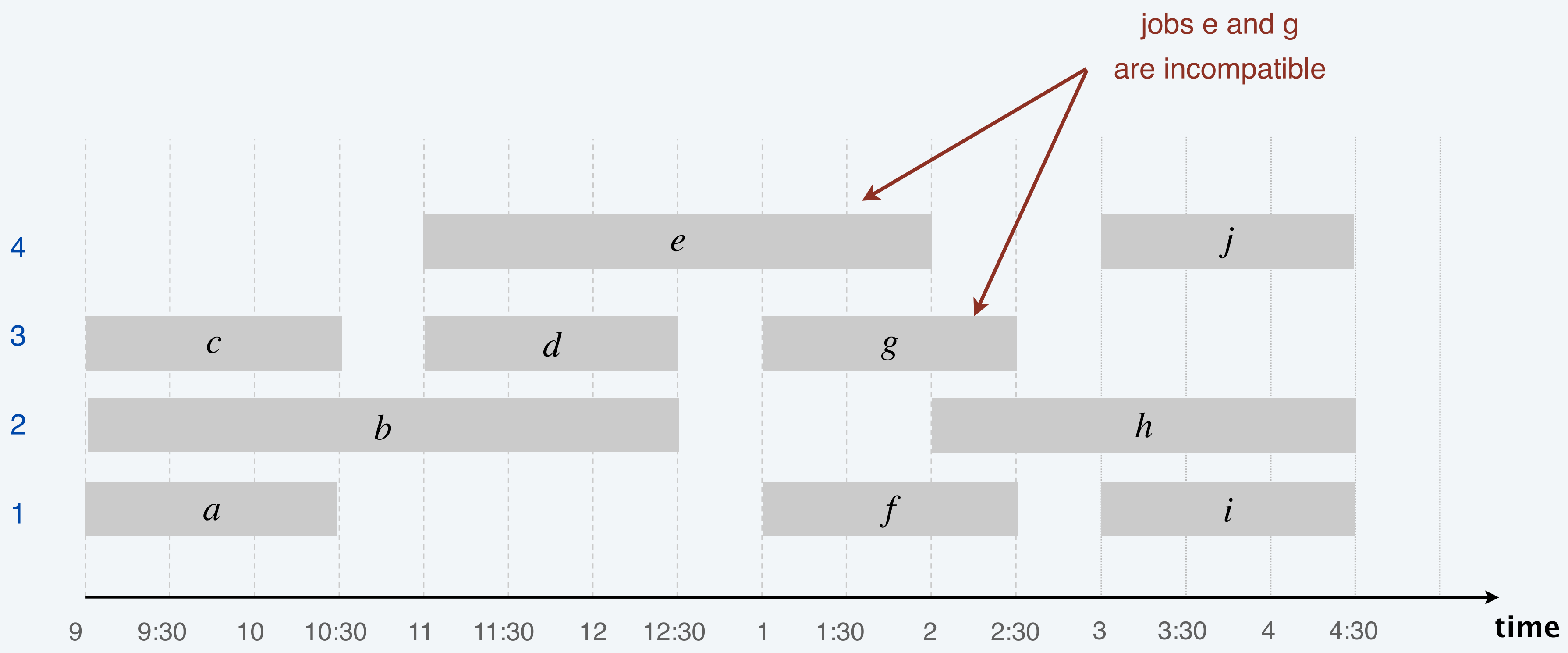
4. GREEDY ALGORITHMS I

- *coin changing*
- *interval scheduling*
- *interval partitioning*
- *scheduling to minimize lateness*
- *optimal caching*
- *stable matching*

Interval partitioning

- Lecture j starts at s_j and finishes at f_j.
- Goal: find minimum number of classrooms to schedule all lectures so that no two lectures occur at the same time in the same room.

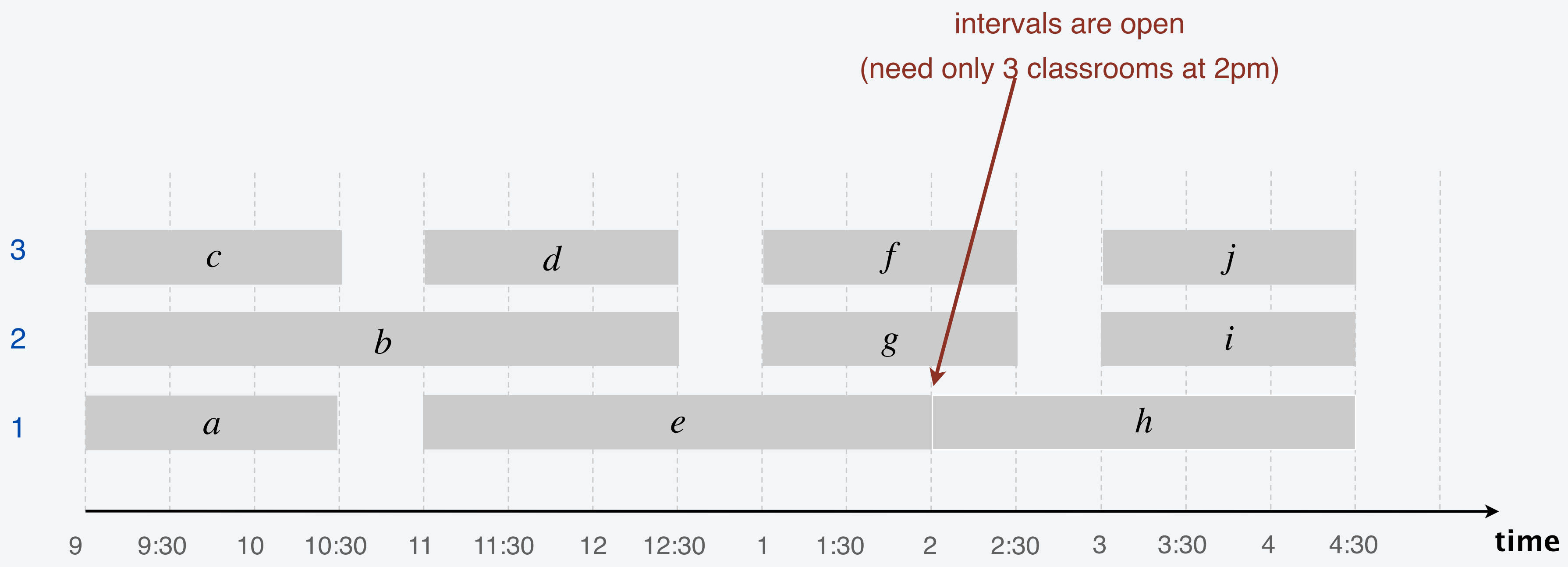
Ex. This schedule uses 4 classrooms to schedule 10 lectures.



Interval partitioning

- Lecture j starts at s_j and finishes at f_j.
- Goal: find minimum number of classrooms to schedule all lectures so that no two lectures occur at the same time in the same room.

Ex. This schedule uses 3 classrooms to schedule 10 lectures.



Greedy algorithms I: quiz 4

Consider lectures in some order, assigning each lecture to first available classroom (opening a new classroom if none is available). Which rule is optimal?

A. [Earliest start time] Consider lectures in ascending order of s_j.

B. [Earliest finish time] Consider lectures in ascending order of f_j.

C. [Shortest interval] Consider lectures in ascending order of $f_j - s_j$.

D. None of the above.

Interval partitioning: earliest-start-time-first algorithm

```
EARLIEST-START-TIME-FIRST ($n$, $s_1$, $s_2$, …, $s_n$ , $f_1$, $f_2$, …, $f_n$)

SORT lectures by start times and renumber so that $s_1 \le s_2 \le \ldots \le s_n$.
$d \leftarrow 0$.   ⟵   number of allocated classrooms
FOR $j = 1$ TO $n$
    IF (lecture $j$ is compatible with some classroom)
        Schedule lecture $j$ in any such classroom $k$.
    ELSE
        Allocate a new classroom $d + 1$.
        Schedule lecture $j$ in classroom $d + 1$.
        $d \leftarrow d + 1$.
RETURN schedule.
```

Interval partitioning: earliest-start-time-first algorithm

Proposition. The earliest-start-time-first algorithm can be implemented in $O(n \log n)$ time.

Pf.

- Sorting by start times takes $O(n \log n)$ time.
- Store classrooms in a priority queue (key = finish time of its last lecture).
  - to allocate a new classroom, INSERT classroom onto priority queue.
  - to schedule lecture j in classroom k, INCREASE-KEY of classroom k to f_j.
  - to determine whether lecture j is compatible with any classroom, compare s_j to FIND-MIN
- Total # of priority queue operations is $O(n)$; each takes $O(\log n)$ time. ▪

Remark. This implementation chooses a classroom k whose finish time of its last lecture is the earliest.

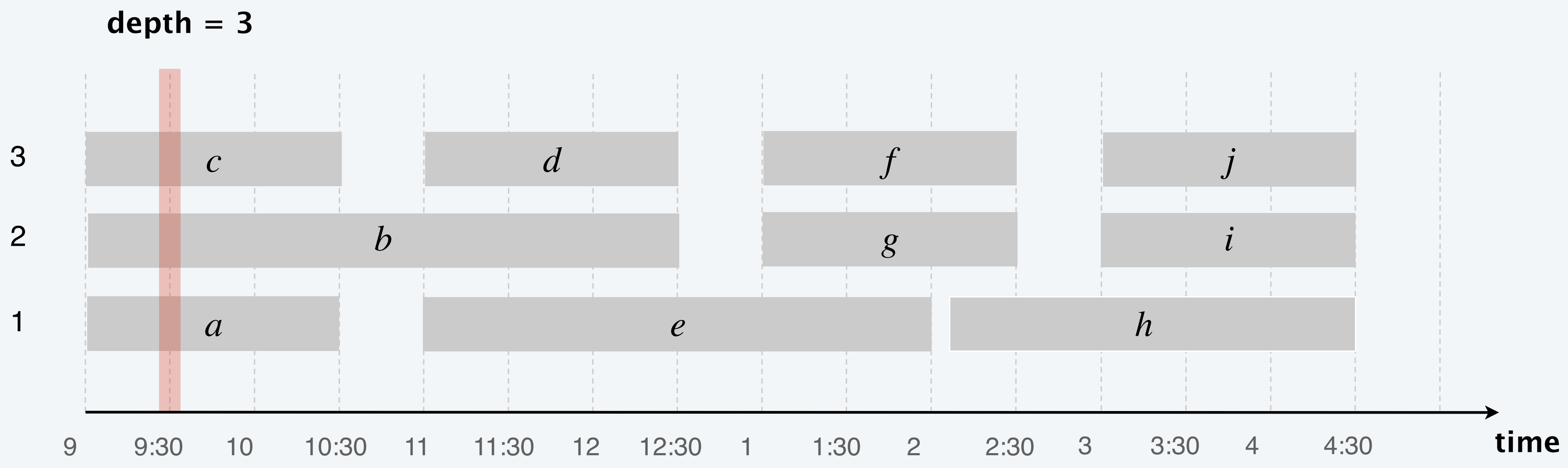
Interval partitioning: lower bound on optimal solution

Def. The depth of a set of open intervals is the maximum number of intervals that contain any given point.

Key observation. Number of classrooms needed ≥ depth.

Q. Does minimum number of classrooms needed always equal depth?

A. Yes! Moreover, earliest-start-time-first algorithm finds a schedule whose number of classrooms equals the depth.

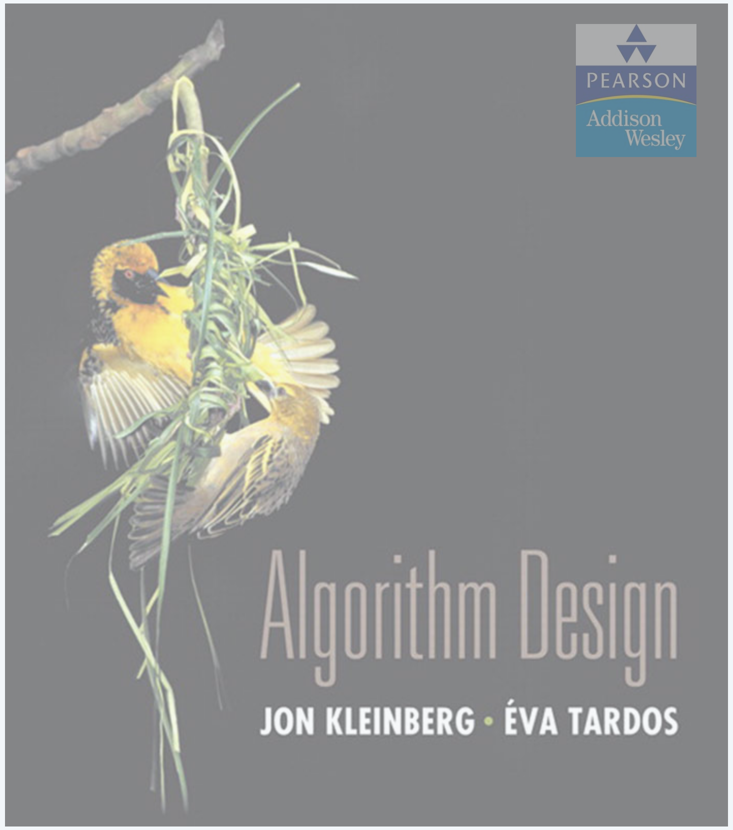


Interval partitioning: analysis of earliest-start-time-first algorithm

Observation. The earliest-start-time first algorithm never schedules two incompatible lectures in the same classroom.

Theorem. Earliest-start-time-first algorithm is optimal.
Pf.

- Let d = number of classrooms that the algorithm allocates.
- Classroom d is opened because we needed to schedule a lecture, say j, that is incompatible with a lecture in each of $d - 1$ other classrooms.
- Thus, these d lectures each end after s_j.
- Since we sorted by start time, each of these incompatible lectures start no later than s_j.
- Thus, we have d lectures overlapping at time $s_j + \varepsilon$.
- Key observation $\Rightarrow$ all schedules use $\geq d$ classrooms. ▪



SECTION 4.2

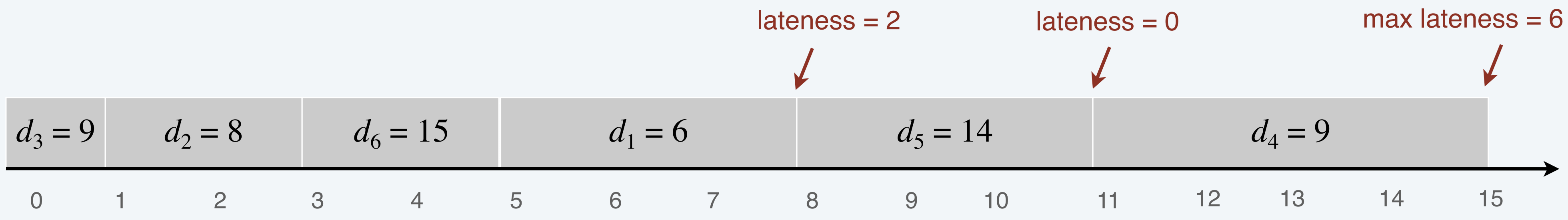
4. GREEDY ALGORITHMS I

- *coin changing*
- *interval scheduling*
- *interval partitioning*
- *scheduling to minimize lateness*
- *optimal caching*
- *stable matching*

Scheduling to minimizing lateness

- Single resource processes one job at a time.
- Job j requires t_j units of processing time and is due at time d_j.
- If j starts at time s_j, it finishes at time $f_j = s_j + t_j$.
- Lateness: $\ell_j = \max \{ 0, f_j - d_j \}$.
- Goal: schedule all jobs to minimize maximum lateness $L = \max_j \ell_j$.

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Greedy algorithms I: quiz 5

Schedule jobs according to some natural order. Which order minimizes the maximum lateness?

A. [shortest processing time] Ascending order of processing time t_j.

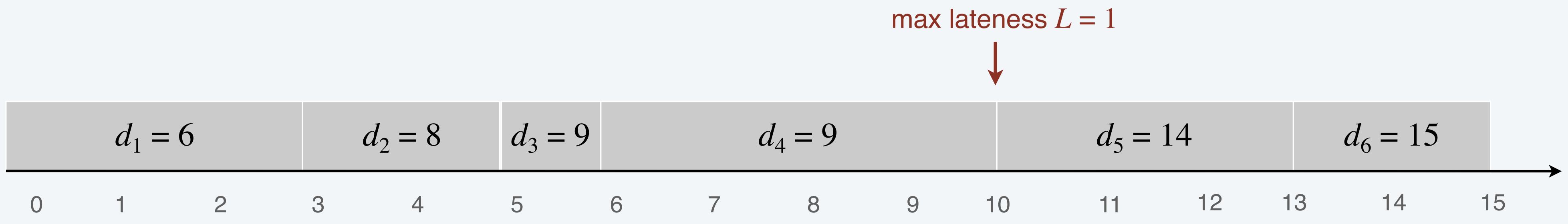
B. [earliest deadline first] Ascending order of deadline d_j.

C. [smallest slack] Ascending order of slack: $d_j - t_j$.

D. None of the above.

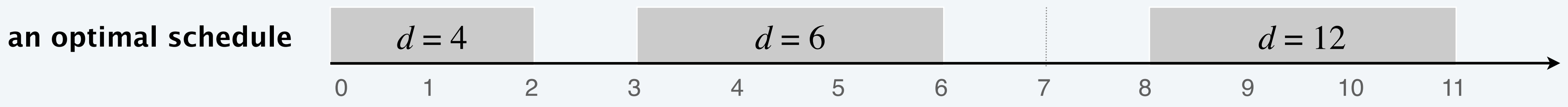
Minimizing lateness: earliest deadline first

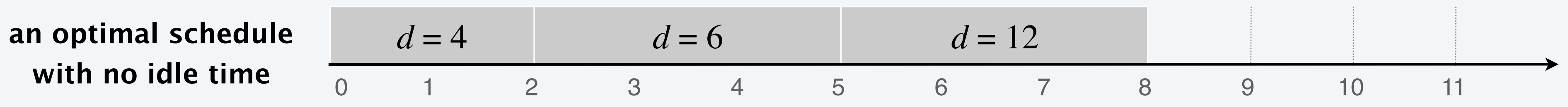
```
EARLIEST-DEADLINE-FIRST ($n, t_1, t_2, \ldots, t_n, d_1, d_2, \ldots, d_n$)

SORT jobs by due times and renumber so that $d_1 \leq d_2 \leq \ldots \leq d_n$.
$t \leftarrow 0$.
FOR $j = 1$ TO $n$
    Assign job $j$ to interval $[t, t + t_j]$.
    $s_j \leftarrow t$ ;  $f_j \leftarrow t + t_j$.
    $t \leftarrow t + t_j$.
RETURN intervals $[s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n]$.
```



Minimizing lateness: no idle time

Observation 1. There exists an optimal schedule with no idle time.

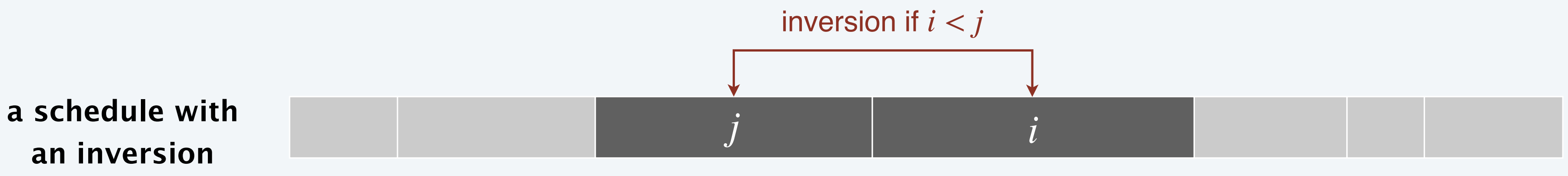




Observation 2. The earliest-deadline-first schedule has no idle time.

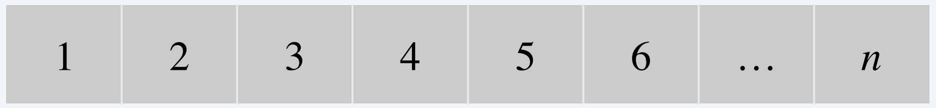
Minimizing lateness: inversions

Def. Given a schedule S, an inversion is a pair of jobs i and j such that: $i < j$ but j is scheduled before i.



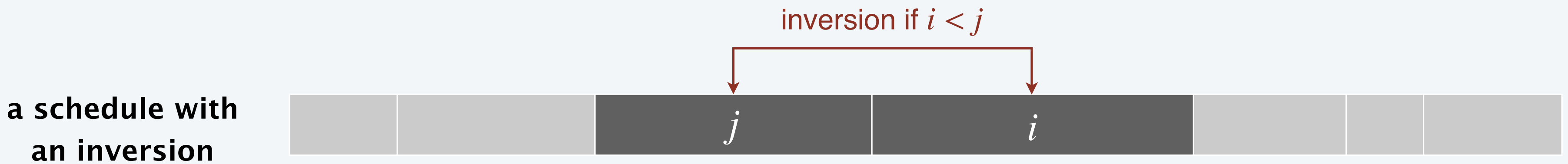
recall: we assume the jobs are numbered so that $d_1 \le d_2 \le \ldots \le d_n$

Observation 3. The earliest-deadline-first schedule is the unique idle-free schedule with no inversions.



Minimizing lateness: inversions

Def. Given a schedule S, an inversion is a pair of jobs i and j such that: $i < j$ but j is scheduled before i.



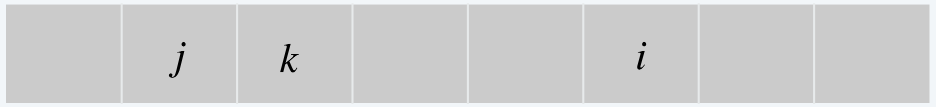
recall: we assume the jobs are numbered so that $d_1 \leq d_2 \leq \ldots \leq d_n$

Observation 4. If an idle-free schedule has an inversion, then it has an adjacent inversion.

two inverted jobs scheduled consecutively

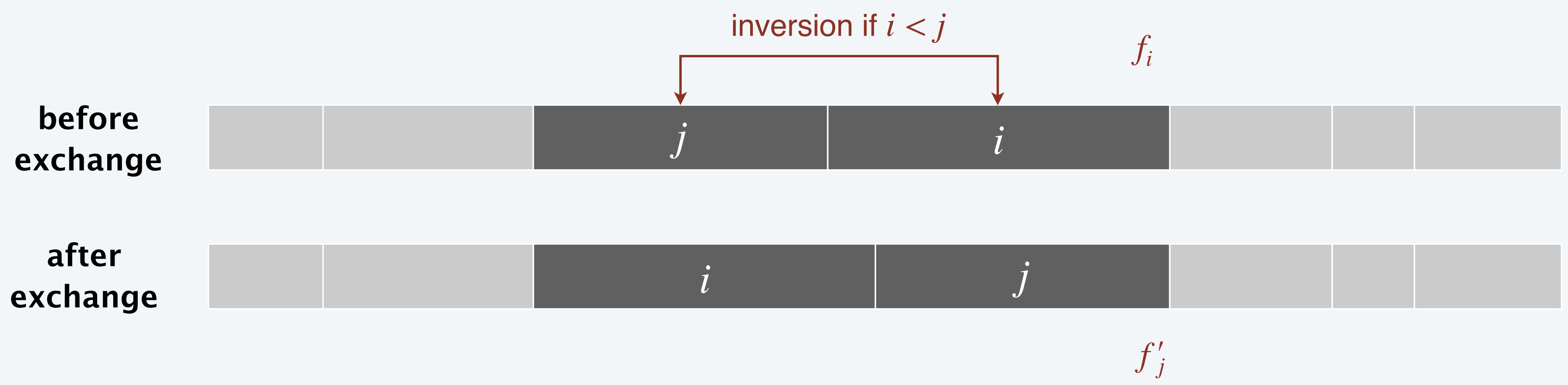
Pf.

- Let $i– j$ be a closest inversion.
- Let k be element immediately to the right of j.
- Case 1. $[\, j > k \,]$ Then $j–k$ is an adjacent inversion.
- Case 2. $[\, j < k \,]$ Then $i–k$ is a closer inversion since $i < j < k$. ※



Minimizing lateness: inversions

Def. Given a schedule S, an inversion is a pair of jobs i and j such that: $i < j$ but j is scheduled before i.



Key claim. Exchanging two adjacent, inverted jobs i and j reduces the number of inversions by 1 and does not increase the max lateness.

Pf. Let ℓ be the lateness before the swap, and let ℓ' be it afterwards.

- $\ell'_k = \ell_k$ for all $k \neq i, j$.
- $\ell'_i \leq \ell_i$.
- If job j is late, ℓ'_j

$$
\begin{aligned}
&= f'_j - d_j && \longleftarrow \text{definition} \\
&= f_i - d_j && \longleftarrow j \text{ now finishes at time } f_i \\
&\leq f_i - d_i && \longleftarrow i < j \;\Rightarrow\; d_i \leq d_j \\
&\leq \ell_i\,. && \longleftarrow \text{definition}
\end{aligned}
$$

Minimizing lateness: analysis of earliest-deadline-first algorithm

Theorem. The earliest-deadline-first schedule S is optimal.

optimal schedule can have inversions

Pf. [by contradiction]

Define S^* to be an optimal schedule with the fewest inversions.

- Can assume S^* has no idle time. $\longleftarrow$ Observation 1
- Case 1. [S^* has no inversions] Then $S = S^*$. $\longleftarrow$ Observation 3
- Case 2. [S^* has an inversion]
  - let i–j be an adjacent inversion $\longleftarrow$ Observation 4
  - exchanging jobs i and j decreases the number of inversions by 1 without increasing the max lateness $\longleftarrow$ key claim
  - contradicts "fewest inversions" part of the definition of S^* ※

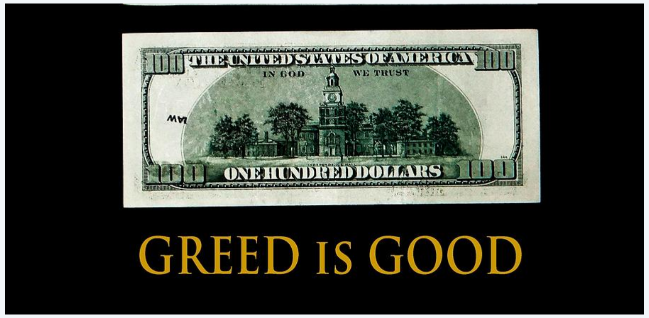
Greedy analysis strategies

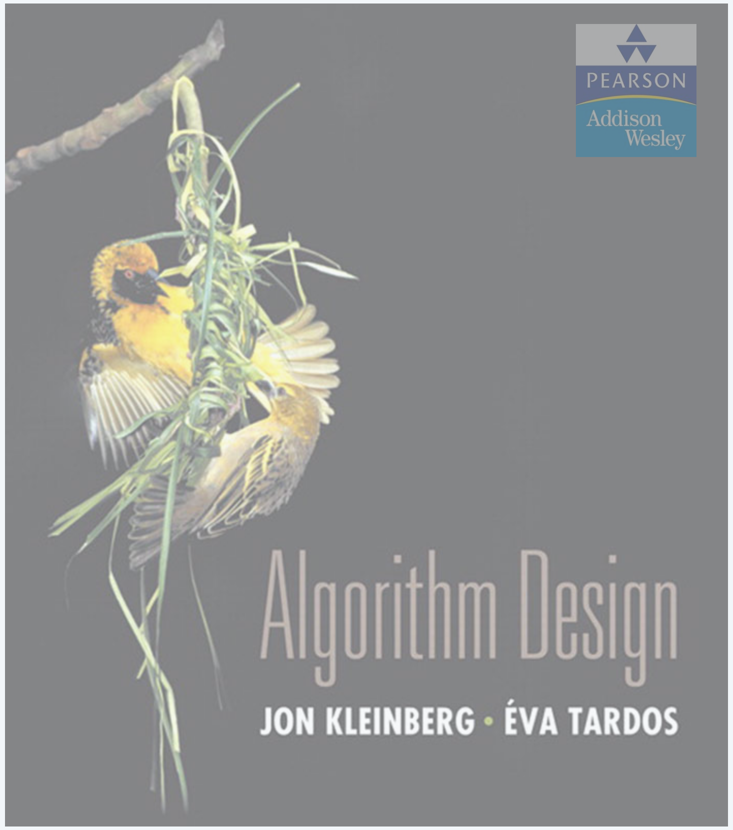
Greedy algorithm stays ahead. Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.

Structural. Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound.

Exchange argument. Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.

Other greedy algorithms. Gale–Shapley, Kruskal, Prim, Dijkstra, Huffman, …





SECTION 4.3

4. GREEDY ALGORITHMS I

- *coin changing*
- *interval scheduling*
- *interval partitioning*
- *scheduling to minimize lateness*
- *optimal caching*
- *stable matching*

Optimal offline caching

Caching.

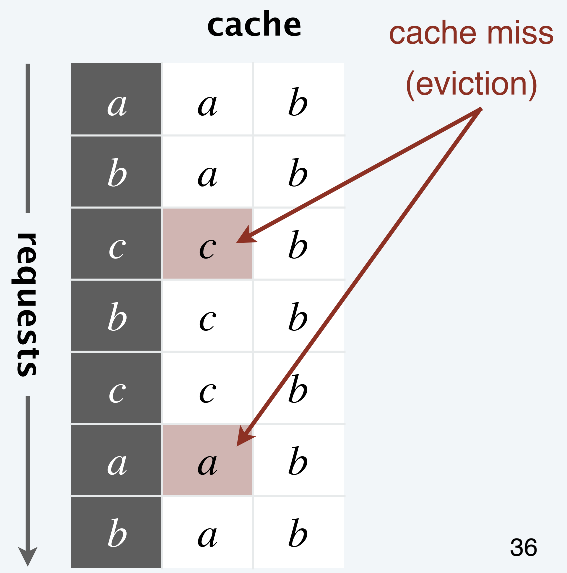
- Cache with capacity to store k items.
- Sequence of m item requests $d_1, d_2, \ldots, d_m$.
- Cache hit: item in cache when requested.
- Cache miss: item not in cache when requested.
  (must evict some item from cache and bring requested item into cache)

Applications. CPU, RAM, hard drive, web, browser, ….

Goal. Eviction schedule that minimizes the number of evictions.

Ex. $k = 2$, initial cache = ab, requests: $a, b, c, b, c, a, b.$

Optimal eviction schedule. 2 evictions.

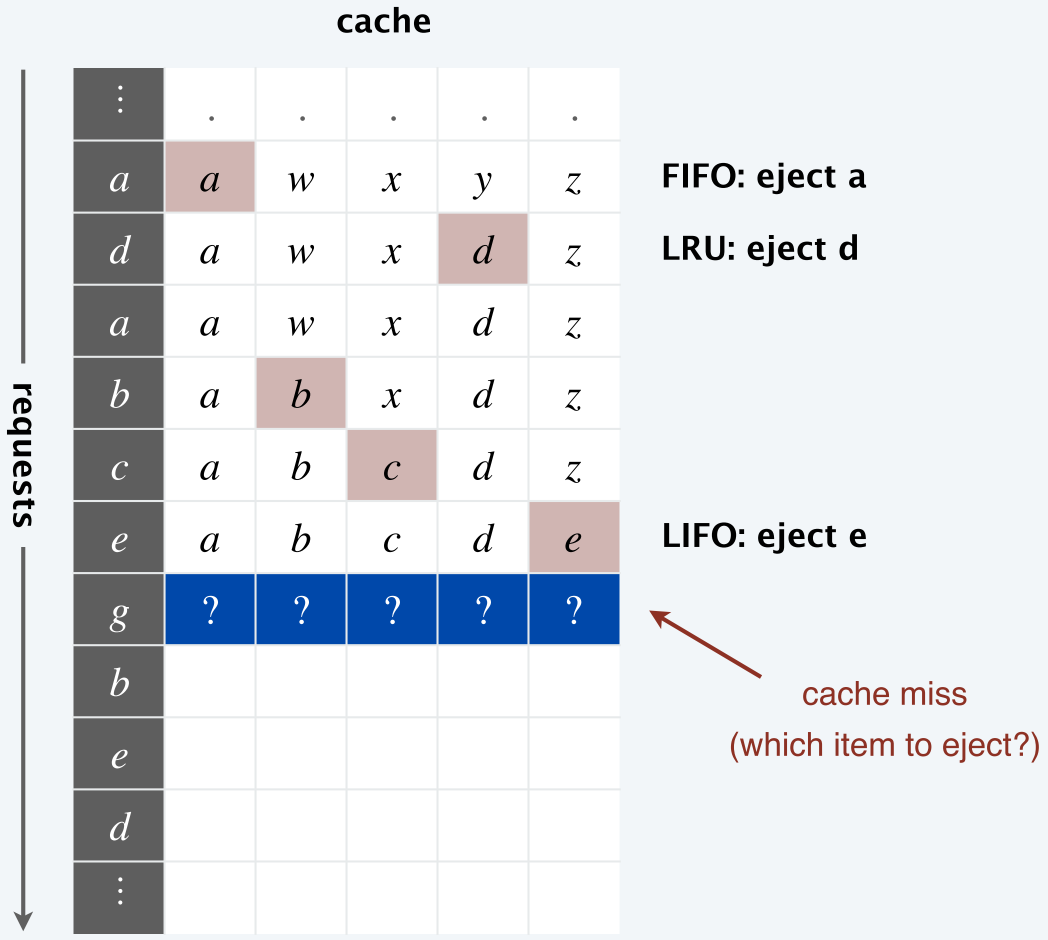


Optimal offline caching: greedy algorithms

LIFO/FIFO. Evict item brought in least (most) recently.

LRU. Evict item whose most recent access was earliest.

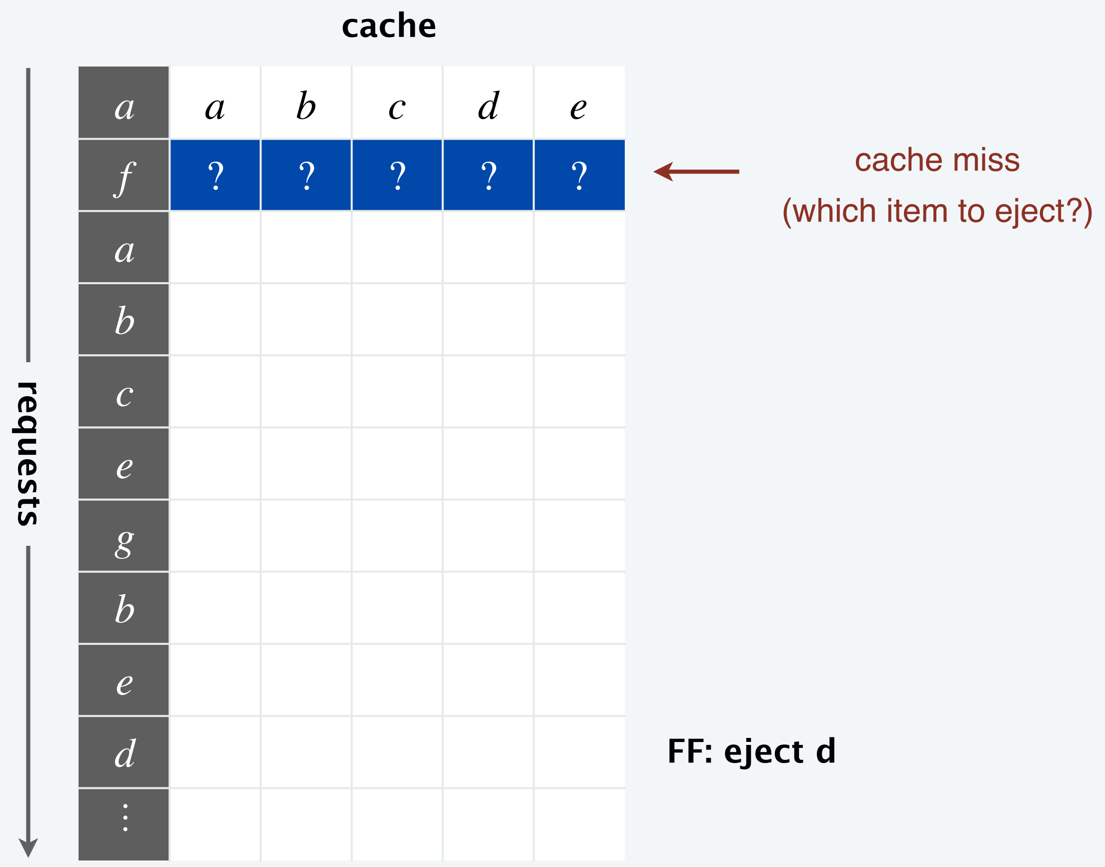
LFU. Evict item that was least frequently requested.



requests	cache				
⋮	.	.	.	.	.
a	a	w	x	y	z
d	a	w	x	d	z
a	a	w	x	d	z
b	a	b	x	d	z
c	a	b	c	d	z
e	a	b	c	d	e
g	?	?	?	?	?
b					
e					
d					
⋮					

Optimal offline caching: farthest-in-future (clairvoyant algorithm)

Farthest-in-future. Evict item in the cache that is not requested until farthest in the future.



Theorem. [Bélády 1966] FF is optimal eviction schedule.

Pf. Algorithm and theorem are intuitive; proof is subtle.

Greedy algorithms I: quiz 6

Which item will be evicted next using farthest-in-future schedule?

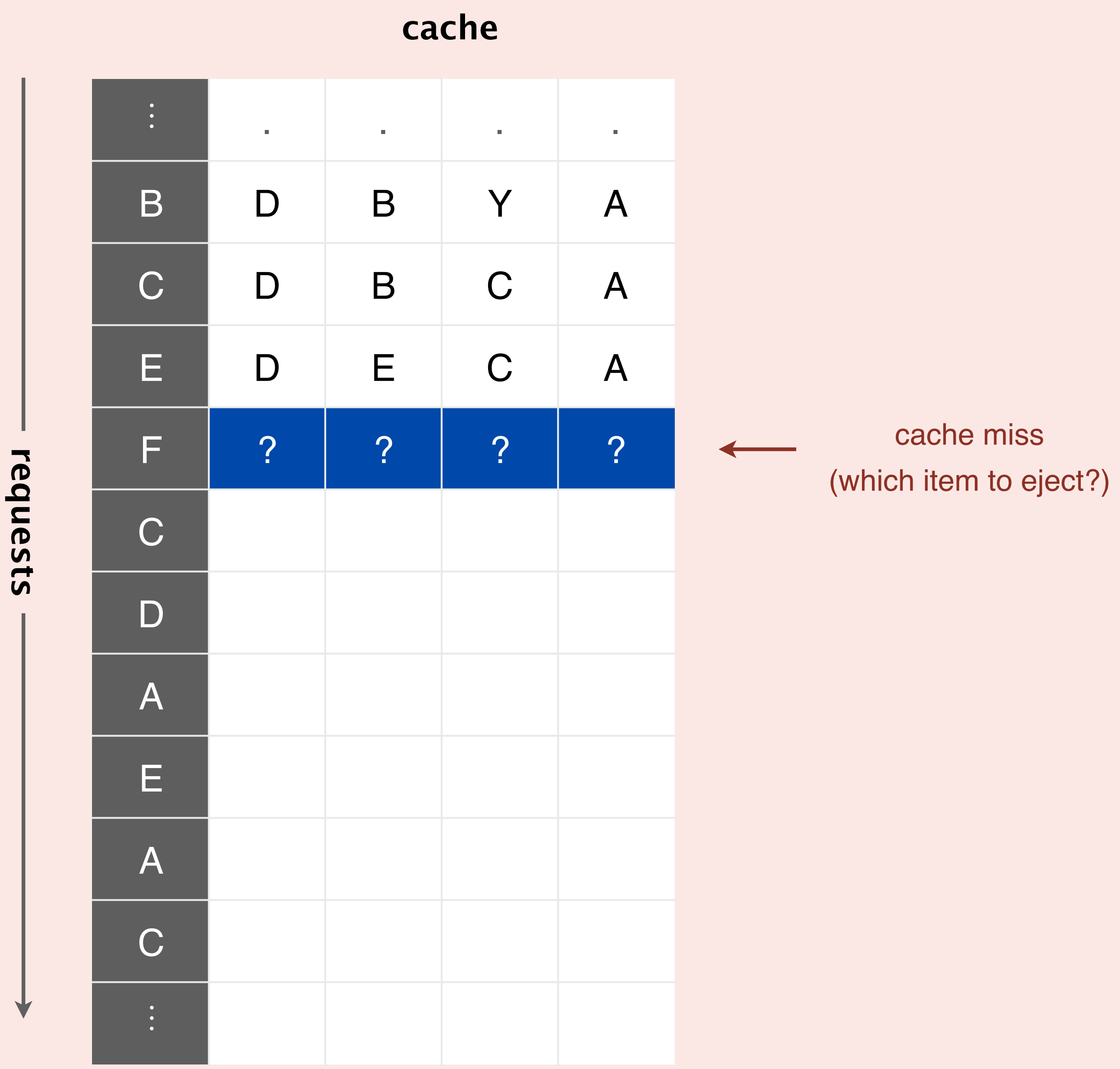
A.

B.

C.

D.

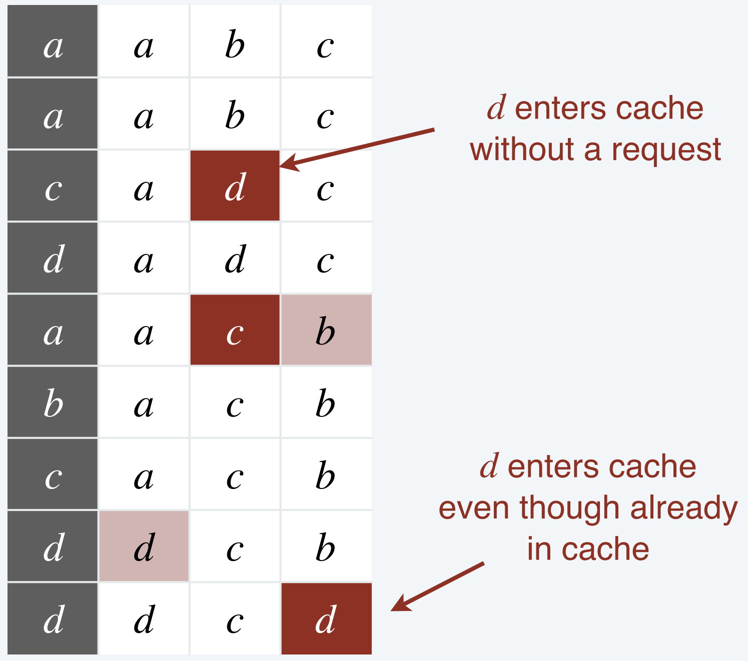
E.



requests	cache			
⋮	.	.	.	.
B	D	B	Y	A
C	D	B	C	A
E	D	E	C	A
F	?	?	?	?
C				
D				
A				
E				
A				
C				
⋮				

Reduced eviction schedules

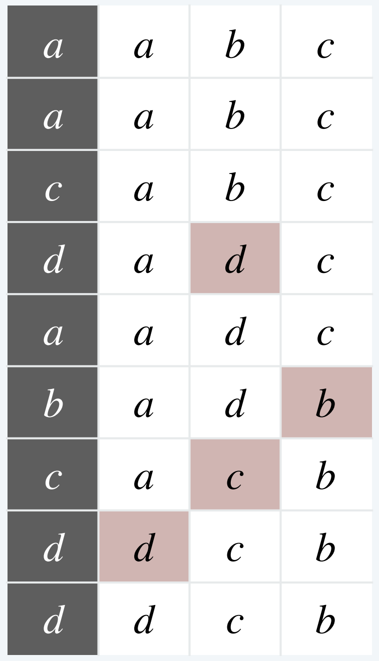
Def. A reduced schedule is a schedule that brings an item d into the cache in step j only if there is a request for d in step j and d is not already in the cache.



d enters cache without a request
d enters cache even though already in cache

a	a	b	c
a	a	b	c
c	a	d	c
d	a	d	c
a	a	c	b
b	a	c	b
c	a	c	b
d	d	c	b
d	d	c	d

an unreduced schedule



a	a	b	c
a	a	b	c
c	a	b	c
d	a	d	c
a	a	d	c
b	a	d	b
c	a	c	b
d	d	c	b
d	d	c	b

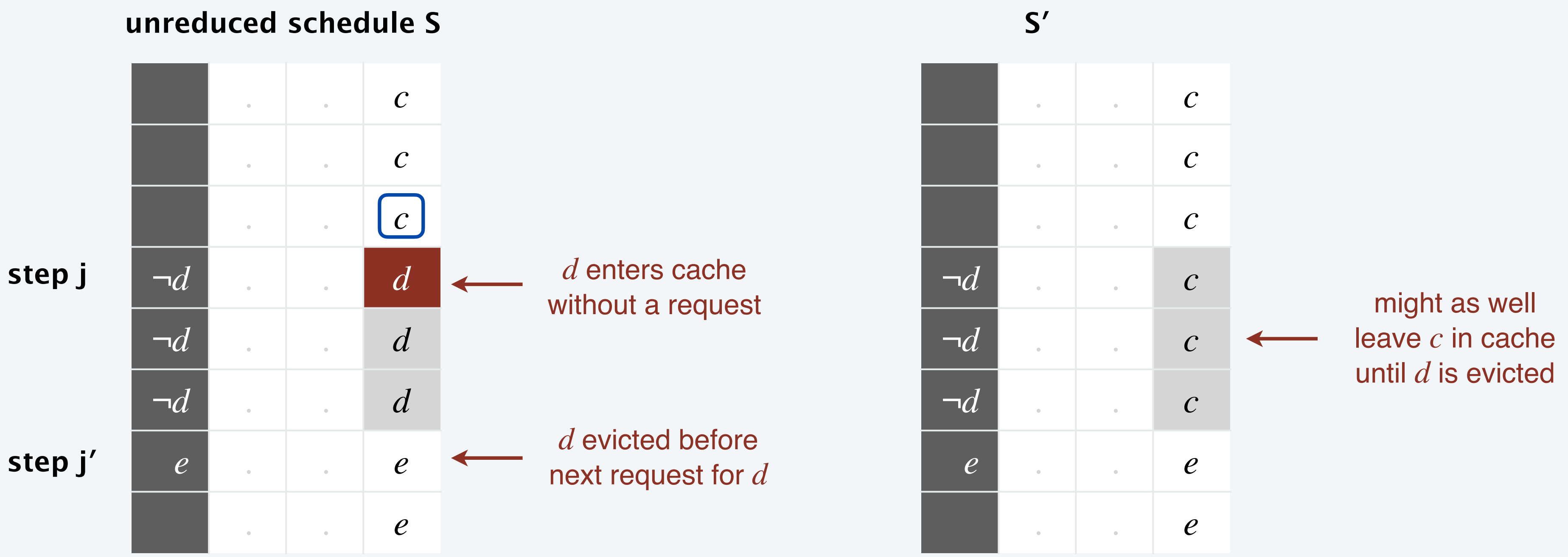
a reduced schedule

Reduced eviction schedules

Claim. Given any unreduced schedule S, can transform it into a reduced schedule S' with no more evictions.

Pf. [by induction on number of steps j]

- Suppose S brings d into the cache in step j without a request.
- Let c be the item S evicts when it brings d into the cache.
- Case 1a: d evicted before next request for d.



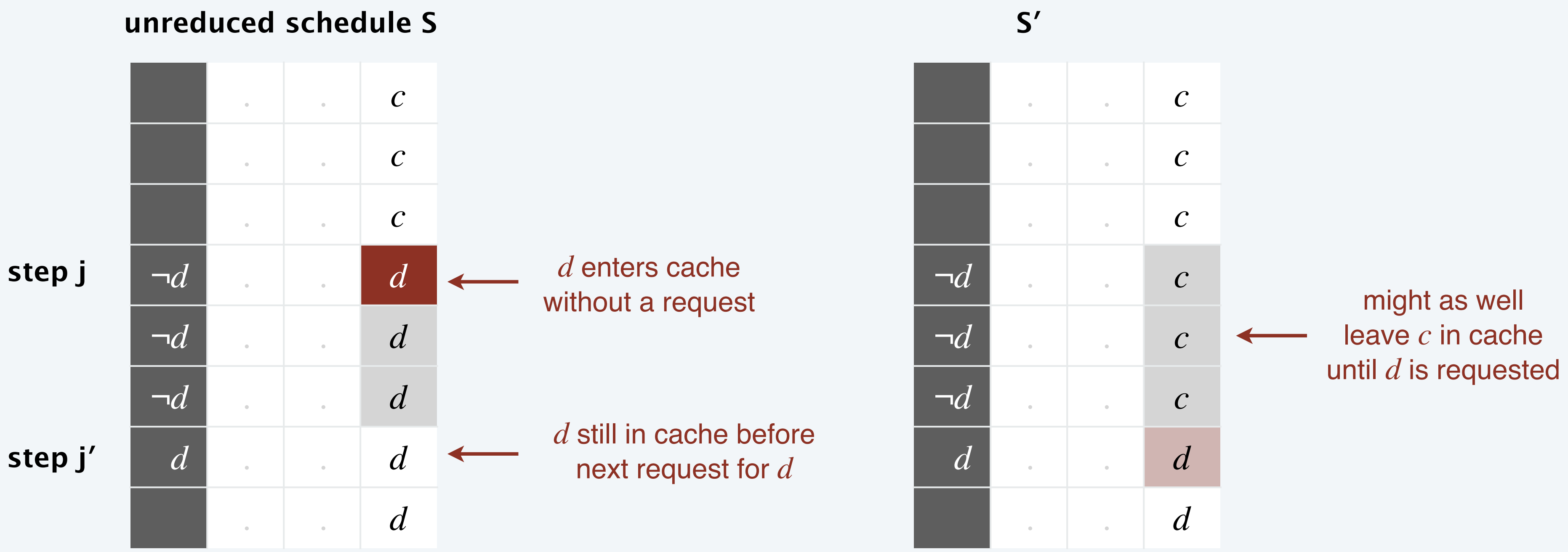
unreduced schedule S
S′
step j
step j′
d enters cache without a request
d evicted before next request for d
might as well leave c in cache until d is evicted

Reduced eviction schedules

Claim. Given any unreduced schedule S, can transform it into a reduced schedule S' with no more evictions.

Pf. [by induction on number of steps j]

- Suppose S brings d into the cache in step j without a request.
- Let c be the item S evicts when it brings d into the cache.
- Case 1a: d evicted before next request for d.
- Case 1b: next request for d occurs before d is evicted.



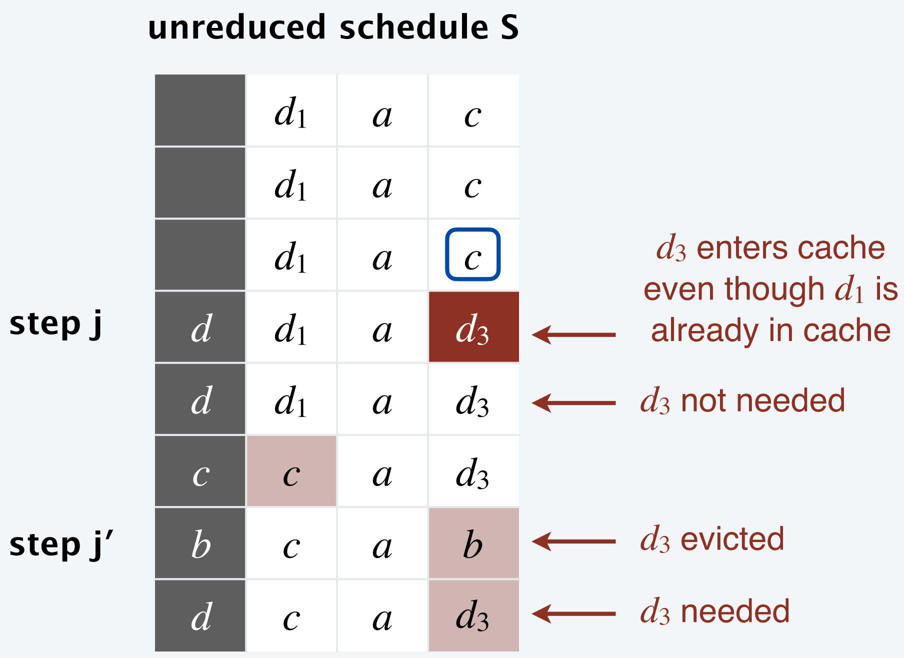
unreduced schedule S
S′
step j
step j′
¬d
d
c
d enters cache without a request
d still in cache before next request for d
might as well leave c in cache until d is requested

Reduced eviction schedules

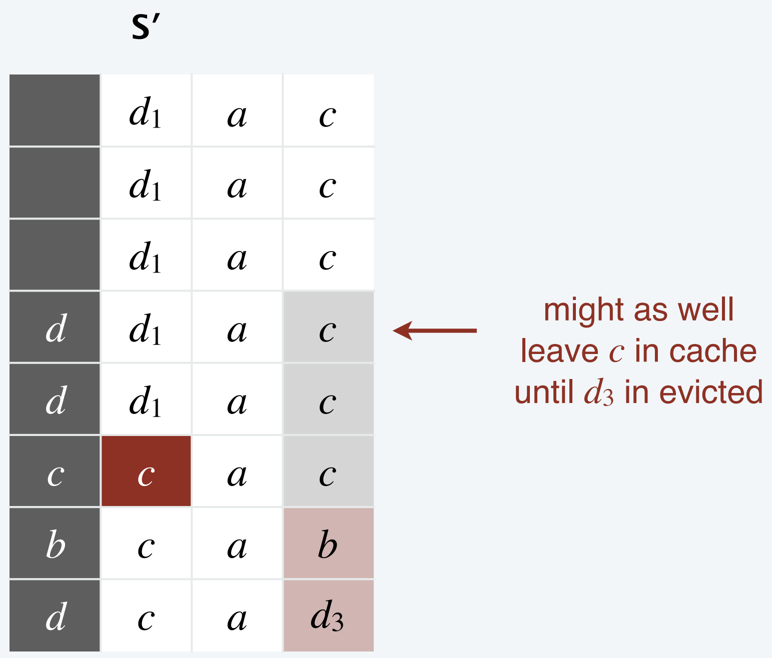
Claim. Given any unreduced schedule S, can transform it into a reduced schedule S' with no more evictions.

Pf. [by induction on number of steps j]

- Suppose S brings d into the cache in step j even though d is in cache.
- Let c be the item S evicts when it brings d into the cache.
- Case 2a: d evicted before it is needed.



		d_1	a	c
		d_1	a	c
		d_1	a	$\boxed{c}$
step j	d	d_1	a	d_3
	d	d_1	a	d_3
	c	c	a	d_3
step j′	b	c	a	b
	d	c	a	d_3



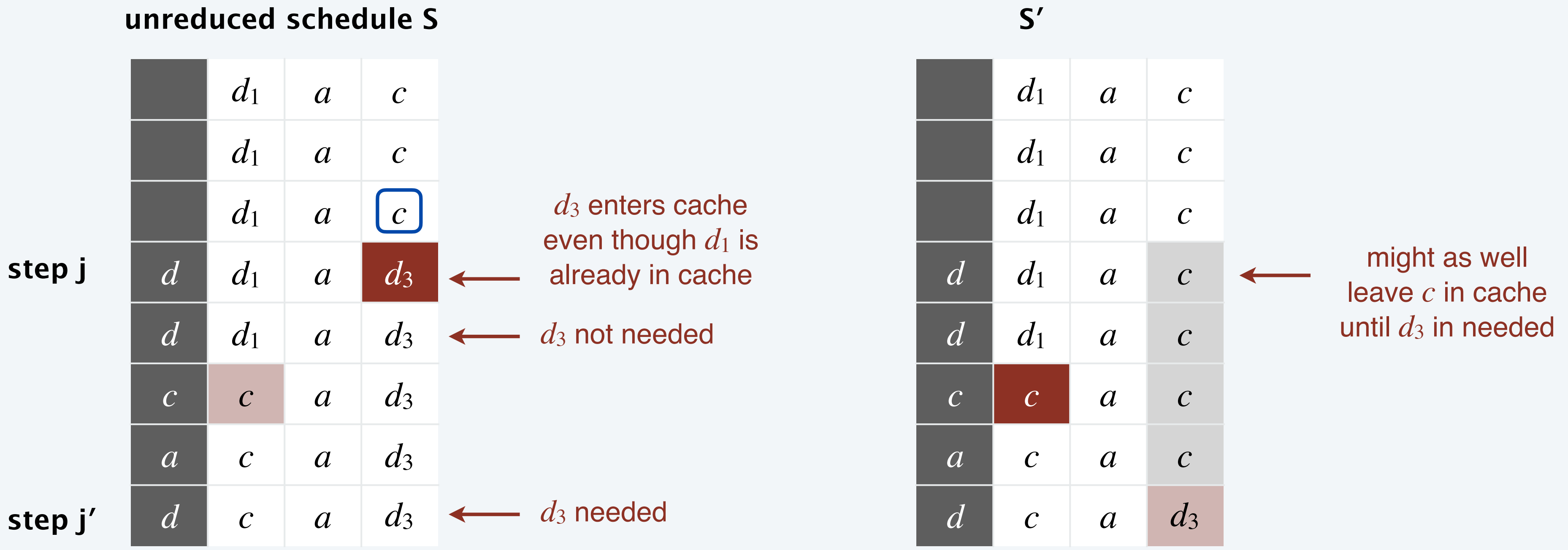
	d_1	a	c
	d_1	a	c
	d_1	a	c
d	d_1	a	c
d	d_1	a	c
c	c	a	c
b	c	a	b
d	c	a	d_3

Reduced eviction schedules

Claim. Given any unreduced schedule S, can transform it into a reduced schedule S' with no more evictions.

Pf. [by induction on number of steps j]

- Suppose S brings d into the cache in step j even though d is in cache.
- Let c be the item S evicts when it brings d into the cache.
- Case 2a: d evicted before it is needed.
- Case 2b: d needed before it is evicted.



Reduced eviction schedules

Claim. Given any unreduced schedule S, can transform it into a reduced schedule S' with no more evictions.

Pf. [by induction on number of steps j]

- Case 1: S brings d into the cache in step j without a request. ✔
- Case 2: S brings d into the cache in step j even though d is in cache. ✔
- If multiple unreduced items in step j, apply each one in turn, dealing with Case 1 before Case 2. ▪

resolving Case 1 might trigger Case 2

Farthest-in-future: analysis

Theorem. FF is optimal eviction algorithm.

Pf. Follows directly from the following invariant.

Invariant. There exists an optimal reduced schedule S that has the same eviction schedule as S_{FF} through the first j steps.

Pf. [by induction on number of steps j]

Base case: $j = 0$.

Let S be reduced schedule that satisfies invariant through j steps.

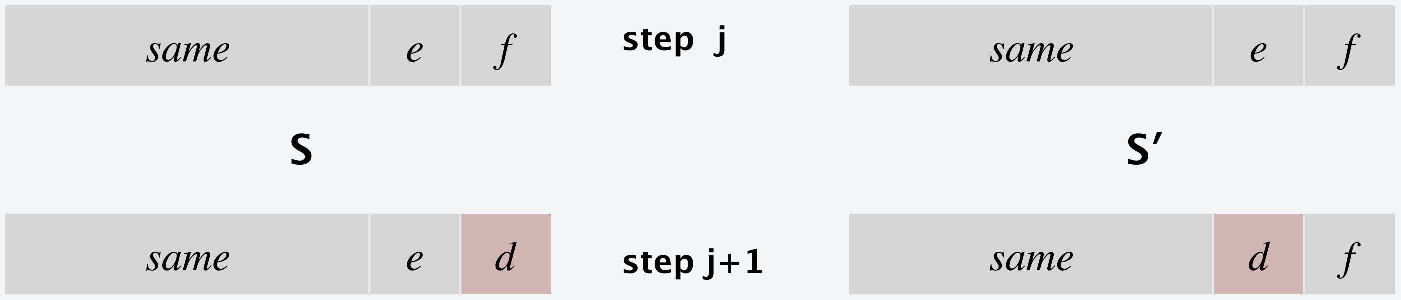
We produce S' that satisfies invariant after $j + 1$ steps.

- Let d denote the item requested in step $j + 1$.
- Since S and S_{FF} have agreed up until now, they have the same cache contents before step $j + 1$.
- Case 1: d is already in the cache.
  $S' = S$ satisfies invariant.
- Case 2: d is not in the cache and S and S_{FF} evict the same item.
  $S' = S$ satisfies invariant.

Farthest-in-future: analysis

Pf. [continued]

- Case 3: d is not in the cache; S_{FF} evicts e; S evicts $f \neq e$.
  - begin construction of S' from S by evicting e instead of f

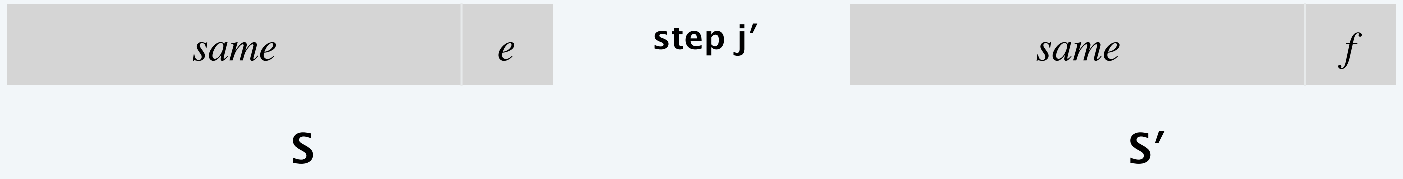


  - now S' agrees with S_{FF} for first $j + 1$ steps; we show that having item f in cache is no worse than having item e in cache
  - let S' behave the same as S until S' is forced to take a different action (because either S evicts e; or because either e or f is requested)

Farthest-in-future: analysis

Let j' be the first step after $j+1$ that S' must take a different action from S;
let g denote the item requested in step j'.

involves either e or f (or both)



- Case 3a: $g = e$.

  S' agrees with S_{FF} through first $j+1$ steps

  Can't happen with FF since there must be a request for f before e.

- Case 3b: $g = f$.

  Element f can't be in cache of S; let e' be the item that S evicts.
  - if $e' = e$, S' accesses f from cache; now S and S' have same cache
  - if $e' \neq e$, we make S' evict e' and bring e into the cache;
    now S and S' have the same cache

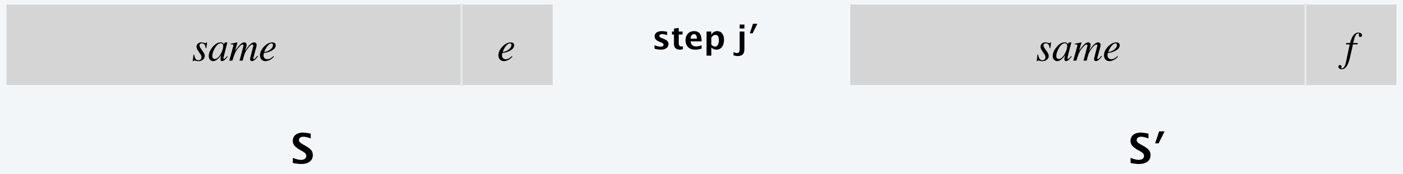
  We let S' behave exactly like S for remaining requests.

S' is no longer reduced, but can be transformed into a reduced schedule that agrees with FF through first $j+1$ steps

Farthest-in-future: analysis

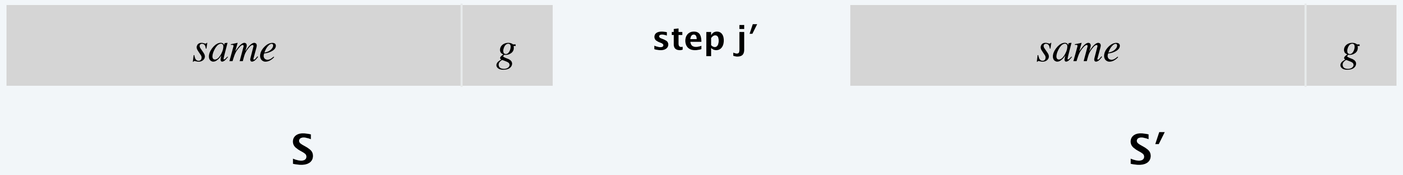
Let j' be the first step after $j+1$ that S' must take a different action from S; let g denote the item requested in step j'.

involves wither e or f (or both)



otherwise S' could have taken the same action

- Case 3c: $g \neq e, f$. S evicts e.
  - make S' evict f.



  - now S and S' have the same cache
  - let S' behave exactly like S for the remaining requests ▪

Caching perspective

Online vs. offline algorithms.

- Offline: full sequence of requests is known a priori.
- Online (reality): requests are not known in advance.
- Caching is among most fundamental online problems in CS.

LIFO. Evict item brought in most recently.

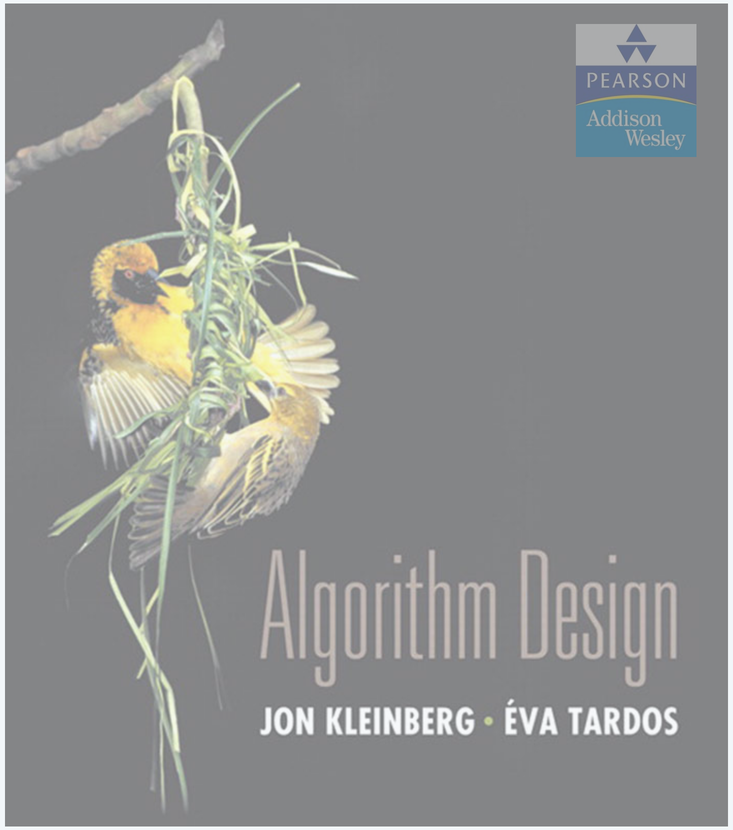
LRU. Evict item whose most recent access was earliest.

FF with direction of time reversed!

Theorem. FF is optimal offline eviction algorithm.

- Provides basis for understanding and analyzing online algorithms.
- LIFO can be arbitrarily bad.
- LRU is k-competitive: for any sequence of requests σ, $LRU(\sigma) \le k\, FF(\sigma) + k$.
- Randomized marking is $O(\log k)$-competitive.

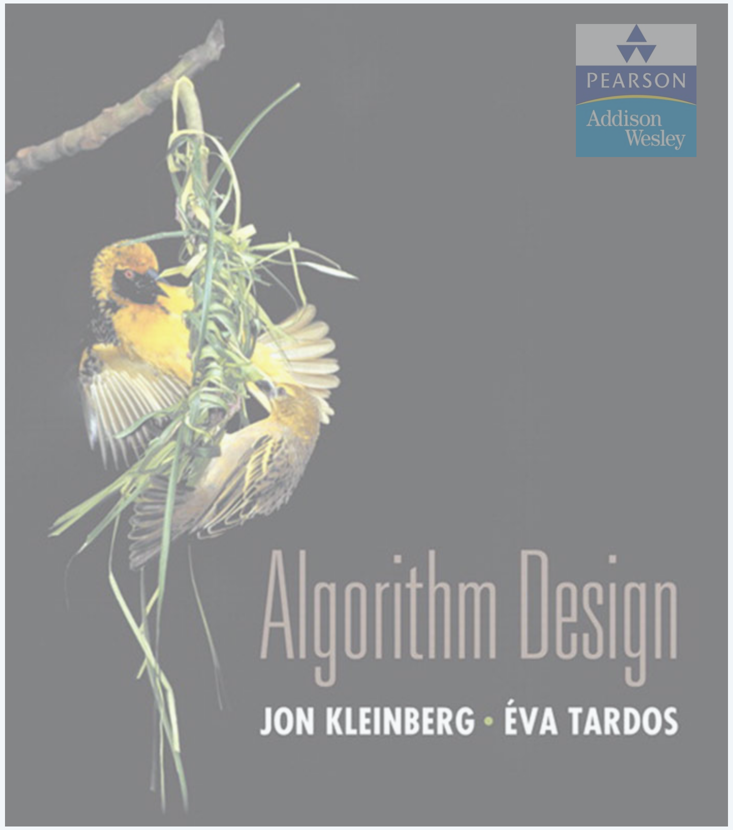
see Section 13.8



SECTION 1.1

4. GREEDY ALGORITHMS I

- *coin changing*
- *interval scheduling*
- *interval partitioning*
- *scheduling to minimize lateness*
- *optimal caching*
- *stable matching*



SECTION 1.1

1. STABLE MATCHING

- *stable matching problem*
- *Gale–Shapley algorithm*
- *hospital optimality*
- *context*

Matching med-school students to hospitals

Goal. Given a set of preferences among hospitals and med-school students, design a self-reinforcing admissions process.

Unstable pair. Hospital h and student s form an unstable pair if both:

- h prefers s to one of its admitted students.
- s prefers h to assigned hospital.

Stable assignment. Assignment with no unstable pairs.

- Natural and desirable condition.
- Individual self-interest prevents any hospital–student side deal.

Stable matching problem: input

Input. A set of n hospitals H and a set of n students S.

- Each hospital $h \in H$ ranks students.
- Each student $s \in S$ ranks hospitals.

one student per hospital (for now)

favorite ↓ least favorite ↓

	1st	2nd	3rd
Atlanta	Xavier	Yolanda	Zeus
Boston	Yolanda	Xavier	Zeus
Chicago	Xavier	Yolanda	Zeus

hospitals' preference lists

favorite ↓ least favorite ↓

	1st	2nd	3rd
Xavier	Boston	Atlanta	Chicago
Yolanda	Atlanta	Boston	Chicago
Zeus	Atlanta	Boston	Chicago

students' preference lists

Perfect matching

Def. A matching M is a set of ordered pairs $h\text{–}s$ with $h \in H$ and $s \in S$ s.t.

- Each hospital $h \in H$ appears in at most one pair of M.
- Each student $s \in S$ appears in at most one pair of M.

Def. A matching M is perfect if $|M| = |H| = |S| = n$.

	1st	2nd	3rd
Atlanta	Xavier	Yolanda	Zeus 🟪
Boston	Yolanda 🟪	Xavier	Zeus
Chicago	Xavier 🟪	Yolanda	Zeus

	1st	2nd	3rd
Xavier	Boston	Atlanta	Chicago 🟪
Yolanda	Atlanta	Boston 🟪	Chicago
Zeus	Atlanta 🟪	Boston	Chicago

a perfect matching M = { A–Z, B–Y, C–X }

Unstable pair

Def. Given a perfect matching M, hospital h and student s form an unstable pair if both:

- h prefers s to matched student.
- s prefers h to matched hospital.

Key point. An unstable pair h–s could each improve by joint action.

	1st	2nd	3rd
Atlanta	Xavier	Yolanda	Zeus
Boston	Yolanda	Xavier	Zeus
Chicago	Xavier	Yolanda	Zeus

	1st	2nd	3rd
Xavier	Boston	Atlanta	Chicago
Yolanda	Atlanta	Boston	Chicago
Zeus	Atlanta	Boston	Chicago

A–Y is an unstable pair for matching M = { A–Z, B–Y, C–X }

Stable matching: quiz 1

Which pair is unstable in the matching { A–X, B–Z, C–Y } ?

A. A–Y.

B. B–X.

C. B–Z.

D. None of the above.

	1st	2nd	3rd
Atlanta	Xavier	Yolanda	Zeus
Boston	Yolanda	Xavier	Zeus
Chicago	Xavier	Yolanda	Zeus

	1st	2nd	3rd
Xavier	Boston	Atlanta	Chicago
Yolanda	Atlanta	Boston	Chicago
Zeus	Atlanta	Boston	Chicago

Stable matching: quiz 1

Which pair is unstable in the matching { A–X, B–Z, C–Y } ?

A. A–Y.

B. B–X.

C. B–Z.

D. None of the above.

	1st	2nd	3rd
Atlanta	Xavier	Yolanda	Zeus
Boston	Yolanda	Xavier	Zeus
Chicago	Xavier	Yolanda	Zeus

	1st	2nd	3rd
Xavier	Boston	Atlanta	Chicago
Yolanda	Atlanta	Boston	Chicago
Zeus	Atlanta	Boston	Chicago

B–X is an unstable pair

Stable matching problem

Def. A stable matching is a perfect matching with no unstable pairs.

Stable matching problem. Given the preference lists of n hospitals and n students, find a stable matching (if one exists).

	1st	2nd	3rd
Atlanta	Xavier	Yolanda	Zeus
Boston	Yolanda	Xavier	Zeus
Chicago	Xavier	Yolanda	Zeus

	1st	2nd	3rd
Xavier	Boston	Atlanta	Chicago
Yolanda	Atlanta	Boston	Chicago
Zeus	Atlanta	Boston	Chicago

a stable matching M = { A–X, B–Y, C–Z }

Stable roommate problem

Q. Do stable matchings always exist?

A. Not obvious a priori.

Stable roommate problem.

- $2n$ people; each person ranks others from 1 to $2n - 1$.
- Assign roommate pairs so that no unstable pairs.

	1st	2nd	3rd
A	B	C	D
B	C	A	D
C	A	B	D
D	A	B	C

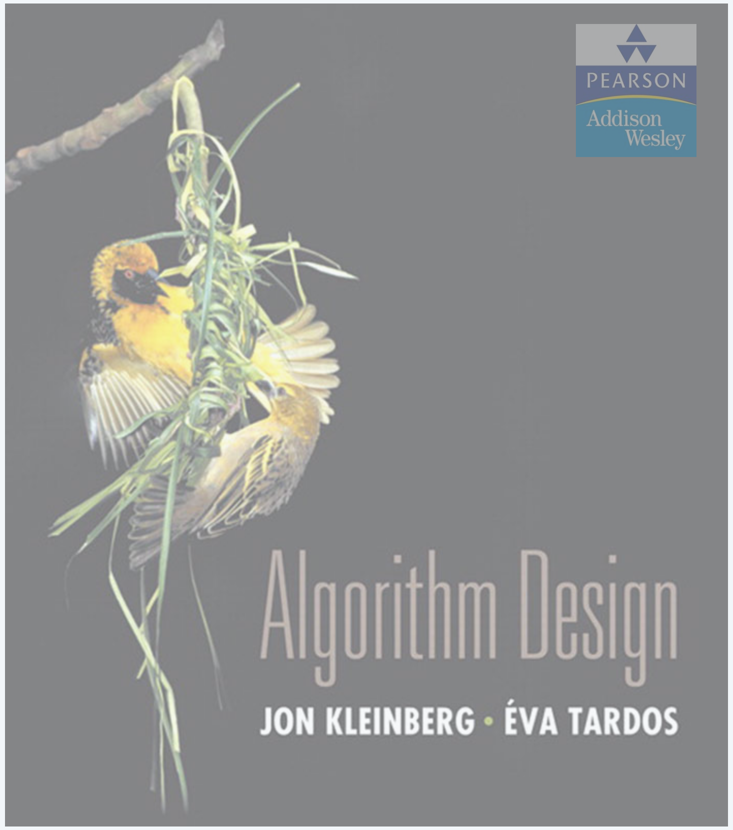
no perfect matching is stable

A–B, C–D $\Rightarrow$ B–C unstable

A–C, B–D $\Rightarrow$ A–B unstable

A–D, B–C $\Rightarrow$ A–C unstable

Observation. Stable matchings need not exist.



SECTION 1.1

1. STABLE MATCHING

- stable matching problem
- Gale–Shapley algorithm
- hospital optimality
- context

Gale–Shapley deferred acceptance algorithm

An intuitive method that guarantees to find a stable matching.

```
GALE–SHAPLEY (preference lists for hospitals and students)

INITIALIZE $M$ to empty matching.
WHILE (some hospital $h$ is unmatched and hasn't proposed to every student)
    $s \leftarrow$ first student on $h$'s list to whom $h$ has not yet proposed.
    IF ($s$ is unmatched)
        Add $h$–$s$ to matching $M$.
    ELSE IF ($s$ prefers $h$ to current partner $h'$)
        Replace $h'$–$s$ with $h$–$s$ in matching $M$.
    ELSE
        $s$ rejects $h$.

RETURN stable matching $M$.
```

Proof of correctness: termination

Observation 1. Hospitals propose to students in decreasing order of preference.

Observation 2. Once a student is matched, the student never becomes unmatched; only "trades up."

Claim. Algorithm terminates after at most n^2 iterations of WHILE loop.

Pf. Each time through the WHILE loop, a hospital proposes to a new student. Thus, there are at most n^2 possible proposals. ▪

	1st	2nd	3rd	4th	5th
A	V	W	X	Y	Z
B	W	X	Y	V	Z
C	X	Y	V	W	Z
D	Y	V	W	X	Z
E	V	W	X	Y	Z

	1st	2nd	3rd	4th	5th
V	B	C	D	E	A
W	C	D	E	A	B
X	D	E	A	B	C
Y	E	A	B	C	D
Z	A	B	C	D	E

n(n–1) + 1 proposals

Proof of correctness: perfect matching

Claim. Gale–Shapley outputs a matching.

Pf.

- Hospital proposes only if unmatched. $\Rightarrow$ matched to ≤ 1 student
- Student keeps only best hospital. $\Rightarrow$ matched to ≤ 1 hospital

Claim. In Gale–Shapley matching, all hospitals get matched.

Pf. [by contradiction]

- Suppose, for sake of contradiction, that some hospital $h \in H$ is unmatched upon termination of Gale–Shapley algorithm.
- Then some student, say $s \in S$, is unmatched upon termination.
- By Observation 2, s was never proposed to.
- But, h proposes to every student, since h ends up unmatched. ※

Claim. In Gale–Shapley matching, all students get matched.

Pf. [by counting]

- By previous claim, all n hospitals get matched.
- Thus, all n students get matched. ▪

Proof of correctness: stability

Claim. In Gale–Shapley matching M^*, there are no unstable pairs.

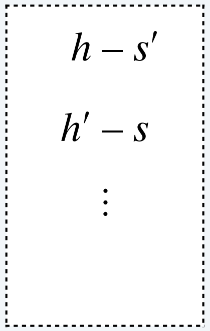
Pf. Consider any pair h–s that is not in M^*.

- Case 1: h never proposed to s.
  $\Rightarrow$ h prefers its Gale–Shapley partner s' to s. $\leftarrow$ hospitals propose in decreasing order of preference
  $\Rightarrow$ h–s is not unstable.

- Case 2: h proposed to s.
  $\Rightarrow$ s rejected h (either right away or later)
  $\Rightarrow$ s prefers Gale–Shapley partner h' to h.
  $\Rightarrow$ h–s is not unstable.

students only trade up

- In either case, the pair h–s is not unstable. ▪

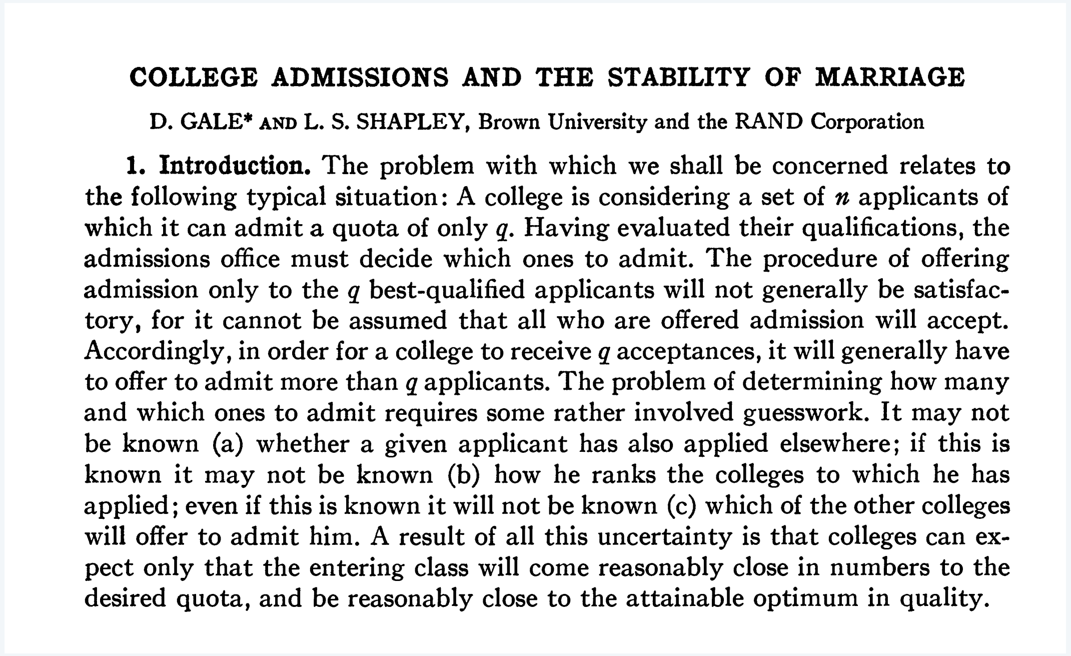


Gale–Shapley matching M*

Summary

Stable matching problem. Given n hospitals and n students, and their preference lists, find a stable matching if one exists.

Theorem. [Gale–Shapley 1962] The Gale–Shapley algorithm guarantees to find a stable matching for any problem instance.



COLLEGE ADMISSIONS AND THE STABILITY OF MARRIAGE

D. GALE* AND L. S. SHAPLEY, Brown University and the RAND Corporation

1. Introduction. The problem with which we shall be concerned relates to the following typical situation: A college is considering a set of n applicants of which it can admit a quota of only q. Having evaluated their qualifications, the admissions office must decide which ones to admit. The procedure of offering admission only to the q best-qualified applicants will not generally be satisfactory, for it cannot be assumed that all who are offered admission will accept. Accordingly, in order for a college to receive q acceptances, it will generally have to offer to admit more than q applicants. The problem of determining how many and which ones to admit requires some rather involved guesswork. It may not be known (a) whether a given applicant has also applied elsewhere; if this is known it may not be known (b) how he ranks the colleges to which he has applied; even if this is known it will not be known (c) which of the other colleges will offer to admit him. A result of all this uncertainty is that colleges can expect only that the entering class will come reasonably close in numbers to the desired quota, and be reasonably close to the attainable optimum in quality.

Stable matching: quiz 2

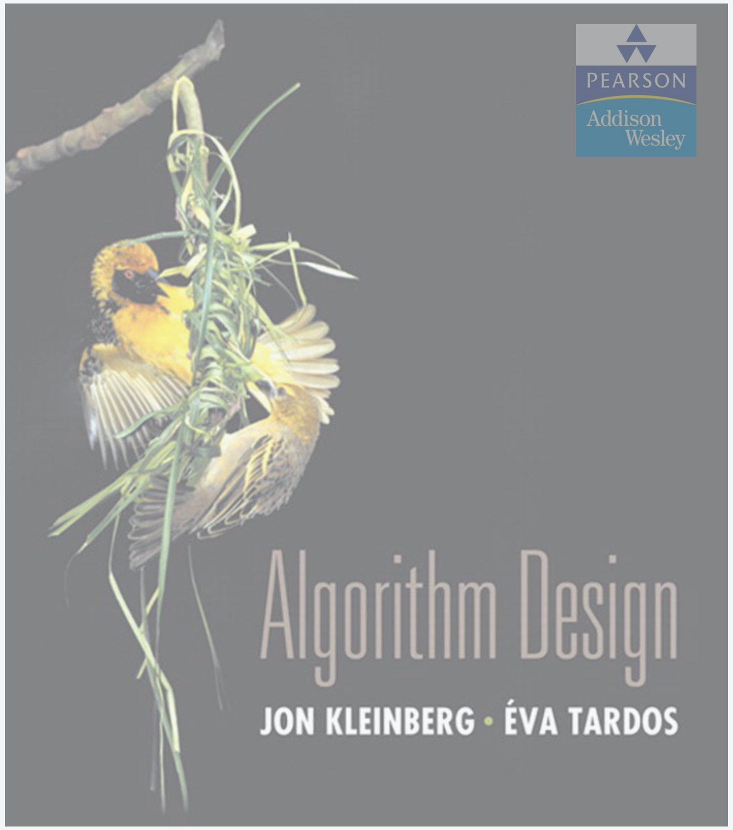
Do all executions of Gale–Shapley lead to the same stable matching?

A. No, because the algorithm is nondeterministic.

B. No, because an instance can have several stable matchings.

C. Yes, because each instance has a unique stable matching.

D. Yes, even though an instance can have several stable matchings and the algorithm is nondeterministic.



SECTION 1.1

1. STABLE MATCHING

- *stable matching problem*
- *Gale–Shapley algorithm*
- *hospital optimality*
- *context*

Understanding the solution

For a given problem instance, there may be several stable matchings.

	1st	2nd	3rd
A	X	Y	Z
B	Y	X	Z
C	X	Y	Z

	1st	2nd	3rd
X	B	A	C
Y	A	B	C
Z	A	B	C

an instance with two stable matchings: S = { A–X, B–Y, C–Z } and S′ = { A–Y, B–X, C–Z }

Understanding the solution

Def. Student s is a valid partner for hospital h if there exists any stable matching in which h and s are matched.

Ex.

- Both X and Y are valid partners for A.
- Both X and Y are valid partners for B.
- Z is the only valid partner for C.

	1st	2nd	3rd
A	X	Y	Z
B	Y	X	Z
C	X	Y	Z

	1st	2nd	3rd
X	B	A	C
Y	A	B	C
Z	A	B	C

an instance with two stable matchings: S = { A–X, B–Y, C–Z } and S′ = { A–Y, B–X, C–Z }

Stable matching: quiz 3

Who is the best valid partner for W in the following instance?

A.

B.

C.

D.

6 stable matchings
{ A–W, B–X, C–Y, D–Z }
{ A–X, B–W, C–Y, D–Z }
{ A–X, B–Y, C–W, D–Z }
{ A–Z, B–W, C–Y, D–X }
{ A–Z, B–Y, C–W, D–X }
{ A–Y, B–Z, C–W, D–X }

	1st	2nd	3rd	4th
A	Y	Z	X	W
B	Z	Y	W	X
C	W	Y	X	Z
D	X	Z	W	Y

	1st	2nd	3rd	4th
W	D	A	B	C
X	C	B	A	D
Y	C	B	A	D
Z	D	A	B	C

Stable matching: quiz 3

Who is the best valid partner for W in the following instance?

A.

B.

C.

D.

6 stable matchings

{ A–W, B–X, C–Y, D–Z }

{ A–X, B–W, C–Y, D–Z }

{ A–X, B–Y, C–W, D–Z }

{ A–Z, B–W, C–Y, D–X }

{ A–Z, B–Y, C–W, D–X }

{ A–Y, B–Z, C–W, D–X }

	1st	2nd	3rd	4th
A	Y	Z	X	W
B	Z	Y	W	X
C	W	Y	X	Z
D	X	Z	W	Y

best valid partners

valid partners

	1st	2nd	3rd	4th
W	D	A	B	C
X	C	B	A	D
Y	C	B	A	D
Z	D	A	B	C

Understanding the solution

Def. Student s is a valid partner for hospital h if there exists any stable matching in which h and s are matched.

Hospital-optimal assignment. Each hospital receives best valid partner.

- Is it a perfect matching?
- Is it stable?

Claim. All executions of Gale–Shapley yield hospital-optimal assignment.

Corollary. Hospital-optimal assignment is a stable matching!

Hospital optimality

Claim. Gale–Shapley matching M^* is hospital-optimal.

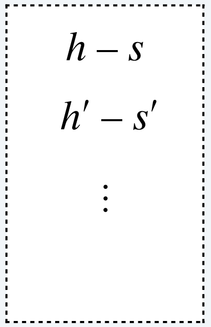
Pf. [by contradiction]

- Suppose a hospital is matched with student other than best valid partner.
- Hospitals propose in decreasing order of preference.

  $\Rightarrow$ some hospital is rejected by a valid partner during Gale–Shapley
- Let h be first such hospital, and let s be the first valid partner that rejects h.
- Let M be a stable matching where h and s are matched.
- When s rejects h in Gale–Shapley, s forms (or re-affirms) commitment to a hospital, say h'.

  $\Rightarrow$ $\boxed{s \text{ prefers } h' \text{ to } h.}$ $\longleftarrow$ students only trade up
- Let s' be partner of h' in M.
- h' had not been rejected by any valid partner (including s') at the point when h is rejected by s. $\longleftarrow$ because this is the first rejection by a valid partner
- **Thus**, h' had not yet proposed to s' when h' proposed to s.

  $\Rightarrow$ $\boxed{h' \text{ prefers } s \text{ to } s'.}$ $\longleftarrow$ hospitals propose in decreasing order of preference
- Thus, h'–s is unstable in M, a contradiction. ▪



stable matching M

Student pessimality

Q. Does hospital-optimality come at the expense of the students?

A. Yes.

Student-pessimal assignment. Each student receives worst valid partner.

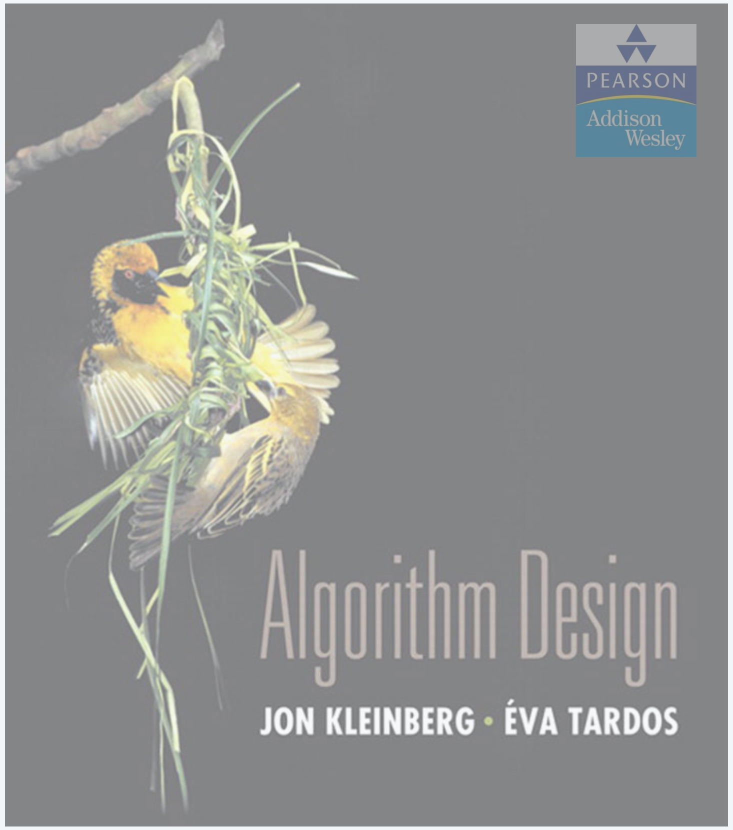
Claim. Gale–Shapley finds student-pessimal stable matching M^*.

Pf. [by contradiction]

- Suppose h–s matched in M^* but h is not the worst valid partner for s.
- There exists stable matching M in which s is paired with a hospital, say h', whom s prefers less than h.
  $\Rightarrow \boxed{s \text{ prefers } h \text{ to } h'.}$
- Let s' be the partner of h in M.
- By hospital-optimality, s is the best valid partner for h.
  $\Rightarrow \boxed{h \text{ prefers } s \text{ to } s'.}$
- Thus, h–s is an unstable pair in M, a contradiction. ▪

$$h' - s$$
$$h - s'$$
$$\vdots$$

stable matching M



SECTION 1.1

1. STABLE MATCHING

- *stable matching problem*
- *Gale–Shapley algorithm*
- *hospital optimality*
- *context*

Extensions

Extension 1. Some agents declare others as unacceptable.

Extension 2. Some hospitals have more than one position.

Extension 3. Unequal number of positions and students.

med-school student unwilling to work in Cleveland

≥ 43K med-school students; only 31K positions

Def. Matching M is unstable if there is a hospital h and student s such that:

- h and s are acceptable to each other; and
- Either s is unmatched, or s prefers h to assigned hospital; and
- Either h does not have all its places filled, or h prefers s to at least one of its assigned students.

Theorem. There exists a stable matching.

Pf. Straightforward generalization of Gale–Shapley algorithm.