# CSE 421

## Greedy Algorithms / Caching Problem

Yin Tat Lee

# Optimal Caching/Paging

## Memory systems

- Many levels of storage with different access times
- Smaller storage has shorter access time
- To access an item it must be brought to the lowest level of the memory system

| Type | Latency | Capacity |
|---|---|---|
| Registers | 0.25 ns | 36 KB |
| L1 Cache | 1 ns | 192 KB |
| L2 Cache | 3 ns | 1.5 MB |
| L3 Cache | 14 ns | 15 MB |
| DRAM | 66 ns | 32 GB |
| SDD | 0.15 ms | 480 GB |
| Internet | 7 ms | |

My home computer

# Optimal Caching/Paging

## Memory systems

- Many levels of storage with different access times
- Smaller storage has shorter access time
- To access an item it must be brought to the lowest level of the memory system

## Consider the problem between 2 levels

- Main memory with $\boldsymbol{n}$ data items
- Cache can hold $\boldsymbol{k} < \boldsymbol{n}$ items
- Assume no restrictions about where items can be
- Suppose cache is full initially
  - Holds $\boldsymbol{k}$ data items to start with

# Optimal Offline Caching

## Caching

- Cache with capacity to store $k$ items.
- Sequence of $m$ item requests $d_1, d_2, \cdots, d_m$.
- Cache hit: item already in cache when requested.
- Cache miss: item not already in cache when requested: must bring requested item into cache, and evict some existing item, if full.

## Goal

- Eviction schedule that minimizes number of evictions.

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

Optimal eviction schedule: $2$ cache misses.

| requests | cache | |
|---|---|---|
| a | a | b |
| b | a | b |
| c | c | b |
| b | c | b |
| c | c | b |
| a | a | b |
| a | a | b |
| b | a | b |

Why 2 is optimal?

# Optimal Offline Caching: Farthest-In-Future

**Which item we should evict?**

## Farthest-in-future

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

current cache:

| a | b | c | d | e | f |
|---|---|---|---|---|---|

future queries: `g a b c e d a b b a c d e a f a d e f g h ...`

cache miss (g)

eject this one (f)

## Theorem

- [Bellady, 1960s] FIF is an optimal eviction schedule.

## Exchange Argument

- We can swap choices to convert other schedules to Farthest-In-Future without losing quality

# Why Exchange Argument?

Greedy cannot handle problems with many local minimum.

Exchange argument basically proving there is no local min.

# Warm up ($n = k + 1$)

## Farthest-in-future

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

current cache:

| a | b | c | d | e | f |
|---|---|---|---|---|---|

future queries: **g a b c e d a b b a c d e a f a d e f g h . . .**

cache miss

eject this one

When $n = k + 1$,

between the cache miss and the farthest-item in the future,

"**g a b c e d a b b a c d e a f**"

contains all the item.

Hence, any algorithm must miss once.

Problem: What's the problem of this proof for $n > k + 1$.

# Reduced Eviction Schedules

## Definition

- A reduced schedule is a schedule that only inserts an item into the cache in a step in which that item is requested.

## Intuition

- Can transform an unreduced schedule into a reduced one with no more cache misses.

| | | | |
|---|---|---|---|
| a | a | b | c |
| a | a | x | c |
| c | a | d | c |
| d | a | d | b |
| a | a | c | b |
| b | a | x | b |
| c | a | c | b |
| a | a | b | c |
| a | a | b | c |

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 |
| a | a | c | b |
| a | a | c | b |

a reduced schedule

# Reduced Eviction Schedules

Claim

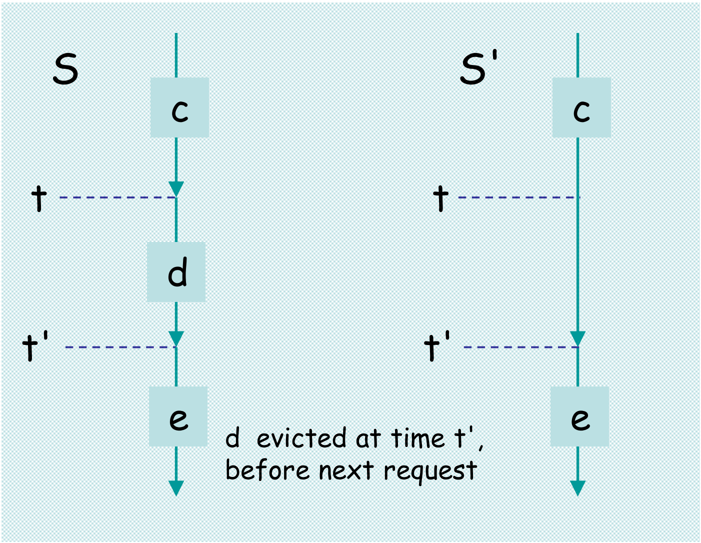
- Given any unreduced schedule $S$, can transform it into a reduced schedule $S'$ with no more cache misses.

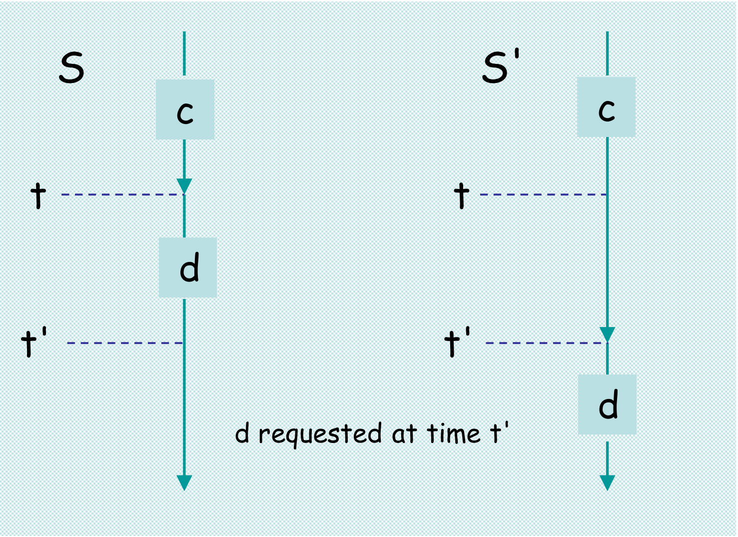
Proof (by induction on number of unreduced items)

- Suppose $S$ brings $d$ into the cache at time $t$, without a request.
- Let $c$ be the item $S$ evicts when it brings $d$ into the cache.

Case 1: $d$ evicted at time $t'$, before next request for $d$.

Case 2: $d$ requested at time $t'$ before $d$ is evicted. ▪



# Farthest-In-Future: Analysis

Theorem

- FIF is optimal eviction algorithm.

Proof. (by induction on number or requests $j$)

> Invariant: There exists an optimal reduced schedule $\boldsymbol{S}$ that makes the same eviction schedule as $\boldsymbol{S_{FIF}}$ through the first $\boldsymbol{j+1}$ requests.

Let $\boldsymbol{S}$ be reduced schedule that satisfies invariant through $\boldsymbol{j}$ requests. We produce $\boldsymbol{S'}$ that satisfies invariant after $\boldsymbol{j+1}$ requests.

- Consider $(\boldsymbol{j+1})^{\text{st}}$ request $\boldsymbol{d = d_{j+1}}$.
- Since $\boldsymbol{S}$ and $\boldsymbol{S_{FIF}}$ have agreed up until now, they have the same cache contents before request $\boldsymbol{j+1}$.

Case 1: ($\boldsymbol{d}$ is already in the cache).

$\boldsymbol{S' = S}$ satisfies invariant. (used $\boldsymbol{S}$ is reduced here)

Case 2: ($\boldsymbol{d}$ is not in the cache and $\boldsymbol{S}$ and $\boldsymbol{S_{FIF}}$ evict the same element).

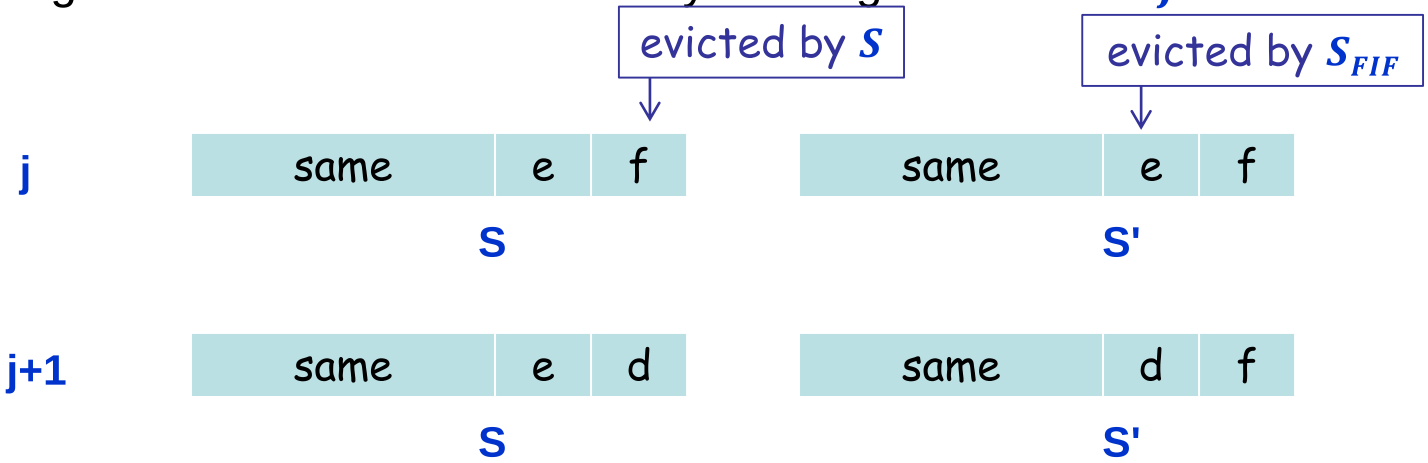
$\boldsymbol{S' = S}$ satisfies invariant.

# Farthest-In-Future: Analysis

Proof. (continued)

Case 3: ($d$ is not in the cache; $S_{FIF}$ evicts $e$; $S$ evicts $f \neq e$).

- begin construction of $S'$ from $S$ by evicting $e$ instead of $f$




- now $S'$ agrees with $S_{FIF}$ on first $j+1$ requests; we show that having element $f$ in cache is no worse than having element $e$
  - Continue building $S'$ to be the same as $S$ until forced to be different

# Farthest-In-Future: Analysis

Proof. (continued)

Let $j'$ be the first time after $j+1$ that $S$ and $S'$ must take a different action, and let $g$ be item requested at time $j'$.

| j' | same | e | | same | f |
|---|---|---|---|---|---|
| | S | | | S' | |

Case 3a: $g = e$.

Can't happen: $e$ was evicted by Farthest-In-Future so there must be a request for $f$ before $e$.

Case 3b: $g = f$.

Element $f$ can't be in cache of $S$, so let $e'$ be the element that $S$ evicts.

- if $e' = e$, $S'$ accesses $f$ from cache; now $S$ and $S'$ have same cache
- if $e' \neq e$, $S'$ evicts $e'$ and brings $e$ into the cache; now $S$ and $S'$ have the same cache

Note: $S'$ is no longer reduced, but can be transformed into a reduced schedule that agrees with $S_{FIF}$ through step $j+1$

# Farthest-In-Future: Analysis

Proof. (continued)

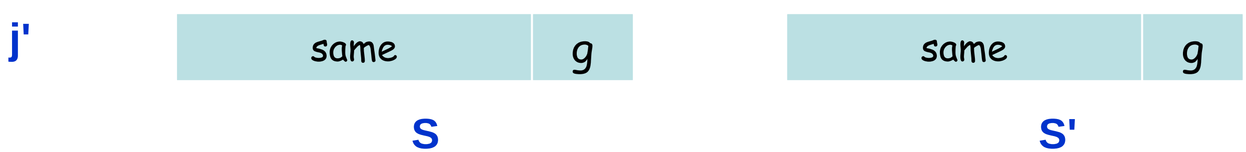
Let $j'$ be the first time after $j+1$ that $S$ and $S'$ must take a different action, and let $g$ be item requested at time $j'$.

Action of S must involve e or f (or both)


Case 3c: $g \neq e$ and $g \neq f$.

$S$ must evict $e$.
Make $S'$ evict $f$; now $S$ and $S'$ have the same cache. ▪

otherwise $S'$ would take the same action




In each case can now extend $S'$ using rest of $S$ at no extra cost.
$S'$ is optimal, reduced, and agrees with $S_{FIF}$ for $j+1$ steps
Optimality of $S_{FIF}$ follows by induction.

# Online Caching

- 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.

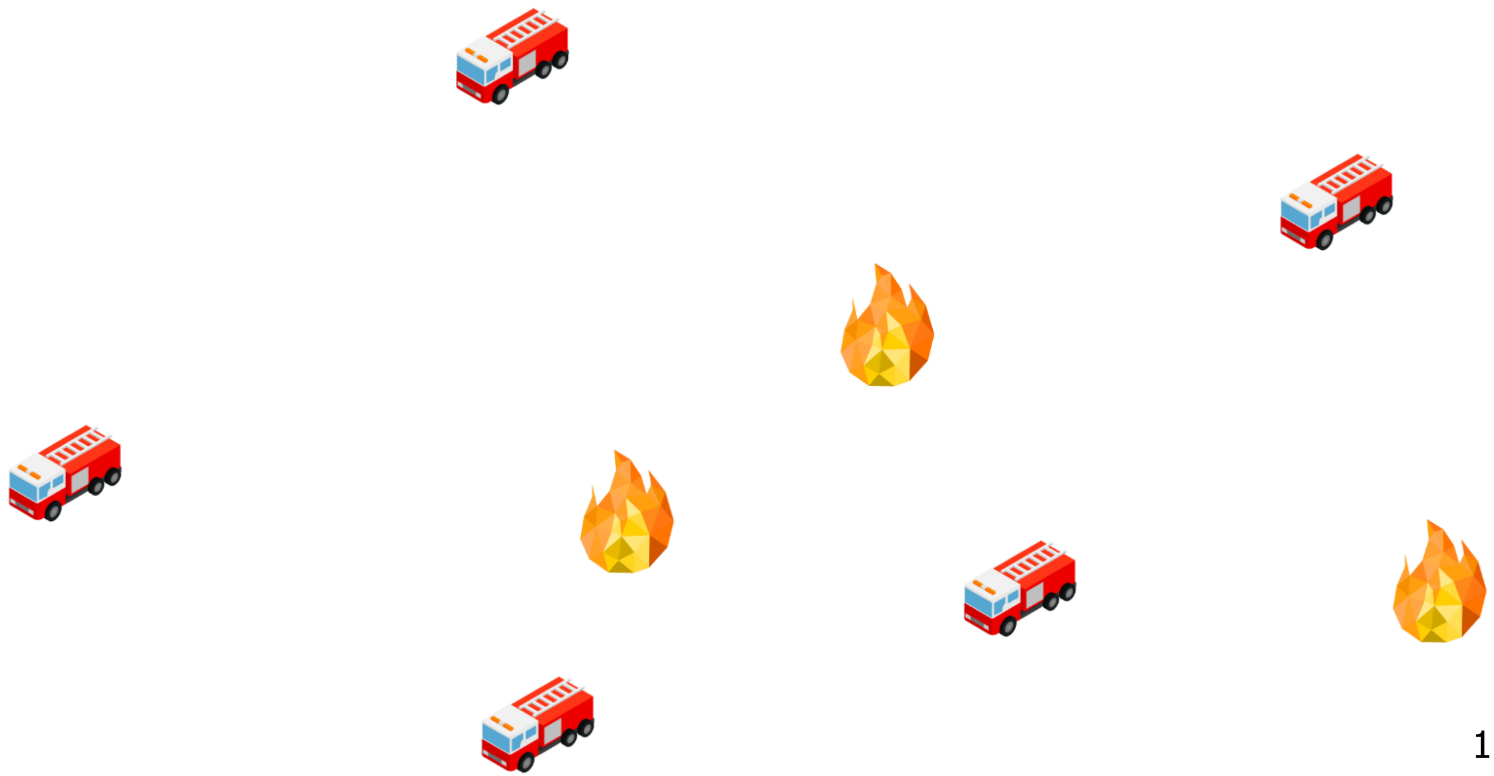
- LIFO. Evict page brought in most recently.
- LRU. Evict page whose most recent access was earliest.

FIF with direction of time reversed!

- Theorem. FIF is optimal offline eviction algorithm.
  - Provides basis for understanding and analyzing online algorithms.
  - LRU is k-competitive. [Section 13.8]
  - LIFO is arbitrarily bad.

# $k$-server problem

- There are $k$ fire trucks.
- When fire happens, We need to move a truck there.
- Define $ALG$ is the total movement of all fire trucks.
- Define $OPT$ is the total movement of the optimal plan if we know where the fire happens in advance.
- Define the competitive ratio is $ALG/OPT$.



# Greedy does not work

**Fusible HSTs and the randomized k-server conjecture**

James R. Lee

(Submitted on 6 Nov 2017 (v1), last revised 21 Feb 2018 (this version, v2))

James R. Lee

$ALG = +\infty$.
$OPT = O(1)$.

So, the competitive ratio is $\frac{ALG}{OPT} = +\infty$.

For long time, the best competitive ratio is $O(k)$.

It was conjectured that one can get $\log^{O(1)}(k)$.

This man did it.