Announcements

- Homework 3 online, due today
- Exam 2 Next Friday

Exam 2

- In class
- 6 one-sided pages of notes
- No textbooks or electronic aids
- Assigned seats
- 3Qs in 45 minutes

Topics

- Divide and Conquer
  - Basic paradigm
  - Master Theorem
  - Karatsuba Multiplication
  - MergeSort
  - Order statistics
  - Binary Search
  - Closest Pair of Points
- Greedy algorithms
  - Basic paradigm
  - Exchange arguments
  - Interval packing
  - Optimal Caching

Last Time

- Greedy Algorithms
- Interval Scheduling

Greedy Algorithms

Often when trying to find the optimal solution to some problem you need to consider all your possible choices and how they might interact with other choices down the line.

But sometimes you don’t. Sometimes you can just take what looks like the best option for now and repeat.

Greedy Algorithms

General Algorithmic Technique:

1. Find decision criterion
2. Make best choice according to criterion
3. Repeat until done

Surprisingly, this sometimes works.

Interval Scheduling

Problem: Given a collection C of intervals, find a subset $S \subseteq C$ so that:

1. No two intervals in S overlap.
2. Subject to (1), $|S|$ is as large as possible.

Algorithm: Repeatedly add the interval with the smallest maximum that doesn't overlap with already chosen intervals.

Proofs

As it is very easy to write down plausible greedy algorithms for problems, but more difficult to find correct ones, it is very important to be able to *prove* that your algorithm is correct.

Fortunately, there is a standard proof technique for greedy algorithms.

Today

- Exchange arguments
- Optimal caching
- Huffman codes

Exchange Argument

- Greedy algorithm makes a sequence of decisions D_1, D_2, D_3,…,D_n eventually reaching solution G.

Exchange Argument

- Greedy algorithm makes a sequence of decisions D_1, D_2, D_3,…,D_n eventually reaching solution G.
- Need to show that for arbitrary solutions A that G ≥ A.

Exchange Argument

- Greedy algorithm makes a sequence of decisions D_1, D_2, D_3,…,D_n eventually reaching solution G.
- Need to show that for arbitrary solutions A that G ≥ A.
- Find sequence of solutions
  $A=A_0$, A_1, A_2,…,A_n = G
  so that:

Exchange Argument

- Greedy algorithm makes a sequence of decisions D_1, D_2, D_3,…,D_n eventually reaching solution G.
- Need to show that for arbitrary solutions A that G ≥ A.
- Find sequence of solutions
  $A=A_0, A_1, A_2,\ldots,A_n = G$
  so that:
  - $A_i \leq A_{i+1}$
  - A_i agrees with $D_1,D_2,\ldots,D_i$

Exchange Argument

In particular, we need to show that given any A_i consistent with $D_1,\ldots,D_i$ we can find an A_{i+1} so that:

- A_{i+1} is consistent with $D_1,\ldots,D_{i+1}$
- $A_{i+1} \geq A_i$

Exchange Argument

In particular, we need to show that given any A_i consistent with $D_1,\ldots,D_i$ we can find an A_{i+1} so that:

- A_{i+1} is consistent with $D_1,\ldots,D_{i+1}$
- $A_{i+1} \geq A_i$

Then we inductively construct sequence

$A=A_0 \leq A_1 \leq A_2 \leq \ldots \leq A_n = G$

Exchange Argument

In particular, we need to show that given any A_i consistent with $D_1,\ldots,D_i$ we can find an A_{i+1} so that:

- A_{i+1} is consistent with $D_1,\ldots,D_{i+1}$
- $A_{i+1} \geq A_i$

Then we inductively construct sequence

$A=A_0 \leq A_1 \leq A_2 \leq \ldots \leq A_n = G$

Thus, $G \geq A$ for any A. So G is optimal.

Exchange Argument for Interval Packing

- What decisions does greedy algorithm make?

Exchange Argument for Interval Packing

- What decisions does greedy algorithm make?
  - D_i, i^{th} interval equals J_i.

Exchange Argument for Interval Packing

- What decisions does greedy algorithm make?
  - D_i, i^{th} interval equals J_i.
- Need to show that IF we have a solution that agrees with first i decisions, can make it agree with i+1 without making it worse.

Exchange Argument for Interval Packing

- What decisions does greedy algorithm make?
  - D_i, i^{th} interval equals J_i.
- Need to show that IF we have a solution that agrees with first i decisions, can make it agree with i+1 without making it worse.
- Have solution: $J_1, J_2, \ldots, J_i, K_{i+1}, \ldots, K_m$
  - Need to modify to use interval J_{i+1}.

Inductive Step

Greedy solution: $J_1, J_2, \ldots, J_n$ $J_i = [x_i, y_i]$

Current solution: $J_1, J_2, \ldots, J_i, K_{i+1}, \ldots, K_m$ $K_i = [w_i, z_i]$

Inductive Step

Greedy solution: $J_1,J_2,\ldots,J_n$ $J_i = [x_i,y_i]$

Current solution: $J_1,J_2,\ldots,J_i,K_{i+1},\ldots,K_m$ $K_i = [w_i,z_i]$

Claim: $J_1,J_2,\ldots,J_i,J_{i+1},K_{i+2},\ldots,K_m$ is a valid solution.

Inductive Step

Greedy solution: $J_1,J_2,\ldots,J_n$ $J_i = [x_i,y_i]$

Current solution: $J_1,J_2,\ldots,J_i,K_{i+1},\ldots,K_m$ $K_i = [w_i,z_i]$

Claim: $J_1,J_2,\ldots,J_i,J_{i+1},K_{i+2},\ldots,K_m$ is a valid solution.

Proof: Need to verify that J_{i+1} doesn't overlap anything:

- $x_{i+1} > y_i$: This is because J_i, J_{i+1} don't overlap.

Inductive Step

Greedy solution: $J_1, J_2, \ldots, J_n$ $J_i = [x_i, y_i]$

Current solution: $J_1, J_2, \ldots, J_i, K_{i+1}, \ldots, K_m$ $K_i = [w_i, z_i]$

Claim: $J_1, J_2, \ldots, J_i, J_{i+1}, K_{i+2}, \ldots, K_m$ is a valid solution.

Proof: Need to verify that J_{i+1} doesn't overlap anything:

- $x_{i+1} > y_i$: This is because J_i, J_{i+1} don't overlap.
- $w_{i+2} > y_{i+1}$: This is because $w_{i+2} > z_{i+1} \geq y_{i+1}$.

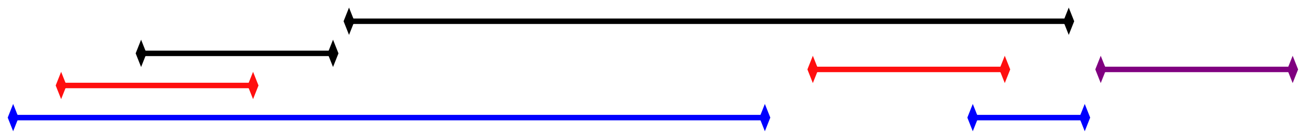
Example

Greedy Solution

Example

Greedy Solution

Arbitrary Solution



Example

Greedy Solution

Arbitrary Solution

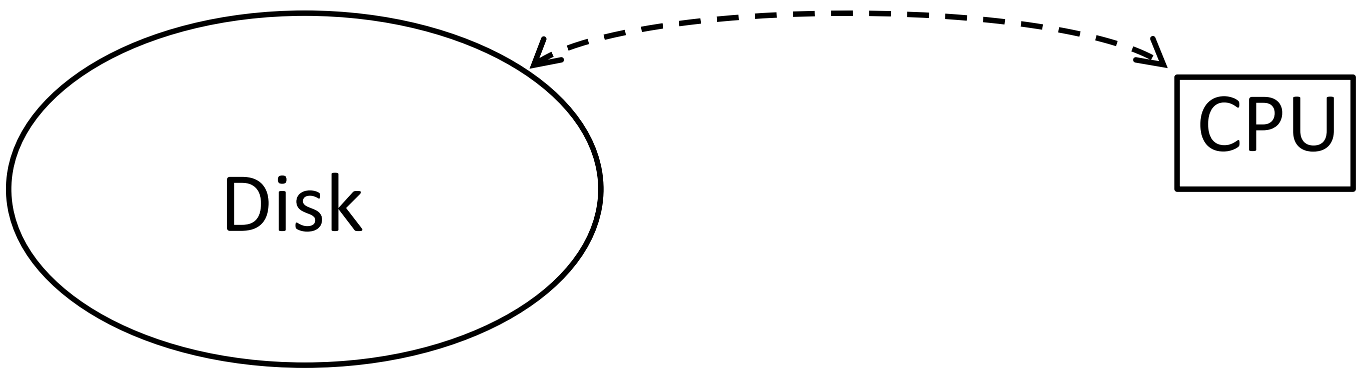
Example

Greedy Solution

Arbitrary Solution

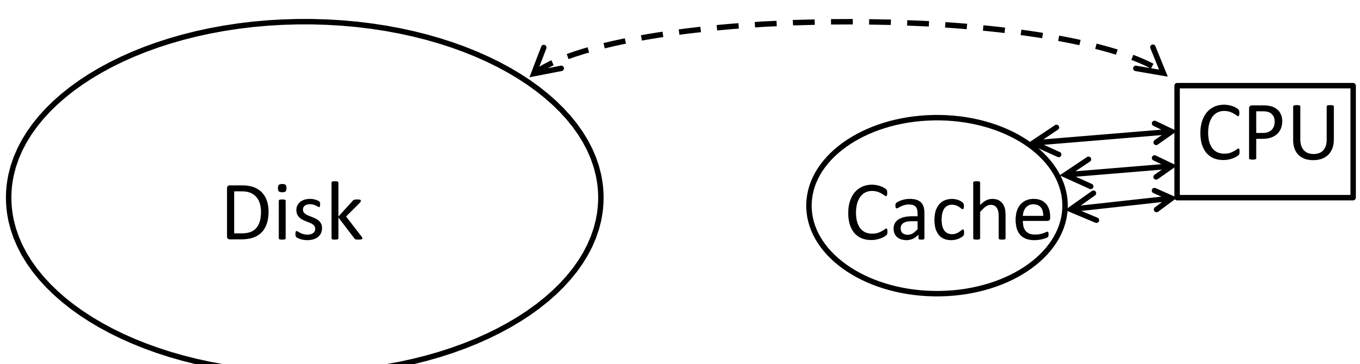
Optimal Caching (not in textbook)

- Communication between disk and CPU is slow.



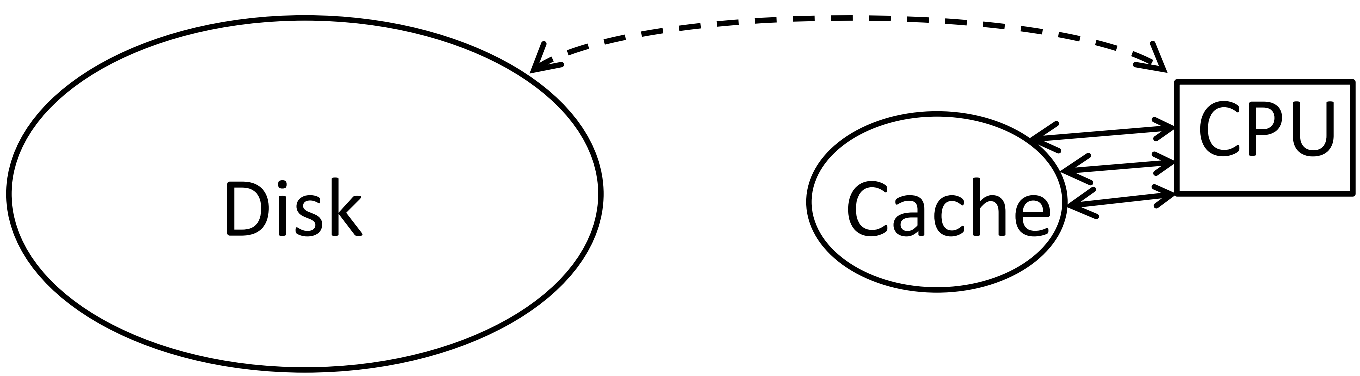
Optimal Caching (not in textbook)

- Communication between disk and CPU is slow.
- Have much smaller cache close to CPU.



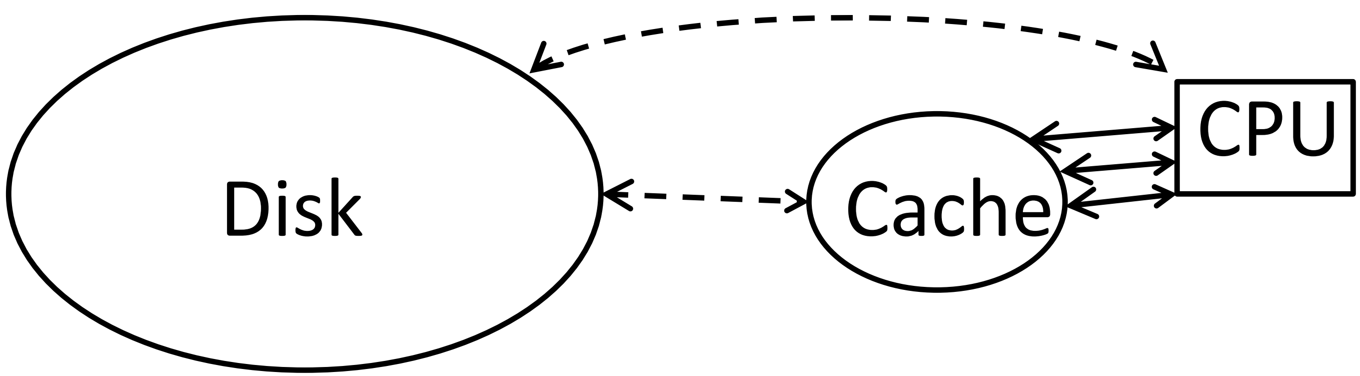
Optimal Caching (not in textbook)

- Communication between disk and CPU is slow.
- Have much smaller cache close to CPU.
- Store only a bit in cache at a time.



Optimal Caching (not in textbook)

- Communication between disk and CPU is slow.
- Have much smaller cache close to CPU.
- Store only a bit in cache at a time.
- If need to access some other location, will need to load into cache (slow).



Model

- k words in cache at a time.

Cache:

Location: 1011
Location: 0001
Location: 1110
Location: 0101

Model

- k words in cache at a time.
- CPU asks for memory access.

CPU Needs:

Location: 0001

Cache:

Location: 1011
Location: 0001
Location: 1110
Location: 0101

Model

- k words in cache at a time.
- CPU asks for memory access.
- If in cache already, easy.

CPU Needs:

Location: 0001

Cache:

Location: 1011

Location: 0001

Location: 1110

Location: 0101

Model

- k words in cache at a time.
- CPU asks for memory access.
- If in cache already, easy.
- Otherwise, need to load into cache replacing something else, slow.

CPU Needs:

Location: 0001

Location: 1001

Cache:

Location: 1011

Location: 0001

Location: 1110

Location: 0101

Model

- k words in cache at a time.
- CPU asks for memory access.
- If in cache already, easy.
- Otherwise, need to load into cache replacing something else, slow.

CPU Needs:

Location: 0001

Location: 1001

Cache:

Location: 1011

Location: 0001

Location: 1110

Location: 0101

Model

- k words in cache at a time.
- CPU asks for memory access.
- If in cache already, easy.
- Otherwise, need to load into cache replacing something else, slow.

CPU Needs:

Location: 0001

Location: 1001

Cache:

Location: 1011

Location: 0001

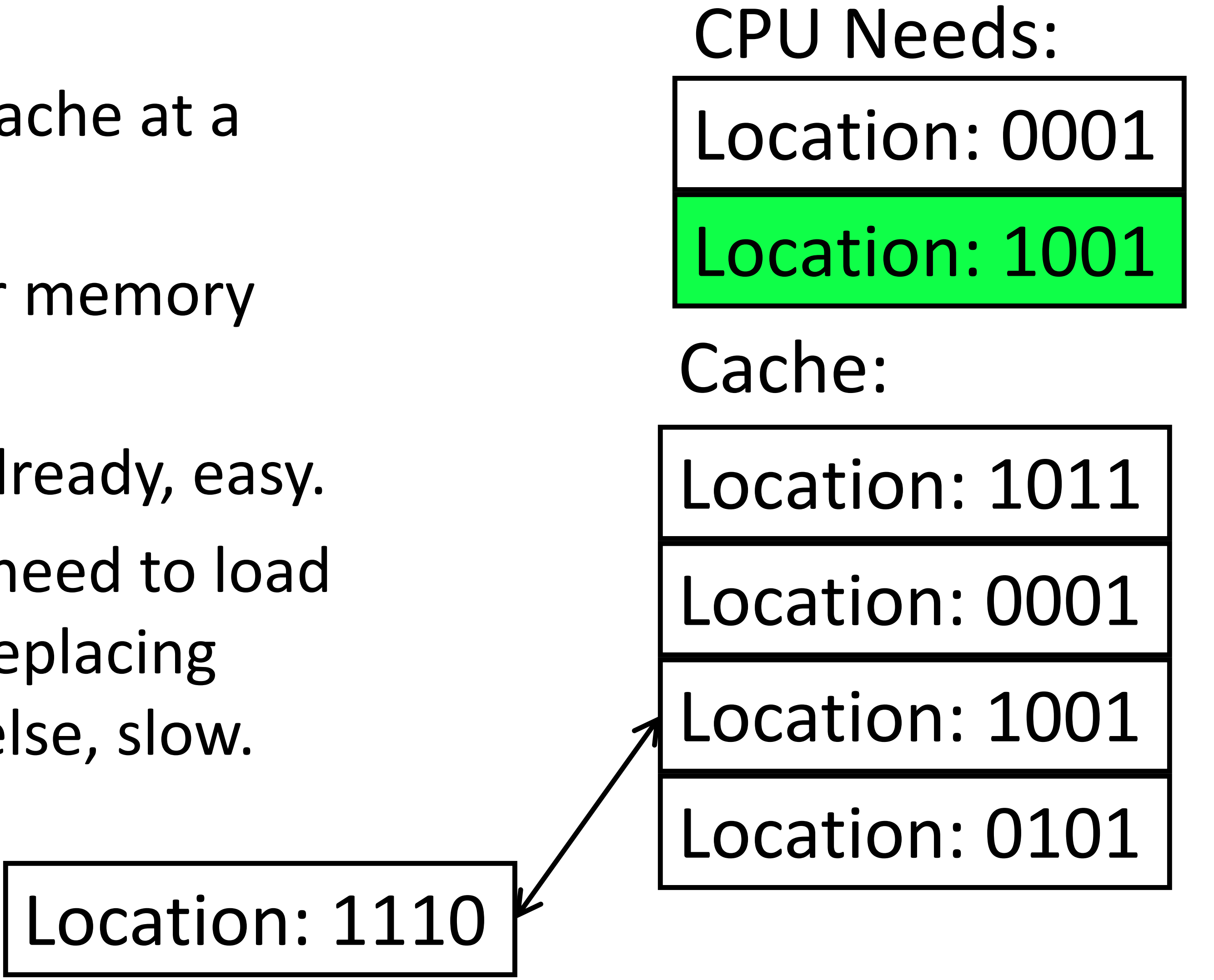
Location: 1001

Location: 0101

Location: 1110

Model

- k words in cache at a time.
- CPU asks for memory access.
- If in cache already, easy.
- Otherwise, need to load into cache replacing something else, slow.



Optimal Caching

Problem: Given sequence of memory accesses and cache size, find a cache schedule that involves fewest possible number of swaps with disk.

Optimal Caching

Problem: Given sequence of memory accesses and cache size, find a cache schedule that involves fewest possible number of swaps with disk.

CPU	A	B	A	C	A	D	E	C	B	C	A	C
Cache 1												
Cache 2												

Optimal Caching

Problem: Given sequence of memory accesses and cache size, find a cache schedule that involves fewest possible number of swaps with disk.

CPU	A	B	A	C	A	D	E	C	B	C	A	C
Cache 1	A											
Cache 2	-											

Optimal Caching

Problem: Given sequence of memory accesses and cache size, find a cache schedule that involves fewest possible number of swaps with disk.

CPU	A	B	A	C	A	D	E	C	B	C	A	C
Cache 1	A	A										
Cache 2	-	B										

Optimal Caching

Problem: Given sequence of memory accesses and cache size, find a cache schedule that involves fewest possible number of swaps with disk.

CPU	A	B	A	C	A	D	E	C	B	C	A	C
Cache 1	A	A	A									
Cache 2	-	B	B									

Optimal Caching

Problem: Given sequence of memory accesses and cache size, find a cache schedule that involves fewest possible number of swaps with disk.

CPU	A	B	A	C	A	D	E	C	B	C	A	C
Cache 1	A	A	A	A								
Cache 2	-	B	B	C								

Optimal Caching

Problem: Given sequence of memory accesses and cache size, find a cache schedule that involves fewest possible number of swaps with disk.

CPU	A	B	A	C	A	D	E	C	B	C	A	C
Cache 1	A	A	A	A	A							
Cache 2	-	B	B	C	C							

Optimal Caching

Problem: Given sequence of memory accesses and cache size, find a cache schedule that involves fewest possible number of swaps with disk.

CPU	A	B	A	C	A	D	E	C	B	C	A	C
Cache 1	A	A	A	A	A	A						
Cache 2	-	B	B	C	C	D						

Optimal Caching

Problem: Given sequence of memory accesses and cache size, find a cache schedule that involves fewest possible number of swaps with disk.

CPU	A	B	A	C	A	D	E	C	B	C	A	C
Cache 1	A	A	A	A	A	A	A					
Cache 2	-	B	B	C	C	D	E					

Optimal Caching

Problem: Given sequence of memory accesses and cache size, find a cache schedule that involves fewest possible number of swaps with disk.

CPU	A	B	A	C	A	D	E	C	B	C	A	C
Cache 1	A	A	A	A	A	A	A	C				
Cache 2	-	B	B	C	C	D	E	E				

Optimal Caching

Problem: Given sequence of memory accesses and cache size, find a cache schedule that involves fewest possible number of swaps with disk.

CPU	A	B	A	C	A	D	E	C	B	C	A	C
Cache 1	A	A	A	A	A	A	A	C	C			
Cache 2	-	B	B	C	C	D	E	E	B			

Optimal Caching

<u>Problem:</u> Given sequence of memory accesses and cache size, find a cache schedule that involves fewest possible number of swaps with disk.

CPU	A	B	A	C	A	D	E	C	B	C	A	C
Cache 1	A	A	A	A	A	A	A	C	C	C		
Cache 2	-	B	B	C	C	D	E	E	B	B		

Optimal Caching

Problem: Given sequence of memory accesses and cache size, find a cache schedule that involves fewest possible number of swaps with disk.

CPU	A	B	A	C	A	D	E	C	B	C	A	C
Cache 1	A	A	A	A	A	A	A	C	C	C	C	
Cache 2	-	B	B	C	C	D	E	E	B	B	A	

Optimal Caching

Problem: Given sequence of memory accesses and cache size, find a cache schedule that involves fewest possible number of swaps with disk.

CPU	A	B	A	C	A	D	E	C	B	C	A	C
Cache 1	A	A	A	A	A	A	A	C	C	C	C	C
Cache 2	-	B	B	C	C	D	E	E	B	B	A	A

Optimal Caching

<u>Problem:</u> Given sequence of memory accesses and cache size, find a cache schedule that involves fewest possible number of swaps with disk.

CPU	A	B	A	C	A	D	E	C	B	C	A	C
Cache 1	A	A	A	A	A	A	A	C	C	C	C	C
Cache 2	-	B	B	C	C	D	E	E	B	B	A	A

8 Cache misses.

Observation

- No need to get new entries in cache ahead of time.
- Only make replacements when new value is called for.
- Only question algorithm needs to answer is which memory cells to overwrite.

Question

What is a good candidate greedy procedure for deciding which cell to overwrite?

Furthest In The Future (FITF)

- For each cell consider the next time that memory location will be called for.
- Replace cell whose next call is the furthest in the future.

Furthest In The Future (FITF)

- For each cell consider the next time that memory location will be called for.
- Replace cell whose next call is the furthest in the future.

	X	A	Y	A	B	B	X	C
A								
B								
C								

Furthest In The Future (FITF)

- For each cell consider the next time that memory location will be called for.
- Replace cell whose next call is the furthest in the future.

	X	**A**	Y	A	B	B	X	C
A								
B								
C								

Furthest In The Future (FITF)

- For each cell consider the next time that memory location will be called for.
- Replace cell whose next call is the furthest in the future.

	X	**A**	Y	A	**B**	B	X	C
A								
B								
C								

Furthest In The Future (FITF)

- For each cell consider the next time that memory location will be called for.
- Replace cell whose next call is the furthest in the future.

	X	A	Y	A	B	B	X	C
A								
B								
C								

Furthest In The Future (FITF)

- For each cell consider the next time that memory location will be called for.
- Replace cell whose next call is the furthest in the future.

	X	A	Y	A	B	B	X	C
A	A							
B	B							
C	X							

Proof of Optimality

- Exchange argument

Proof of Optimality

- Exchange argument
- n^{th} decision: What to do at n^{th} time step.

Proof of Optimality

- Exchange argument
- n^{th} decision: What to do at n^{th} time step.
- Given schedule S that agrees with FITF for first n time steps, create schedule S' that agrees for n+1 and has no more cache misses.

Case 1: S agrees with FITF on step n+1

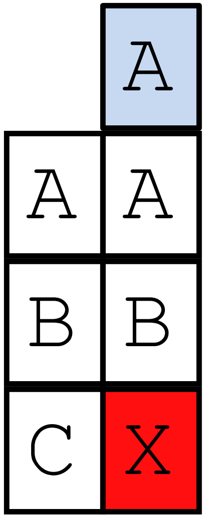
Nothing to do. S’ = S.

Case 2: S Makes Unnecessary Replacement

If S replaces some element of memory that was not immediately called for, put it off.

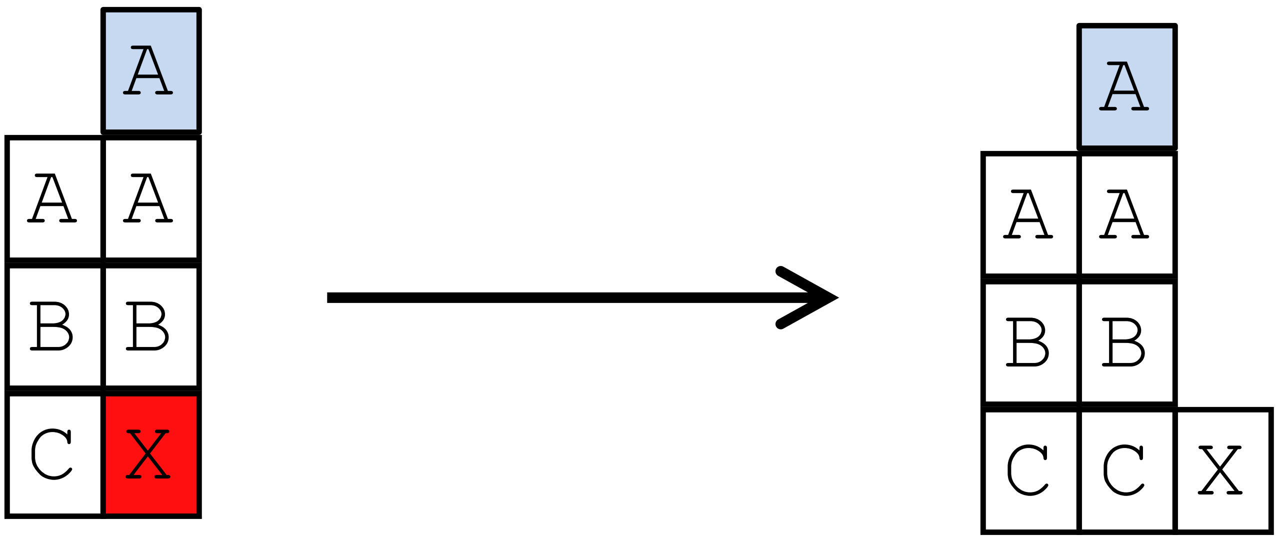
Case 2: S Makes Unnecessary Replacement

If S replaces some element of memory that was not immediately called for, put it off.



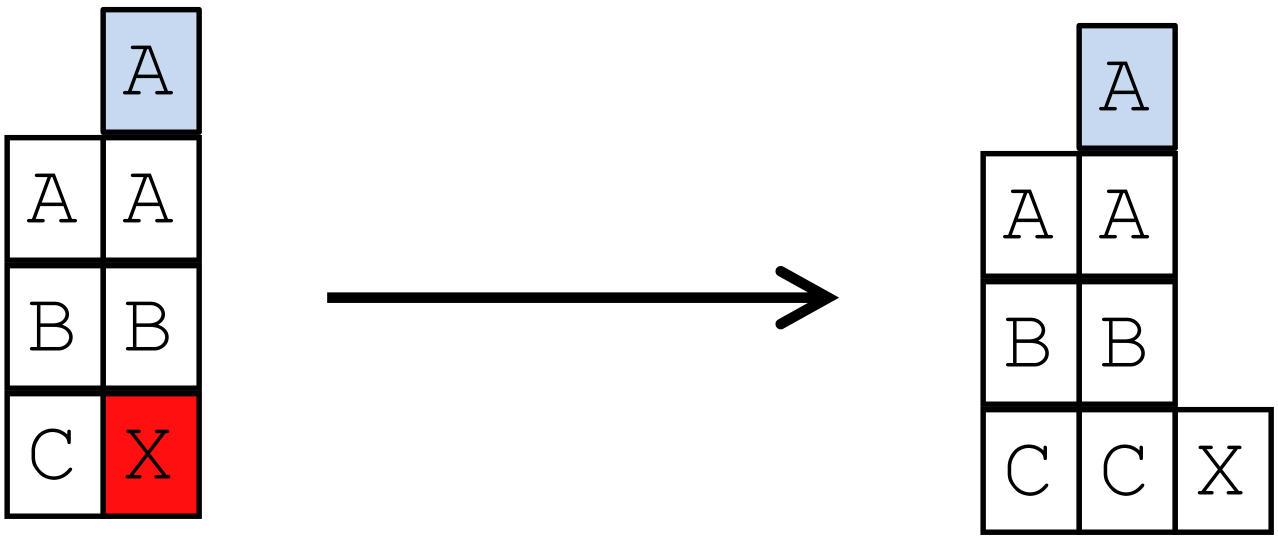
Case 2: S Makes Unnecessary Replacement

If S replaces some element of memory that was not immediately called for, put it off.



Case 2: S Makes Unnecessary Replacement

If S replaces some element of memory that was not immediately called for, put it off.



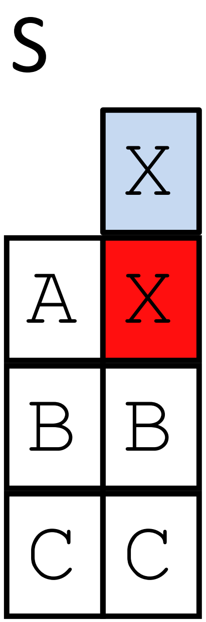
Can assume that S only replaces elements if there's a cache miss.

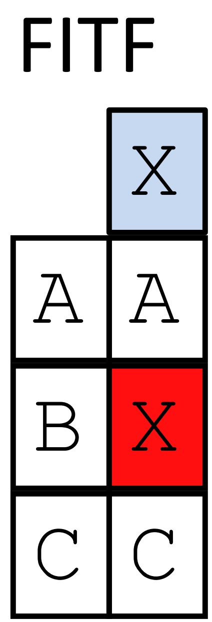
Case 3

The remaining case is that there is a cache miss at step n+1 and that S replaces the *wrong* thing.

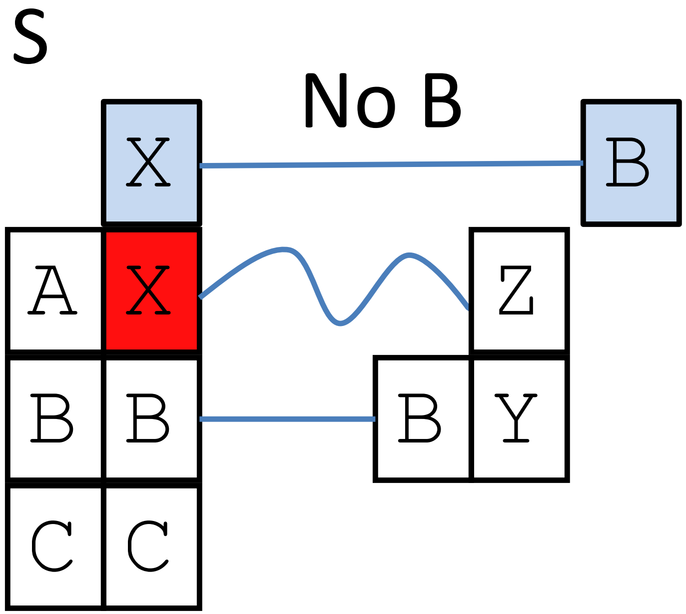
Case 3

The remaining case is that there is a cache miss at step n+1 and that S replaces the *wrong* thing.

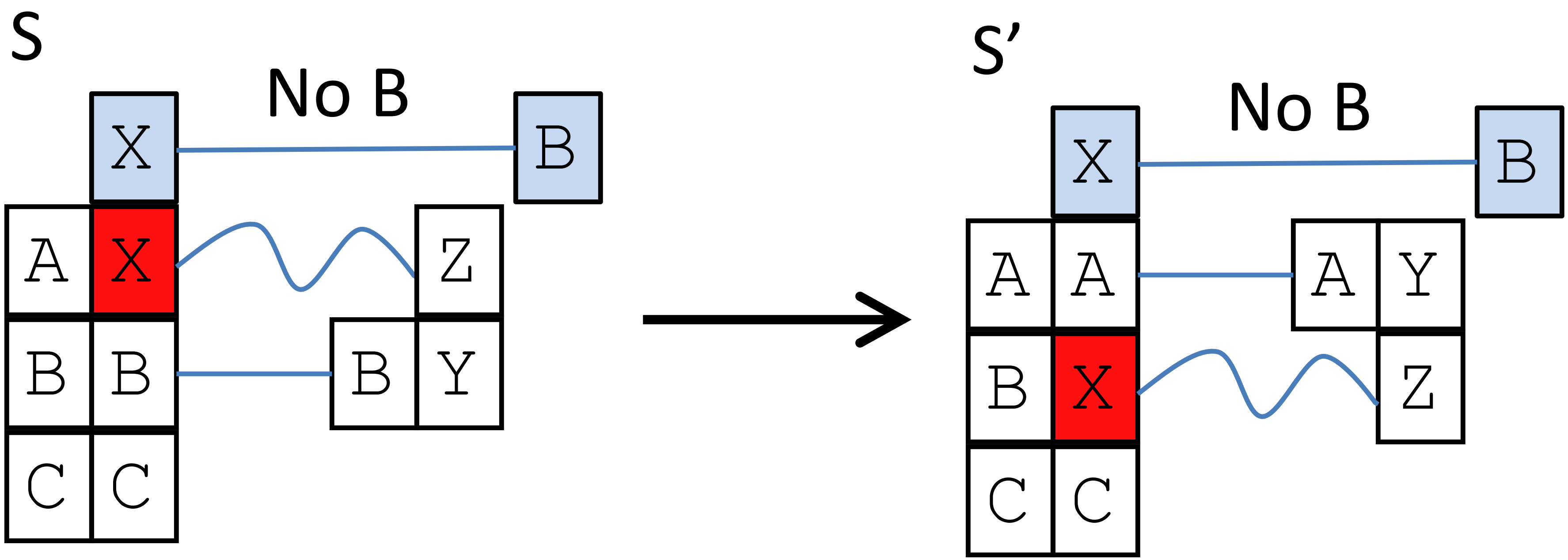




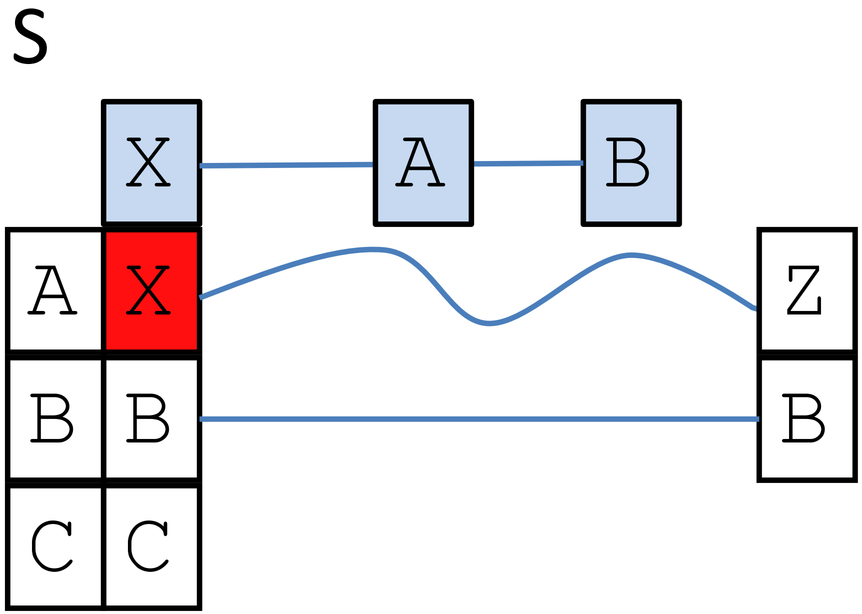
Case 3a: S throws out B before using it



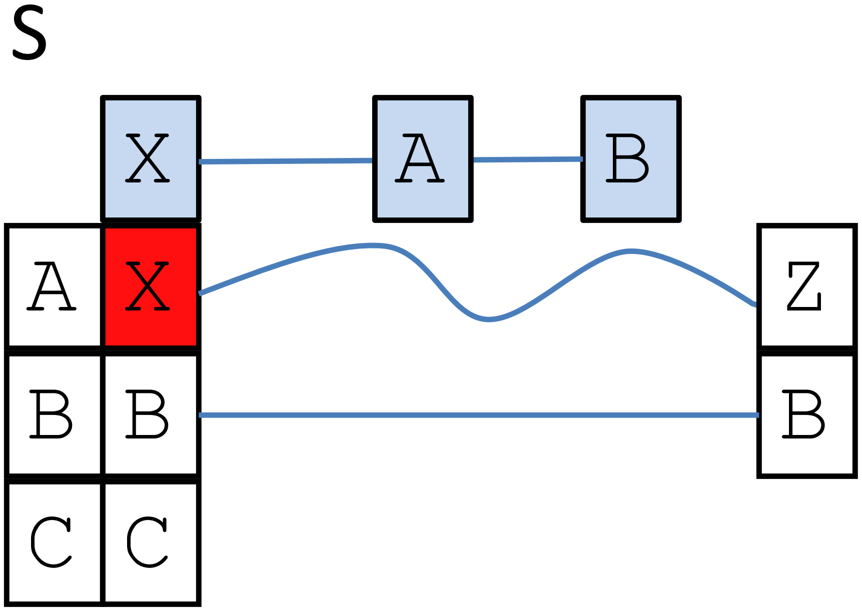
Case 3a: S throws out B before using it



Case 3b: S keeps B until it is used



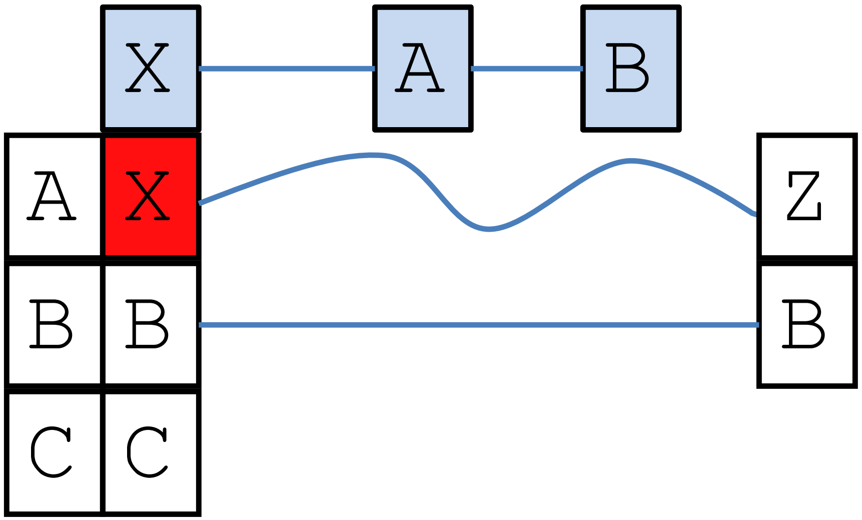
Case 3b: S keeps B until it is used



- B is FITF

Case 3b: S keeps B until it is used

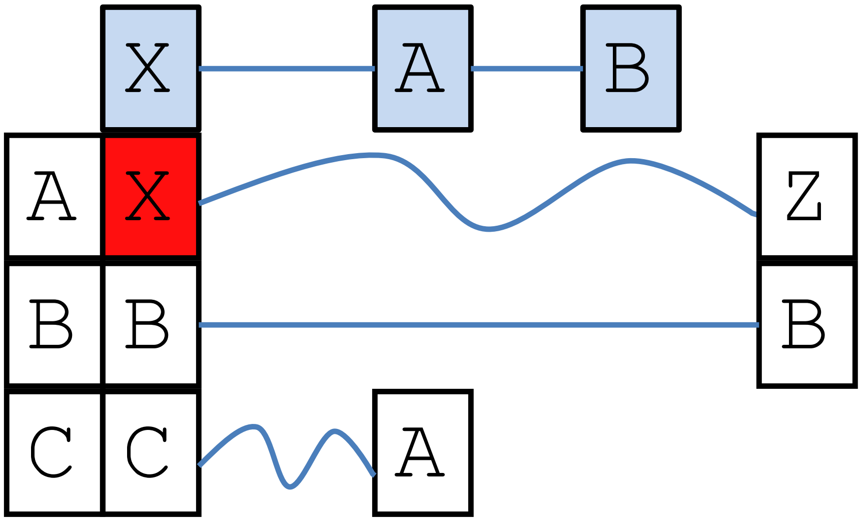
S



- B is FITF
- A is used sometime before B.

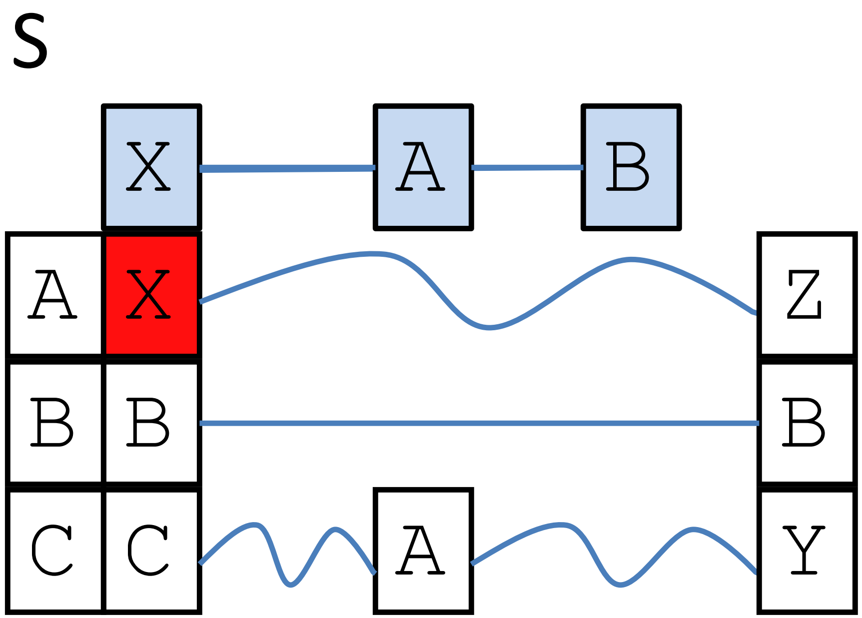
Case 3b: S keeps B until it is used

S

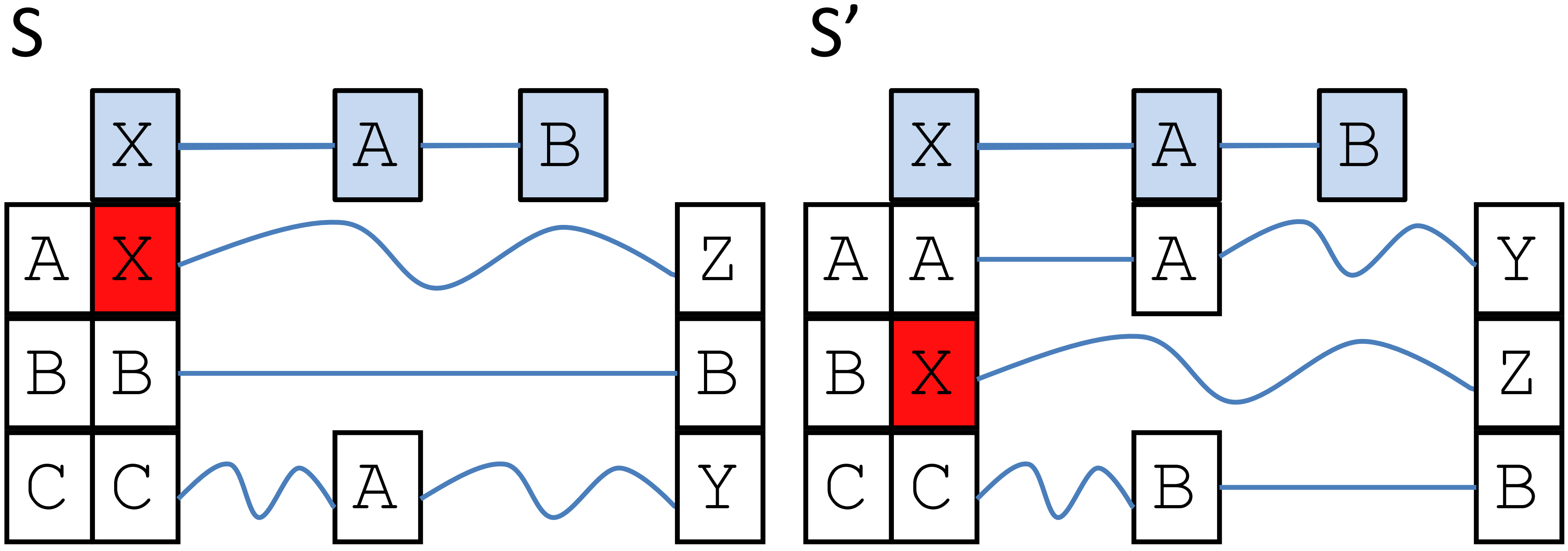


- B is FITF
- A is used sometime before B.
- A must be loaded into memory somewhere else.

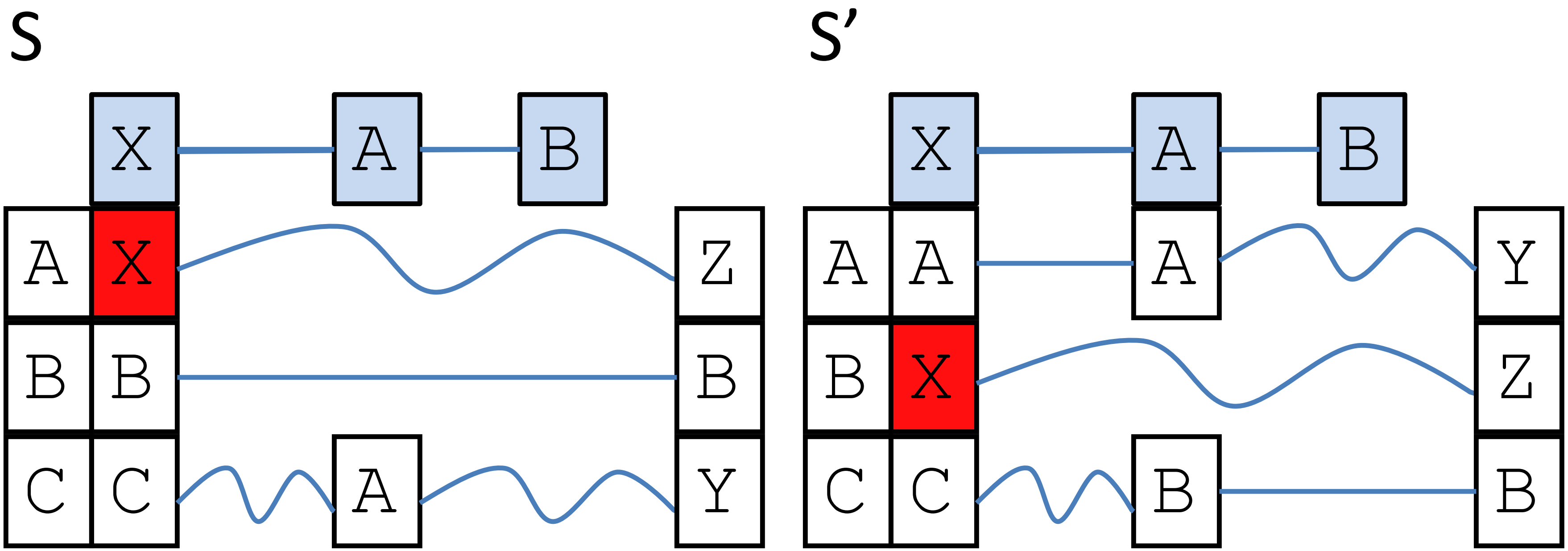
Case 3b: S keeps B until it is used



Case 3b: S keeps B until it is used



Case 3b: S keeps B until it is used



Instead of replacing A and then bringing it back, we can replace B and then bring it back.

Least Recently Used

Unfortunately, FITF requires that you know exactly what future memory accesses are needed. This makes it hard to use in practice.

Least Recently Used

Unfortunately, FITF requires that you know exactly what future memory accesses are needed. This makes it hard to use in practice.

Instead, people often throw out the Least Recently Used (LRU) memory location. This is *not* always optimal, but it can be shown to be competitive with the optimal.