Stack Algorithm Analytics

Peter J. Denning

Analytics

How to compute fault functions fast?

Analytics

- The simulation methods illustrated above are useful for finding the F(m) functions, but they are computationally intensive because they must be repeated for each value of m.

- The simulations show that the memory contents of LRU and MIN satisfy the inclusion property. When we increase m by 1, we add a row to the diagram without changing any of the rows already there.

- Can we exploit this for a simpler computational method?

- Yes. The F(m) functions of replacement policies with the inclusion property can be computed in a single pass over the address trace.

Inclusion Property

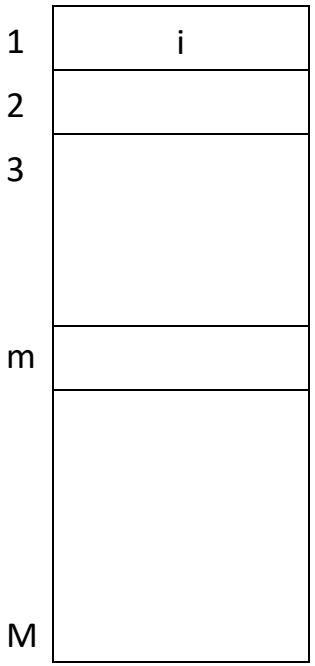
- At all times, the contents of m-page memory are a subset of the contents of (m+1)-page memory

"Adding a page to the memory allocation never makes things worse."

Stack Algorithms

- The pages of the program can, at each moment of time, be organized into a **single** ordered list such that the first m pages on the list are the contents of the m-page memory.
- Satisfies the inclusion property: the first m pages are a subset of the first m+1 pages.
- The list is called “the stack”. The stack represents the layers of inclusion as memory grows.

LRU Stack

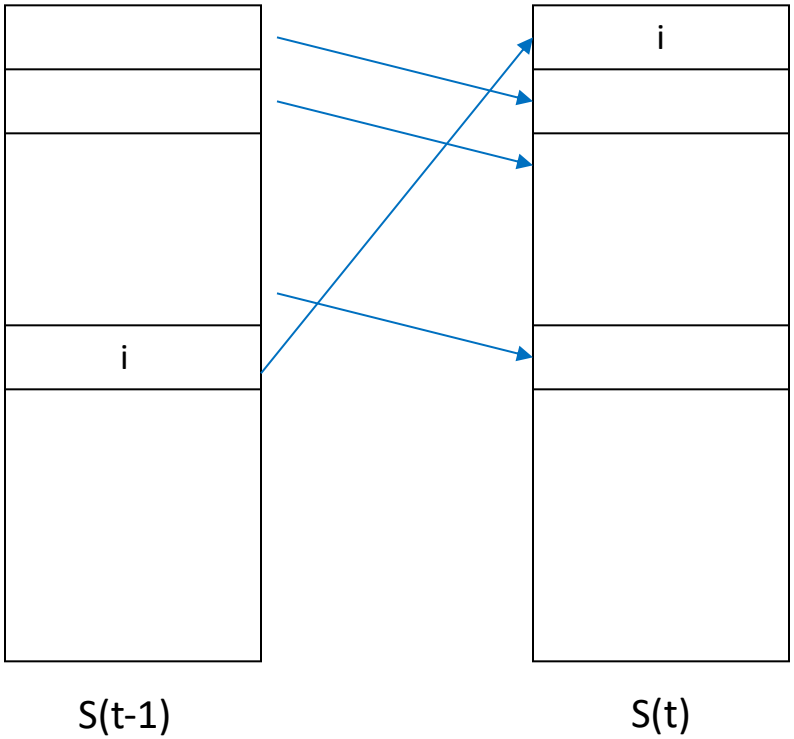


S(t)

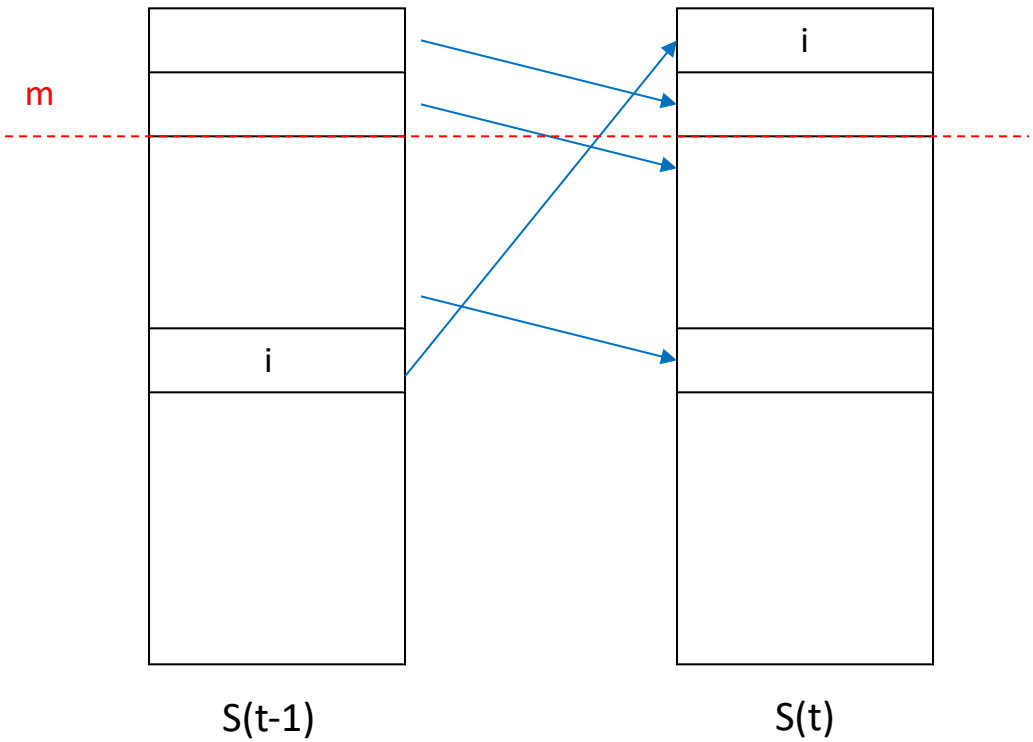
LRU stack lists **all** the pages of the program at time t in order of most to least recently used, from top to bottom.

The current page r(t)=i is always on top. (Why? Because the referenced page is always the most recent and is always in memory.)

The first m pages in the stack are the contents of the LRU memory at time t. (Why? Because the policy always removes the least recently used, leaving only the m most recently used.)

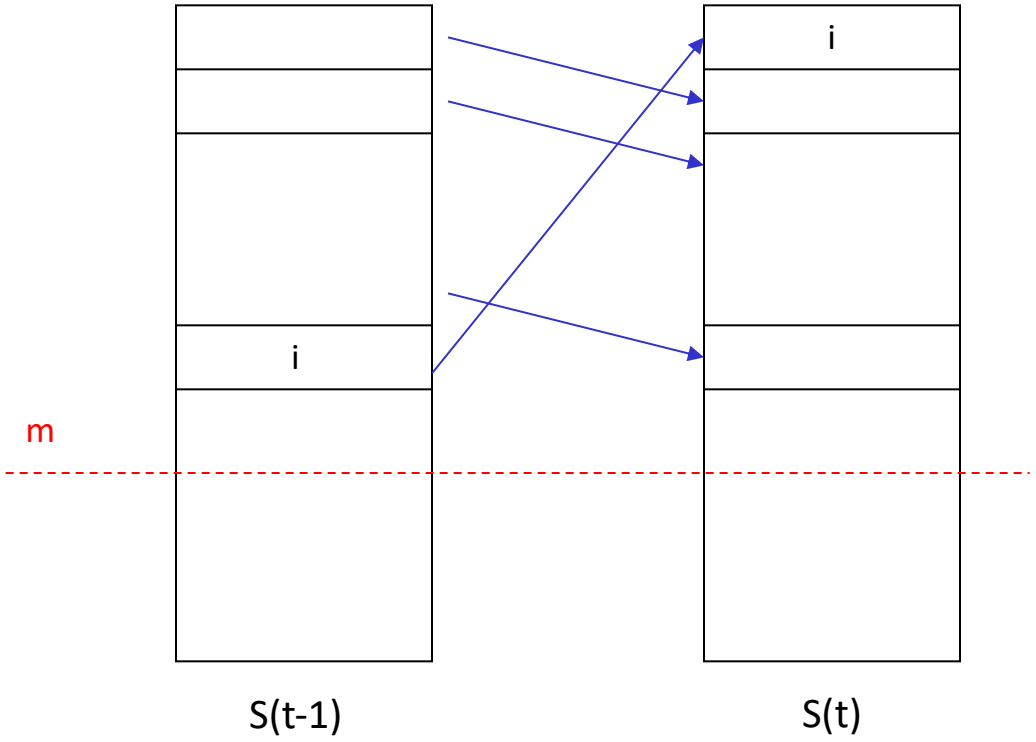


Look at the dynamics. If page i is referenced at time t, we can update the previous stack by moving i from its position to the top and pushing all the intervening pages down by 1 position. That preserves the LRU order of the stack.



Draw a horizonal red line slicing the stack at memory size m. All the pages above the line are in memory.

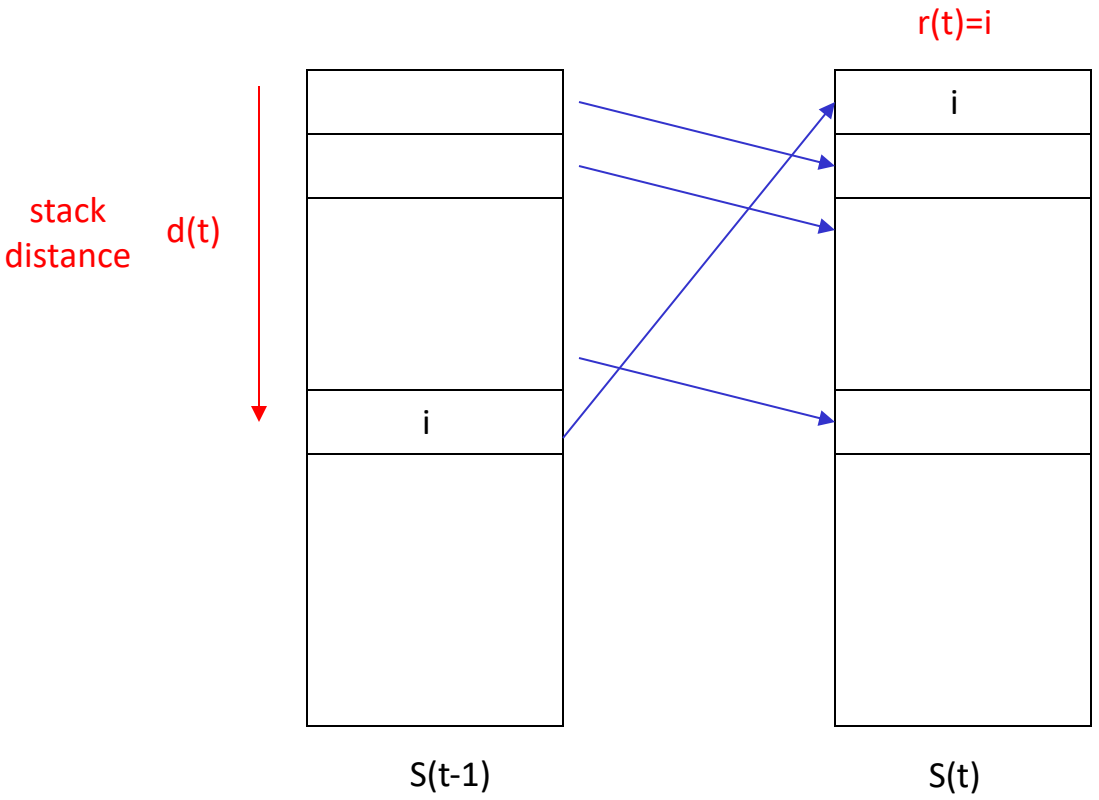
Notice that if the red line for m is above the referenced position, a page crosses the red line upward into the memory and a page crosses downward out of the memory. The upward moving page is the one brought by the page fault. The downward moving page is the LRU page among those in the m-page memory allocation.



If the memory size m is below the referenced position, no pages move up or down across the red line. This is because no page fault occurs for the referenced page, which is already in memory.

The pages below the referenced position are not reordered because their positions relative to page i are unchanged.

Stack distance



Stack distance d(t) is the position of r(t) in the previous stack.

d(t) can be as small 1 (for repeated reference) or as large as M.

From previous pictures we see that there is a page fault only when d(t)>m

Therefore the number of times d(t)>m is the total number of faults, F(m)

First references are special: treat them as distance = infinity because there is no finite size memory that avoids the faults

- How to count stack distances?
  - Counter c(k) counts number of times d(t)=k, for k=1,2,…,M
  - Define counter c(M+1) to catch all the first references, which have distances = infinity
  - Data collection: initially all counters 0; simulate the LRU stacks; at time t if d(t)=k, add 1 to counter c(k).
  - Then the number of times d(t)>m is simply the sum of the counters bigger than m

r(t):	1	2	3	4	1	2	5	1	2	3	4	5
	1	2	3	4	1	2	5	1	2	3	4	5
		1	2	3	4	1	2	5	1	2	3	4
stacks:			1	2	3	4	1	2	5	1	2	3
				1	2	3	4	4	4	5	1	2
							3	3	3	4	5	1
d(t):	x	x	x	x	4	4	x	3	3	5	5	5

In one pass track the LRU stack and its distances.

On first references, distance is infinite, denoted by x.

k	c(k)	F(k)
1	0	12
2	0	12
3	2	10
4	2	8
5	3	5
x	5	

Collect frequency histogram for all possible stack distances, k.

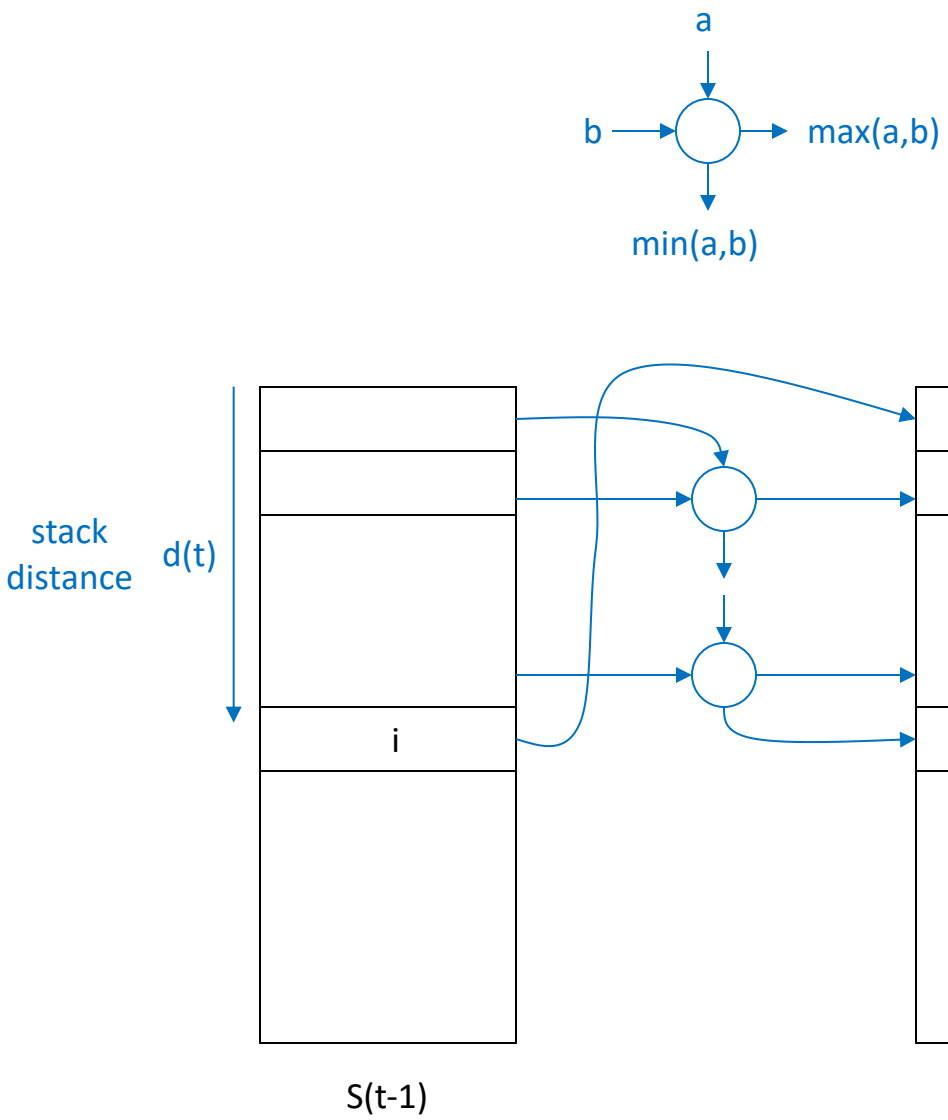
c(k) = no. distances equal to k

F(k) = sum of the counts for all distances **larger** than k.

General stack

- Now consider any replacement policy that operates from a priority rule. A priority rule is a list of all M pages at each time t, that orders the pages from most to least important at that time.
- If r(t) causes a page fault, use the priority rule to select the least important page for replacement.
- The diagram to follow shows the dynamics of a stack update when there is a priority rule. It is more complicated than LRU.

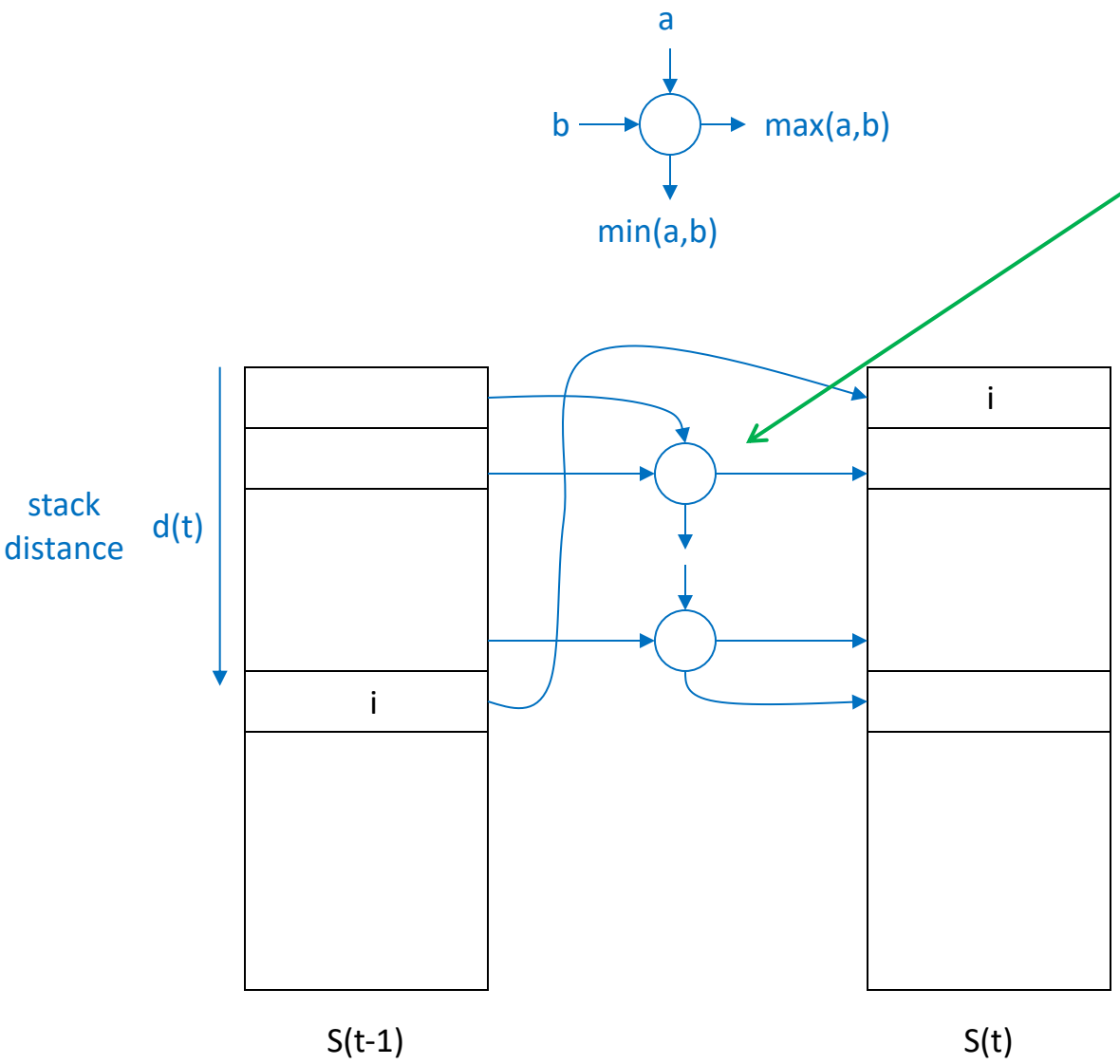
- Referenced page moves to top because it is always in memory, including when m=1.
- Any page **below** the current distance **does not move** because there is no fault in any of those memory sizes.
- Pages in between **move down or stay put**, but cannot move up because an up move would represent a page-in for a non referenced page.
- A page moves as far down as its priority permits – i.e., it settles in a position whose page was lower priority.



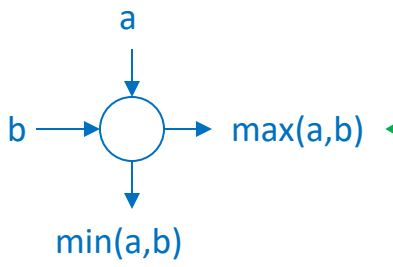
Referenced page always moves to the top because it must be in memory even when m=1.

Only two pages compete for each slot: the one already there and the one moving down. Most important takes the slot, least important moves down.

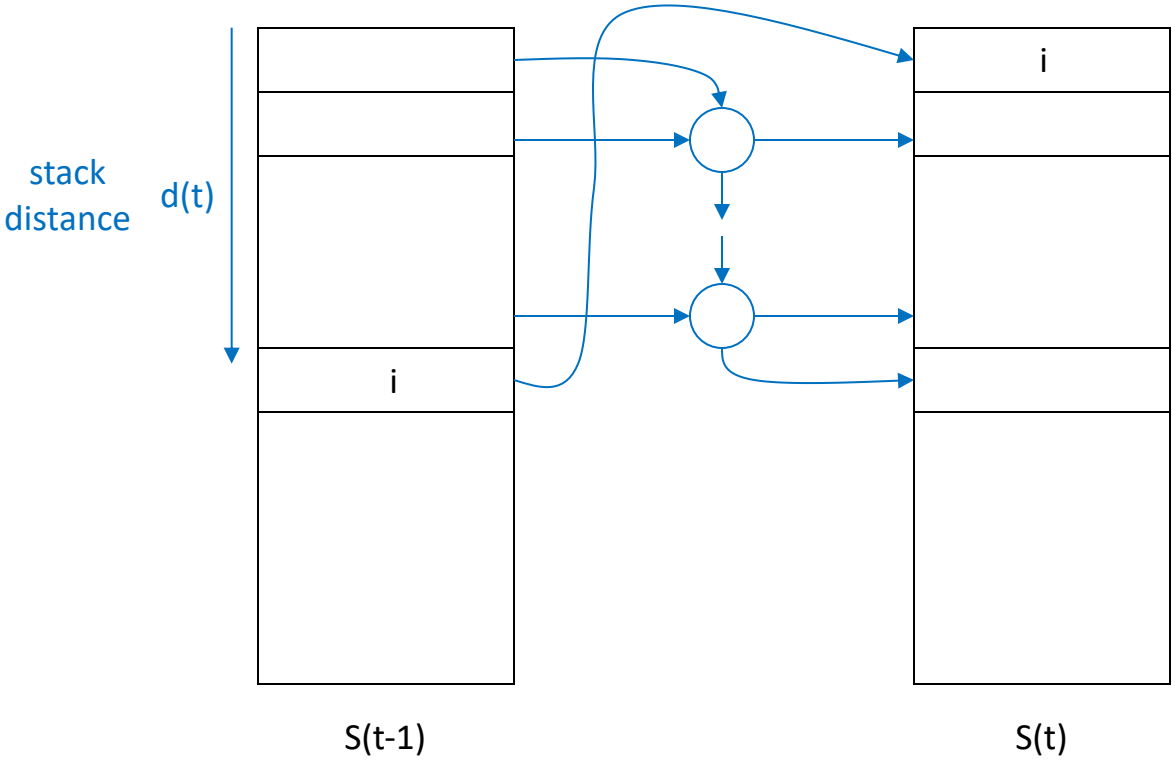
No page below the one at position d(t) moves. Why? There is no page fault if m>d(t) and therefore no up or down at those memory sizes.



Of the two previous top pages a and b, the lesser important min(a,b) moves down because it will be replaced from the m=2 memory. The more important max(a,b) remains.



For position k<d(t), a is the least important page in memory of size k-1. If it is less important than the page b at position k, it continues to move down, otherwise it stays at position k and b moves down.



- LRU is a stack algorithm; its priority rule at time t organizes the pages by increasing backward distance.
- MIN (optimal) is a stack algorithm; its priority rule at time t organizes the pages by increasing forward distance.
- Even the RAND (random) policy is a stack algorithm; its priority rule at time t is a random permutation of the pages.

MIN example

r(t):	1	2	3	4	1	2	5	1	2	3	4	5
	1	2	3	4	1	2	5	1	2	3	4	5
stacks:		1	1	1	4	1	1	5	5	5	5	4
			2	2	2	4	2	2	1	1	1	1
				3	3	3	3	3	3	2	2	2
							4	4	4	4	3	3
d(t):	x	x	x	x	2	3	x	2	3	4	5	2

This diagram shows the behavior of MIN on the example address trace.

After its last reference, treat page as least important among those in memory (future reference time infinite). It no longer influences MIN.

Highlight shows example of an update to MIN stack.

r(7)=5 is first reference, thus d(7)=x, 5 goes in position 1

Now (2,1) compete for position 2; 1 wins because it is sooner in future, 2 loses

Now (2,4) compete for position 3; 2 wins, 4 loses

Now (4,3) compete for position 4; 3 wins, 4 loses

Now 4 goes into position 5, which was previously unoccupied

r(t):	1	2	3	4	1	2	5	1	2	3	4	5
	1	2	3	4	1	2	5	1	2	3	4	5
stacks:		1	1	1	4	1	1	5	5	5	5	4
			2	2	2	4	2	2	1	1	1	1
				3	3	3	3	3	3	2	2	2
							4	4	4	4	3	3
d(t):	x	x	x	x	2	3	x	2	3	4	5	2

Highlight shows example of an update to MIN stack.

r(9)=2 is not first reference, thus 2 goes in position 1

Now (1,5) compete for position 2; 5 wins, 1 loses

Now 1 goes into position 3, which 2 just vacated

Trajectory-Based MIN Distance Calculation

There is another way to do this, based on tracking trajectories of pages in the MIN stack

Red shows a trajectory of page 1 prior to its first reuse

Blue shows a trajectory of page 2 prior to its first reuse

Trajectory is a stack path between successive references to a page. At each step the page stays in the same position or moves lower; never up. Page moves up only when it is reused.

r(t):	1	2	3	4	1	2	5	1	**2**	3	4	5
	1	2	3	4	1	2	5	**1**	2	3	4	5
stacks:		1	1	1	4	1	1	**5**	5	5	5	4
			2	2	2	4	2	**2**	1	1	1	1
				3	3	3	3	**3**	3	2	2	2
							4	**4**	4	4	3	3
d(t):	x	x	x	x	2	3	x	2	3	4	5	2

Algorithm for trajectories:

Write entire address trace into row 1. Rest of tableau is blank.

For each t consider r(t)=p

If this is first reference to p do nothing

Otherwise, go back to prior reference to p, and for each time up to t

Copy p to next column in highest blank position without moving p up

Position of p at t-1 is d(t)

If prior reference is at time t', algorithm places p at its proper position in columns t'+1 through t-1.

Let's illustrate and then we'll prove it works.

	1	2	3	4	1	2	5	1	2	3	4	5
t=4:	x	x	x	x								

Start by copying address trace into stack position 1. The five rows of tableau are the stack positions. Rest of tableau is blank. Distances at the bottom. First four distances are x (infinite)

	1	2	3	4	1	2	5	1	2	3	4	5
		1	1	1								
t=5:	x	x	x	x	2							

Prior reference to p=1 at time 1. Build trajectory from time 2 to time 4. Copy p=1 into highest blank space of column 2 (position 2). Repeat for columns 3 and 4. Its position at t=4 is the stack distance d(5)=2.

	1	2	3	4	1	2	5	1	2	3	4	5
		1	1	1								
			2	2	2							
t=6:	x	x	x	x	2	3						

Copy p=2 into highest blank space of column 3 (position 3). Repeat for columns 4 and 5. Its position at t=5 is the stack distance d(6)=3.

	1	2	3	4	1	2	5	1	2	3	4	5
		1	1	1								
			2	2	2							
t=7:	x	x	x	x	2	3	x					

Page 5 is first reference. It's distance is d(7)=x. No finite trajectory to record.

	1	2	3	4	1	2	5	1	2	3	4	5
		1	1	1		1	1					
			2	2	2							
t=8:	x	x	x	x	2	3	x	2				

Page 1 prior reference at time 5. Copy p=1 into highest blank space of next column. Repeat. At time 7 its position is 2, giving d(8)=2.

	1	2	3	4	1	2	5	1	2	3	4	5
		1	1	1		1	1					
			2	2	2		2	2				
t=9:	x	x	x	x	2	3	x	2	3			

Page2 prior reference at time 6. Copy p=2 into first vacant position 3 of next column. Note skip past position 2, which is already occupied. Repeat. At time 8 its position is 3, giving d(9)=3.

	1	2	3	4	1	2	5	1	2	3	4	5
		1	1	1		1	1					
			2	2	2		2	2				
				3	3	3	3	3	3			
t=10:	x	x	x	x	2	3	x	2	3	4		

Page 3 prior reference at time 3. At time 4 page 3 drops to position 4, the highest vacant. It propagates at position 4 to time 9. Thus d(10)=4.

	1	2	3	4	1	2	5	1	2	3	4	5
		1	1	1	4	1	1					
			2	2	2	4	2	2				
				3	3	3	3	3	3			
							4	4	4	4		
t=11:	x	x	x	x	2	3	x	2	3	4	5	

Page 4 prior reference at time 4. At time 5 page 4 drops to position 2, the highest vacant. Then to position 3, then to position 5. Propagates at position 5 to time 10. Thus d(11)=5.

	1	2	3	4	1	2	5	1	2	3	4	5
		1	1	1	4	1	1	5	5	5	5	
			2	2	2	4	2	2				
				3	3	3	3	3	3			
							4	4	4	4		
t=12:	x	x	x	x	2	3	x	2	3	4	5	2

Page 5 prior reference at time 7. Copy p=5 into first vacant position 2 of next column. Repeat. At time 11 its position is 2, giving d(12)=2.

THAT'S IT. Trajectories all filled in. Blank spots don't matter for the distances.

Why Trajectory Algorithm Works

NOTATION: Notation D(p,t) is the position of page p in the stack at time t. Because no page can move up stack unless referenced, $D(p,t-1) \leq D(p,t)$ except when r(t)=p in which case D(p,t)=1. A trajectory is the sequence of positions D(p,u) where t'<u<t, and t' and t are two successive uses of the page p. Let T(t) be the trajectory for the page p=r(t).

Inductive proof. Assume all trajectories correct prior to time t. Construct trajectory at time t and show it is correct. Basis: D(p,1)=1 where p=r(1).

The trajectory T(t) covers times t', …, t. Consider time u, t'<u<t. Suppose D(p,u)=j. If position (j,u+1) is blank, no trajectory prior to time t has used that position. That means a future trajectory for a page q may pass through that position. Page q must therefore compete with p for position (j,u+1) and q loses because its future reuse time is later than t. Therefore p correctly occupies position (j,u+1).

If position (j,u+1) is already occupied by page q, then then p and q must compete. Because q belongs to a trajectory already in place, q is used between before t, which means it precedes next reference to p, and thus wins. Therefore p must move down. The same argument applies for each down until a blank is found, when previous case applies.

An Interesting Consequence

A consequence of the Trajectory algorithm is that **MIN distances can be computed in real time** without actually seeing the future. As soon as we see p=r(t) there is enough information in the tableau to fill in its correct trajectory and output the MIN stack distance d(t).

Isn't that amazing?

This does not mean we can implement MIN. MIN still needs to see the future to determine the priorities of pages in memory.

Les Belady, inventor of MIN, got a patent for a device that could be part of the MMU of the CPU, and would output the MIN fault count at each time t. He used a variation of the algorithm discussed here. He envisioned that the device would allow operating system engineers to know how far their actual paging algorithm is from the ideal. As far as I know, IBM never built that circuit.

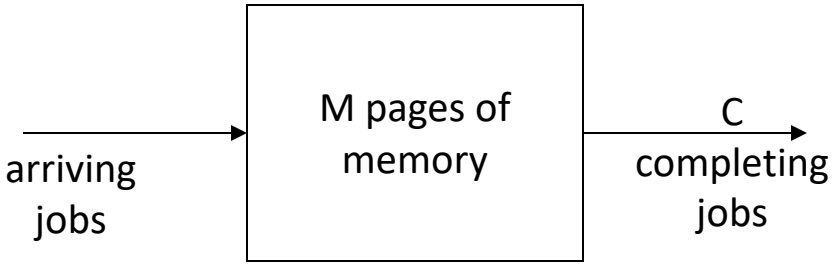
Advanced Topic:
What's the right size memory allocation?

Use system throughput as measure of performance.

Footprint and system throughput

- Footprint of a process
  - Space-time accumulated by a process
  - Time measured in ticks (1 clock tick per memory access)
  - Process charged 1 "page-tick" for each clock tick per page held in memory
  - A page-tick is unit of "rent" for holding memory space
  - Real time footprint ≠ virtual time footprint because page fault delays add to real time footprint but not virtual time footprint

memory space time law



Observe system for T seconds:

Total space-time available = MT

Mean footprint per job Y = MT/C

System throughput X = C/T

Thus M = XY

Throughput maximum when footprint minimum.

What memory allocation minimizes a process's footprint?

ESTIMATE OF FOOTPRINT

F(m) = k means k faults observed in observation period T with memory allocation m

footprint without page faults = mT

footprint of one page fault = mD (D = main-secondary access-time ratio, typically 10^6)

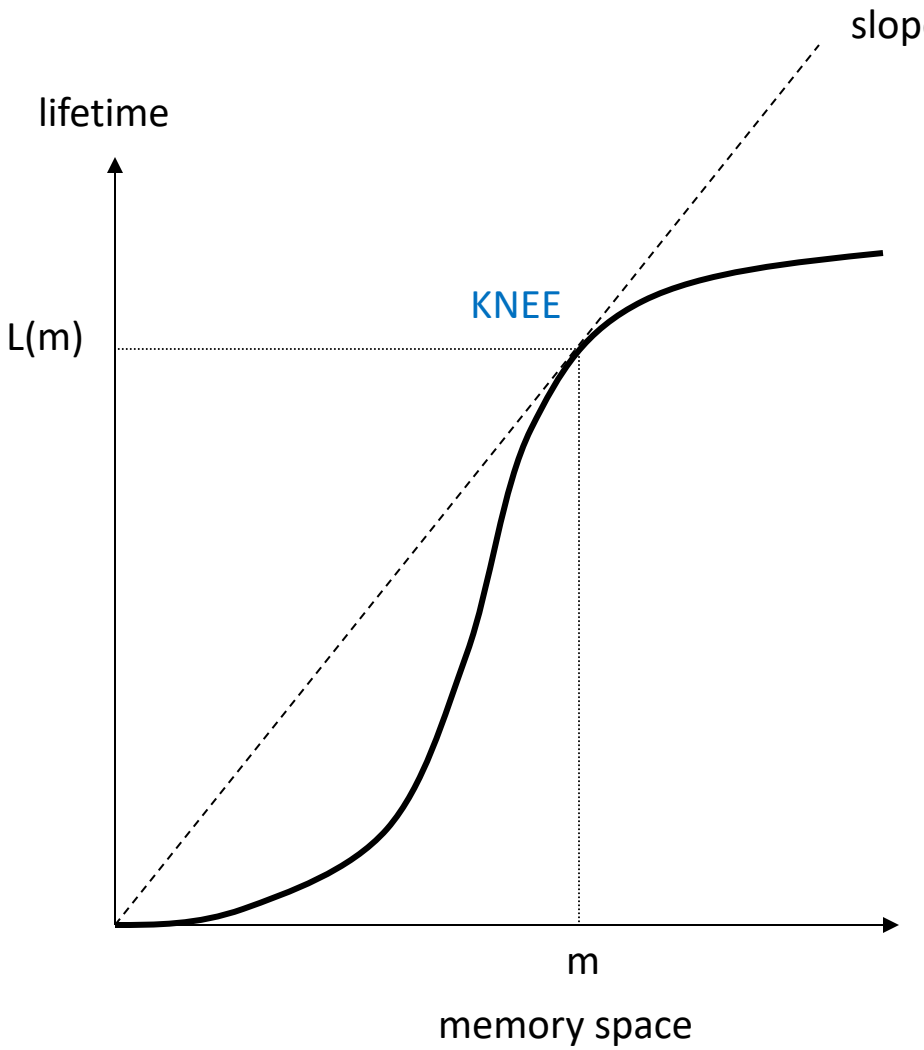
footprint for all page faults = mDF(m)

Let L(m) = lifetime = mean time between faults = T/F(m)

Therefore Y(m) = total footprint
= mT+mDF(m)
= mT+DmT/L(m))
= mT(1+D/L(m))

If D/L(m) much larger than 1 (typical), Y(m) = (TD)(m/L(m)), which is minimum when L(m)/m is maximum.

This occurs when m is at the lifetime curve KNEE (next page).



Lifetime curve L(m) = mean virtual time between page faults when m pages of memory are allocated = T/F(m)

Empirically, lifetime curves have an S-shape.

Slope of line from origin to any point m on curve is L(m)/m.

Knee, an inflection point, maximizes L(m)/m, which is the same as minimizing mF(m).

Therefore: choose m near the knee. This choice will minimize its footprint and maximize system throughput.

Not easy to implement.

finis