# Page Replacement

- Local vs Global replacement
  - Local: victim chosen from frames of process experiencing page fault
    - fixed allocation per process
  - Global: victim chosen from frames allocated to any process
    - variable allocation per process
- Many replacement policies
  - Random, FIFO, LRU, Clock, Working set, etc.
- Goal: minimizing number of page faults



# Checkin with one condition variable

```
self.allCheckedIn = Condition(self.lock)

def checkin():
  with self.lock:
    nArrived++
    if nArrived < nThreads:
      while nArrived < nThreads:
        allCheckedIn.wait()
    else:
      allCheckedIn.broadcast()
      nArrived = 0
```

What's wrong with this?



# Checkin: 2 condition variables

```
                    self.allCheckedIn = Condition(self.lock)
                    self.allLeaving = Condition(self.lock)
def checkin():
    nArrived++
    if nArrived < nThreads:          // not everyone has checked in
        while nArrived < nThreads:
            allCheckedIn.wait()      // wait for everyone to check in
    else:
        nLeaving = 0                 // this thread is the last to arrive
        allCheckedIn.broadcast()     // tell everyone we're all here!

    nLeaving++
    if nLeaving < nThreads:          // not everyone has left yet
        while nLeaving < nThreads:
            allLeaving.wait()        // wait for everyone to leave
    else:
        nArrived = 0                 // this thread is the last to leave
        allLeaving.broadcast()       // tell everyone we're outta here!
```

# How do we pick a victim?

- We want:
  - low page fault-rate
  - page faults as inexpensive as possible
- We need:
  - a way to compare the relative performance of different page replacement algorithms
  - some absolute notion of what a "good" page replacement algorithm should accomplish

# Comparing Page Replacement Algorithms

- Record a trace of the pages accessed by a process
  - E.g. 3,1,4,2,5,2,1,2,3,4 (or c,a,d,b,e,b,a,b,c,b)
- Simulate behavior of page replacement algorithm on trace
- Record number of page faults generated



# Optimal Page Replacement

- Replace page needed furthest in future

| Time | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Requests | | | c | a | d | b | e | b | a | b | c | d |
| Page Frames | 0 | a | a | a | a | a | a | a | a | a | a | d |
| | 1 | b | b | b | b | b | b | b | b | b | b | b |
| | 2 | c | c | c | c | c | c | c | c | c | c | c |
| | 3 | d | d | d | d | d | e | e | e | e | e | e |
| Faults | | | | | | | X | | | | | X |
| Time page needed next | | | | | | a = 7<br>b = 6<br>c = 9<br>d = 10 | | | | | a = ∞<br>b = 11<br>c = 13<br>e = 15 | |

b d c b e



# FIFO Replacement

- Replace pages in the order they come into memory

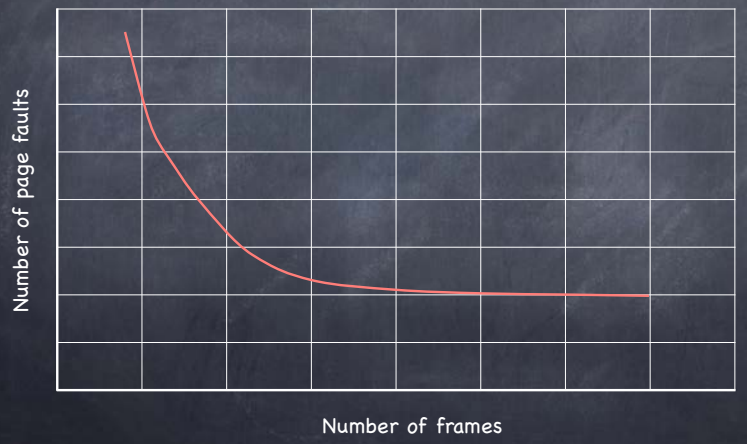
Assume:

a @ -3
b @ -2
c @ -1
d @ 0

| Time | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Requests | | | c | a | d | b | e | b | a | b | c | d |
| Page Frames | 0 | a | a | a | a | a | e | e | e | e | e | d |
| | 1 | b | b | b | b | b | b | b | a | a | a | a |
| | 2 | c | c | c | c | c | c | c | c | b | b | b |
| | 3 | d | d | d | d | d | d | e | e | e | c | c |
| Faults | | | | | | | X | | X | X | X | X |



# + Frames
# - Page Faults



# For example...

FIFO

| Time | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Request | | | a | b | c | d | a | b | e | a | b | c | d | e |
| Page Frames | 0 | | a | a | a | d | d | d | e | e | e | e | e | e |
| | 1 | | | b | b | b | a | a | a | a | a | c | c | c |
| | 2 | | | | c | c | c | b | b | b | b | b | d | d |
| Faults | | | X | X | X | X | X | X | X | | | X | X | |

- 3 frames - 9 page faults!



# Belady's Anomaly

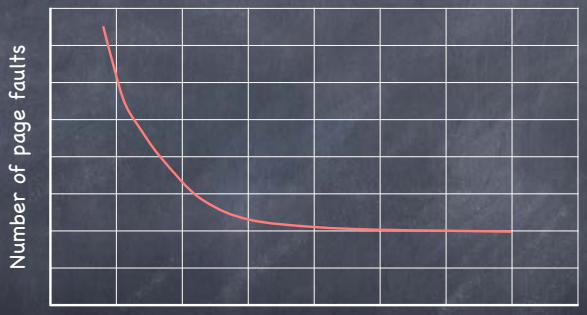
FIFO

| Time | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Request | | | a | b | c | d | a | b | e | a | b | c | d | e |
| Page Frames | 0 | | a | a | a | a | a | a | e | e | e | e | d | d |
| | 1 | | | b | b | b | b | b | a | a | a | a | a | e |
| | 2 | | | | c | c | c | c | c | b | b | b | b | b |
| | 3 | | | | | d | d | d | d | d | d | c | c | c |
| Faults | | | X | X | X | X | | | X | X | X | X | X | X |

- 4 frames - 10 page faults!



# + Frames
# - Page Faults?




- Yes, but only for stack page replacement policies
  - set of pages in memory with n frames is a subset of set of pages in memory with n+1 frames



# Locality of Reference

- If a process access a memory location, then it is likely that
  - the same memory location is going to be accessed again in the near future (temporal locality)
  - nearby memory locations are going to be accessed in the future (spatial locality)
- 90% of the execution of a program is sequential
- Most iterative constructs consist of a relatively small number of instructions

# LRU: Least Recently Used

- Replace page not referenced for the longest time

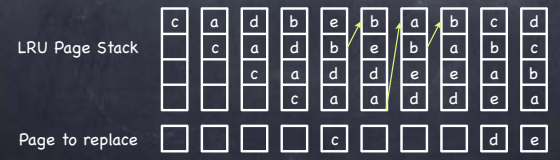
| Time | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Requests | | | c | a | d | b | e | b | a | b | c | d |
| Page Frames | 0 | a | a | a | a | a | a | a | a | a | a | a |
| | 1 | b | b | b | b | b | b | b | b | b | b | b |
| | 2 | c | c | c | c | c | e | e | e | e | e | d |
| | 3 | d | d | d | d | d | d | d | d | d | c | c |
| Faults | | | | | | | X | | | | X | X |
| Time page last used | | | | | | a = 2<br>b = 4<br>c = 1<br>d = 3 | | | | a = 7<br>b = 8<br>e = 5<br>d = 3 | a = 7<br>b = 8<br>e = 5<br>c = 9 | |



# Implementing LRU

- Maintain a "stack" of recently used pages

| Time | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Requests | | | c | a | d | b | e | b | a | b | c | d |
| Page Frames | 0 | a | a | a | a | a | a | a | a | a | a | a |
| | 1 | b | b | b | b | b | b | b | b | b | b | b |
| | 2 | c | c | c | c | c | e | e | e | e | e | d |
| | 3 | d | d | d | d | d | d | d | d | d | c | c |
| Faults | | | | | | | X | | | | X | X |






# Implementing LRU

- Add a (64-bit) timestamp to each page table entry
  - HW counter incremented on each instruction
  - Page table entry timestamped with counter when referenced
  - Replace page with lowest timestamp



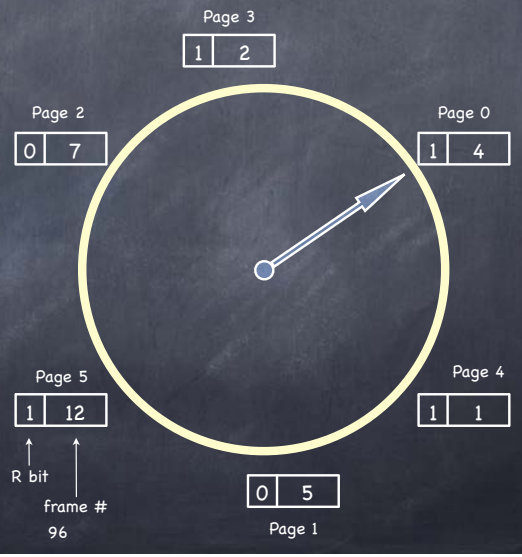
# Implementing LRU

- Add a (64-bit) timestamp to each page table entry
  - HW counter incremented on each instruction
  - Page table entry timestamped with counter when referenced
  - Replace page with lowest timestamp
- Approximate LRU through aging
  - keep a k-bit tag in each table entry
  - at every "tick": i) Shift tag right one bit
    ii) Copy Referenced (R) bit in tag
    iii) Reset Refereced bits to 0
  - If needed, evict page with lowest tag

| | R bits at Tick 0 | R bits at Tick 1 | R bits at Tick 2 | R bits at Tick 4 | R bits at Tick 5 |
|---|---|---|---|---|---|
| | 10111 | 11001 | 11010 | 10001 | 01100 |
| Page 0 | 10000000 | 11000000 | 11100000 | 11110000 | 01111000 |
| Page 1 | 00000000 | 10000000 | 11000000 | 01100000 | 10110000 |
| Page 2 | 10000000 | 01000000 | 00100000 | 00100000 | 10001000 |
| Page 3 | 00000000 | 00000000 | 10000000 | 01000000 | 00100000 |
| Page 4 | 10000000 | 11000000 | 01100000 | 10110000 | 01011000 |
| Page 5 | 10000000 | 01000000 | 10100000 | 01010000 | 00101000 |

# The Clock Algorithm

- Organize pages in memory as a circular list
- When page is referenced, set its reference bit R to 1
- On page fault
  - if R = 0: evict the page
  - if R = 1: clear R
  - advance hand



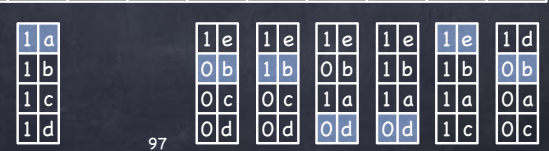



# Clock Page Replacement

| Time | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Requests | | | c | a | d | b | e | b | a | b | c | d |
| Page Frames | 0 | a | a | a | a | a | e | e | e | e | e | d |
| | 1 | b | b | b | b | b | b | b | b | b | b | b |
| | 2 | c | c | c | c | c | c | c | a | c | a | a |
| | 3 | d | d | d | d | d | d | d | d | d | c | c |
| Faults | | | | | | | X | | X | | X | X |

Page table entries for resident pages

Hand clock:

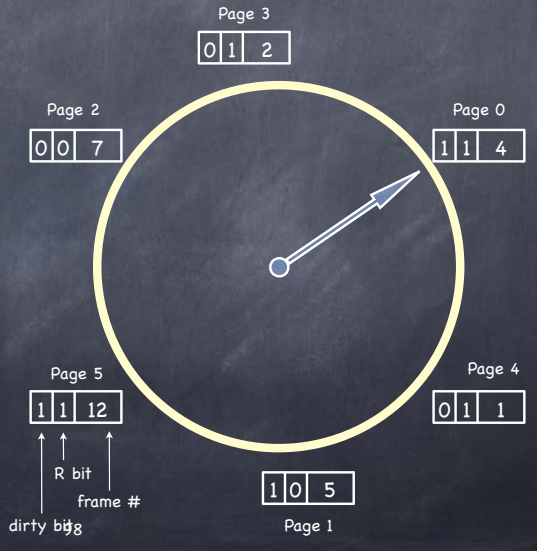





# The Second Chance Algorithm

- Dirty pages get "second chance" before eviction
  - replacing dirty pages is expensive!

| Before Clock sweep | | After Clock sweep | |
|---|---|---|---|
| dirty | R | dirty | R |
| 0 | 0 | replace page | |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 |



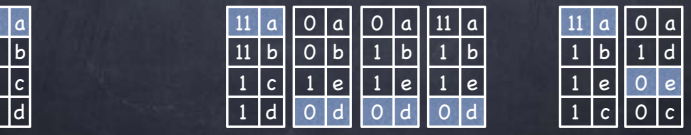



# Second Chance Page Replacement

| Time | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Requests | | | c | $a^w$ | d | $b^w$ | e | b | $a^w$ | b | c | d |
| Page Frames | 0 | a | a | a | a | a | a | a | a | a | a | a |
| | 1 | b | b | b | b | b | b | b | b | b | b | d |
| | 2 | c | c | c | c | c | e | e | e | e | e | e |
| | 3 | d | d | d | d | d | d | d | d | d | c | c |
| Faults | | | | | | | X | | | | X | X |

Page table entries for resident pages

Hand clock:



# Local vs. Global Page Replacement

- Local: Select victim only among allocated frames
  - Equal or proportional frame allocation
- Global: Select any free frame, even if allocated to another process
  - Processes have no control over their own page fault rate



# Brother, can you spare a frame?

FIFO

| Time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Requests | | a | b | c | d | a | b | c | d | a | b | c | d |
| 0 | a | a | a | a | d | d | d | c | c | c | b | b | b |
| 1 | b | b | b | b | b | a | a | a | d | d | d | c | c |
| 2 | c | c | c | c | c | c | b | b | b | a | a | a | d |
| Faults | | | | | X | X | X | X | X | X | X | X | X |



# Brother, can you spare a frame?

FIFO

| Time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Requests | | a | b | c | d | a | b | c | d | a | b | c | d |
| 0 | a | a | a | a | a | a | a | a | a | a | a | a | a |
| 1 | b | b | b | b | b | b | b | b | b | b | b | b | b |
| 2 | c | c | c | c | c | c | c | c | c | c | c | c | c |
| 3 | - | - | - | - | d | d | d | d | d | d | d | d | d |
| Faults | | | | | X | | | | | | | | |

So, what's wrong with global replacement?



# Memory as a Cache

- Demand paging enables frames to cache part of a process VA space
- If the cache is large enough, hit ratio is high
  - few page faults
- What if there aren't enough frames to go around?
  - should decrease degree of multiprogramming
    - swapped out process can then release their frames

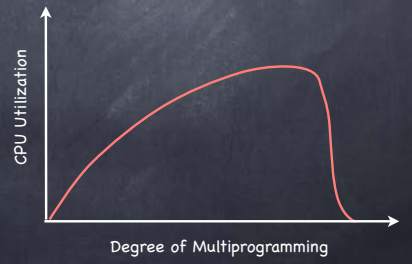
# Instead...

- When not enough frames...
  - high page fault rate
  - low CPU utilization
  - OS may increase degree of multiprogramming!



# Instead...

- When not enough frames...
  - high page fault rate
  - low CPU utilization
  - OS may increase degree of multiprogramming!
- Thrashing
  - process spends all its time swapping pages in and out






# Locality of Reference

- If a process access a memory location, then it is likely that
  - the same memory location is going to be accessed again in the near future (temporal locality)
  - nearby memory locations are going to be accessed in the future (spatial locality)
- 90% of the execution of a program is sequential
- Most iterative constructs consist of a relatively small number of instructions



# Tracking Locality

- When a process executes it moves from locality (set of pages used together) to locality
  - the size of the process' locality (a.k.a. its working set) can change over time
- Goal: track the size of the process' working set, dynamically acquiring and releasing frames as necessary

# The Working Set Model

- Choose Δ page references as WS window
- $WSS_i$ = # of distinct pages referenced by $p_i$ in latest Δ
  - Δ too small does not cover locality
  - Δ too large covers many localities
- Thrashing if $\sum_i WSS_i$ > # frames
  - if so, suspend one of the processes
- If enough free frames, increase degree of multiprogramming



# WS Page Replacement
# Δ = 4

| Time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Requests | | c | c | d | b | c | e | c | e | a | d |
| Pages in Memory: Page a | $t=0$ ● | | | | | | | | | | |
| Page b | | | | | | | | | | | |
| Page c | | | | | | | | | | | |
| Page d | $t=-1$ ● | | | | | | | | | | |
| Page e | $t=-2$ ● | | | | | | | | | | |
| Faults | | | | | | | | | | | |



# WS Page Replacement
# Δ = 4

| Time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Requests | | c | c | d | b | c | e | c | e | a | d |
| Pages in Memory: Page a | $t=0$ ● | ● | ● | ● | | | | | | ● | ● |
| Page b | | | | | ● | ● | ● | ● | | | |
| Page c | | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| Page d | $t=-1$ ● | ● | ● | ● | ● | ● | ● | | | | ● |
| Page e | $t=-2$ ● | ● | | | | | ● | ● | ● | ● | ● |
| Faults | | X | | | X | | X | | | X | X |



# Computing the WS

- Use interval timer $\tau$, the R bit, and $k$ extra bits per page
- $\Delta = \tau \times k$
- When $\tau$ elapses, shift $k$ bits right, copy R bit in MSB and reset R bit
- If one of the $k$ bits is 1, the corresponding page is in WS

# Tracking Page Fault Frequency

- When too high, increase WS; decrease when too low.

Keep time $t_{\text{last}}$ of last page fault

On page fault:

if $t_{\text{current}} - t_{\text{last}} > \tau^*$ (threshold), then unmap all pages not referenced in $[t_{\text{last}} - t_{\text{current}}]$

else add faulting page to the working set



# PFF Page Replacement $\tau^* = 2$

| Time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Requests | | c | c | d | b | c | e | c | e | a | d |
| Pages in Memory: Page a | • | | | | | | | | | | |
| Page b | | | | | | | | | | | |
| Page c | | | | | | | | | | | |
| Page d | • | | | | | | | | | | |
| Page e | • | | | | | | | | | | |
| Faults | | | | | | | | | | | |
| $t_{\text{current}} - t_{\text{last}}$ | | | | | | | | | | | |



# PFF Page Replacement $\tau^* = 2$

| Time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Requests | | c | c | d | b | c | e | c | e | a | d |
| Pages in Memory: Page a | • | • | • | • | | | | | | • | • |
| Page b | | | | | • | • | • | • | • | | |
| Page c | | • | • | • | • | • | • | • | • | • | • |
| Page d | • | • | • | • | • | • | • | • | • | | • |
| Page e | • | • | • | • | | | • | • | • | • | • |
| Faults | | X | | | X | | X | | | X | X |
| $t_{\text{current}} - t_{\text{last}}$ | | 1 | | | 3 | | 2 | | | 3 | 1 |