# Memory Policy Basics

P. J. Denning

For CS471/571

# Potential Paging Bottlenecks

- Three possible bottlenecks of paging systems:
  - fetching
  - mapping
  - replacing
- Demand paging reduces cost of fetching to minimum (pre-fetching adds cost if page not used).
- TLB-assisted MMU reduces cost of mapping to a few percent of RAM access time.
- Replacement policy is the main factor influencing performance.

# Why is Replacement Important?

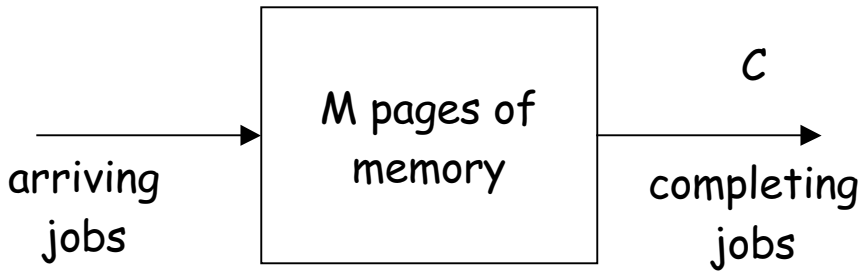
- Most pages are reused repeatedly.
- Replacing a page implies future page fault to recover it.
- Choice of page to replace therefore influences amount of paging.

# Principle of Optimality (1)

- Replace the page that will not be used again for the longest time.
  - Maximizes forward interval to next fault.
  - Minimizes total number of faults.
- Not realizable: future page reference patterns are not known.
- Policies that estimate future forward distance based on past observation are suboptimal.

# Principle of Optimality (2)

- Minimize the space-time of a job.
  - Measured in page-seconds
  - Accumulate one page-second for each page held in memory for one second
  - A kind of "rent" for using memory space.
- In fixed-size memory allocation, the previous principle is minimizes faults and therefore the total space-time resulting from faults.




Observe system for T seconds:
Total space-time available = MT
Space-time per job Y = MT/C
System throughput X = C/T

Thus X = M/Y.

Throughput maximum when space time minimum.

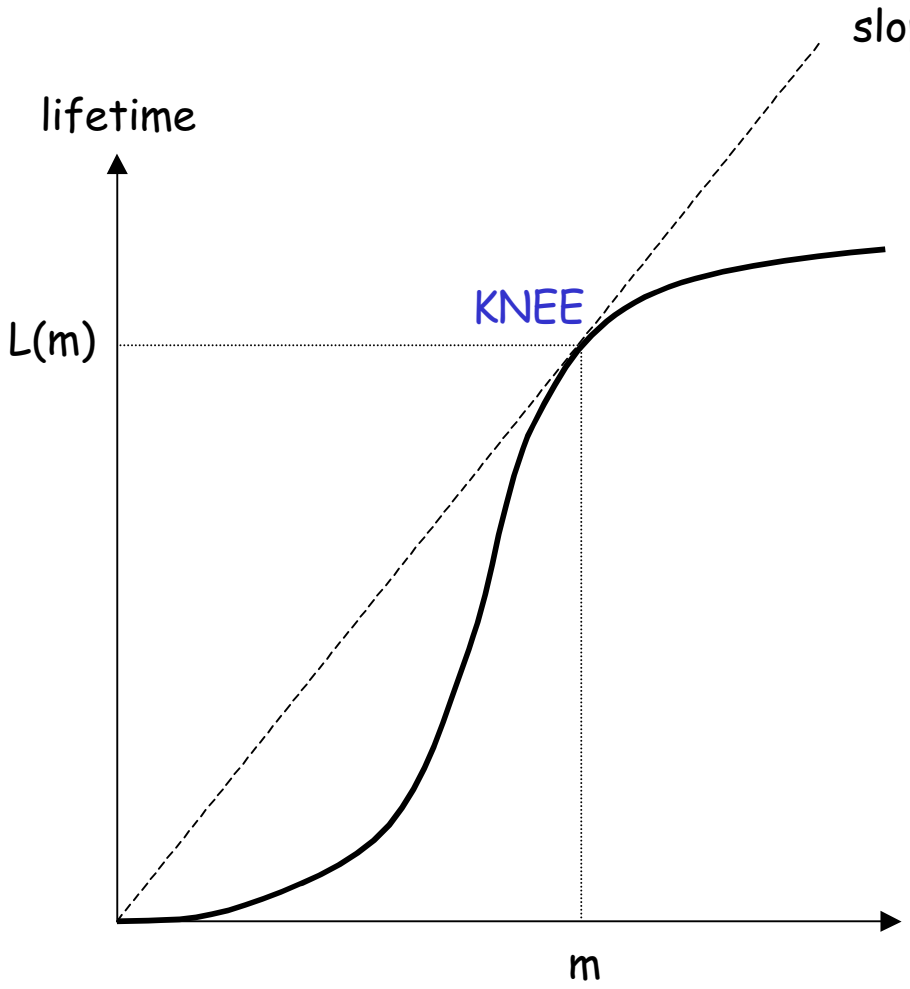
Observe job execute for T seconds virtual time
F(m) = number of page faults when m pages allocated to job

space-time without page faults = mT
space-time for one page fault = mD, where D is the delay
in virtual seconds to process the page fault
space time for all page faults = mDF(m)

Therefore Y(m) = total space-time
= mT+mDF(m)
= mT(1+D/L(m))
where L(m) = T/F(m) = mean time between faults = lifetime function

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

This occurs when m is chosen near the LIFETIME CURVE KNEE (next page).




Slope = L(m)/m is maximum
at the knee of the lifetime curve.

Therefore, space-time minimized
if we choose memory allocation near
knee of job's lifetime curve.

The knee criterion is approximate because the space-time formula on which it depends is approximate.

It also works for variable-space policies because the lifetime curve can be defined for mean time between faults versus mean memory allocation.

# Policies

- Fixed partition
  - FIFO
  - FIFO with usage bit (CLOCK)
  - LRU
  - MIN (the optimum)
- Variable partition
  - Global X (X = LRU, FIFO, CLOCK, etc)
  - WS
  - PFF
  - VMIN

Parameter m, the fixed space size allocated to the job, in pages.

Find F(m) = number of page faults.

Parameter T, the fixed time intervql to measure memory demand.

Find F(T) = number of page faults, and m(T) = mean memory Consumed.

# Fixed Partition Policies

# FIFO

- FIFO = first in first out
- Page frames used ($f_0$,…,$f_{m-1}$)
- Index i tells which frame is next for replacement
- On page fault, replace $f_i$ and set i=(i+1)%m
- Attraction: very simple implementation

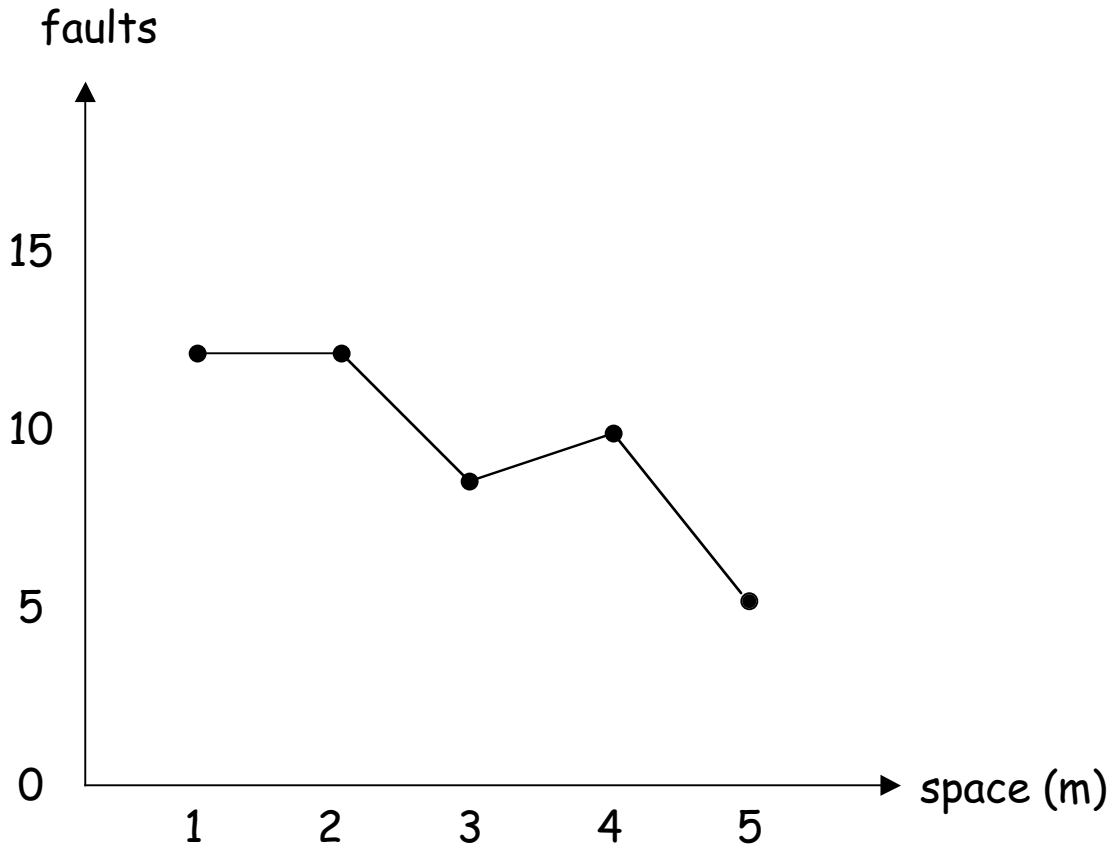
# FIFO Example

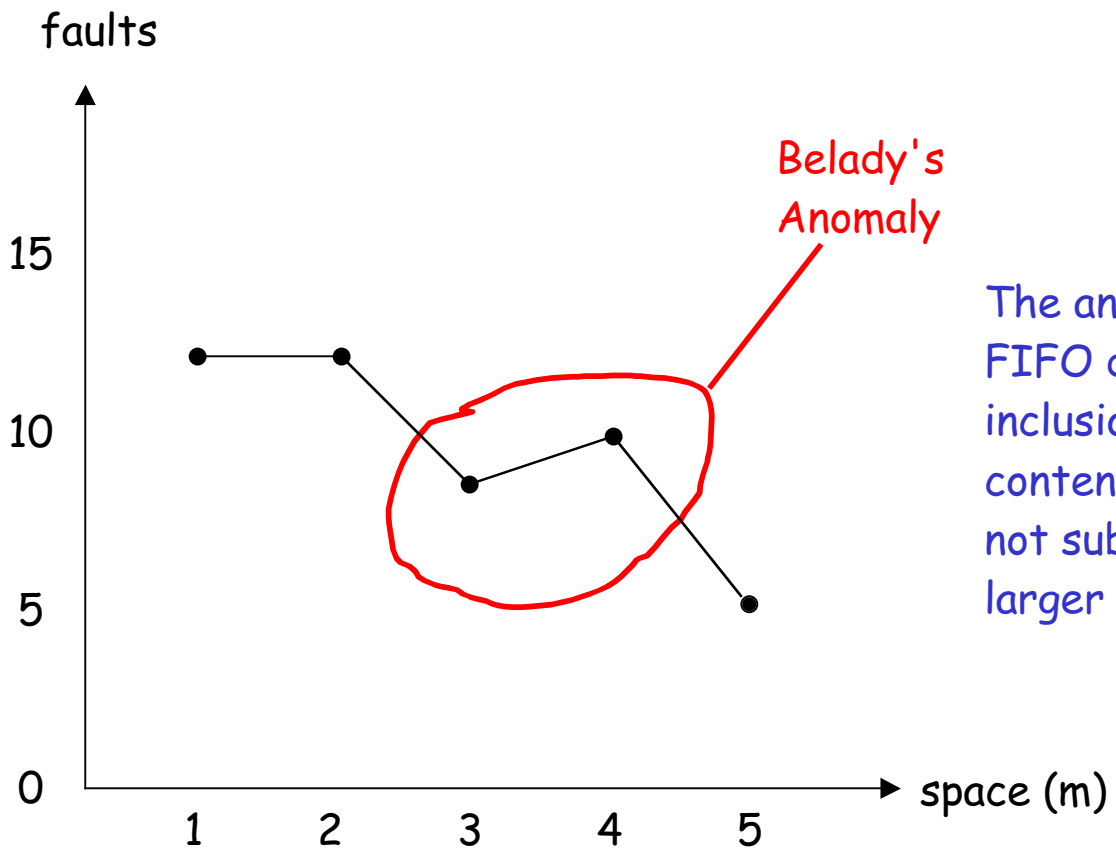
| r(t): | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 4 | 4 | 5 | 1 | 2 | 3 | 4 | 5 |
| m=4: | | 1 | 2 | 3 | 3 | 3 | 4 | 5 | 1 | 2 | 3 | 4 |
| | | | 1 | 2 | 2 | 2 | 3 | 4 | 5 | 1 | 2 | 3 |
| | | | | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 1 | 2 |
| faults: | x | x | x | x | | | x | x | x | x | x | x |

**F(4)=10**

| | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 5 | 5 | 3 | 4 | 4 |
| | | 1 | 2 | 3 | 4 | 1 | 2 | 2 | 2 | 5 | 3 | 3 |
| | | | 1 | 2 | 3 | 4 | 1 | 1 | 1 | 2 | 5 | 5 |
| | x | x | x | x | x | x | x | | | x | x | |

**F(3)=9**






The anomaly occurs because FIFO does not satisfy the inclusion property: memory contents for smaller m are not subsets of those for larger m (see example).

# CLOCK

- CLOCK = FIFO with usage bits
- Same as FIFO, except i-update rule is
  while $f_i$.U=1 do {$f_i$.U=0; i=(i+1)%m}
- Attraction: almost as simple as FIFO and protects used pages.

# LRU

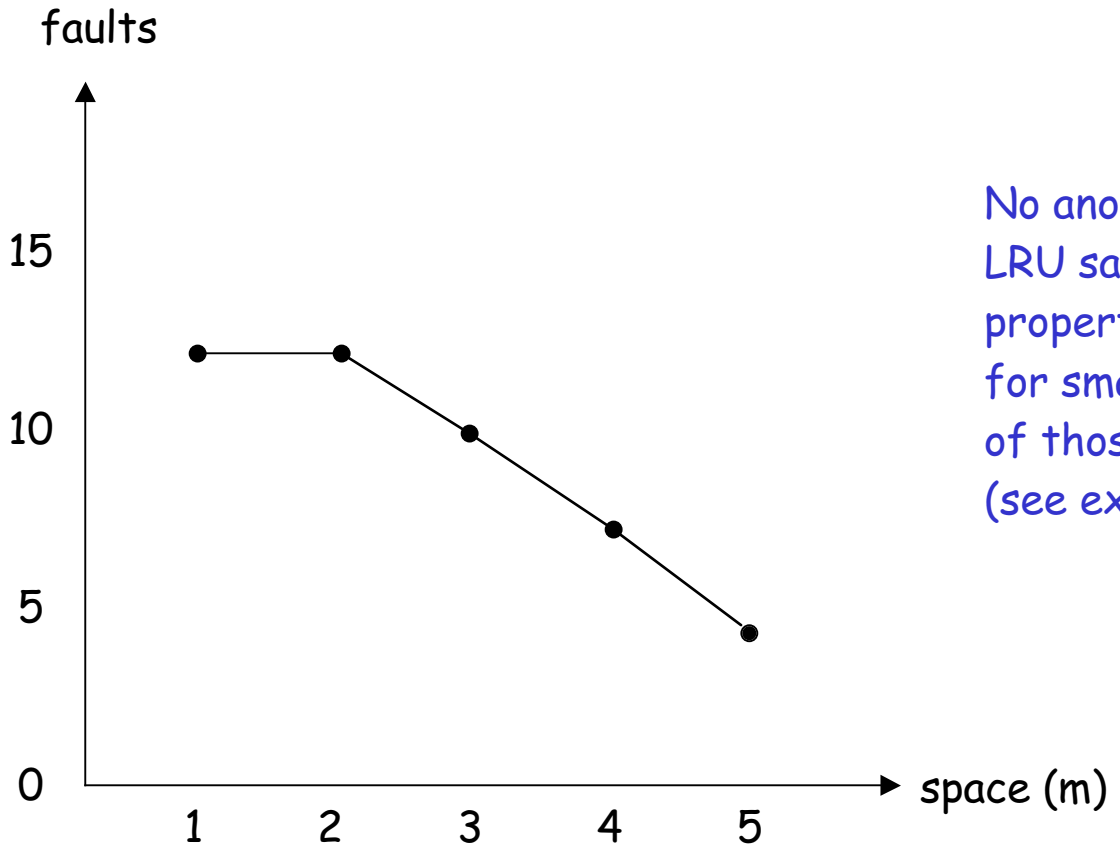
- LRU = least recently used
- Maintain vector called "LRU stack" ordering all p pages from most to least recently referenced
- If $S(t)=(f_0,...,f_{p-1})$ is stack, page to be replaced from m-page memory is $f_{m-1}$.
- Attraction: good performance, no anomalies

| 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 |
| m=4: | | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 |
| | | | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 |
| | | | | 1 | 2 | 3 | 4 | 4 | 4 | 5 | 1 | 2 |
| faults: | x | x | x | x | | | x | | | x | x | x |

**F(4)=8**

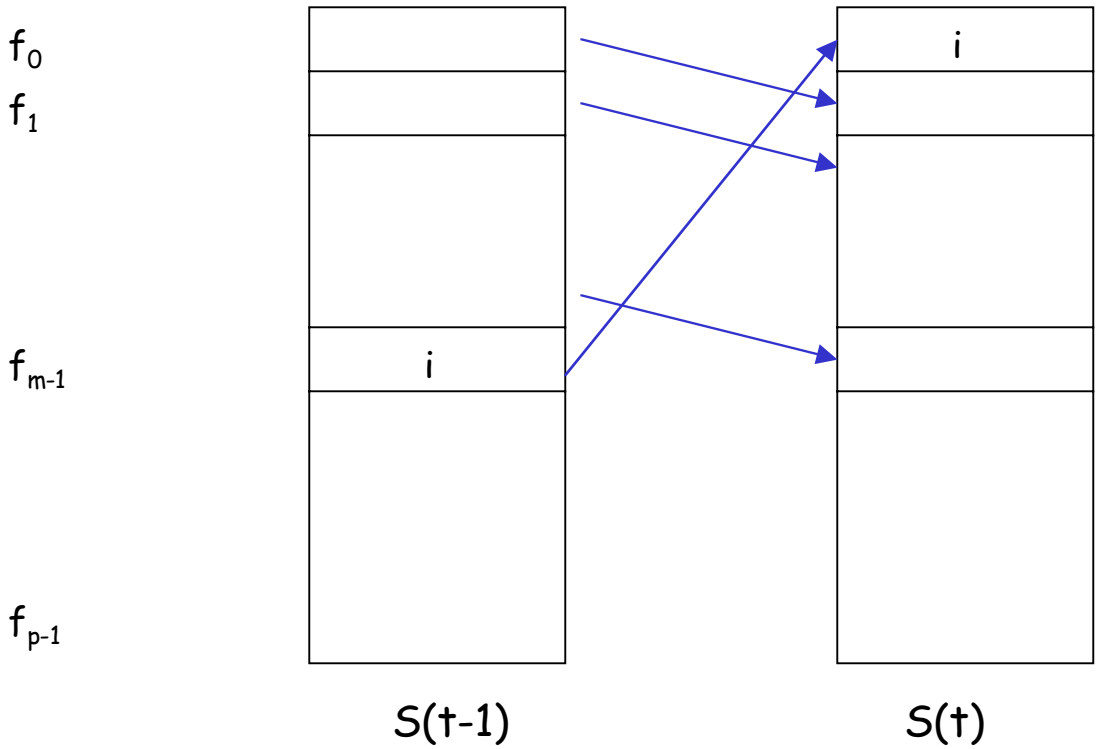
| | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 | 5 |
| | | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 |
| | | | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 |
| | x | x | x | x | x | x | x | | | x | x | x |

**F(3)=10**

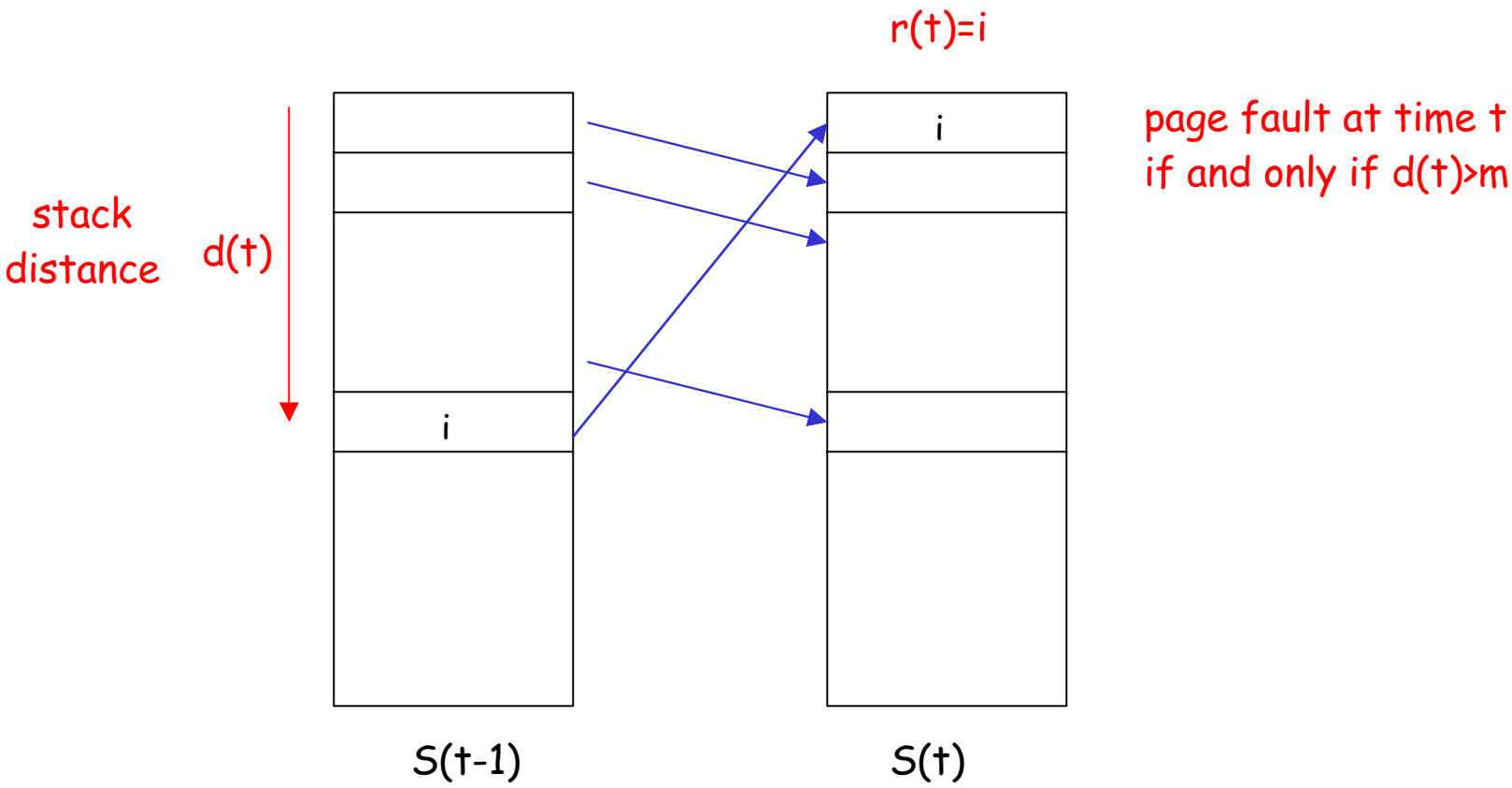



No anomaly occurs because LRU satisfies the inclusion property: memory contents for smaller m are subsets of those for larger m (see example).

# LRU

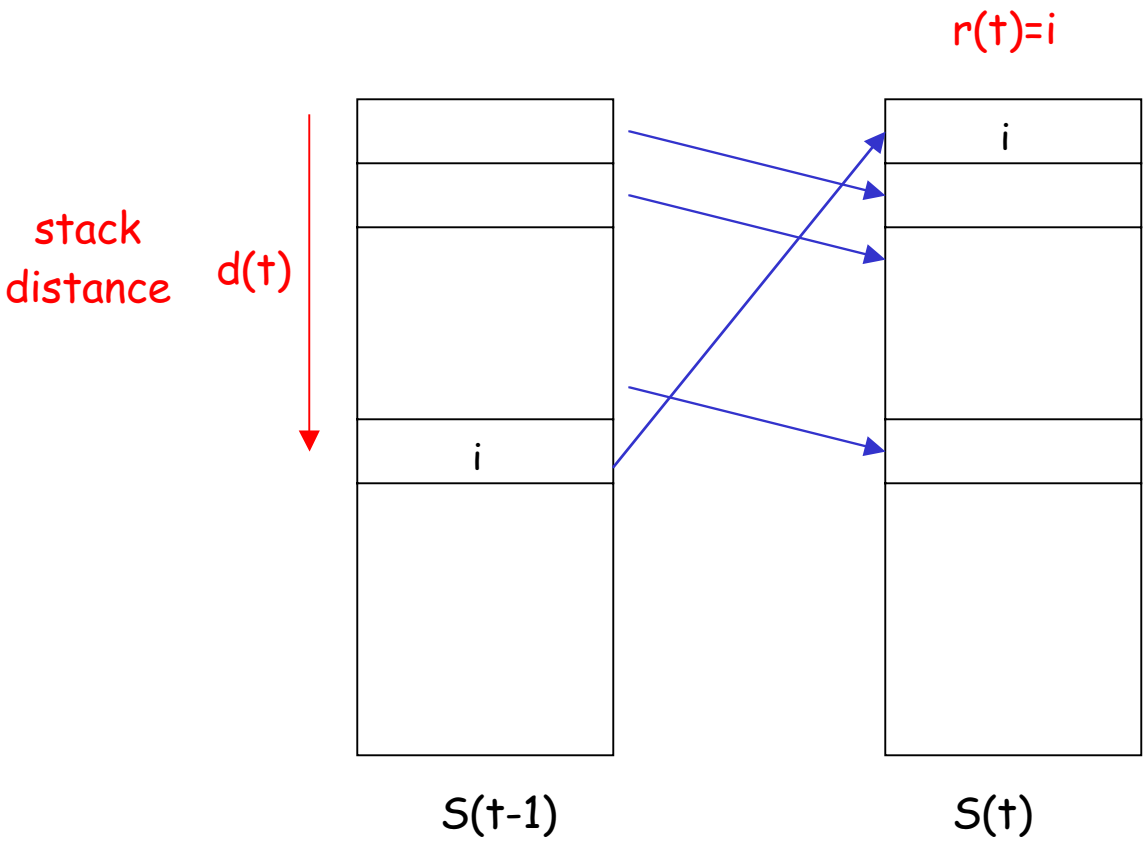


# LRU




page fault at time t
if and only if d(t)>m

# LRU




page fault at time t
if and only if d(t)>m

histogram {c(k)} in
which c(k) = number
of occurrences of
d(t)=k

sum{c(k)} = length of
reference string

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 |
| | | | 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. F(k) is the sum of the counts for all distances larger than k.

# Stack Algorithms

- Fixed-space memory policy called a stack algorithm if it satisfies the inclusion property.
- Its pages can be ordered into a vector S(t) such that the first m entries of S(t) are the memory contents at time t.
- The stack S(t-1) is updated to S(t) by applying a priority rule over *all* the pages.
- Page fault at time t exactly if d(t)>m.

# Stack Algorithms (2)

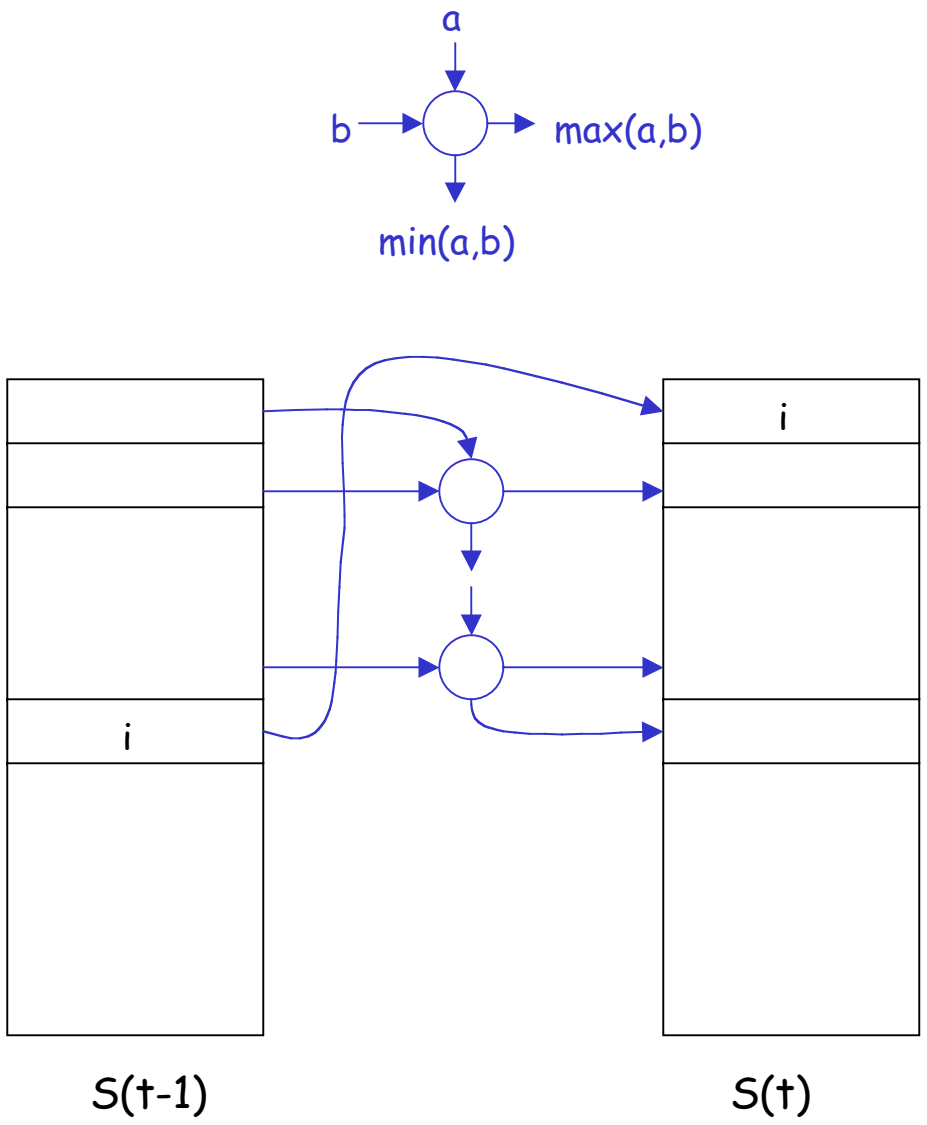
- LRU is a stack algorithm; its priority rule at time t organizes the pages by increasing backward distance.
- MIN (optimal) is a stack algorithm; its priority rule at time t organizes the pages by increasing forward distance.
- Even the RAND (random) policy is a stack algorithm; its priority rule at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

# MIN

- MIN = minimum possible faults (optimum)
- Replace page with longest forward distance until next reference.
- Attraction: benchmark for realizable policies.

# MIN (2)

- Stack update rule more complex.
- Referenced page moves to top because it is always in memory, even if m=1.
- Any page below the current distance does not move because there is no fault in any of those memory sizes and memory contents cannot change in those memories.
- Pages in between move down or stay put, but cannot move up because an up move would represent a page-in for that memory size for a non referenced page.
- A page moves as far down as its priority permits.

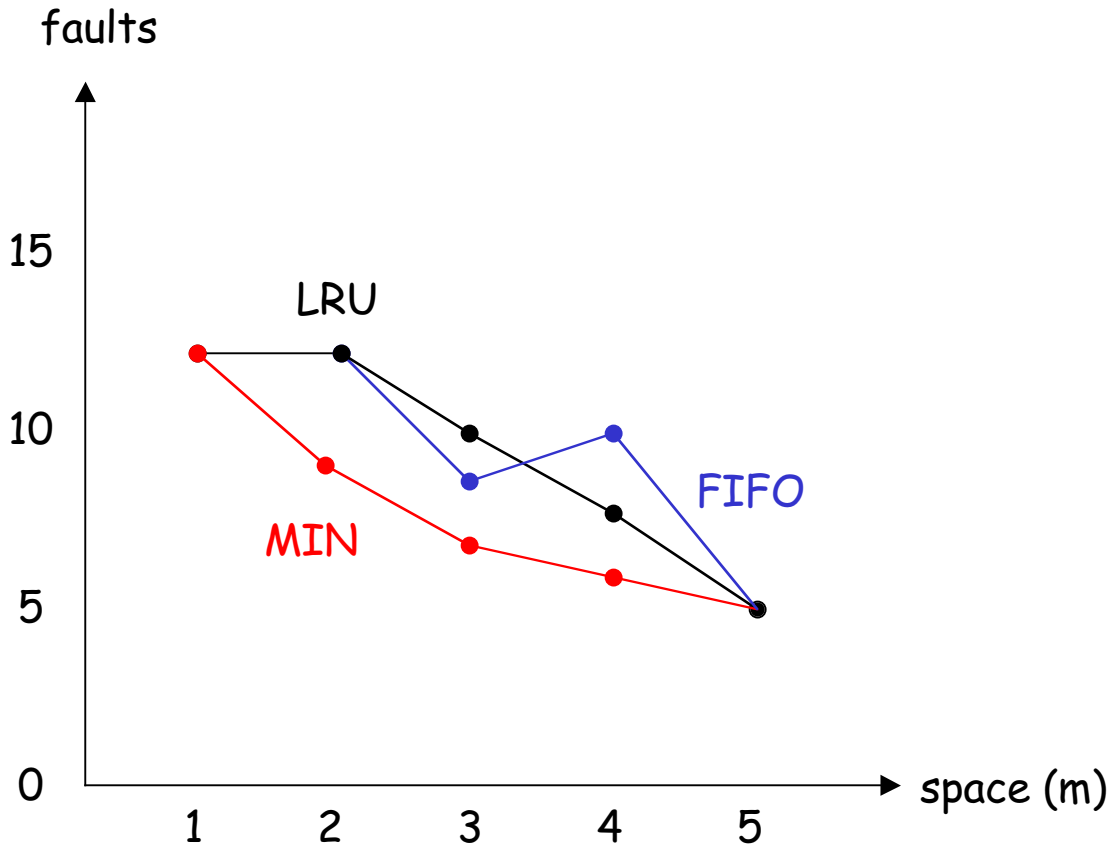


notice the update: page 3 comes from infinity; page 2 must move down, and it goes farther down than 1 because 1 is referenced again sooner.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 |

| k | c(k) | F(k) |
|---|---|---|
| 1 | 0 | 12 |
| 2 | 3 | 9 |
| 3 | 2 | 7 |
| 4 | 1 | 6 |
| 5 | 1 | 5 |
| x | 5 | |

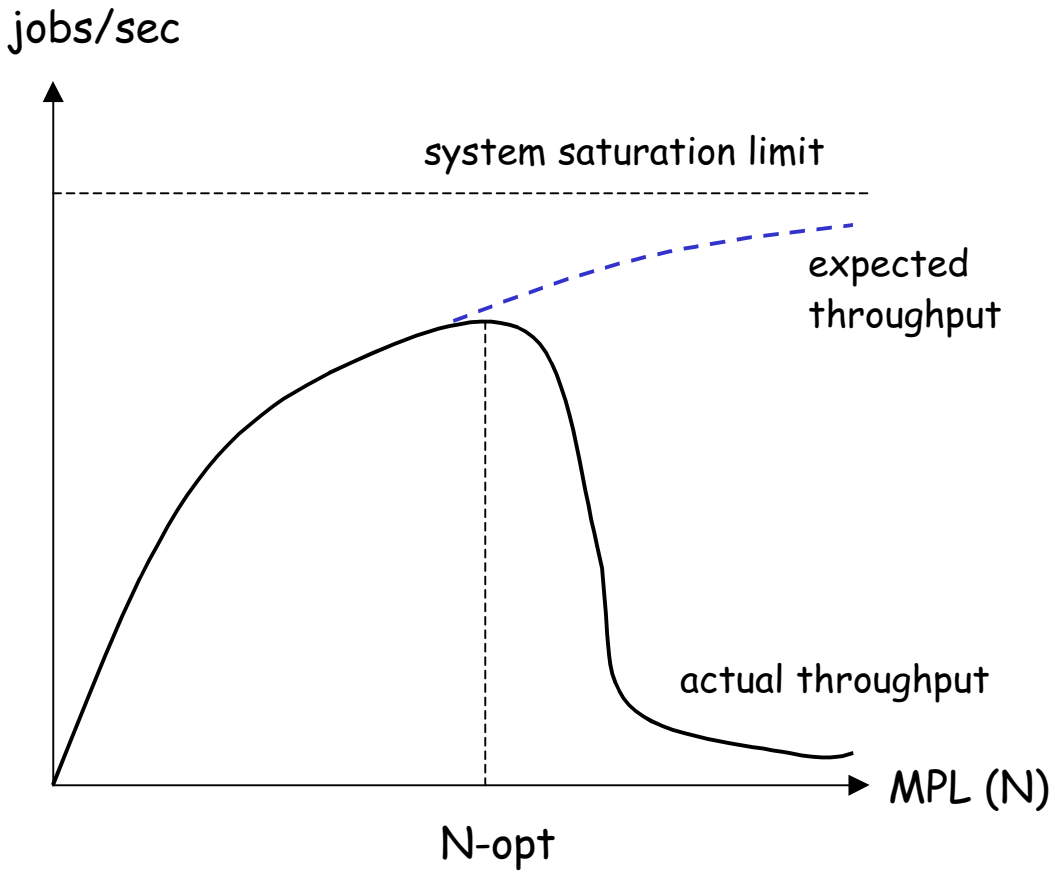
after a page's last reference, its priority is no longer important; here we use priority equal to page number.



# Variable Partition Policies

# GLOBAL "X"

- Lump all pages of all jobs together in one pool and apply policy "X" to the pool.
- Tends to perform much worse than "X" applied separately to jobs:
  - Order in usage-detecting priority lists influenced much more by scheduling (e.g., round-robin) than by program referencing.
  - Susceptible to thrashing because there is no way to protect individual localities.




Thrashing is unexpected, sudden drop in system throughput with increased multiprogramming level.

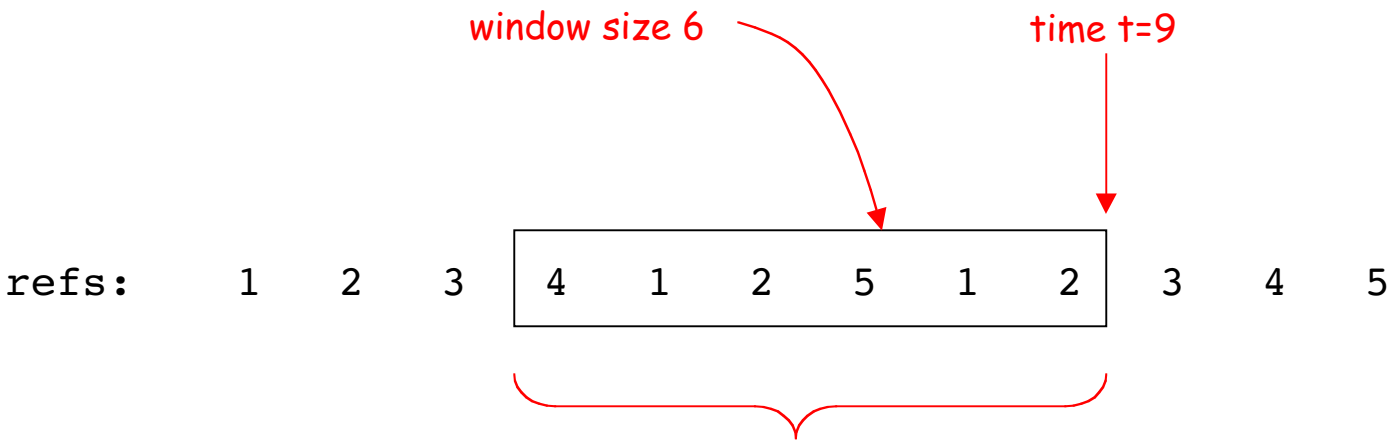
Thrashing occurs when MPL so large that no program has enough space for its locality sets; hence the jobs generate a high level of paging and all wind up waiting for disk service rather than make progress on the CPU.

# WS

- WS = Working Set
- The working set at time t is the set of pages observed in a backward looking window of fixed length T.
- The working set tracks localities: when T is contained within a phase, the WS is the locality set. At transitions, WS contains some of both localities.
- Can be implemented with usage bits sampled every T seconds in virtual time.

# WS (2)

- A WS memory policy is designed for variable partition multiprogramming.
- The policy grants each job its WS, and allows the MPL to rise only until memory is full of WS's.
- The policy prevents thrashing.



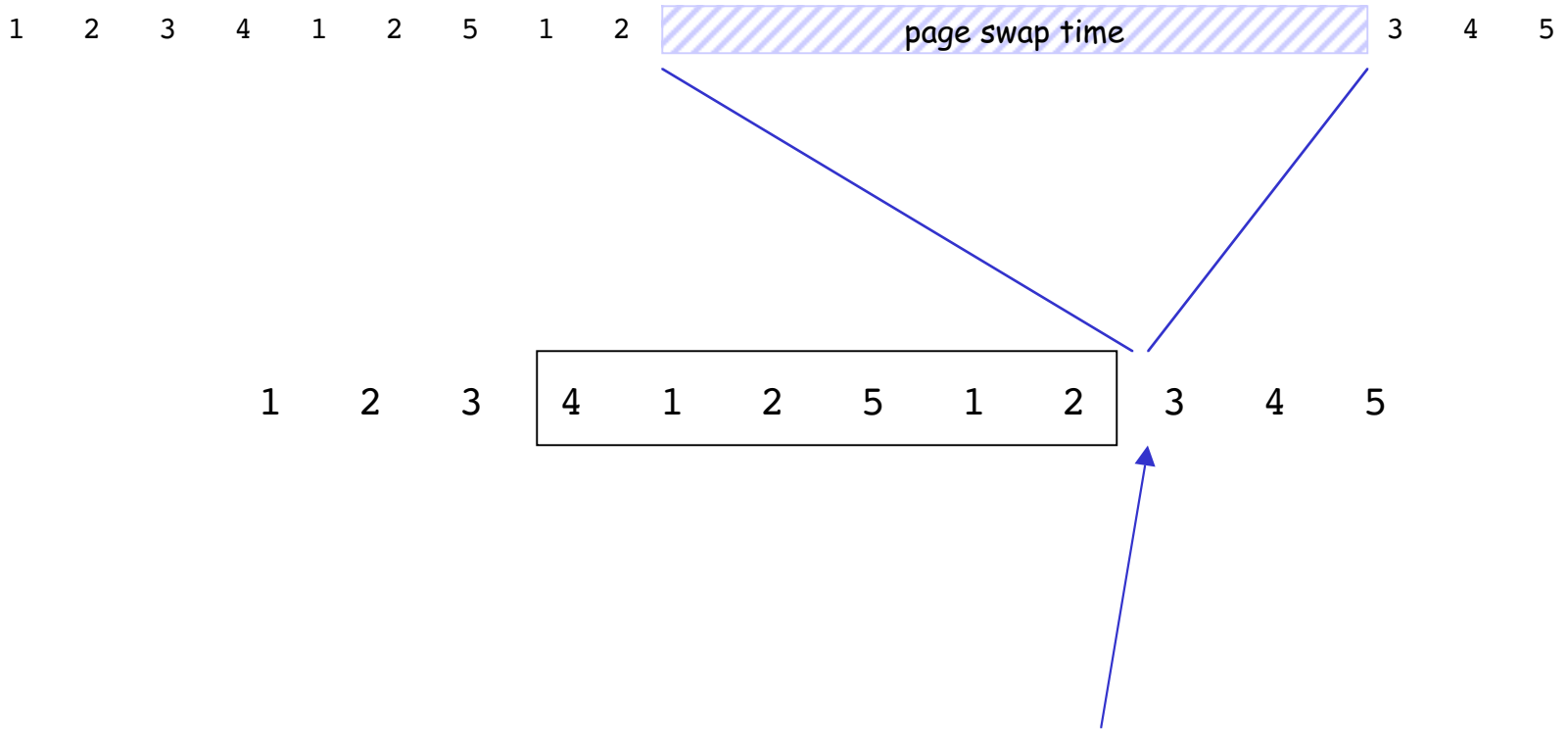

WS = pages 1, 2, 4, 5

W(t,T)

W(9,6) = {1,2,4,5}

WS "sees" a subset of the pages through its window and adjusts memory allocation to match.

A page fault will occur at t=10 because page 3 is not in the working set W(9,6).

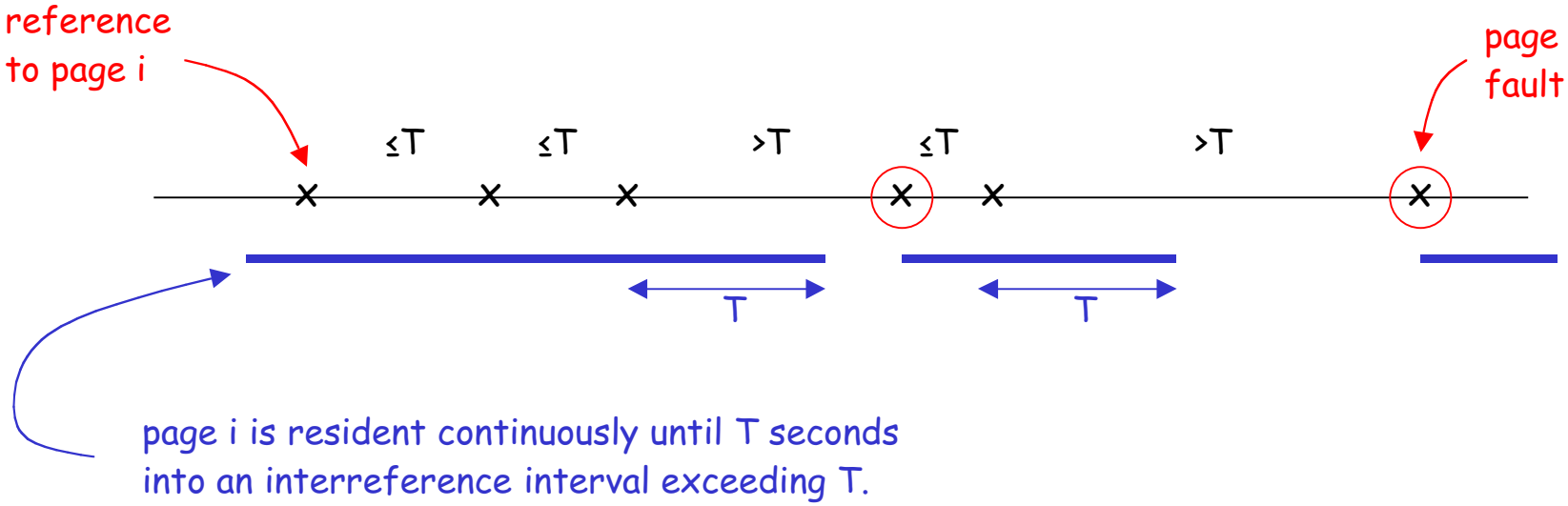



In real time, a page swap delay is inserted between t=9 and t=10 because of the page fault.

WS follows an inclusion property: W(t,T-1) a subset of W(t,T)

Is there a distance function such that d(t)>T exactly when there is a page fault at time t?

Yes: use the virtual time interval between references to the same page. When a page is referenced, it generates a page fault exactly if that distance is larger than T.

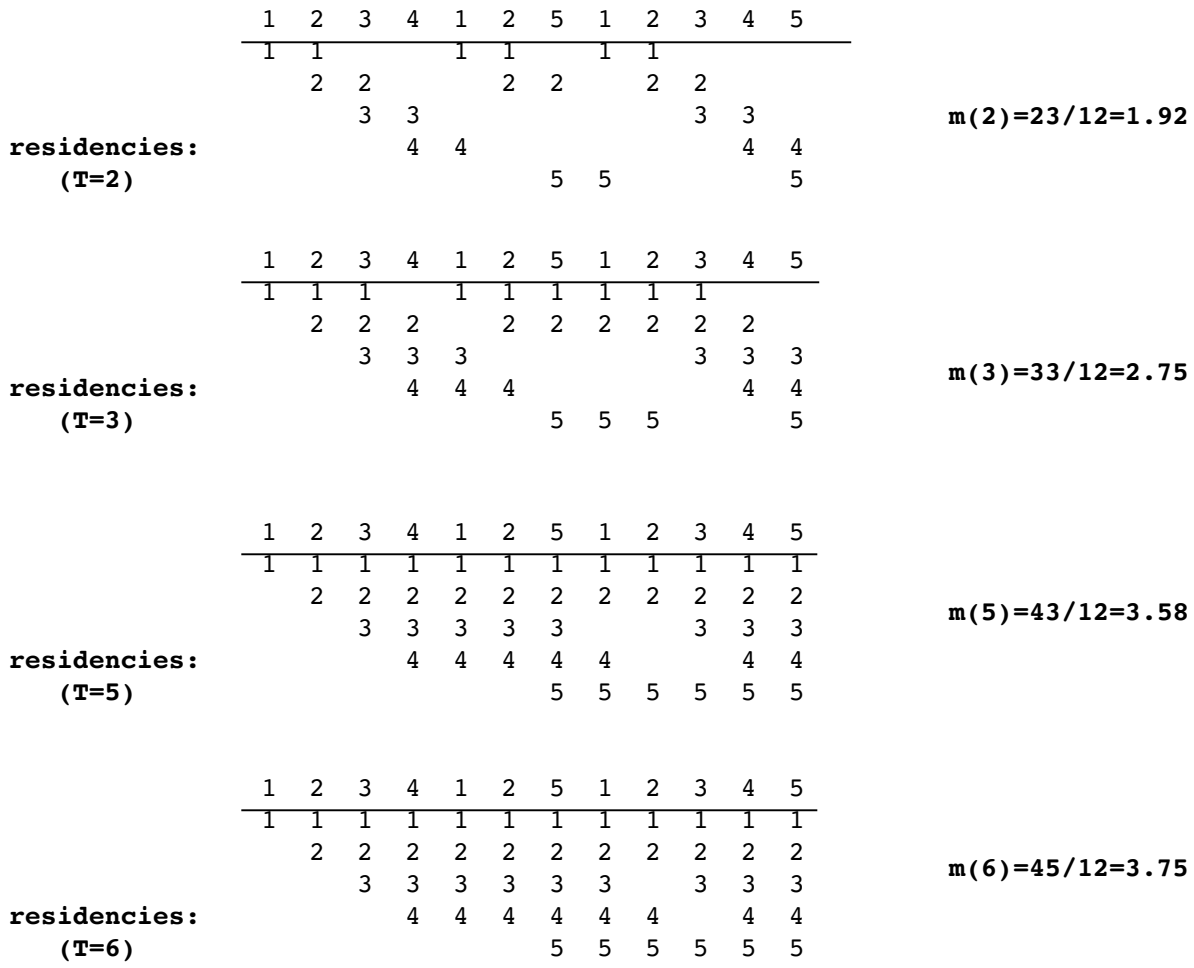


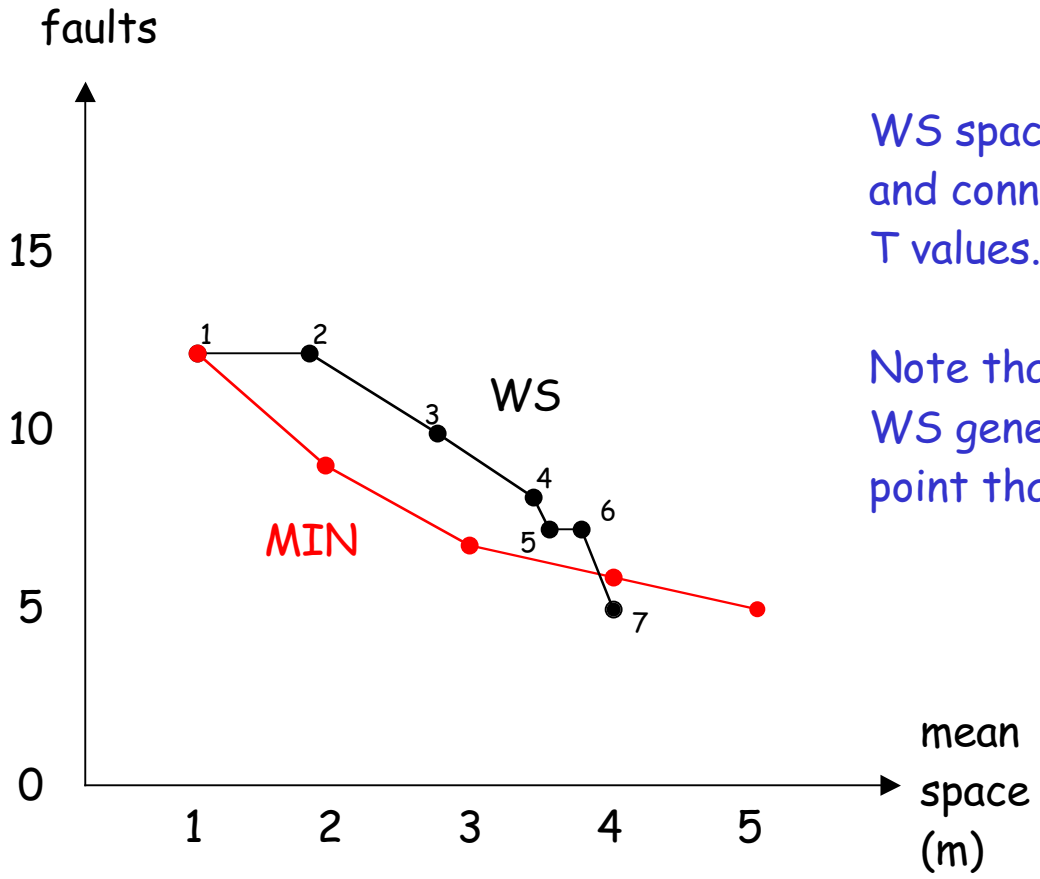
| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **r(t):** | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 | 5 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |
| **residencies:** | | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| **(T=4)** | | | 3 | 3 | 3 | 3 | | | | 3 | 3 | 3 |
| | | | | 4 | 4 | 4 | 4 | | | | 4 | 4 |
| | | | | | | | 5 | 5 | 5 | 5 | | 5 |
| **d(t):** | x | x | x | x | 4 | 4 | x | 3 | 3 | 7 | 7 | 5 |

| k | c(k) | F(k) |
|---|---|---|
| 1 | 0 | 12 |
| 2 | 0 | 12 |
| 3 | 2 | 10 |
| 4 | 2 | 8 |
| 5 | 1 | 7 |
| 6 | 0 | 7 |
| 7 | 2 | 5 |
| x | 5 | |

Note that WS size varies over time.
The mean space m(T) is the total page-time divided by time, or m(4) = 40/12 = 3.33 pages.

Thus for T=4, the corresponding point in the faults versus mean space graph is (F(T), m(T)) or (8, 3.33).


1 2 3 4 1 2 5 1 2 3 4 5
m(2)=23/12=1.92
residencies:
(T=2)
1 2 3 4 1 2 5 1 2 3 4 5
m(3)=33/12=2.75
residencies:
(T=3)
1 2 3 4 1 2 5 1 2 3 4 5
m(5)=43/12=3.58
residencies:
(T=5)
1 2 3 4 1 2 5 1 2 3 4 5
m(6)=45/12=3.75
residencies:
(T=6)




WS space-fault points plotted and connected; numbers are T values.

Note that at one point (T=7) WS generates a better operating point than (fixed-space) MIN

# PFF

- PFF = page fault frequency
- WS requires regular virtual time usage-bit sampling (at window-size separations).
- PFF samples at page-fault times.
- To prevent a short inter-fault interval from making it seem that most pages are unused, PFF uses a threshold T. Inter-fault intervals less than the threshold are ignored.

# PFF (2)

- Let T = threshold value
- Let time t be moment of page fault and t* be time of previous page fault.
- If t-t* ≤ T, take no replacement action; just add the faulting page to memory.
- If t-t* > T, replace all pages with usage bit 0, reset all usage bits to 0; and add the faulting page to memory.

# PFF (3)

- PFF does not satisfy an inclusion property.
- PFF can exhibit anomalies (like Belady's anomaly for FIFO).

# VMIN

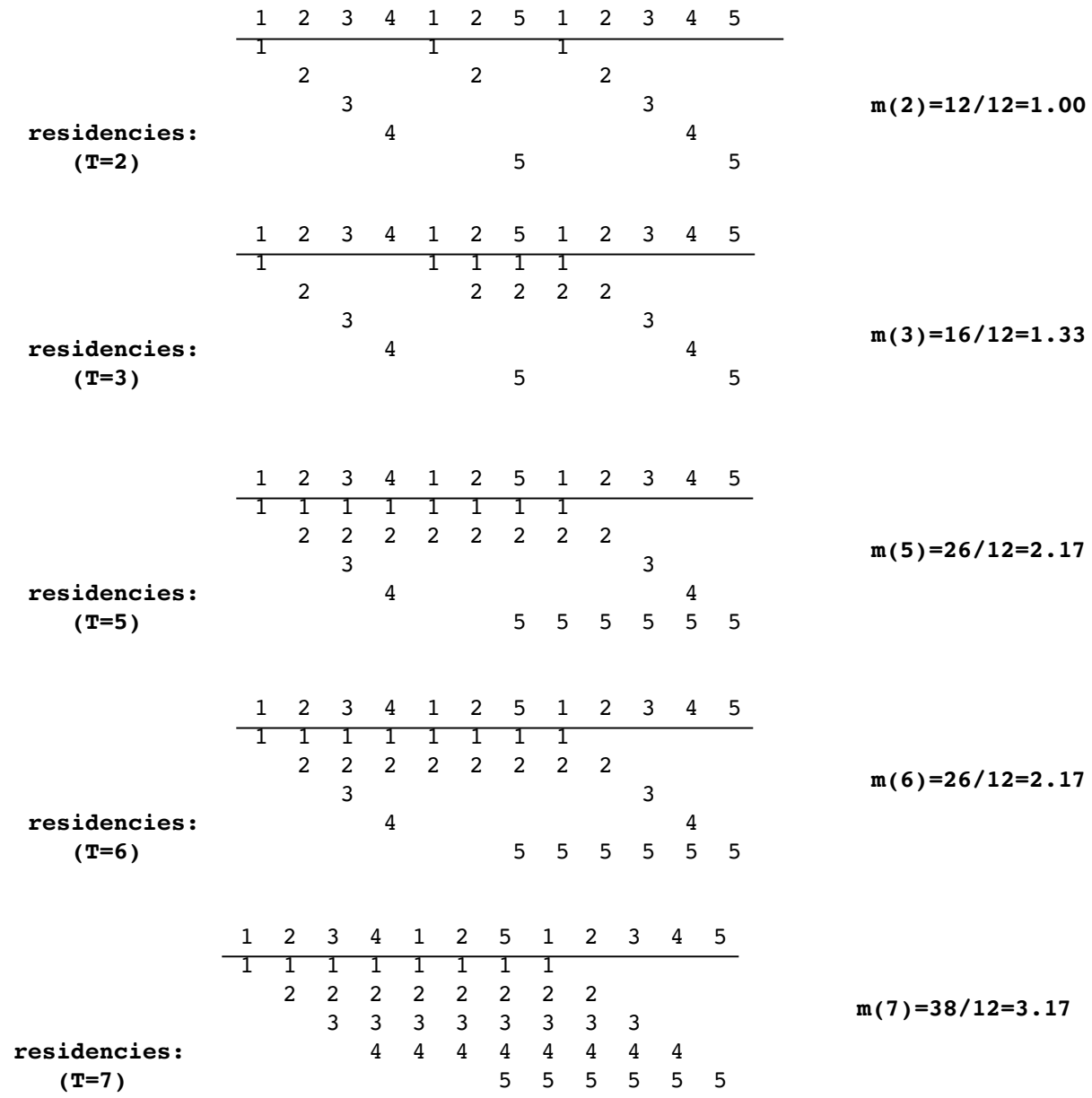
- VMIN = variable space MIN (optimal)
- Parameter T, a window size as in WS
- At a reference to a page:
  - retain page till next reference if forward time to next reference is ≤ T.
  - otherwise replace the page.
- Since criterion for retaining is identical to WS, the fault rate of VMIN = F(T) from WS.

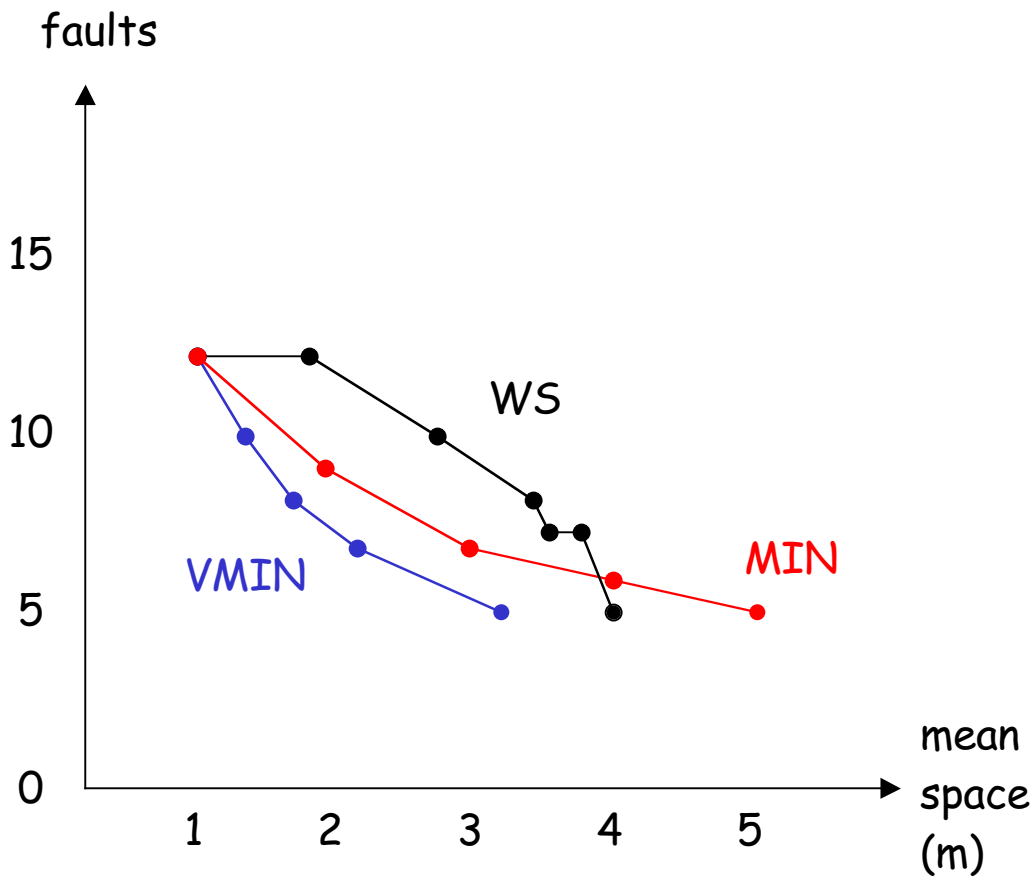
| | r(t): | 1 | 2 | 3 | 4 | 1 | 2 | 5 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |
| residencies: | | | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| (T=4) | | | | 3 | 3 | 3 | 3 | | | | 3 | 3 | 3 |
| | | | | | 4 | 4 | 4 | 4 | | | | 4 | 4 |
| | | | | | | | | 5 | 5 | 5 | 5 | | 5 |
| | d(t): | x | x | x | x | 4 | 4 | x | 3 | 3 | 7 | 7 | 5 |

m(4)=22/12=1.83

The residency chart for VMIN is similar to that for WS, except that VMIN does not hold a page into a long inter-reference interval as does WS.

VMIN has the same page fault sequence but a lower resident set size than WS. The red letters show the contributions to WS space-time that VMIN eliminates.


1 2 3 4 1 2 5 1 2 3 4 5
residencies:
(T=2)
m(2)=12/12=1.00
1 2 3 4 1 2 5 1 2 3 4 5
residencies:
(T=3)
m(3)=16/12=1.33
1 2 3 4 1 2 5 1 2 3 4 5
residencies:
(T=5)
m(5)=26/12=2.17
1 2 3 4 1 2 5 1 2 3 4 5
residencies:
(T=6)
m(6)=26/12=2.17
1 2 3 4 1 2 5 1 2 3 4 5
residencies:
(T=7)
m(7)=38/12=3.17

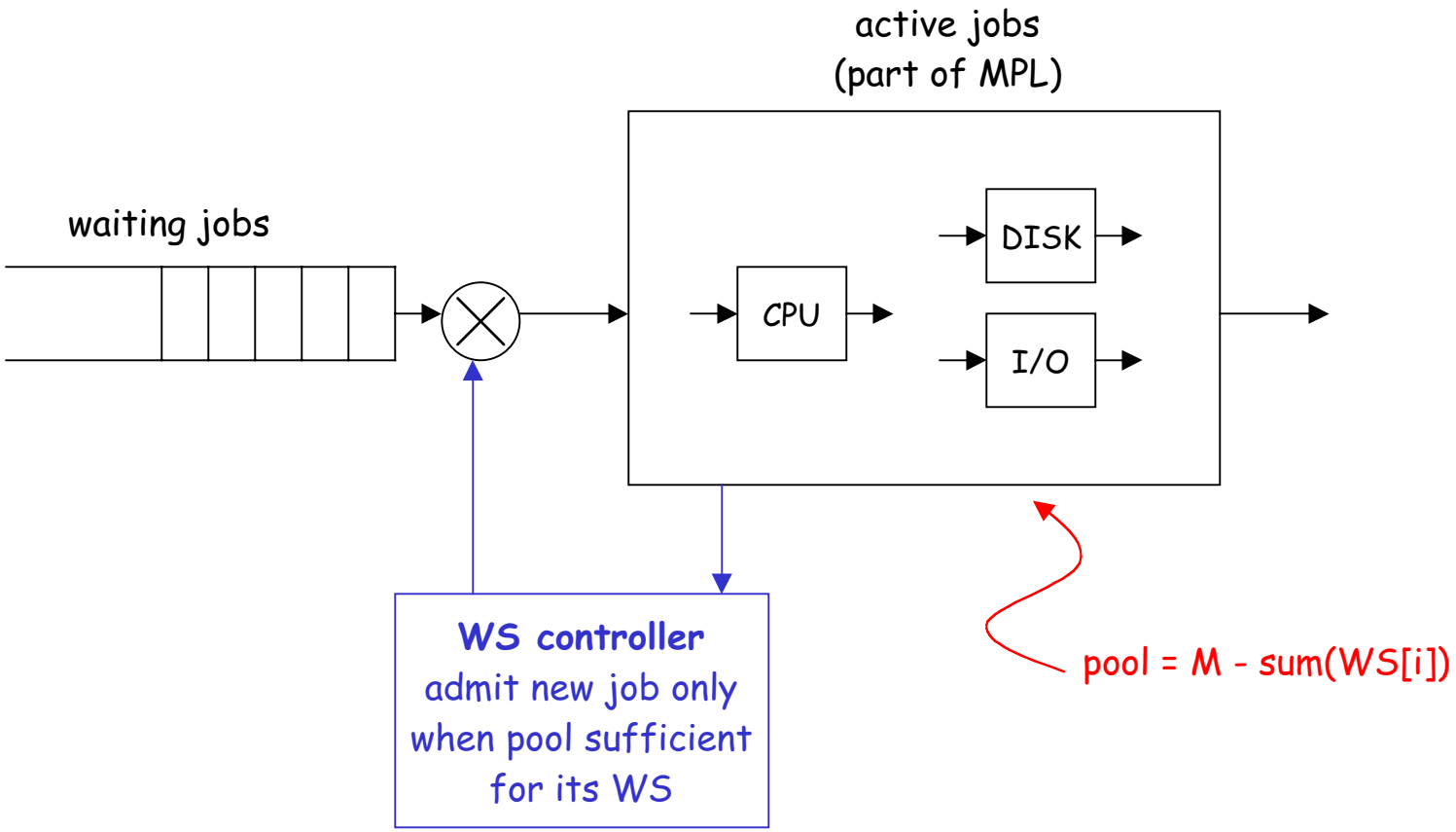


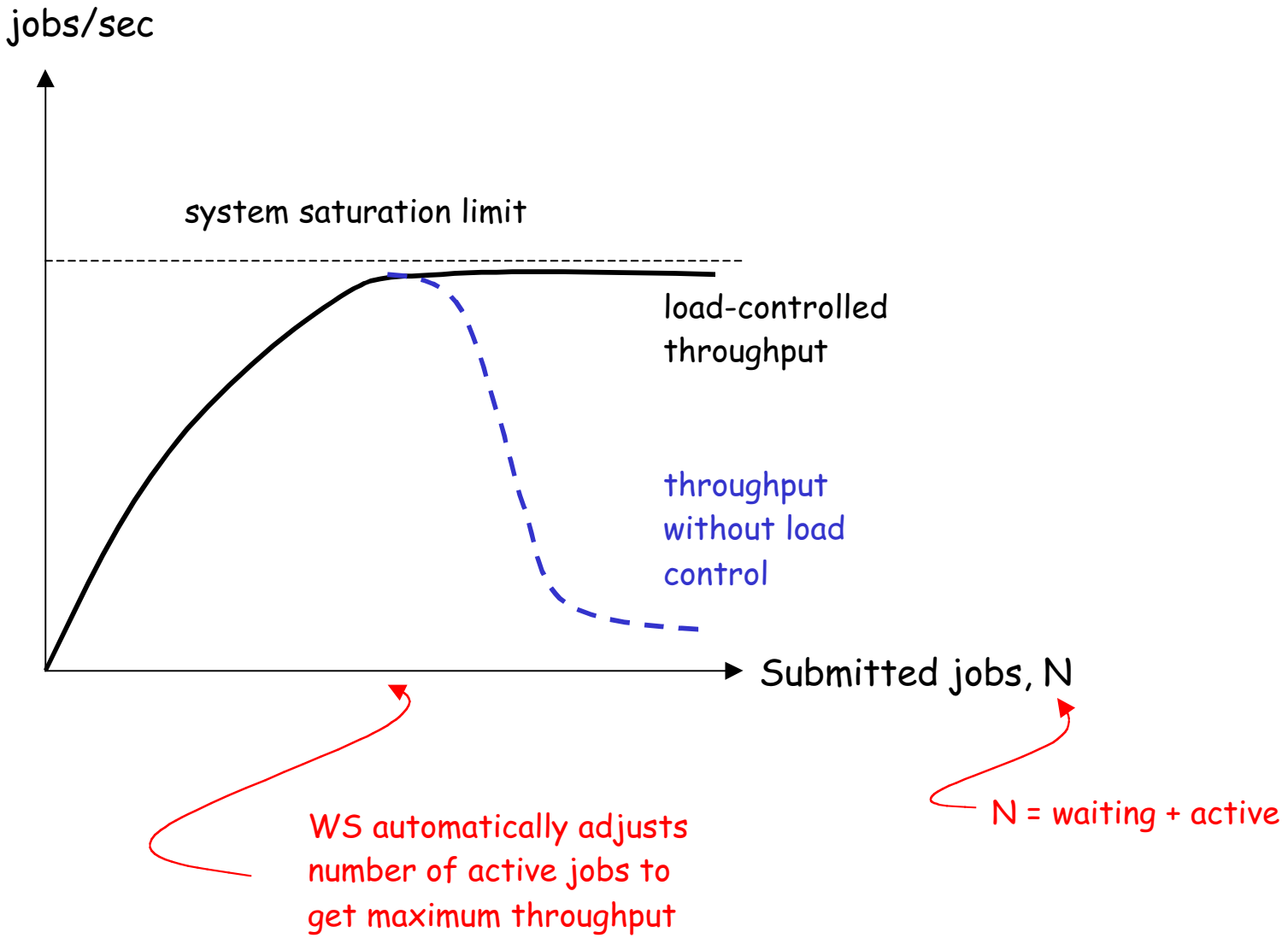
If the program has long phases and T is set less than the phase lengths, then VMIN = WS during the phases.

The VMIN space advantage occurs at the transitions between phases, when VMIN unloads old pages before entering the new phase, while WS retains old pages for a while after entering the new phase.

Experimental studies (1970s) showed that WS is about as close to optimum space-time as a realizable paging strategy can come and will usually be within 5% to 10% of optimum.

# Load-Controlled WS Scheduling





finis