Storage Basics

P. J. Denning

For CS471/571

© 2001-2002, P. J. Denning

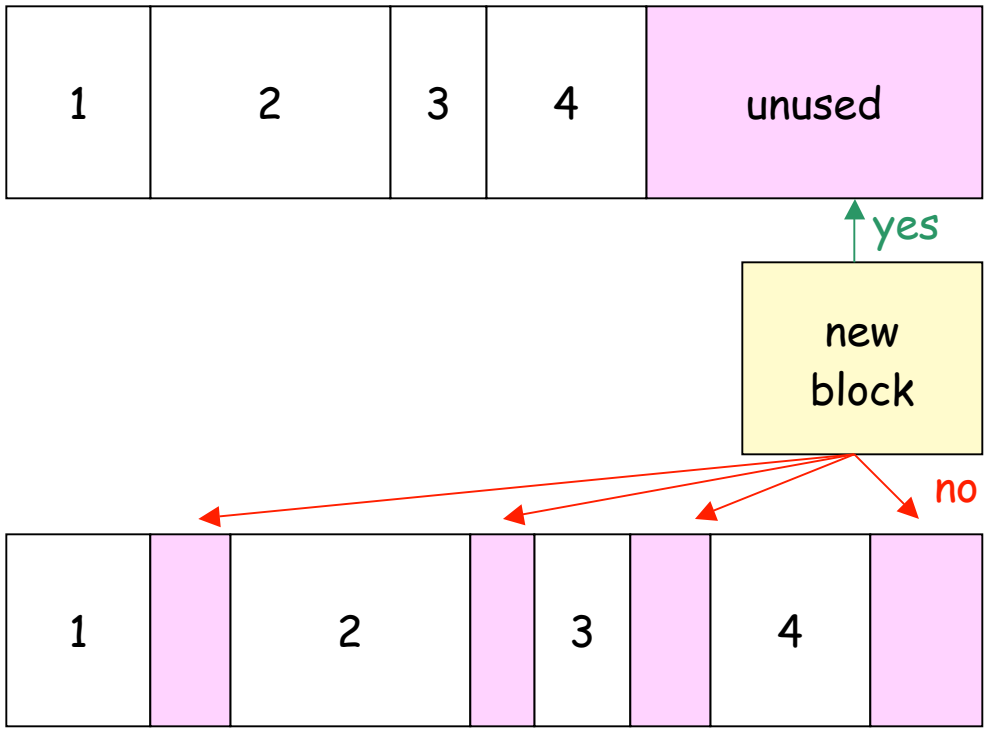
- All computers have memory hierarchies
  - computational store (e.g., RAM, cache)
  - long-term store (e.g., disks, CDs, tapes)
- Why?
  - Speed and cost tradeoffs
  - Different types of storage media
  - Removable and portable storage media
  - Volatility of RAM, persistence of disk
  - Multiprogramming for greater efficiency of resource usage

© 2001-2002, P. J. Denning

Multiprogramming

- Partition main memory among active "jobs"
- Fixed or variable partition?
- Fragmentation? (Checkerboarding)
- Hardware address translation?
- Hardware memory protection?

© 2001-2002, P. J. Denning



four blocks compacted together occupying a multiprogrammed memory; unused portion at end

same blocks scattered; unused space is fragmented and cannot permit loading the new block.

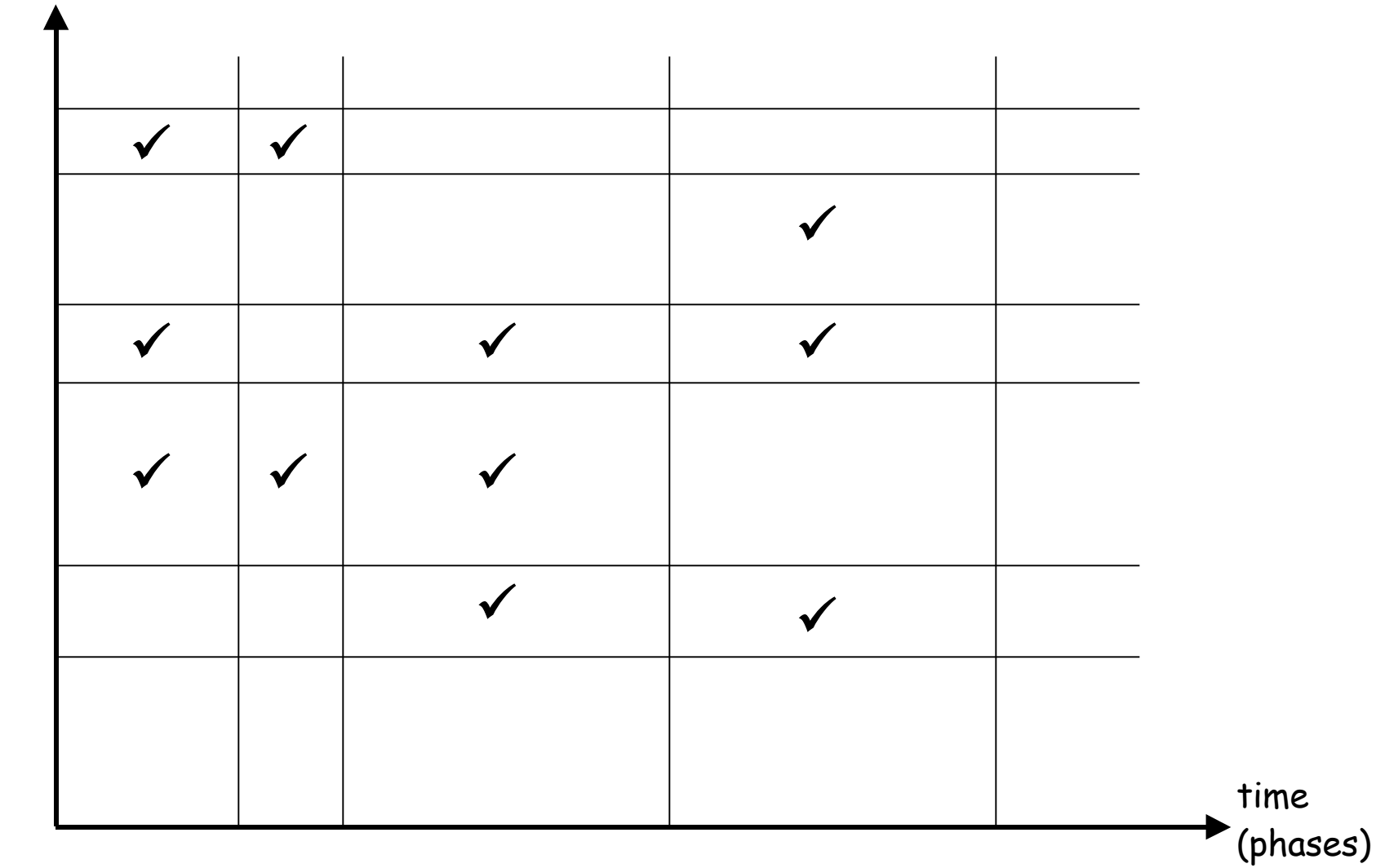
© 2001-2002, P. J. Denning

Implications for Programmers

- Overlay problem
  - Divide program into blocks
  - Divide program time into phases
  - Identify blocks that need to be present in each phase (subject to memory limit)
  - Schedule fetches and replacements
- 40-50% of programming time on this

© 2001-2002, P. J. Denning

LOCALITY DIAGRAM



© 2001-2002, P. J. Denning

- Blocks needing to be present: locality set of a phase.
- Fetch: insert instruction to bring block from disk to RAM
  - Pre-fetch? Demand fetch?
- Replace: insert instruction to copy block from RAM to disk
  - Pre-replace? Demand replace?
- Overlay sequence depends on memory size limit

© 2001-2002, P. J. Denning

Paging

- Invented 1959 on Atlas Computer at University of Manchester
- Divide programs and data into fixed size blocks called pages
- Divide memory into fixed size (same) blocks called frames

© 2001-2002, P. J. Denning

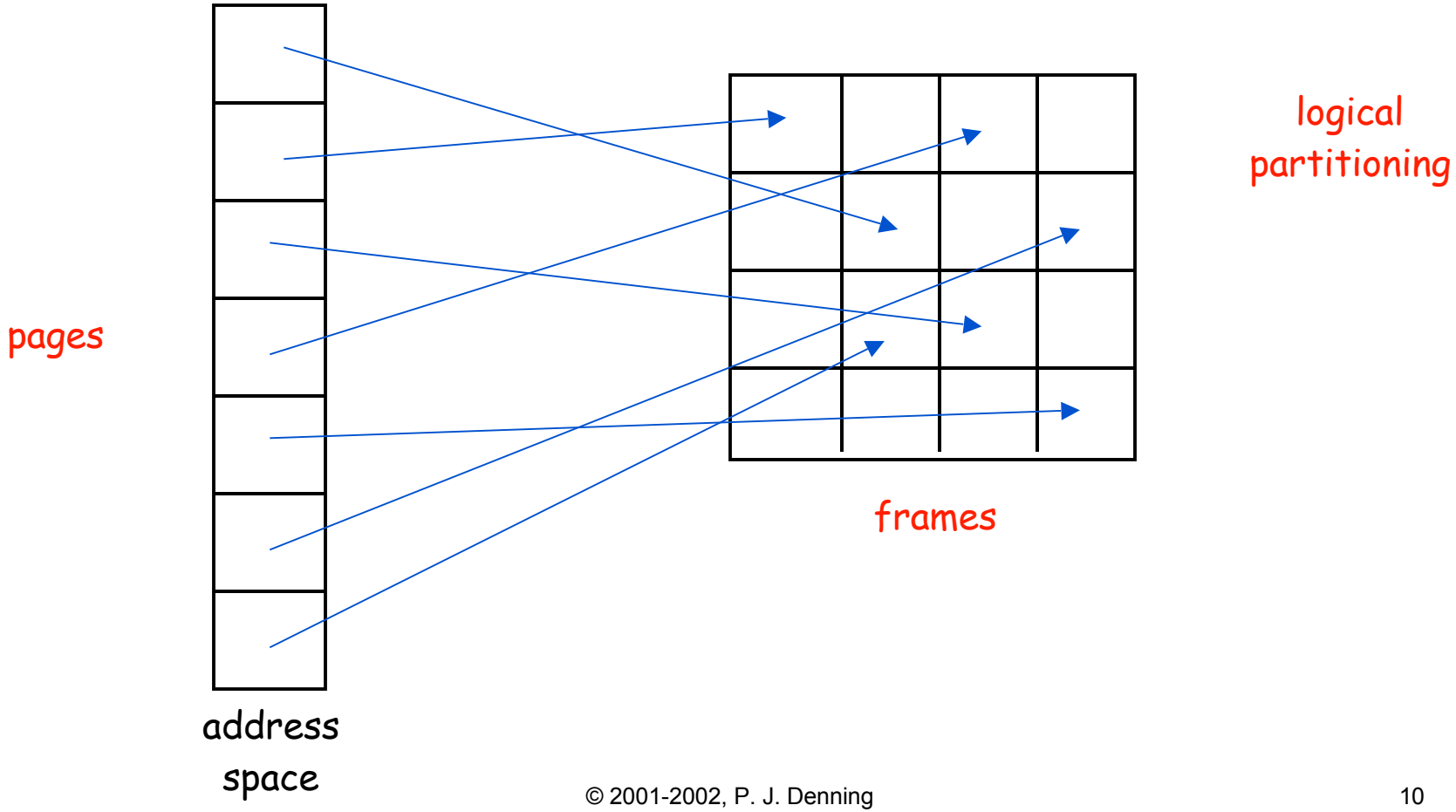
Paging 2

- No (external) fragmentation
- Fetch on demand (cheaper than error-prone pre-fetching)
- Map address space to memory space

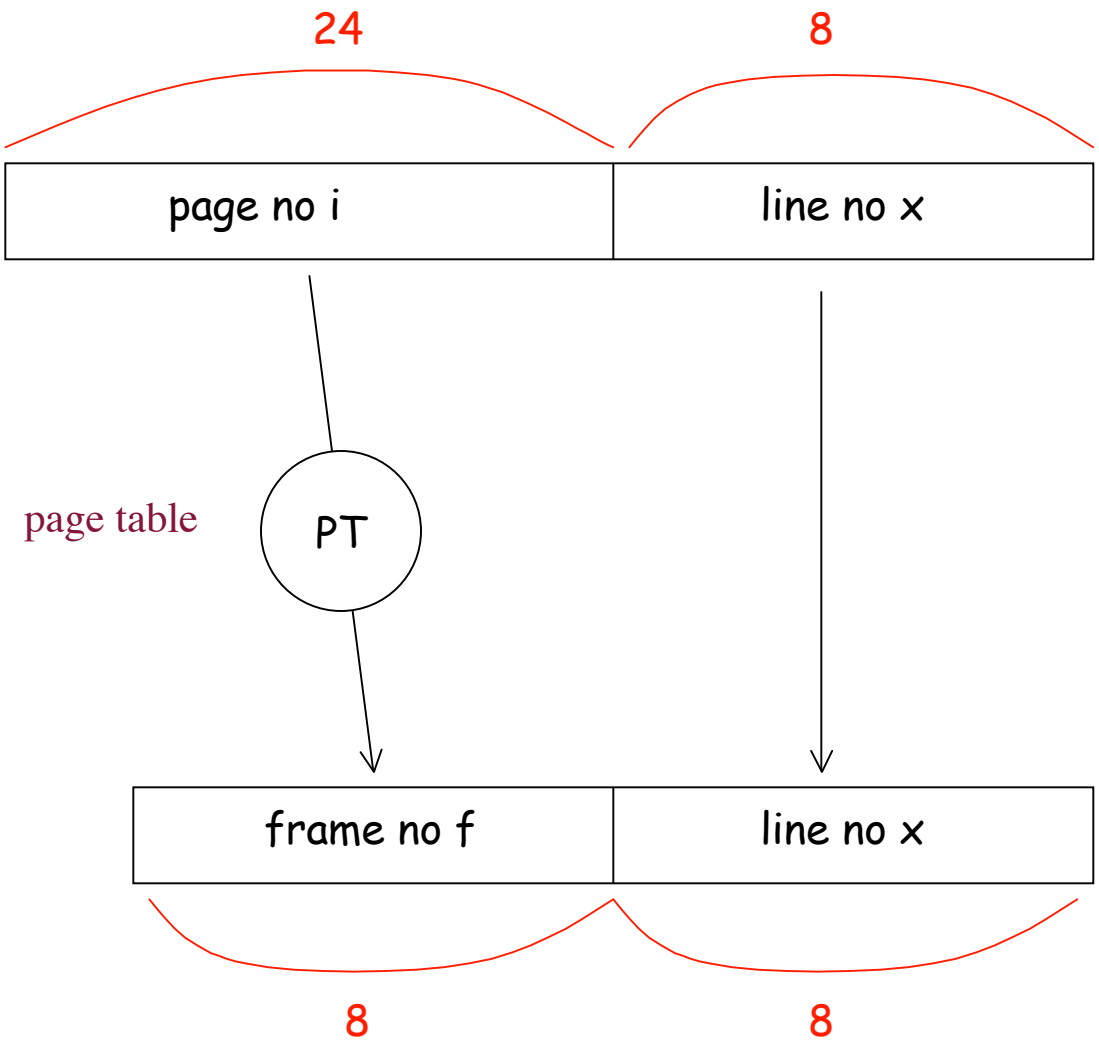
© 2001-2002, P. J. Denning

Paging 3

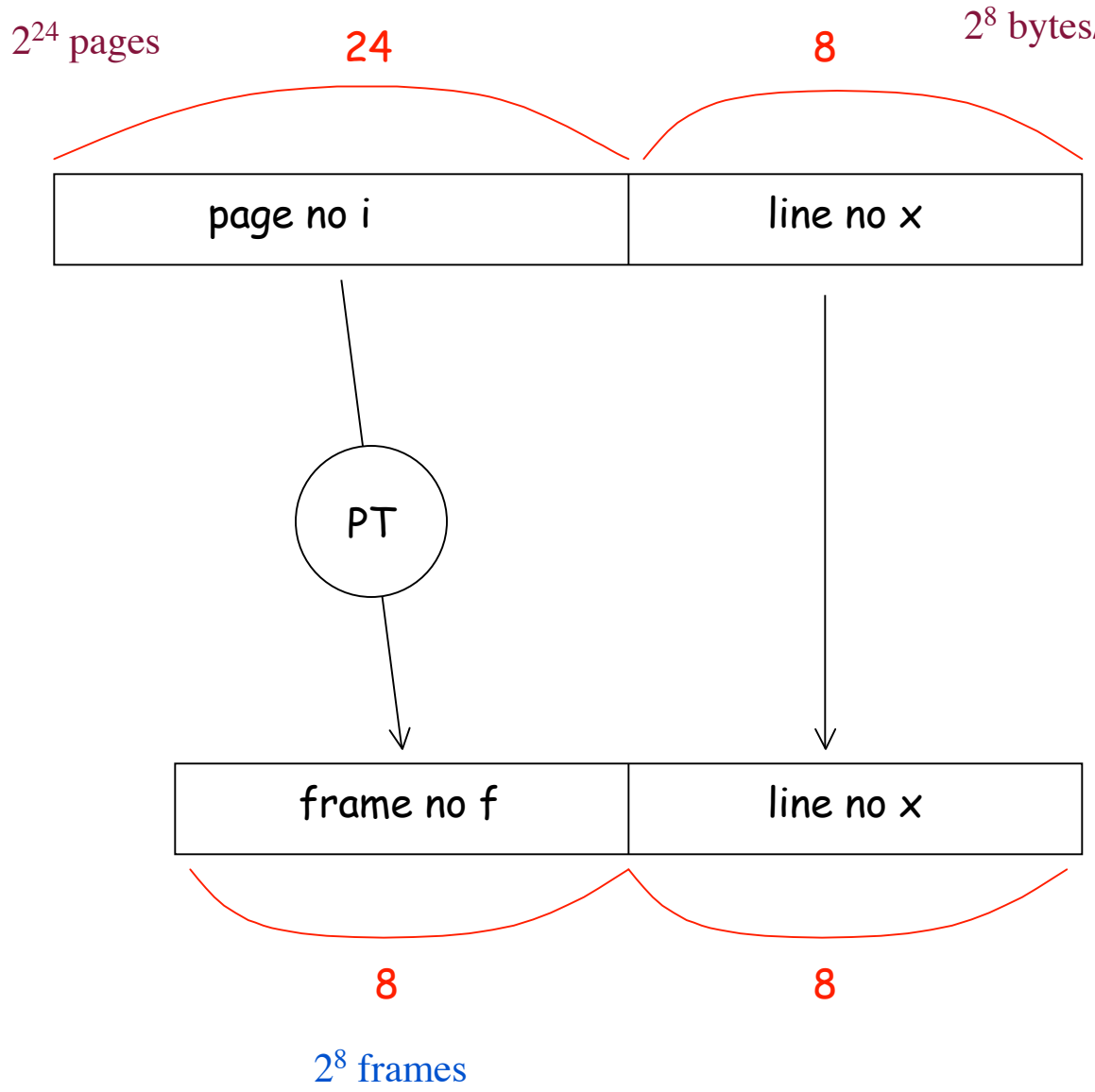
artificial
contiguity



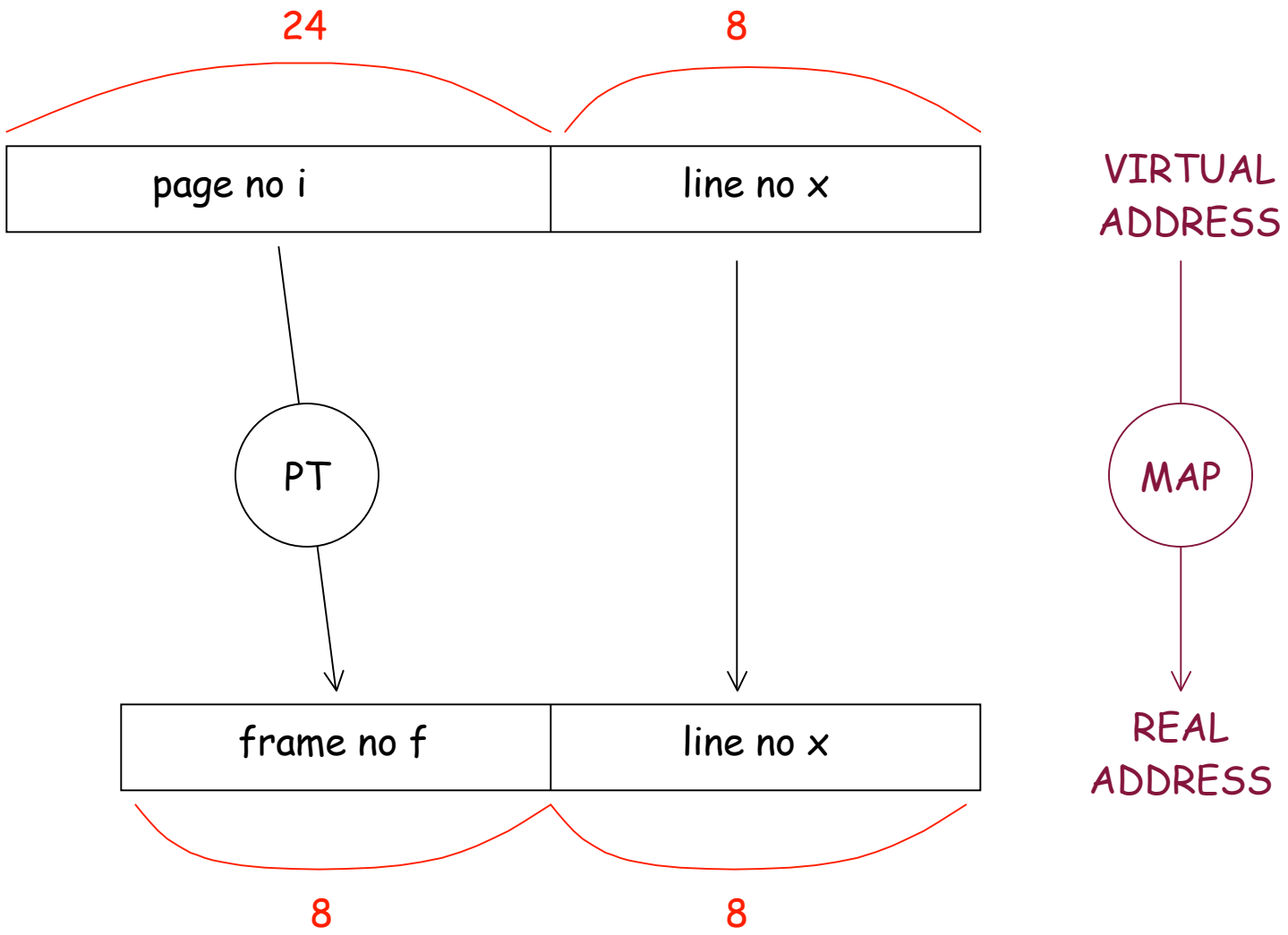
© 2001-2002, P. J. Denning



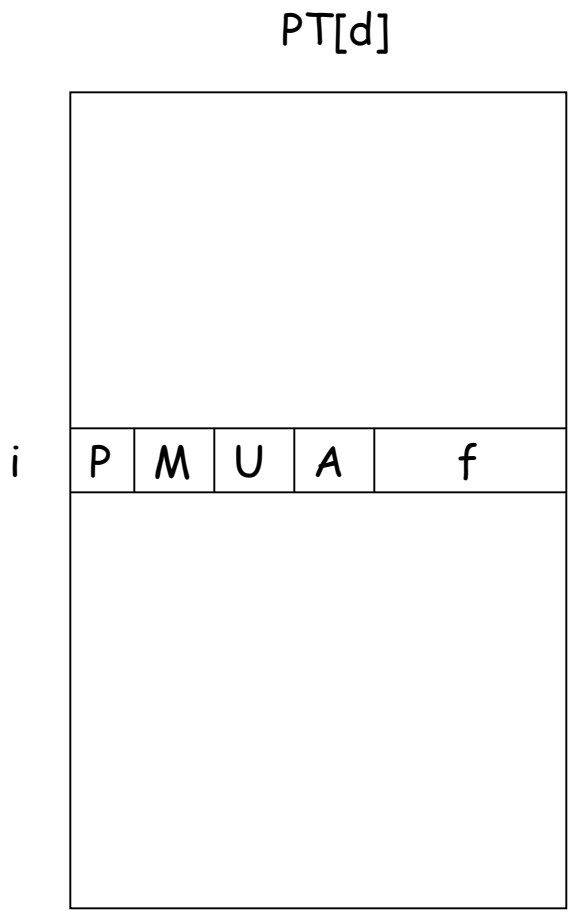
© 2001-2002, P. J. Denning



© 2001-2002, P. J. Denning



© 2001-2002, P. J. Denning



d = domain number

each process has a domain

i = page number

P = presence bit

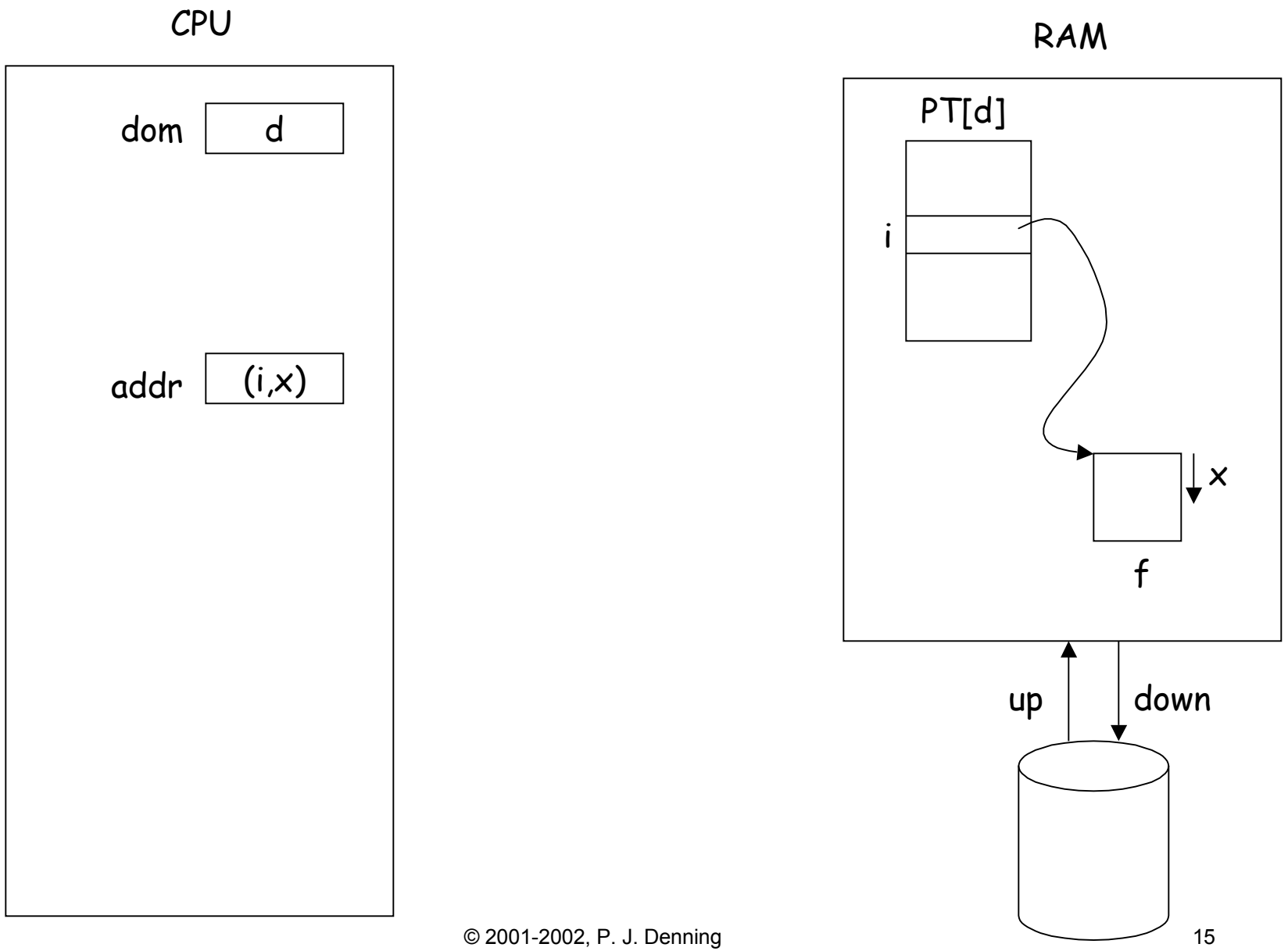
M = modified bit

U = used bit

A = access code

f = frame number

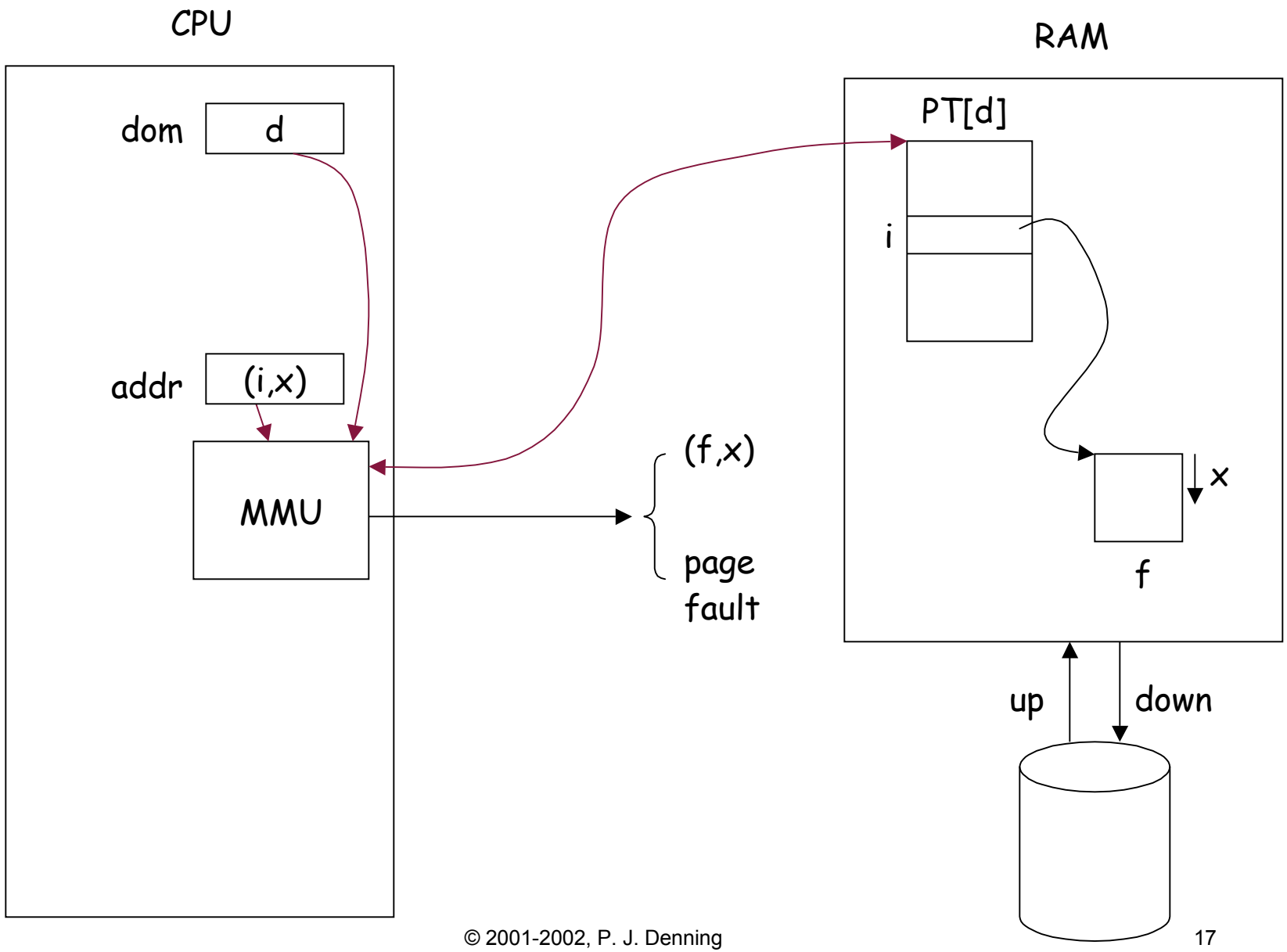
© 2001-2002, P. J. Denning



© 2001-2002, P. J. Denning

- CPU generates address (i,x)
- Objective: map to (f,x) in memory
- PT for CPU's domain d in RAM
- Copy of entire address space in file on disk (the "swap file" or "cache file")

© 2001-2002, P. J. Denning

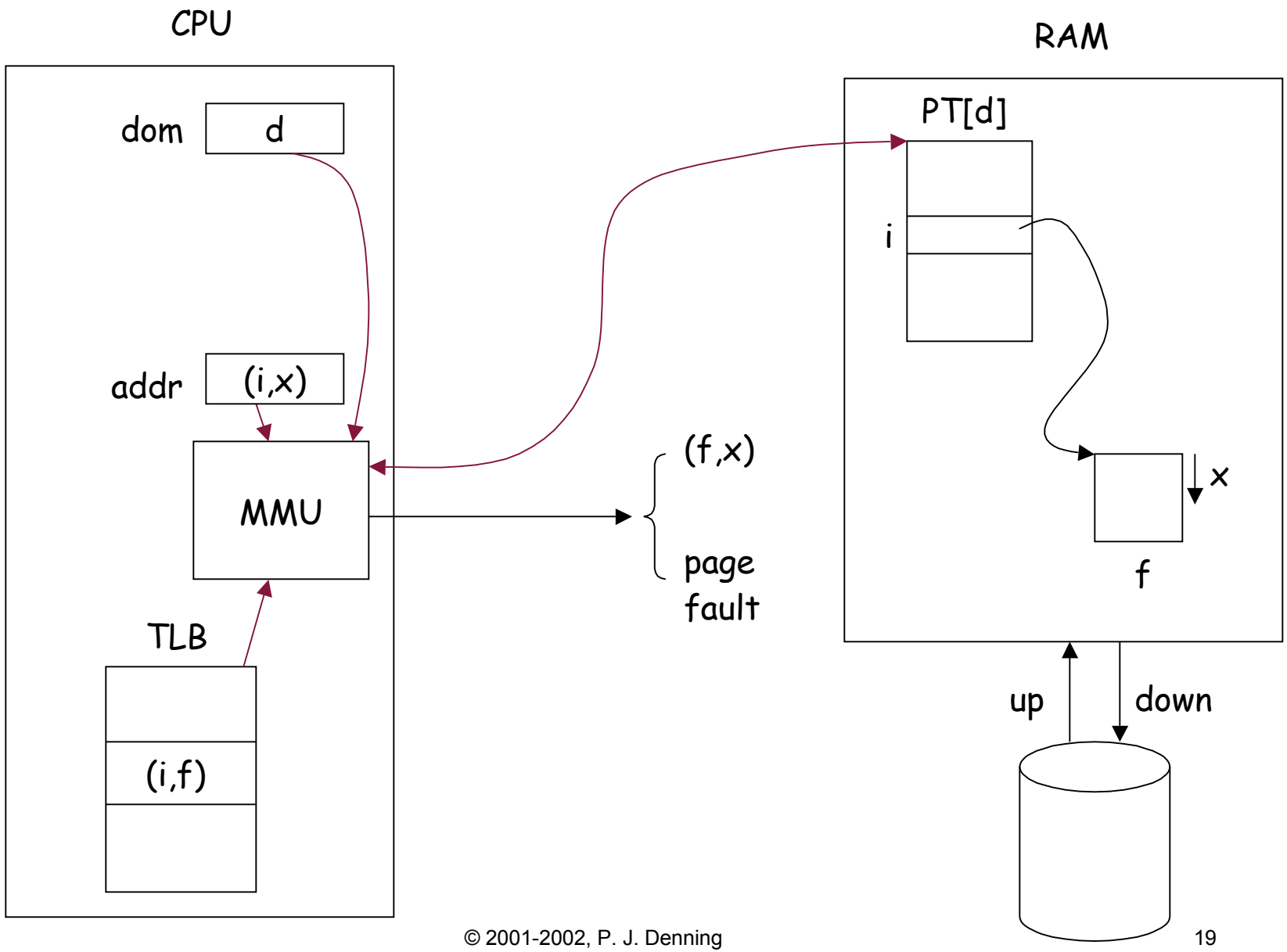


© 2001-2002, P. J. Denning

Memory Mapping Unit (in CPU) translates microcommands (RW, i,x)

```
On (RW,i,x) do
   E = PT[d,i]
   if not (E.A allows RW) then ACCESS FAULT
   if E.P=0 then PAGE FAULT
   (E.U, E.M) = (1, if RW=r then 0 else 1)
   PT[d,i]=E
   generate (PT.f, x)
```

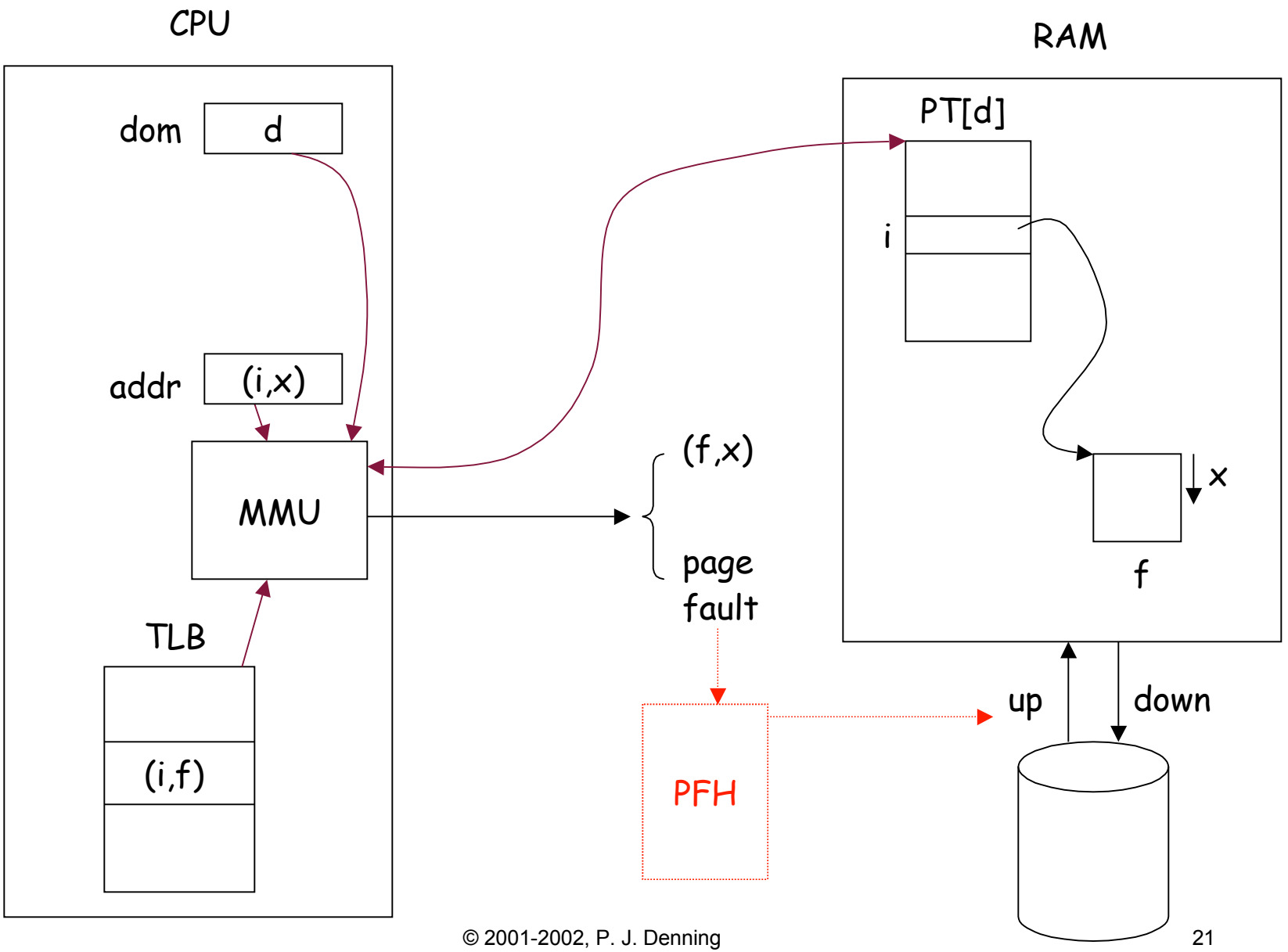
© 2001-2002, P. J. Denning



© 2001-2002, P. J. Denning

- Translation Look Aside Buffer (TLB) is cache of most recent address paths
- Holds most recent (i,f) pairs
  - (Actually (M,i,f) triplets.)
- Bypass a PT lookup if a "hit" occurs.
- Hit ratio h. Miss ratio (1-h).
- Average mapping time:
  - T = h*TLB + (1-h)*RAM
  - Easy to get below 3% with a few hundred TLB entries

© 2001-2002, P. J. Denning



© 2001-2002, P. J. Denning

- Page Fault Handler (PFH) resolves missing page exceptions (i,x) for CPU in domain d:
  - Using replacement policy: select frame (f); if frame modified, copy its occupying page to its disk home -- a "down" move.
  - Load missing page (i) into that frame (f) -- an "up" move.
  - Update PT[d,i]=(1,0,0,A,f)
  - Return to interrupted process; it retries address and now succeeds
- Structure PFH to eliminate as many of the swaps as possible.

© 2001-2002, P. J. Denning

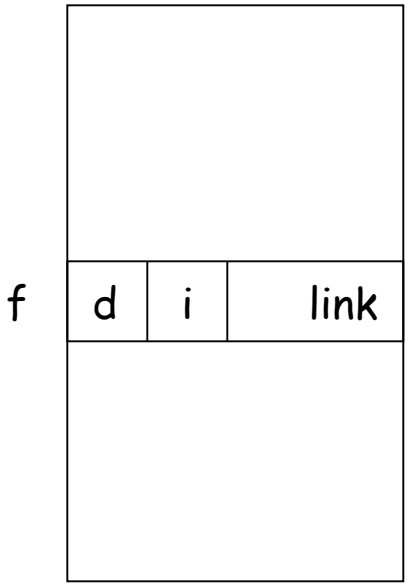
- Paging Manager contains the PFH routine and two daemons:
  - Replace Daemon identifies pages to replace, marks them P=0, and signals WriteBack Daemon for pages with M=1.
  - Replace Daemon targets to keep pool at a given minimum size.
  - WriteBack Daemon copies (P,M)=(0,1) pages to disk, then sets M=0.

© 2001-2002, P. J. Denning

- Because many processes can encounter page faults in parallel, the PFH gets pages from a semaphore-protected pool and notifies Replace Daemon for each withdrawal.
- Replace Daemon may replace multiple pages to replenish pool. When it replaces a modified page, it notifies WriteBack Daemon.
- WriteBack Daemon copies each page back to disk and then puts it in the pool.

© 2001-2002, P. J. Denning

FRAMES



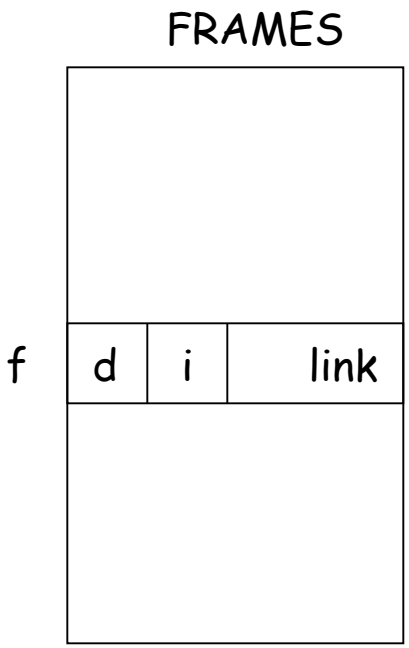
SETS

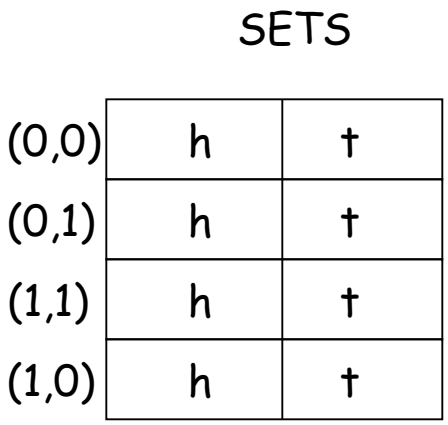
(0,0)	h	t
(0,1)	h	t
(1,1)	h	t
(1,0)	h	t

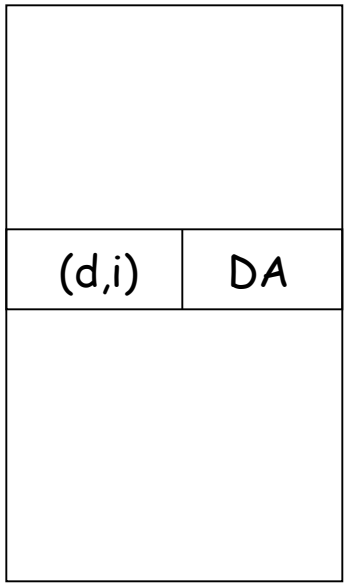
Paging Manager maintains a FRAMES table telling what page occupies each frame. Each frame linked to one of four sets according to their (P,M) values.

A missing page can be reclaimed without swap if FRAMES[PT[d,i].F]=(d,i).

© 2001-2002, P. J. Denning



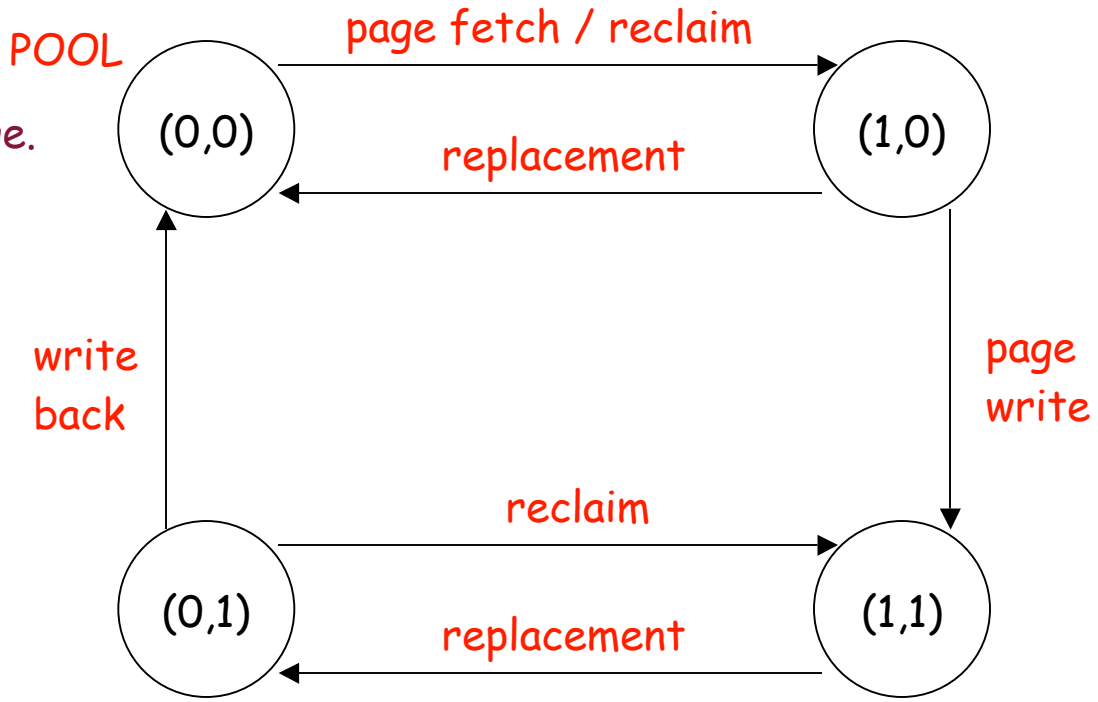




PM also maintains a DISKMAP that tells the disk address of every page.

© 2001-2002, P. J. Denning

The POOL contains all frames that are available to receive the next incoming page.



Frames are divided into sets by (P,M) combinations. Paging Manager manages the state transitions. The reclaim action occurs on a page fault when page is reclaimable; it simply sets P=1.

© 2001-2002, P. J. Denning

PFH(d,i):

```
WAIT(pool)
if FRAMES[f=PT[d,i].f]=(d,i)
   then {PT[d,i].P = 1; unlink f from SET[0,0]}
   else {
      f = unlink first frame of SET[0,0]
      enter (self-pid, r, f, DISKMAP[d,i]) in Disk Work Queue
      WAIT(self-sem)
      PT[d,i] = (1,0,0,A,f)
      poolsize--
      SIGNAL(replace)
      }
link f to SET[1,0]
return
```

Disk Driver protocol: enter request (self-pid, rw, a, b) in Disk Work Queue and wait on self-pid's private semaphore. Parameter a in caller's address space and b is a disk address.

When Disk Driver done, it signals the private semaphore.

© 2001-2002, P. J. Denning

Replace Daemon:

```
1:  WAIT(replace)
    while poolsize < threshold do {
       use replacement rule to unlink frame f from SET(1,-)
       PT[FRAMES[f]].P=0
       if PT[FRAMES[f]].M = 1
          then sendmsg(WriteBack, f)
          else {link f to SET[0,0]; poolsize++; SIGNAL(pool)}
    }
    goto 1
```

The system call sendmsg(p,m) places message m in the inbox of process p.

The system call m=getmsg() in process p returns the message; or waits if the inbox is empty.

© 2001-2002, P. J. Denning

WriteBack Daemon:

```
1:  f = getmsg()
    enter (self-pid, w, f, DISKMAP[FRAMES[f]]) in Disk Work Queue
    WAIT(self-sem)
    PT[FRAMES[f]].M = 0
    link f to SET[0,0]
    poolsize++
    SIGNAL(pool)
    goto 1
```

The frame f had been marked not present by ReplaceDaemon. When copy back to disk is complete, f can be linked to the POOL.

© 2001-2002, P. J. Denning

The Paging Manager is free of deadlock because every WAIT is followed by a SIGNAL that replenishes the count of the semaphore.

In the PFH, the WAIT(pool) is followed by SIGNAL(replace); the Replace Daemon SIGNAL(pool) for an unmodified replacement page and WriteBack Daemon SIGNAL(pool) for a modified page.

This structure gives parallelism between satisfying a fault, replenishing the pool, and writing a page back to disk. This increases the probability the pool is nonempty at the time of a fault.

© 2001-2002, P. J. Denning

OTHER MAPPING METHODS

- Paging
- Segmentation
- Segmentation and paging
- Capability addressing

© 2001-2002, P. J. Denning

Paging

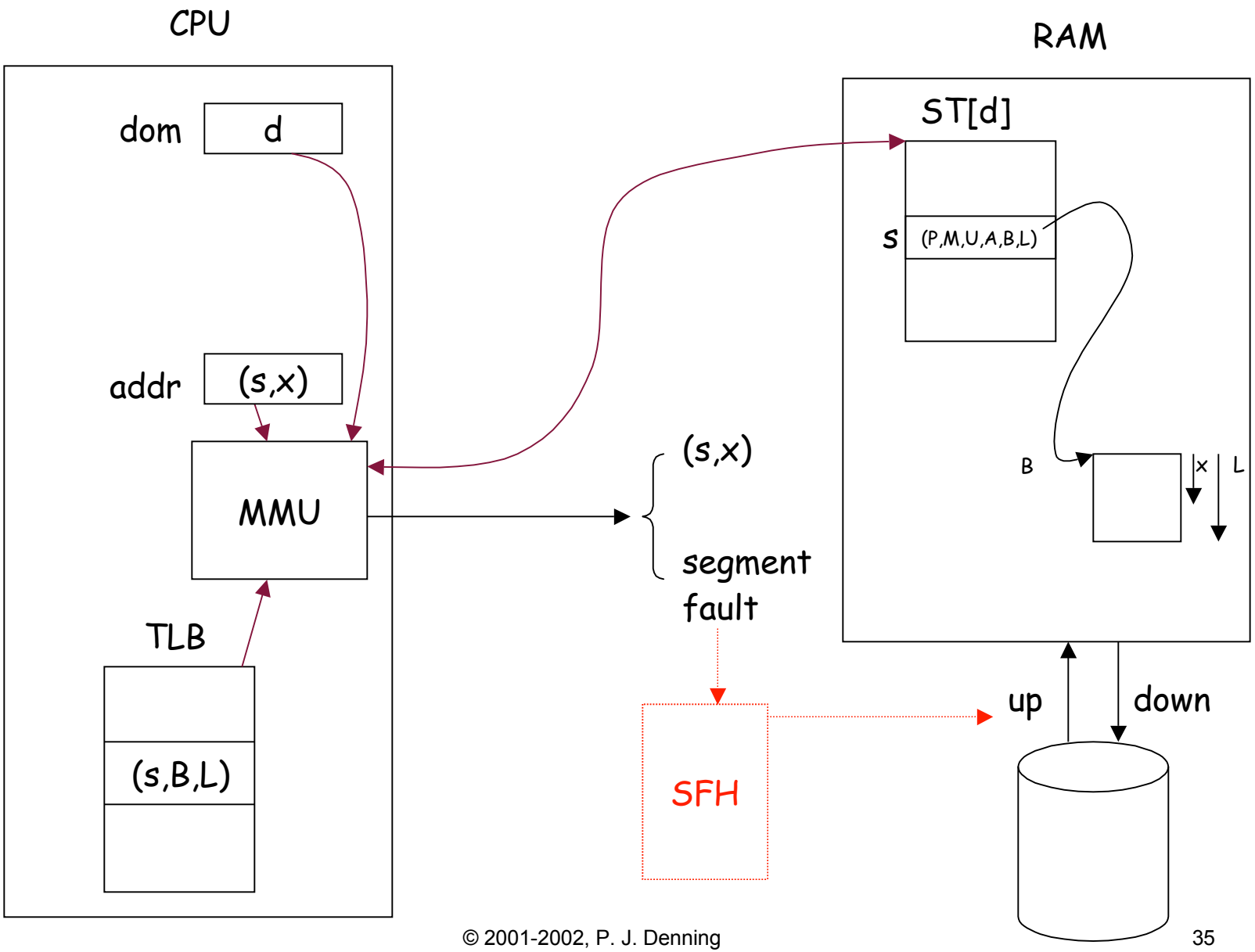
- Address space linear -- all addresses integers from 0 to a max.
- Memory space also linear.
- Both address and memory space divided into equal size blocks -- pages in address space, frames in memory space.
- Pages and frames not visible to programmer.
- Mapping is simple and fast.

© 2001-2002, P. J. Denning

Segmentation

- Address space is nonlinear (2-D) consisting of segments, each a definite size.
- Segments are good containers for objects defined in the program. Compiler assigns objects to segments.
- Addresses are of the form (s,x) -- segment number, offset. Offset cannot exceed segment length.
- Memory is linear, segments stored as contiguous blocks. External fragmentation a problem.

© 2001-2002, P. J. Denning



© 2001-2002, P. J. Denning

Segmentation Modes

- Automatic Mode: Programmer unaware of segment boundaries; compiler assigns objects to segments.
  - Burroughs Algol machines (B5000, B6700)
- Manual Mode: Programmer aware of segments; gives symbolic names (S,X) in program; compiler translates to internal (s,x) codes.
  - Multics (GE645)
- No major machines today

© 2001-2002, P. J. Denning

Dynamic Linking

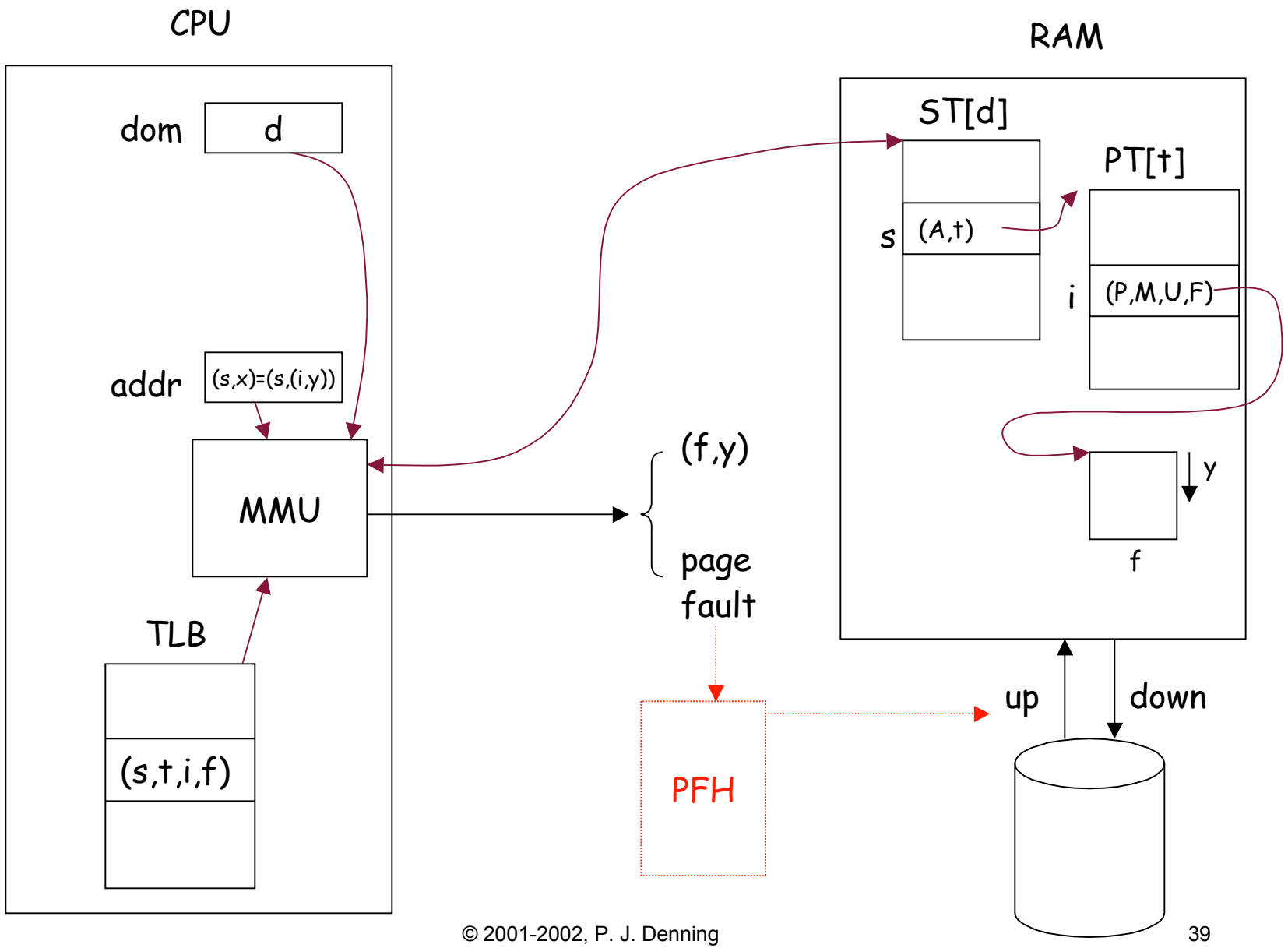
- In manual mode, new possibility arises: program contains symbolic reference (S,X) but no segment number has been assigned to S; S is a file and X a variable within S.
- Compiler can leave the symbolic reference in the code. When it is encountered for the first time, system generates a *linkage fault*.
- Linkage fault handler copies file S into a new segment s, creates ST entry for s, replaces the symbolic reference with its segment number, and returns control to the program.

© 2001-2002, P. J. Denning

Segmentation + Paging

- Combines the two by dividing segments into pages and giving each segment its own page table.
- Eliminates the external fragmentation problem caused by variable size segments in RAM.
- More complex mapping with both ST and PT.

© 2001-2002, P. J. Denning



© 2001-2002, P. J. Denning

MMU-TLB interaction

TLB entries (s,t,i,f)

MMU interrogates TLB with key (s,i)

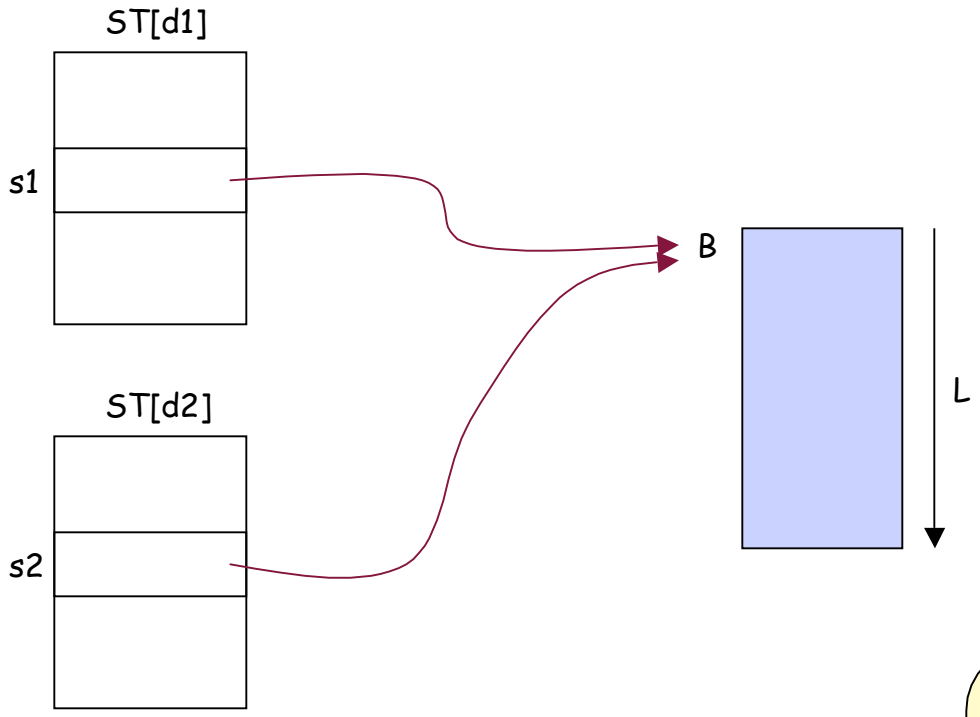
- Bypasses ST and PT lookups if it finds a full match;
  then it gets f immediately.
- Bypasses ST lookup if it finds partial match on segment number s;
  then it must lookup PT entry i to get f.
- Does both ST and PT lookups if it finds no match on segment number s;
  then it must lookup both ST and PT entries to get f.
- Then MMU generates its output (f,y).

© 2001-2002, P. J. Denning

Problem of Sharing

- How do two processes share a segment? Desire to share comes up after programs are written. Can we share without recompiling the programs?
- Difficult with pure segmentation. The parties do not need the same segment number because both their ST entries can point to the same (B,L) region. But if OS needs to move the segment or swap it out, it has to locate all ST and update their entries.
- Easier with segmentation-paging. The parties have different segment numbers pointing to the same PT. Any change to a page's position is recorded once, in the PT entry. All parties will see the change immediately.

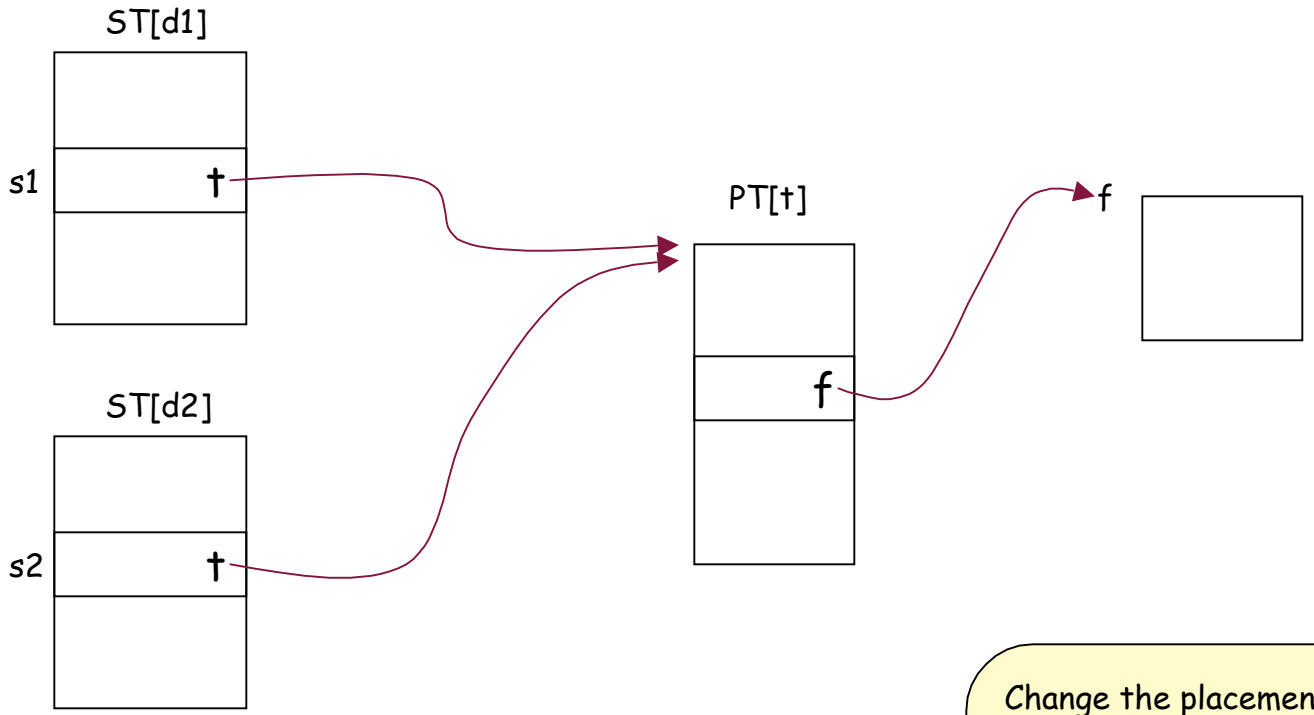
© 2001-2002, P. J. Denning



change the placement or size of the segment ==>

back-propagate the new info to all ST's pointing to the segment.

© 2001-2002, P. J. Denning



Change the placement of the page ==>

Update just one PT entry.

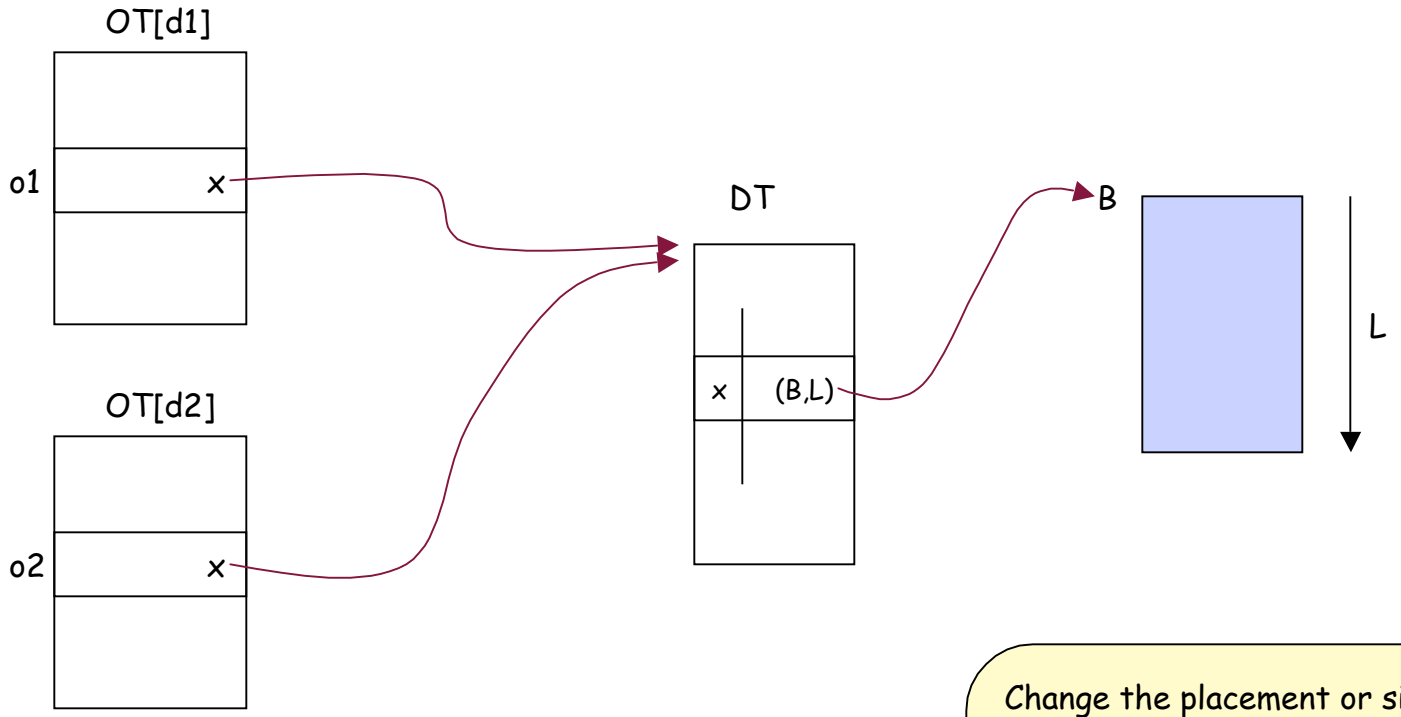
No back-propagation.

© 2001-2002, P. J. Denning

Capability Addressing

- Generalize from segments to objects.
- Facilitate object sharing with location transparency.
- Capabilities = object handles recognized throughout a system or network; protected from alteration.
- Three-level mapping: symbolic name to capability, capability to descriptor, descriptor to object.

© 2001-2002, P. J. Denning



"o1" and "o2" are compiler-assigned codes for symbolic names chosen by the user.

Change the placement or size of the object x ==>

Update just one descriptor (x).

DT a large hash table. Object ids (x) are used just once.

© 2001-2002, P. J. Denning

Modern Capability Examples

- Object-oriented programming:
  - Run-time system generates handles, which map through descriptors to objects.
  - User chooses symbolic names for objects; run-time system associates those names with their handles.
  - Three-level map:
    - name → handle
    - handle → descriptor
    - descriptor → object.

© 2001-2002, P. J. Denning

- E-Mail Systems:
  - Handle: user@host (unique)
  - Three-level mapping:
    - Alias to handle (address book)
    - Handle to IP address (DNS service)
    - IP address to mailbox (IP)

© 2001-2002, P. J. Denning

- Handle Protocol (see handle.net):
  - Give objects a location-dependent, unique name in the Internet.
  - Handle: an object ID chosen within a hierarchical system that makes handle-ids unique.
  - Three-level mapping:
    - Document text reference to handle (handle://unique-id)
    - Handle-id to hostname/pathname (handle server)
    - hostname/pathname to object (http)

© 2001-2002, P. J. Denning

© 2001-2002, P. J. Denning