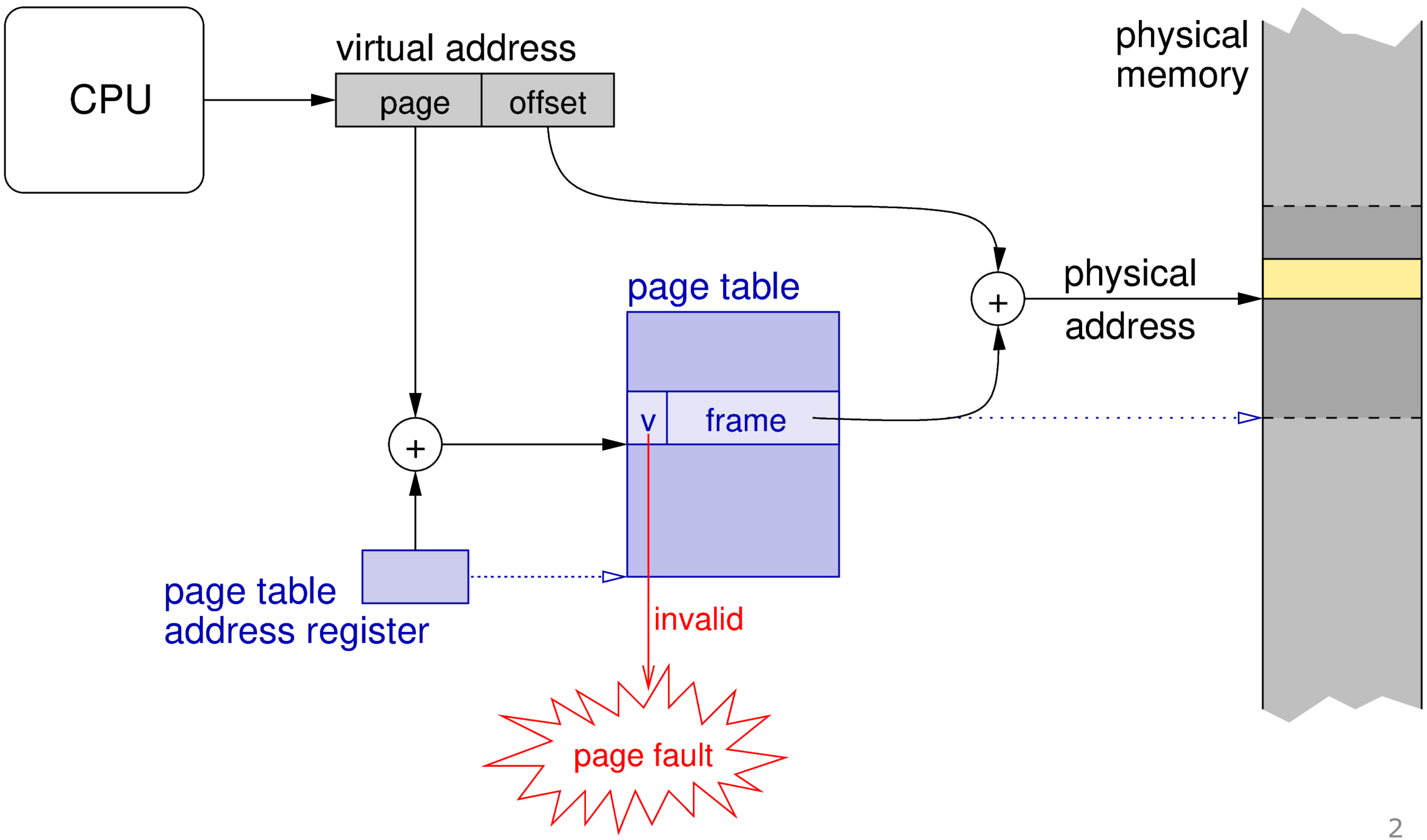
Operating Systems
Paging and Locality

David Hay
Dror Feitelson

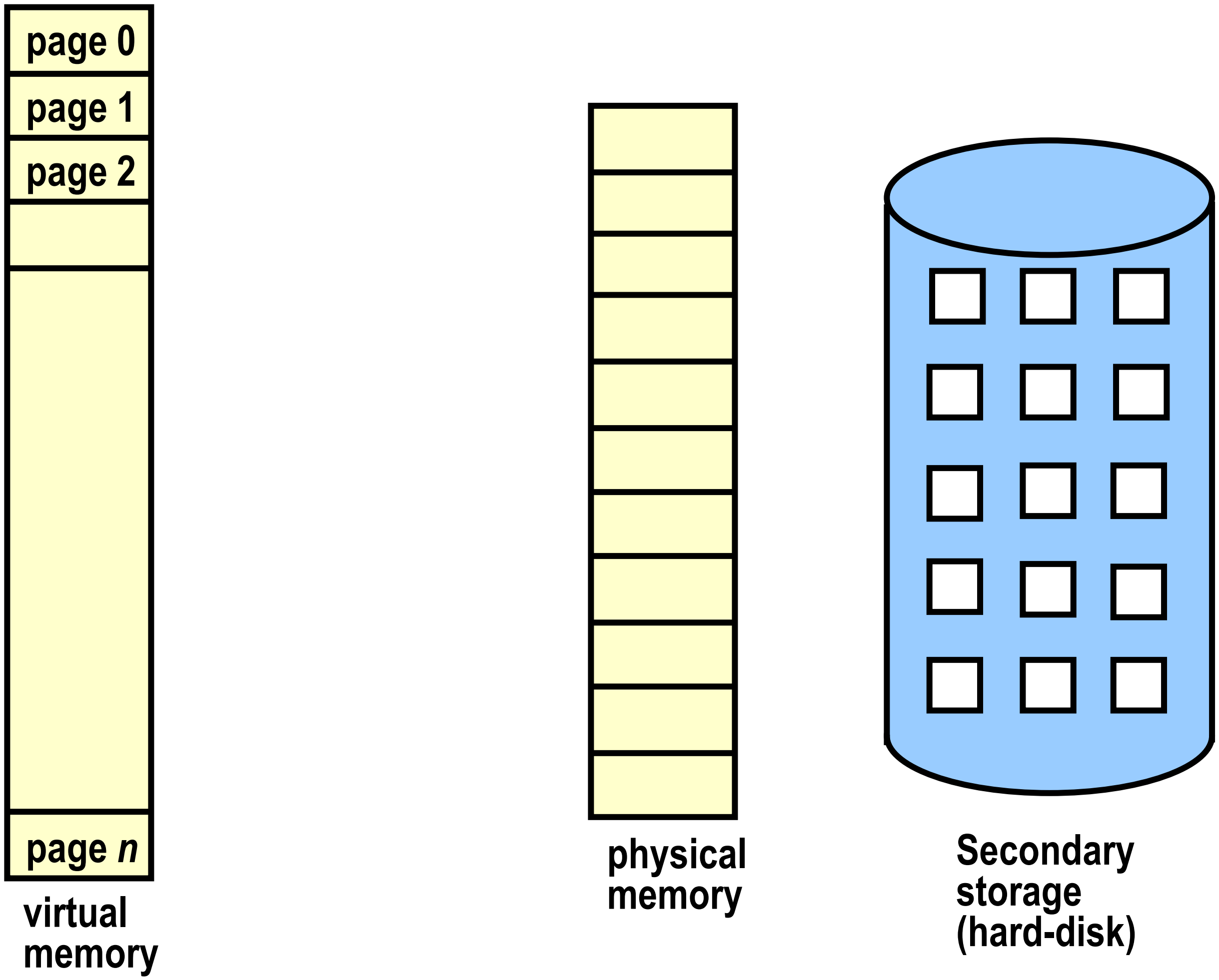
Address Translation



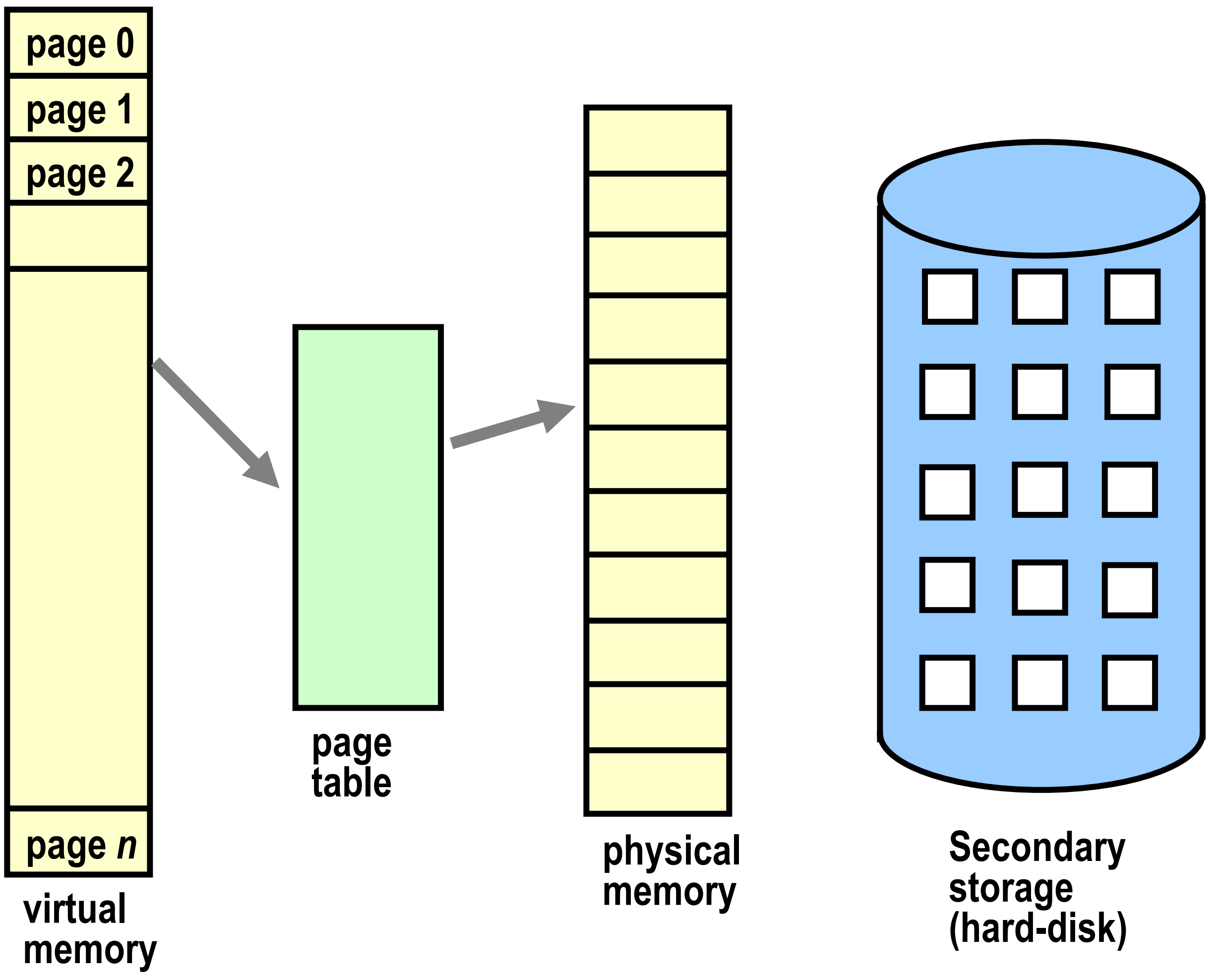
Virtual Memory

- The idea: VIRTUALIZATION
  - Disconnect from the limitations of our physical budget
  - Make it look as if we have all the memory we want
- The implementation: DEMAND PAGING
  - Bring pages to memory when we need them
  - Store them on disk when we don’t

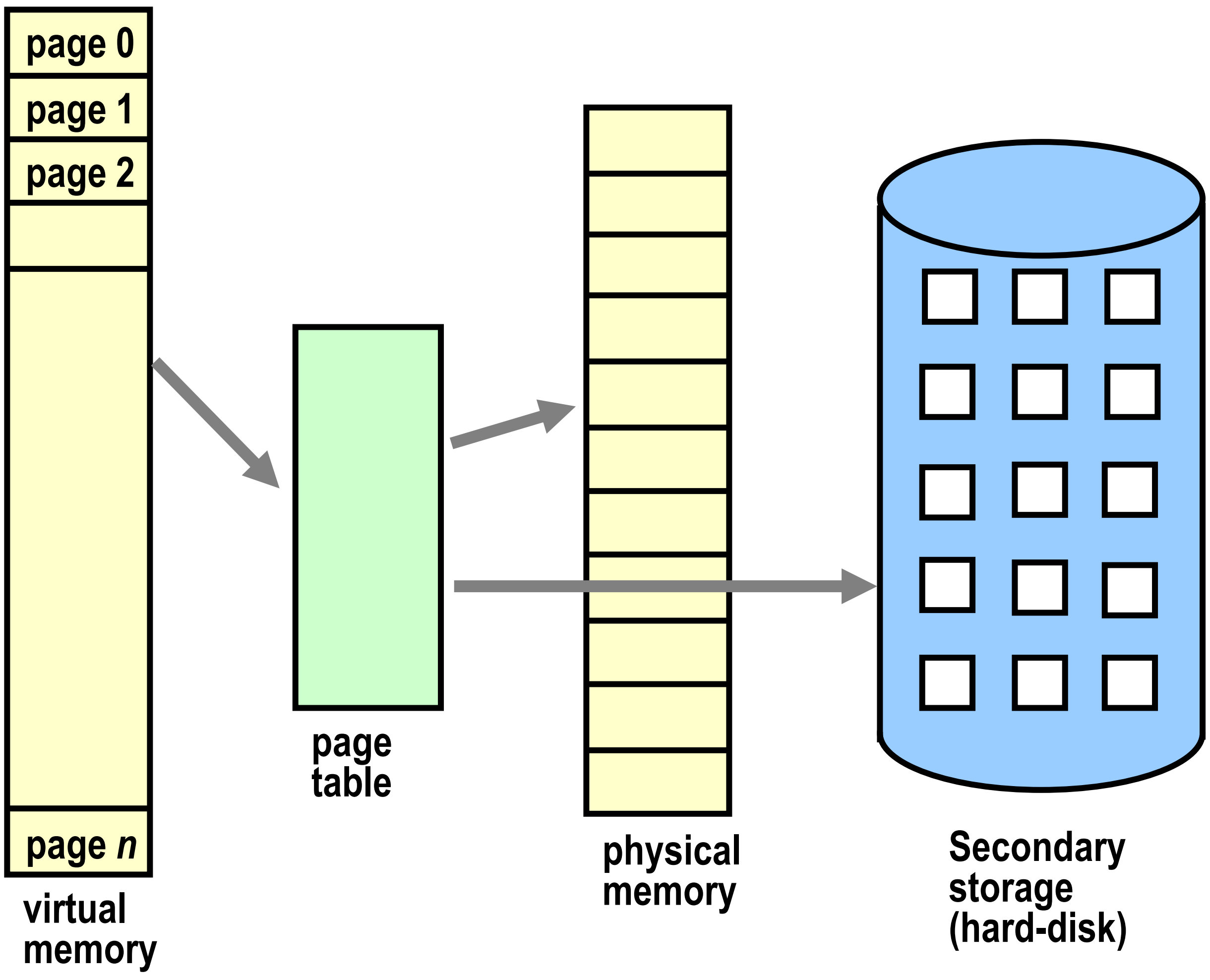
Dynamics of Demand Paging



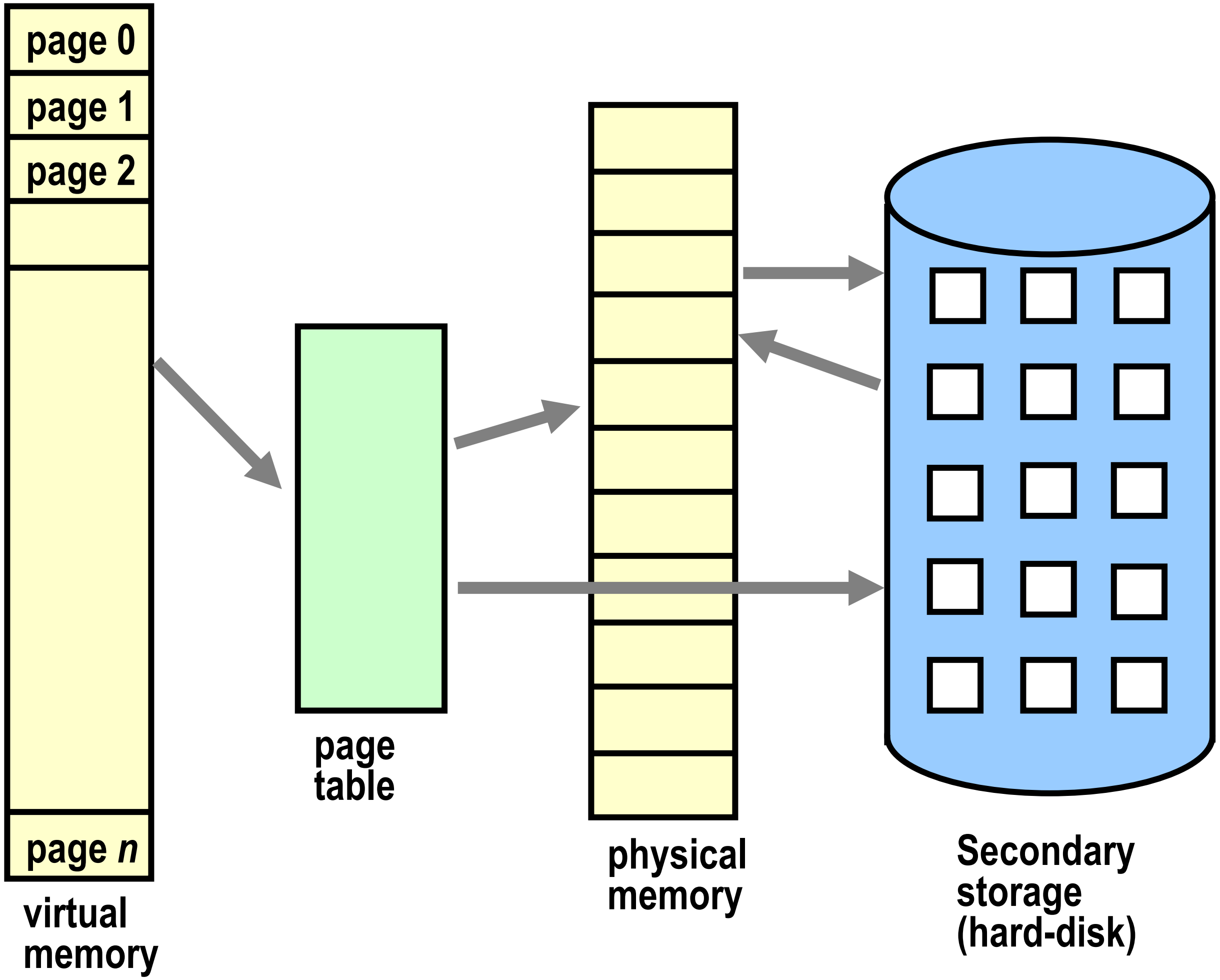
Dynamics of Demand Paging



Dynamics of Demand Paging



Dynamics of Demand Paging



Replacement Algorithms

- How do we choose the *victim*?

Replacement Algorithms

- How do we choose the *victim*?
  - We can just choose at random…

Replacement Algorithms

- How do we choose the *victim*?
  - We can just choose at random…
  - But better not to choose often used pages (will probably need to be brought back in soon)

Replacement Algorithms

- How do we choose the *victim*?
  - We can just choose at random…
  - But better not to choose often used pages (will probably need to be brought back in soon)
- Many policies are possible
  - Optimal
  - Random
  - FIFO (first-in-first-out), second chance FIFO
  - NRU (not recently used)
  - LRU (least recently used), pseudo-LRU
  - LFU (least frequently used)
  - Etc

Performance

p = probability of page fault

Effective access time = p(page fault time)
+(1-p)(memory access time)

Slowdown = Effective access time / memory access time

Typical numbers:

page fault time: 25 ms

access time: 100 ns

p=0.001 → slowdown = 250

p=0.0000004 → slowdown = 1.1

Performance

p = probability of page fault

Effective access time = p(page fault time)
+(1-p)(memory access time)

Slowdown = Effective access time / memory access time

Typical numbers:
page fault time: 25 ms

access time: 100 ns

p=0.001 → slowdown = 250

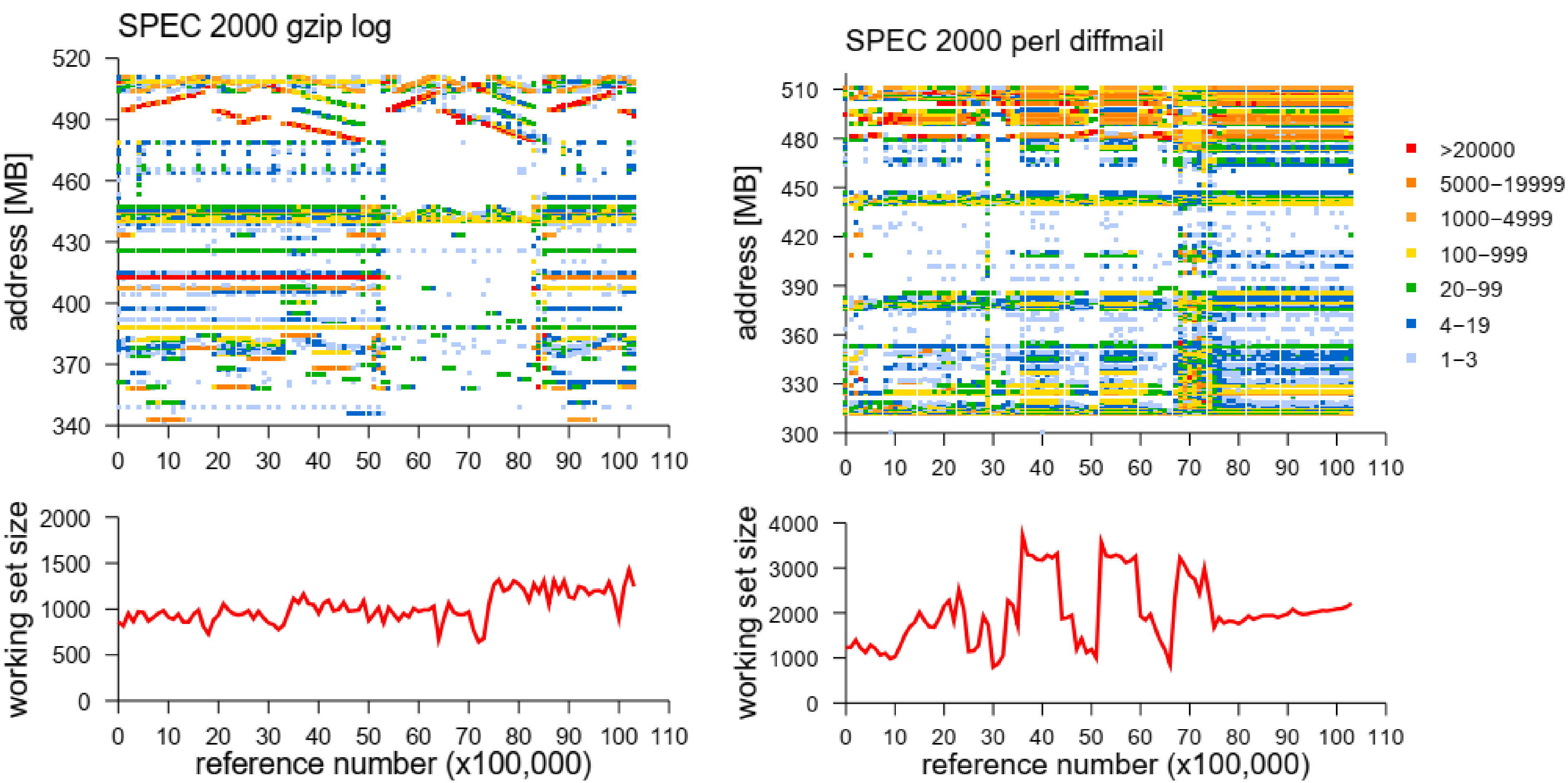
p=0.0000004 → slowdown = 1.1

Performance depends on locality

- Ensure that costly disk operations are rare
- Amortize them across many memory accesses

WORKLOADS

Eyeballing Memory Accesses



Reminder: The Principle of Locality

Temporal locality:

If we accessed a certain address, the chances are high to access it again shortly.

- Data: updating
- Instructions: loops

Spatial locality:

If we accessed a certain address, the chances are high to access its neighbors.

- Data: arrays
- Instructions: sequential execution

Temporal Locality

Actually reflects two separate phenomena

Temporal Locality

Actually reflects two separate phenomena

- Clustered accesses

  A A A A A B B B B B C C C C C D D D D D

Temporal Locality

Actually reflects two separate phenomena

- Clustered accesses

  A A A A A B B B B B C C C C C D D D D D

- Skewed popularity

  B A B B B B C B B B A B B B B B D B C B

Measuring Locality

- How do we know locality really exists?
- How can we characterize the degree of locality given a certain address stream?
- Measure the STACK DISTANCE:
  1. Scan the address stream
     a. Search for each address in the stack; if found note its depth and extract it.
     b. Push the address at the top of the stack.
  2. Output the distribution of depths at which addresses were found.

Measuring Locality

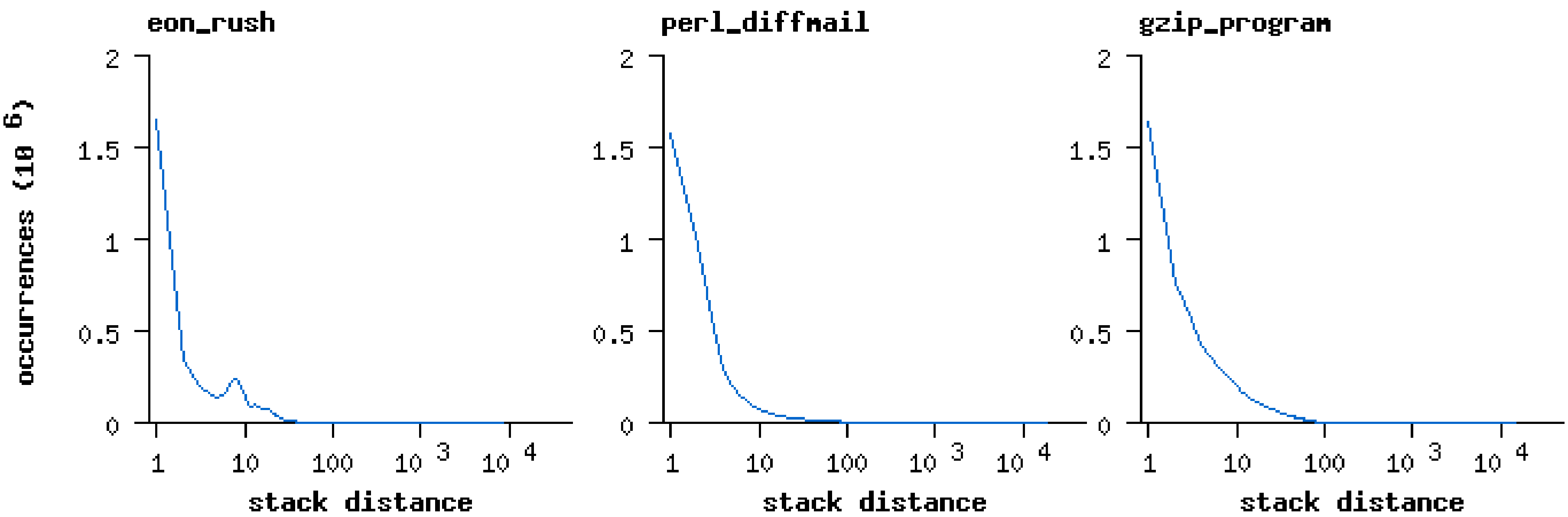
- How do we know locality really exists?
- How can we characterize the degree of locality given a certain address stream?
- Measure the STACK
  1. Scan the address
     a. Search for eac
        found note its
     b. Push the address at top of the stack.
  2. Output the distribution of depths at which addresses were found.

= The distribution of distances between successive accesses to the same address

Example

Access	1	3	1	1
Stack	1	3 1	1 3	1 3
Distance	∞	∞	2	1
Hits	Hit[1]=0 Hit[2]=0 Hit[3]=0 Hit[∞]=1	Hit[1]=0 Hit[2]=0 Hit[3]=0 Hit[∞]=2	Hit[1]=0 Hit[2]=1 Hit[3]=0 Hit[∞]=2	Hit[1]=1 Hit[2]=1 Hit[3]=0 Hit[∞]=2

Stack Distance Examples



- Using SPEC 2000 benchmarks
- Observed stack distances predominantly < 10

Stack Distance and LRU

- The stack distance distribution is all we need to evaluate LRU performance!
- Assume memory of size k
- Top k items in stack are the k least recently used items
- So they are the ones retained by LRU
- p = probability of page fault

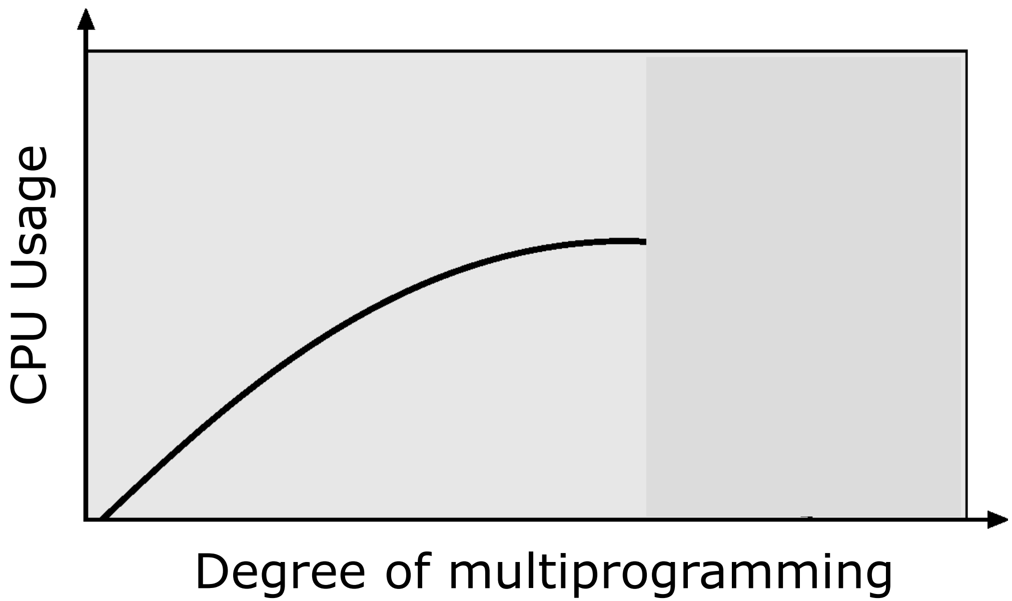
  = probability of $>k$ in the stack

Page Faults and Multiprogramming

- **CPU Usage:** The *fraction* of time the CPU is executing instructions
- Multiprogramming increases CPU Usage

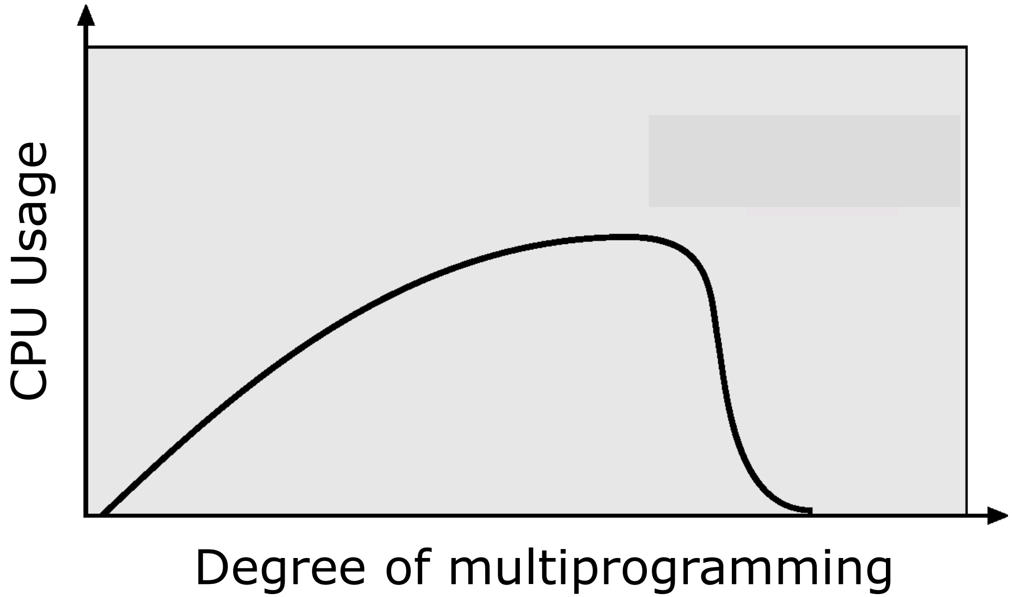
Page Faults and Multiprogramming

- **CPU Usage:** The *fraction* of time the CPU is executing instructions
- Multiprogramming increases CPU Usage



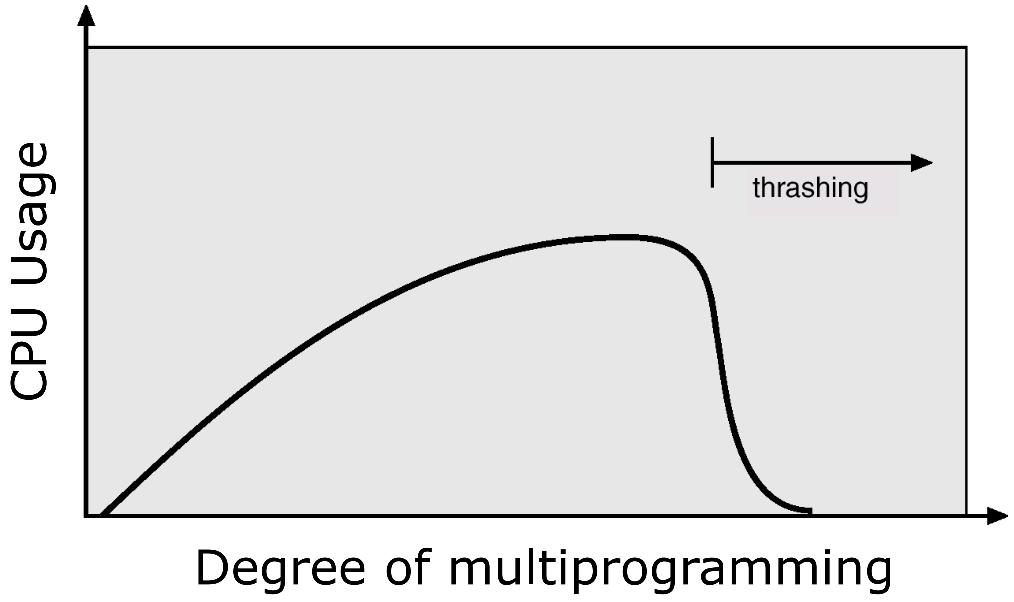
Page Faults and Multiprogramming

- **CPU Usage:** The *fraction* of time the CPU is executing instructions
- Multiprogramming increases CPU Usage



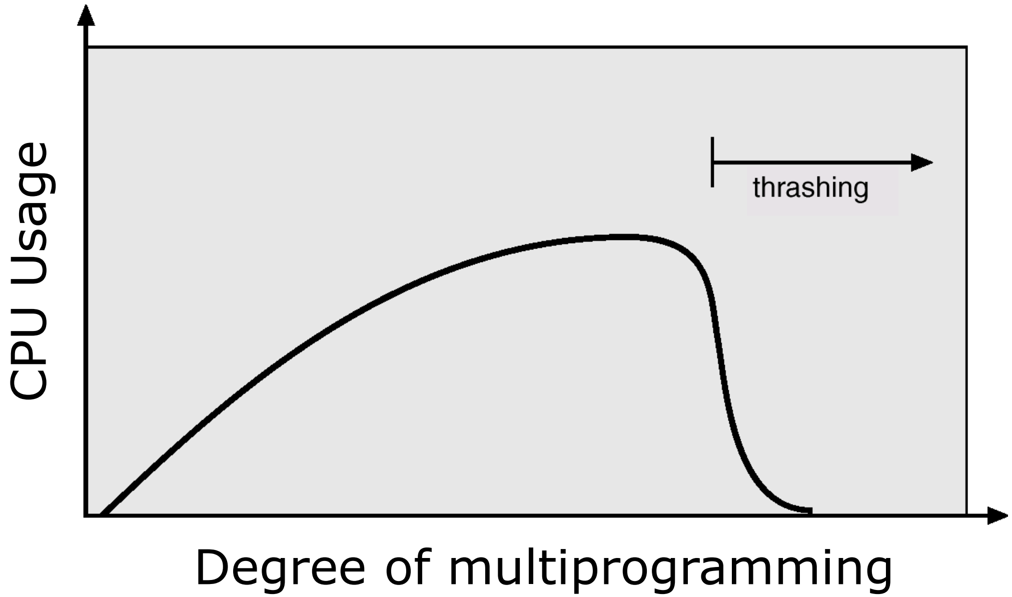
Page Faults and Multiprogramming

- **CPU Usage:** The *fraction* of time the CPU is executing instructions
- Multiprogramming increases CPU Usage



Page Faults and Multiprogramming

- **CPU Usage:** The *fraction* of time the CPU is executing instructions
- Multiprogramming increases CPU Usage



Thrashing: the system is busy swapping pages in and out → no process makes any progress

Thrashing

Thrashing

- Why does paging work?
  - Locality (of reference) model
  - Process migrates from one locality to another
  - Localities may overlap

Thrashing

- Why does paging work?
  - Locality (of reference) model
  - Process migrates from one locality to another
  - Localities may overlap
- Why does thrashing occur?
  - size of locality > total memory size
    (i.e. the combined working sets of all processes exceed the total memory capacity)

Thrashing

- Why does paging work?
  - Locality (of reference) model
  - Process migrates from one locality to another
  - Localities may overlap
- Why does thrashing occur?
  - size of locality > total memory size
    (i.e. the combined working sets of all processes exceed the total memory capacity)
- Solution: limit degree of multiprogramming
  - Suspend processes when exceeding the degree

Diagram of Process States

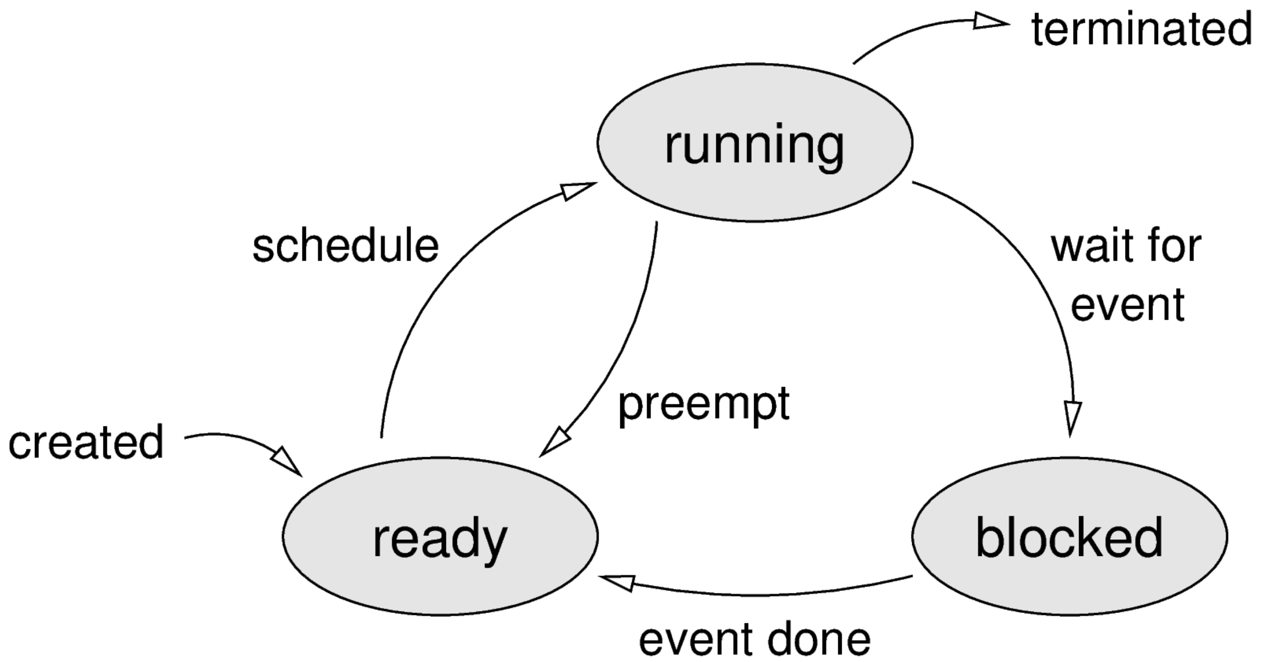


Diagram of Process States

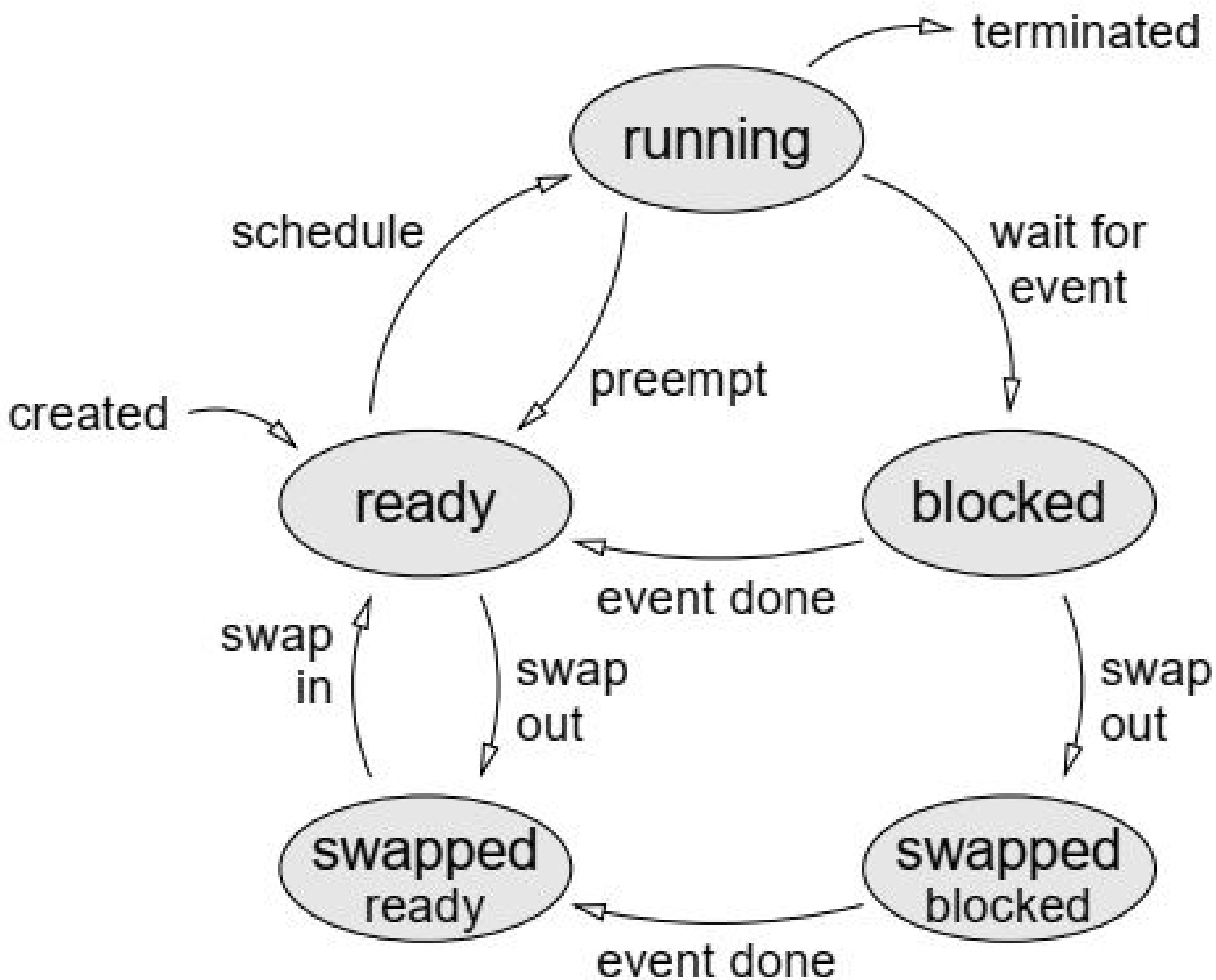
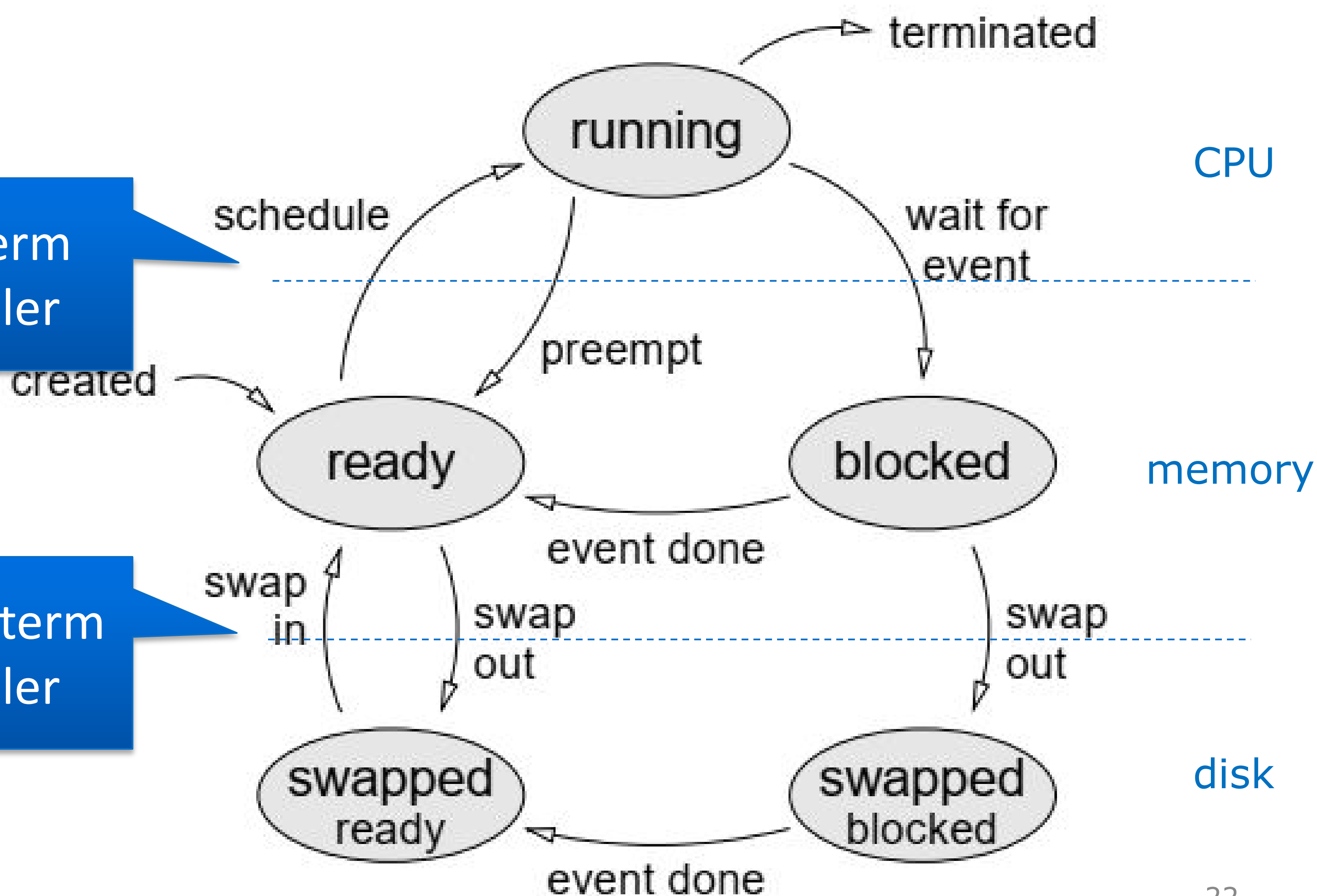


Diagram of Process States



Short-term Scheduler

Medium-term Scheduler

BIG MEMORIES

Page Table Size

Page Table Size

- In 32-bit CPUs, if page size is 4KB → Each process has 2^{20} pages

Page Table Size

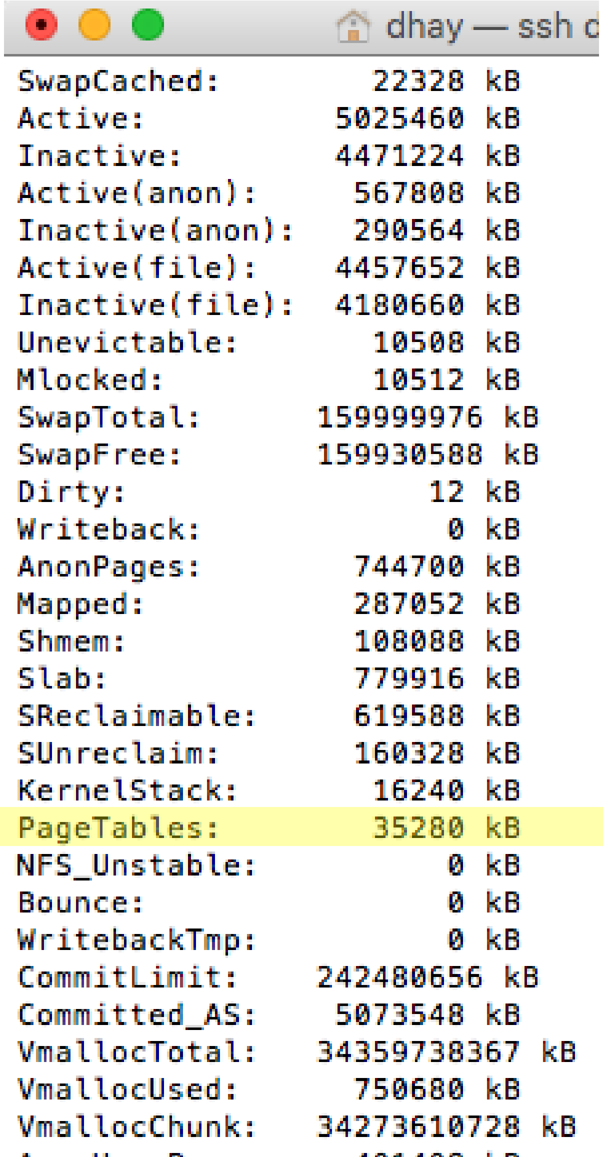
- In 32-bit CPUs, if page size is 4KB → Each process has 2^{20} pages
- If each page entry is 4B, then the page table itself requires 1000 pages…
  - Page table size reflects size of **logical memory**

Page Table Size

- In 32-bit CPUs, if page size is 4KB → Each process has 2^{20} pages
- If each page entry is 4B, then the page table itself requires 1000 pages…
  - Page table size reflects size of **logical memory**
- What happens in a 64-bit CPU?

Page Table Size

- In 32-bit CPUs, if page size is 4KB → Each process has 2^{20} pages
- If each page entry is 4B, then the page table itself requires 1000 pages…
  - Page table size reflects size of **logical memory**
- What happens in a 64-bit CPU?



```
SwapCached:          22328 kB
Active:            5025460 kB
Inactive:          4471224 kB
Active(anon):       567808 kB
Inactive(anon):     290564 kB
Active(file):      4457652 kB
Inactive(file):    4180660 kB
Unevictable:         10508 kB
Mlocked:             10512 kB
SwapTotal:       159999976 kB
SwapFree:        159930588 kB
Dirty:                  12 kB
Writeback:               0 kB
AnonPages:          744700 kB
Mapped:             287052 kB
Shmem:              108088 kB
Slab:               779916 kB
SReclaimable:       619588 kB
SUnreclaim:         160328 kB
KernelStack:         16240 kB
PageTables:          35280 kB
NFS_Unstable:            0 kB
Bounce:                  0 kB
WritebackTmp:            0 kB
CommitLimit:     242480656 kB
Committed_AS:      5073548 kB
VmallocTotal:  34359738367 kB
VmallocUsed:        750680 kB
VmallocChunk:  34273610728 kB
```

Page Table Size – 64-Bit CPU

- 2^{64} bytes logical address space
  - Many CPUs limit the logical address space to 2^{48} bytes
- Page size still 2^{12} bytes; entry in the page table is 2^3 bytes

→The page table itself is 2^{43} pages

Let's say physical memory is 32GB=2^{35} bytes → 2^{23} pages

→ At least 2^{43}-2^{23} pages of the page table are completely invalid

Page	Frame	
0	2	1
1	3	1
2	X	0
3	X	0
4	1	1
5	0	1
6	X	0

Solutions

- Hierarchical Page Tables
  - Break up the logical address space into multiple page tables
  - A simple technique is a two-level page table
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Page Table

Virtual memory

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p
16	q
17	r
18	s
19	t
20	u
21	v
22	w
23	x
24	y
25	z
26	aa

Page Table

Page	Frame	
0	2	1
1	3	1
2	X	0
3	X	0
4	1	1
5	0	1
6	X	0

(per process)

Used by the MMU

Physical memory

0	u
1	v
2	w
3	x
4	q
5	r
6	s
7	t
8	a
9	b
10	c
11	d
12	e
13	f
14	g
15	h

Hierarchical Page Table

Virtual memory

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p
16	q
17	r
18	s
19	t
20	u
21	v
22	w
23	x
24	y
25	z
26	aa

Top Level Page Table

Page1	Frame	v
0	4	1
1	7	1
2	X	0
3	X	0

Page2	Frame	v
0	2	1
1	3	1
2	X	0
3	X	0

Page2	Frame	v
0	X	0
1	1	1
2	0	1
3	X	0

Page2	Frame	v
0	X	0
1	X	0
2	X	0
3	X	0

Used by MMU

Physical memory

0	y
1	z
2	aa
3	bb
4	u
5	v
6	w
7	x
8	a
9	b
10	c
11	d
12	e
13	f
14	g
15	h

Hierarchical Page Table

Virtual memory

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p
16	q
17	r
18	s
19	t
20	u
21	v
22	w
23	x
24	y
25	z
26	aa

Top Level Page Table

Page1	Frame	v
0	4	1
1	7	1
2	X	0
3	X	0

Page2	Frame	v
0	2	1
1	3	1
2	X	0
3	X	0

Page2	Frame	v
0	X	0
1	1	1
2	0	1
3	X	0

Page2	Frame	v
0	X	0
1	X	0
2	X	0
3	X	0

Used by MMU

Physical memory

0	y
1	z
2	aa
3	bb
4	u
5	v
6	w
7	x
8	a
9	b
10	c
11	d
12	e
13	f
14	g
15	h

Level 2 page table number 0

Hierarchical Page Table

Virtual memory

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p
16	q
17	r
18	s
19	t
20	u
21	v
22	w
23	x
24	y
25	z
26	aa

Top Level Page Table

Page1	Frame	v
0	4	1
1	7	1
2	X	0
3	X	0

Page2	Frame	v
0	2	1
1	3	1
2	X	0
3	X	0

Page2	Frame	v
0	X	0
1	1	1
2	0	1
3	X	0

Page2	Frame	v
0	X	0
1	X	0
2	X	0
3	X	0

Used by MMU

Physical memory

0	y
1	z
2	aa
3	bb
4	u
5	v
6	w
7	x
8	a
9	b
10	c
11	d
12	e
13	f
14	g
15	h

Level 2 page table number 0

Hierarchical Page Table

Virtual memory

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p
16	q
17	r
18	s
19	t
20	u
21	v
22	w
23	x
24	y
25	z
26	aa

Address 6
000110

Top Level
Page Table

Page1	Frame	v
0	4	1
1	7	1
2	X	0
3	X	0

Page2	Frame	v
0	2	1
1	3	1
2	X	0
3	X	0

Page2	Frame	v
0	X	0
1	1	1
2	0	1
3	X	0

Page2	Frame	v
0	X	0
1	X	0
2	X	0
3	X	0

Used by MMU

Physical memory

0	y
1	z
2	aa
3	bb
4	u
5	v
6	w
7	x
8	a
9	b
10	c
11	d
12	e
13	f
14	g
15	h

Level 2
page table
number 0

Hierarchical Page Table

Virtual memory

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p
16	q
17	r
18	s
19	t
20	u
21	v
22	w
23	x
24	y
25	z
26	aa

Address 6

000110 → 000110

Page1 Page2 Offset

Top Level Page Table

Page1	Frame	v
0	4	1
1	7	1
2	X	0
3	X	0

Page2	Frame	v
0	2	1
1	3	1
2	X	0
3	X	0

Page2	Frame	v
0	X	0
1	1	1
2	0	1
3	X	0

Page2	Frame	v
0	X	0
1	X	0
2	X	0
3	X	0

Used by MMU

Physical memory

0	y
1	z
2	aa
3	bb
4	u
5	v
6	w
7	x
8	a
9	b
10	c
11	d
12	e
13	f
14	g
15	h

Level 2 page table number 0

Hierarchical Page Table

Virtual memory

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p
16	q
17	r
18	s
19	t
20	u
21	v
22	w
23	x
24	y
25	z
26	aa

Address 6

000110 → 000110

Page1 Page2 Offset

Top Level Page Table

Page1	Frame	v
0	4	1
1	7	1
2	X	0
3	X	0

Page2	Frame	v
0	2	1
1	3	1
2	X	0
3	X	0

Page2	Frame	v
0	X	0
1	1	1
2	0	1
3	X	0

Page2	Frame	v
0	X	0
1	X	0
2	X	0
3	X	0

Used by MMU

Physical memory

0	y
1	z
2	aa
3	bb
4	u
5	v
6	w
7	x
8	a
9	b
10	c
11	d
12	e
13	f
14	g
15	h

Level 2 page table number 0

Hierarchical Page Table

Virtual memory

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p
16	q
17	r
18	s
19	t
20	u
21	v
22	w
23	x
24	y
25	z
26	aa

Address 6

000110 → 000110

Page1 Page2 Offset

Top Level Page Table

Page1	Frame	v
0	4	1
1	7	1
2	X	0
3	X	0

Page2	Frame	v
0	2	1
1	3	1
2	X	0
3	X	0

Page2	Frame	v
0	X	0
1	1	1
2	0	1
3	X	0

Page2	Frame	v
0	X	0
1	X	0
2	X	0
3	X	0

Used by MMU

Physical memory

0	y
1	z
2	aa
3	bb
4	u
5	v
6	w
7	x
8	a
9	b
10	c
11	d
12	e
13	f
14	g
15	h

Level 2 page table number 0

Hierarchical Page Table

Virtual memory

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p
16	q
17	r
18	s
19	t
20	u
21	v
22	w
23	x
24	y
25	z
26	aa

Address 6

000110 → 000110

Page1 Page2 Offset

Top Level Page Table

Page1	Frame	v
0	4	1
1	7	1
2	X	0
3	X	0

Page2	Frame	v
0	2	1
1	3	1
2	X	0
3	X	0

Page2	Frame	v
0	X	0
1	1	1
2	0	1
3	X	0

Page2	Frame	v
0	X	0
1	X	0
2	X	0
3	X	0

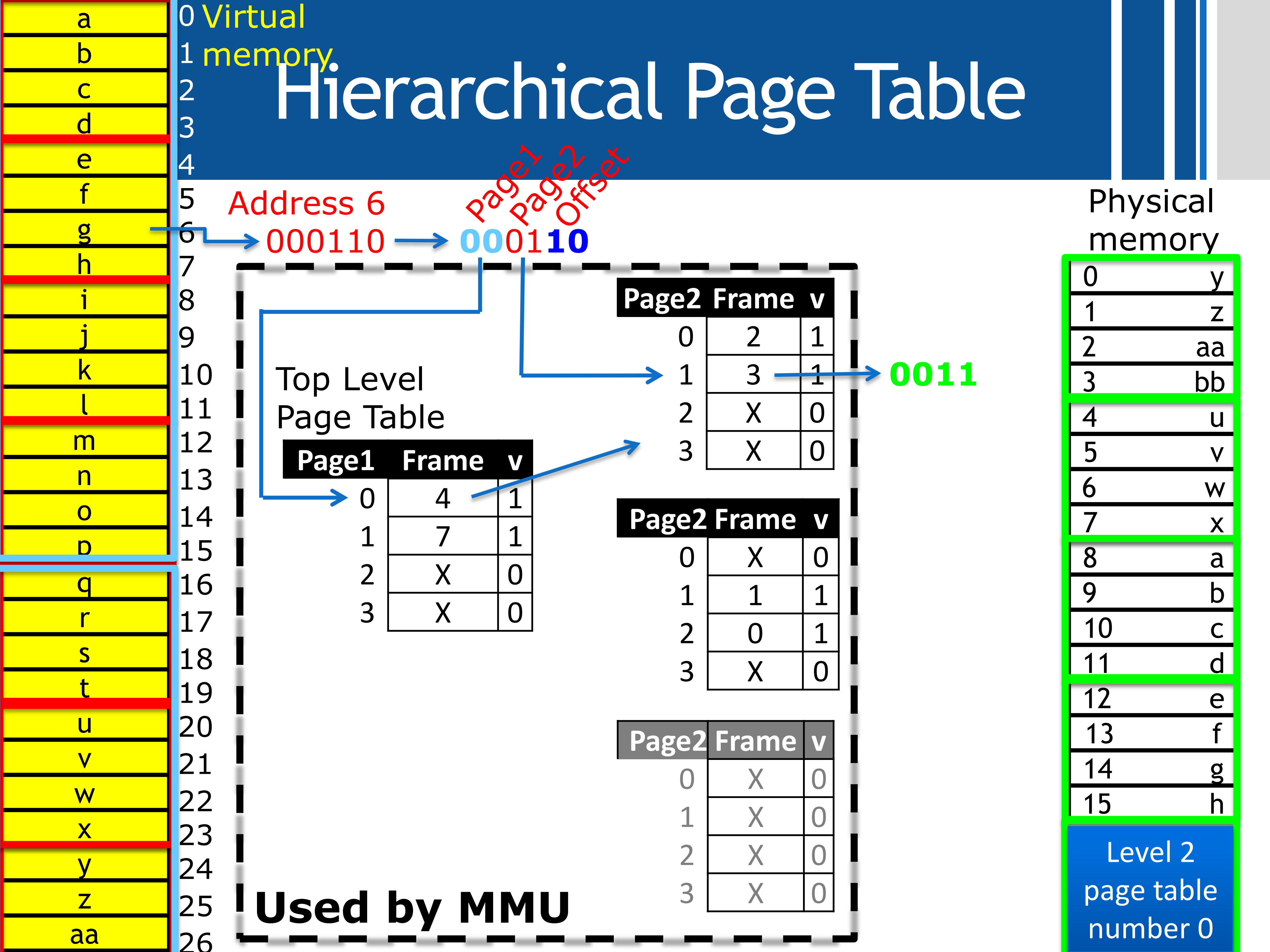
Used by MMU

Physical memory

0	y
1	z
2	aa
3	bb
4	u
5	v
6	w
7	x
8	a
9	b
10	c
11	d
12	e
13	f
14	g
15	h

Level 2 page table number 0

Hierarchical Page Table



Hierarchical Page Table

Virtual memory

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p
16	q
17	r
18	s
19	t
20	u
21	v
22	w
23	x
24	y
25	z
26	aa

Address 6

000110 → 000110 (Page1 Page2 Offset) → 001110

Top Level Page Table

Page1	Frame	v
0	4	1
1	7	1
2	X	0
3	X	0

Page2	Frame	v
0	2	1
1	3	1
2	X	0
3	X	0

Page2	Frame	v
0	X	0
1	1	1
2	0	1
3	X	0

Page2	Frame	v
0	X	0
1	X	0
2	X	0
3	X	0

Used by MMU

Physical memory

0	y
1	z
2	aa
3	bb
4	u
5	v
6	w
7	x
8	a
9	b
10	c
11	d
12	e
13	f
14	g
15	h

Level 2 page table number 0

Hierarchical Page Table

Virtual memory

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p
16	q
17	r
18	s
19	t
20	u
21	v
22	w
23	x
24	y
25	z
26	aa

Address 6

000110 → 000110

Page1 Page2 Offset

Top Level Page Table

Page1	Frame	v
0	4	1
1	7	1
2	X	0
3	X	0

Page2	Frame	v
0	2	1
1	3	1
2	X	0
3	X	0

Page2	Frame	v
0	X	0
1	1	1
2	0	1
3	X	0

Page2	Frame	v
0	X	0
1	X	0
2	X	0
3	X	0

Used by MMU

001110

Physical Address 14

Physical memory

0	y
1	z
2	aa
3	bb
4	u
5	v
6	w
7	x
8	a
9	b
10	c
11	d
12	e
13	f
14	g
15	h

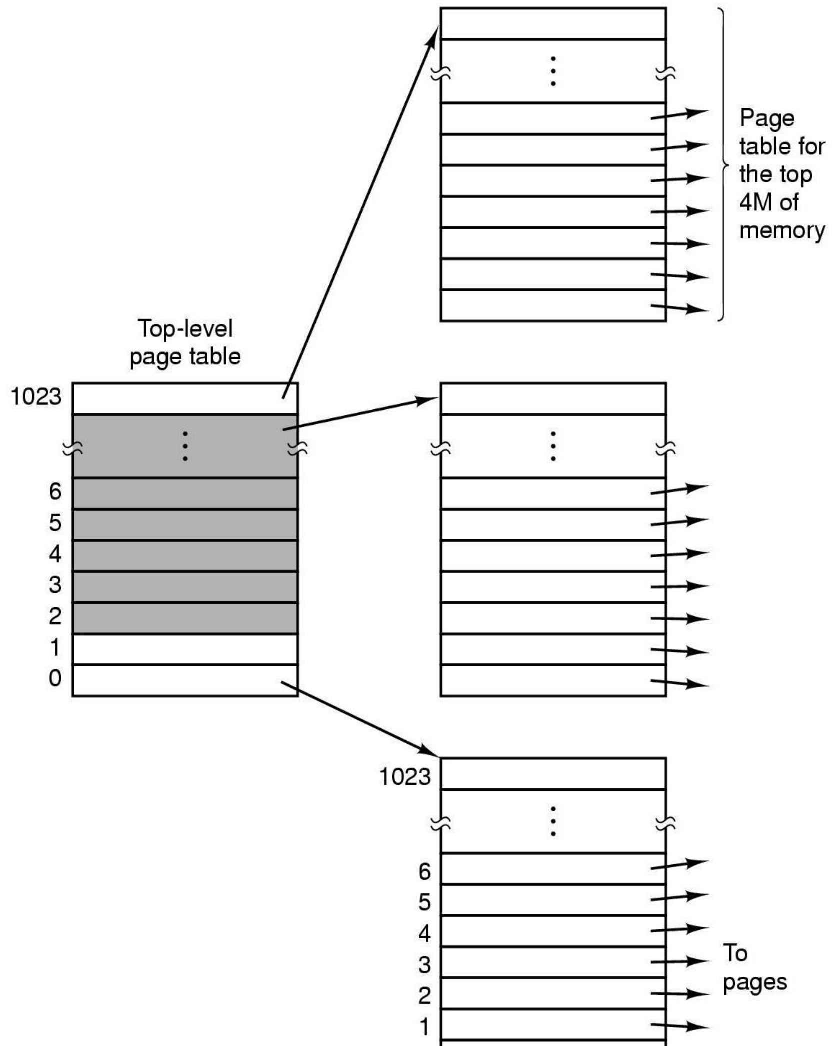
Level 2 page table number 0

Hierarchical Page Table

Instead of holding 1 table with 1M entries, hold 1025 tables with 1K entries

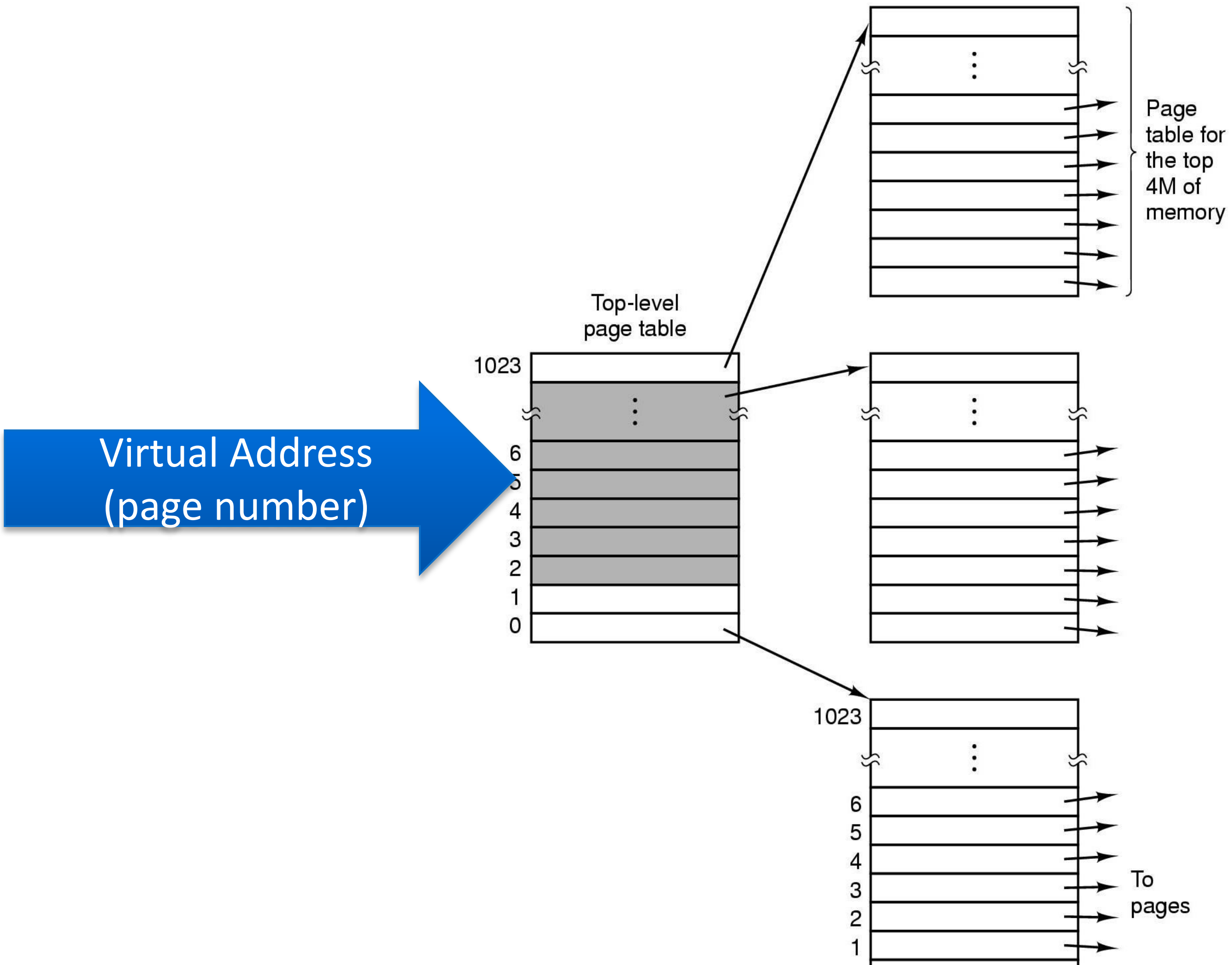
Hierarchical Page Table

Instead of holding 1 table with 1M entries, hold 1025 tables with 1K entries



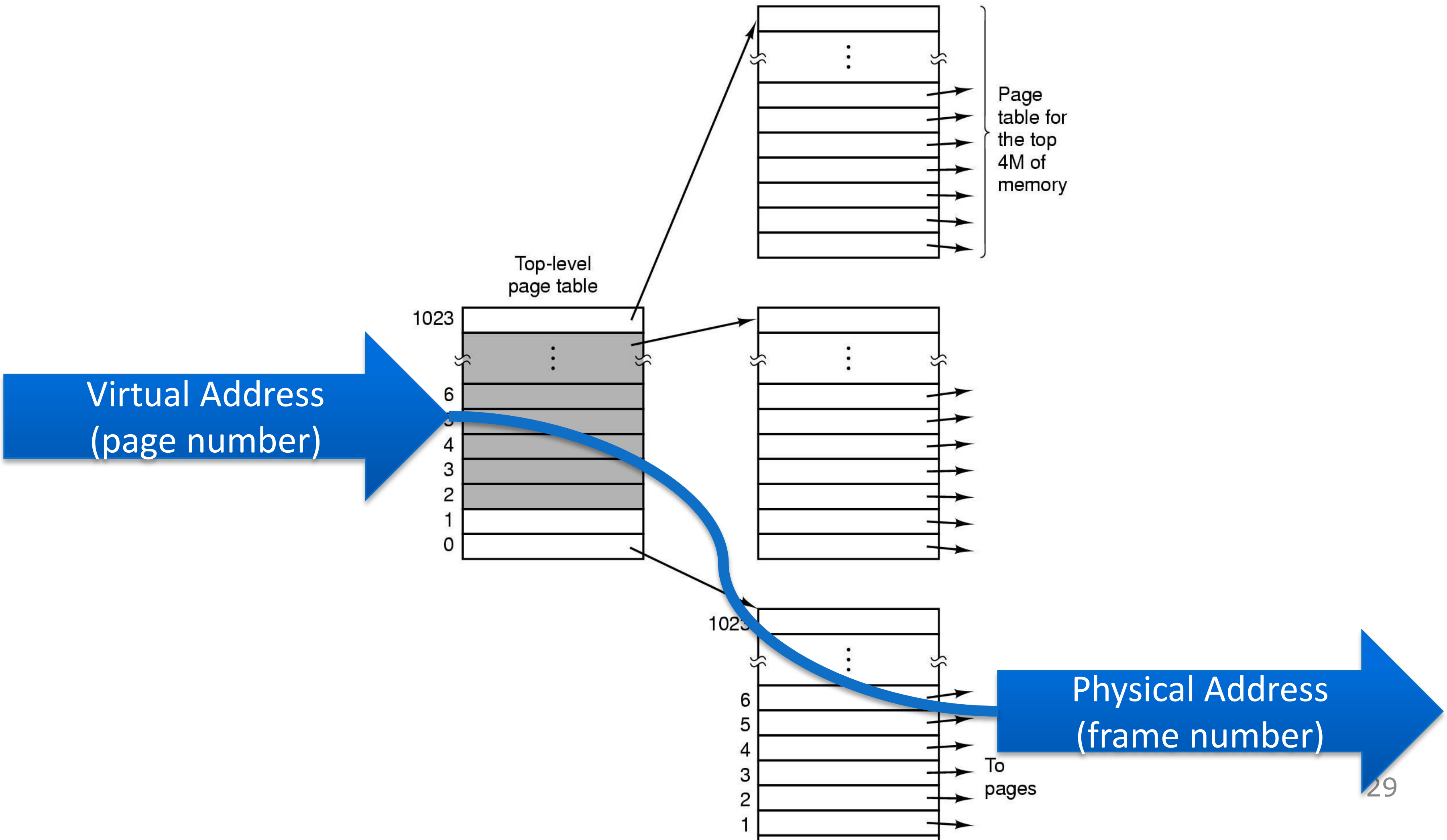
Hierarchical Page Table

Instead of holding 1 table with 1M entries, hold 1025 tables with 1K entries



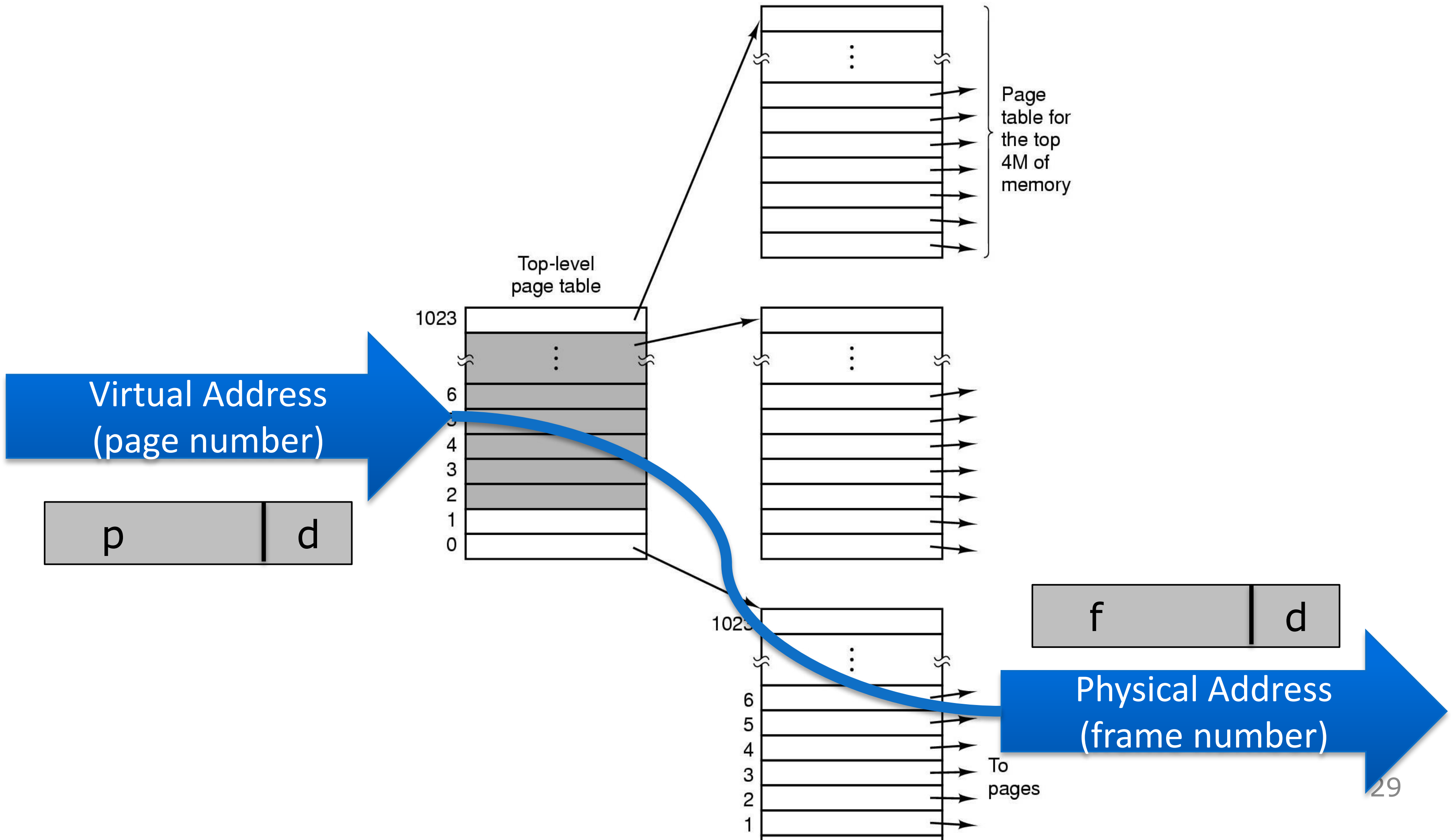
Hierarchical Page Table

Instead of holding 1 table with 1M entries, hold 1025 tables with 1K entries



Hierarchical Page Table

Instead of holding 1 table with 1M entries, hold 1025 tables with 1K entries

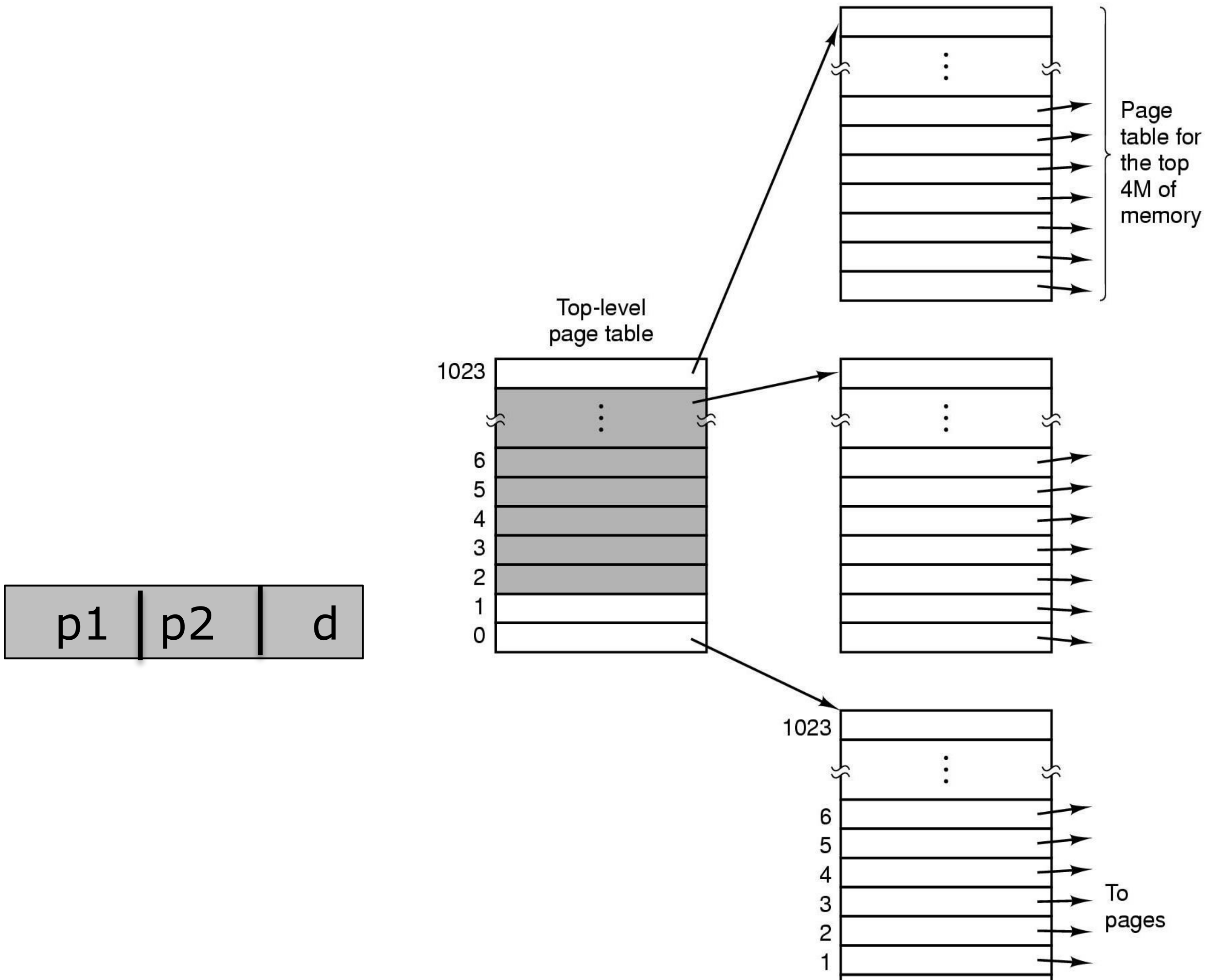


Hierarchical Page Table

- Each table is of size 1 page
- Virtual address is split to three parts:
  - P1: The entry in the top-level table
  - P2: The entry in the second-level table
  - d: The offset within the frame (as before)

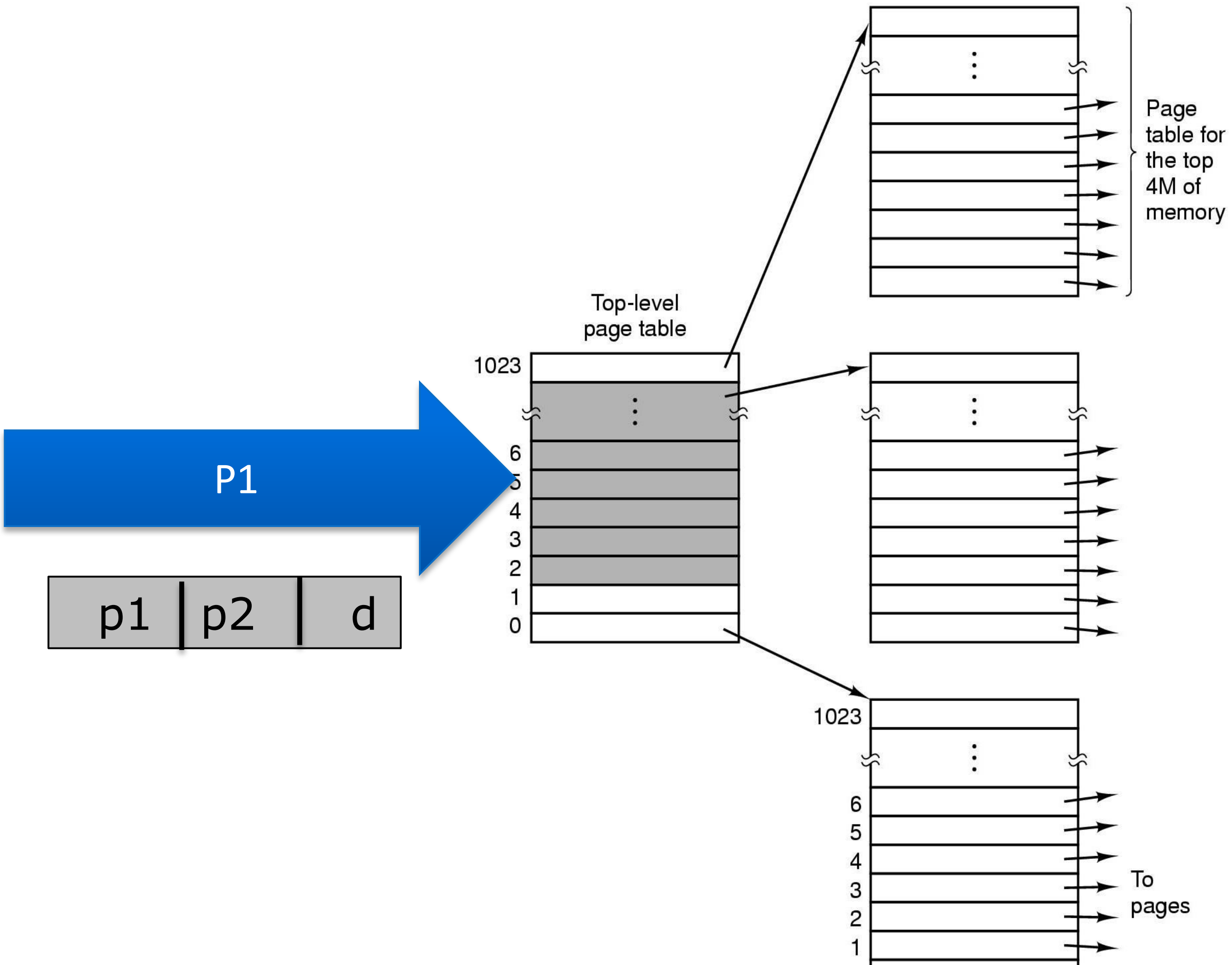
Hierarchical Page Table

Instead of holding 1 table with 1M entries, hold 1025 tables with 1K entries



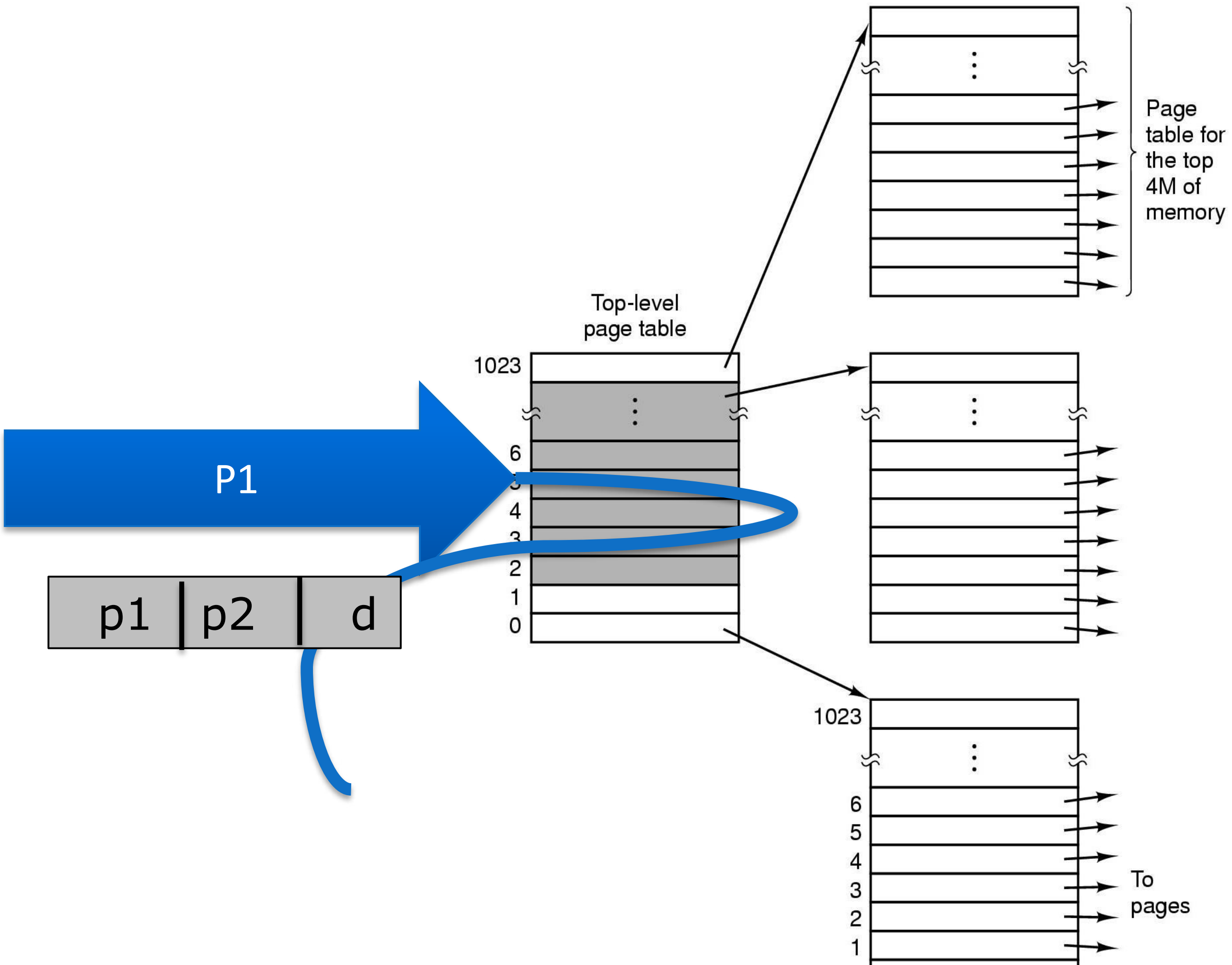
Hierarchical Page Table

Instead of holding 1 table with 1M entries, hold 1025 tables with 1K entries



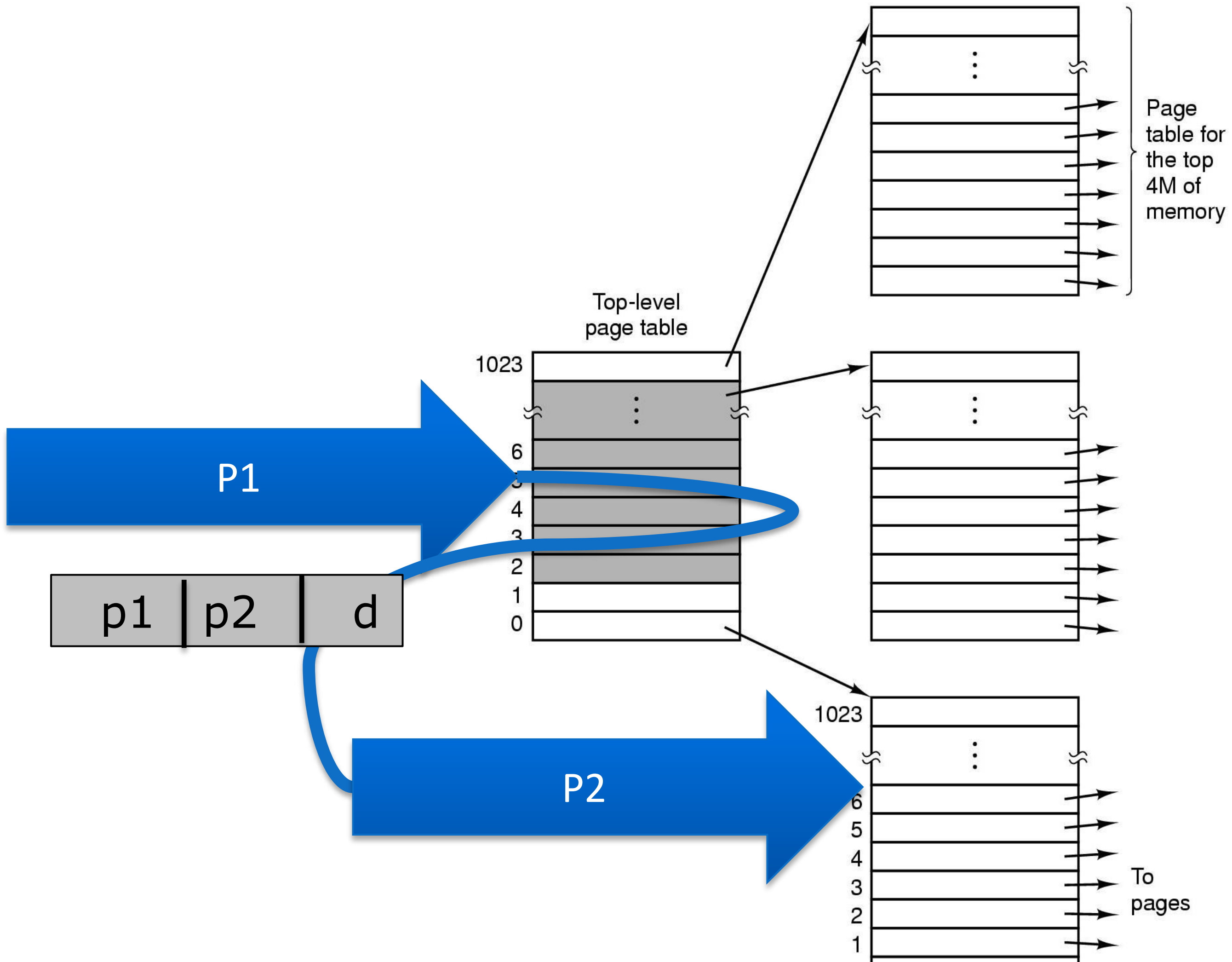
Hierarchical Page Table

Instead of holding 1 table with 1M entries, hold 1025 tables with 1K entries



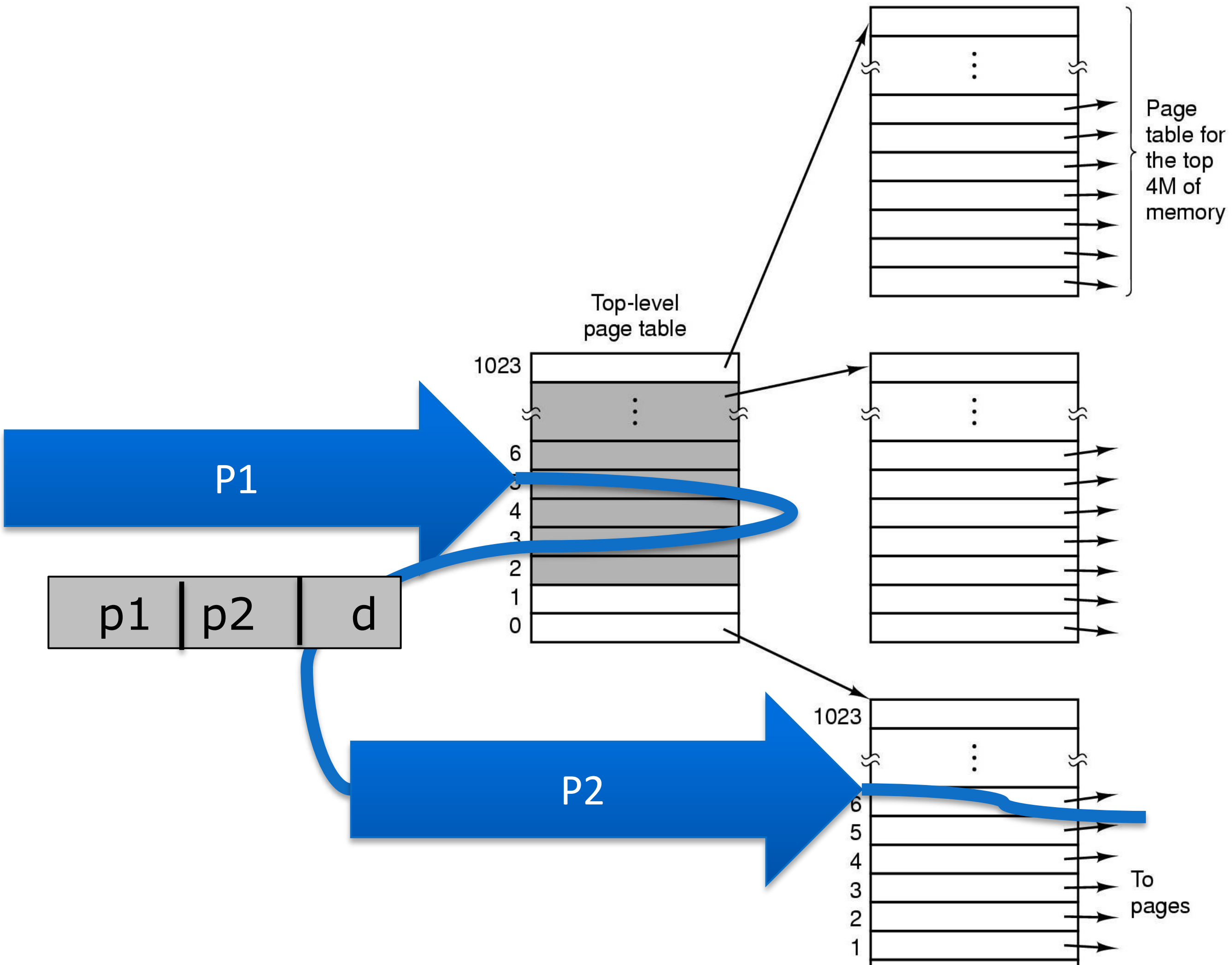
Hierarchical Page Table

Instead of holding 1 table with 1M entries, hold 1025 tables with 1K entries



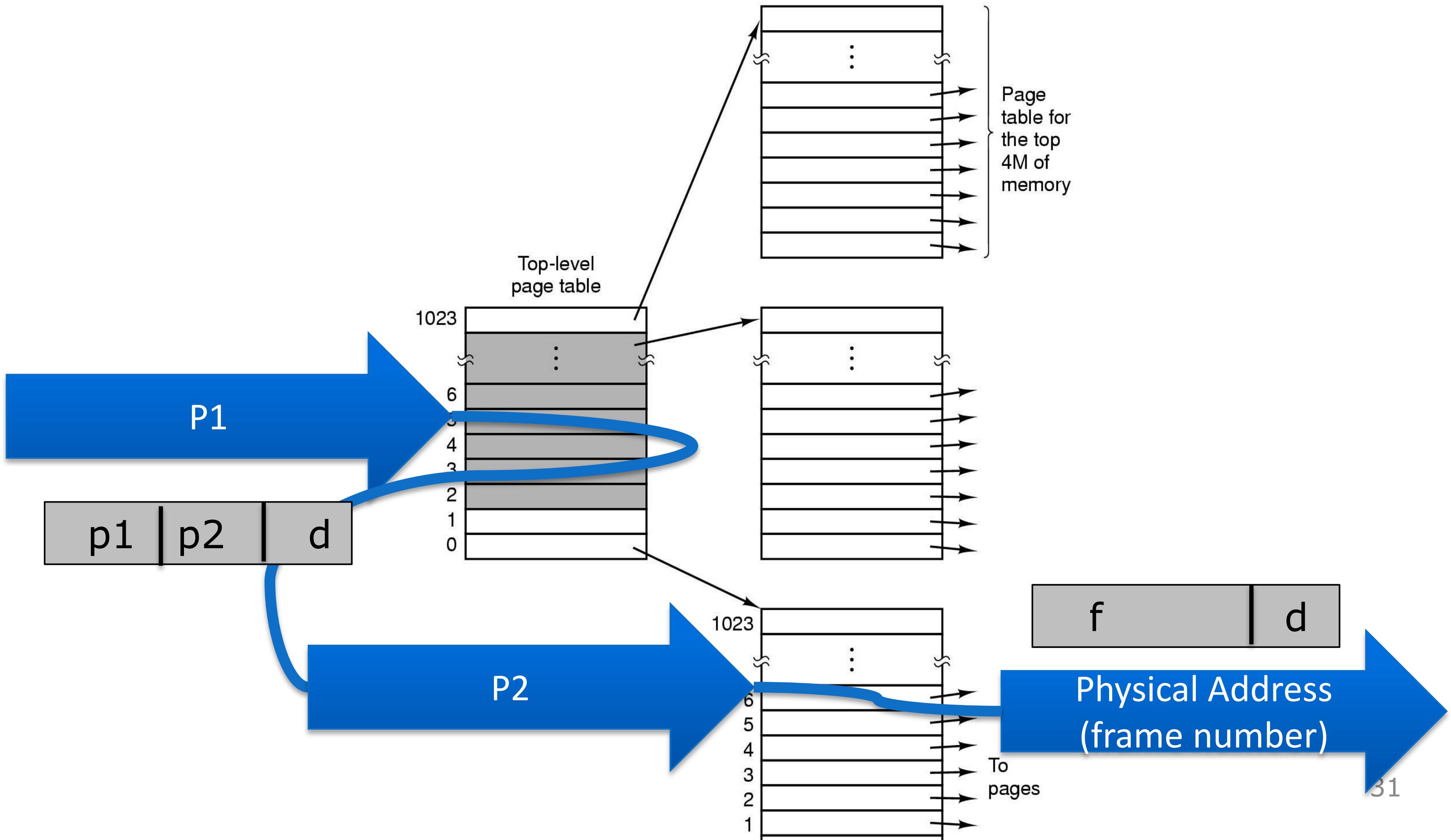
Hierarchical Page Table

Instead of holding 1 table with 1M entries, hold 1025 tables with 1K entries



Hierarchical Page Table

Instead of holding 1 table with 1M entries, hold 1025 tables with 1K entries

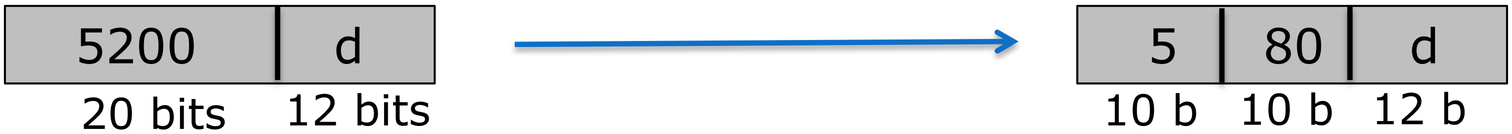


Hierarchical Page Table: Example

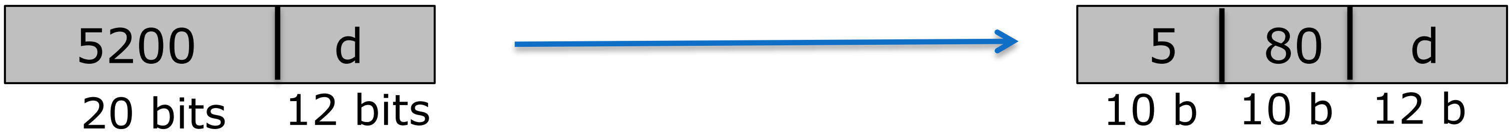
Hierarchical Page Table: Example

5200	d
20 bits	12 bits

Hierarchical Page Table: Example



Hierarchical Page Table: Example



Suppose

- Page 5200 is held in frame 1000
- Top-Level Page table is held in frame 17
- Second level table 5 is held in frame 700

Hierarchical Page Table: Example

5200	d
20 bits	12 bits

→

5	80	d
10 b	10 b	12 b

Suppose

- Page 5200 is held in frame 1000
- Top-Level Page table is held in frame 17
- Second level table 5 is held in frame 700

1. We go to frame 17, read 5th entry → Value is 700

Hierarchical Page Table: Example

5200	d
20 bits	12 bits

→

5	80	d
10 b	10 b	12 b

Suppose

- Page 5200 is held in frame 1000
- Top-Level Page table is held in frame 17
- Second level table 5 is held in frame 700

1. We go to frame 17, read 5^{th} entry → Value is 700
2. We go to frame 700, read 80^{th} entry → Value is 1000

Hierarchical Page Table: Example

5200	d
20 bits	12 bits

→

5	80	d
10 b	10 b	12 b

Suppose

- Page 5200 is held in frame 1000
- Top-Level Page table is held in frame 17
- Second level table 5 is held in frame 700

1. We go to frame 17, read 5th entry → Value is 700
2. We go to frame 700, read 80th entry → Value is 1000
3. We read address

1000	d
20 bits	12 bits

Memory Accesses and Page Faults

Memory Accesses and Page Faults

- Every memory access translates into three memory accesses

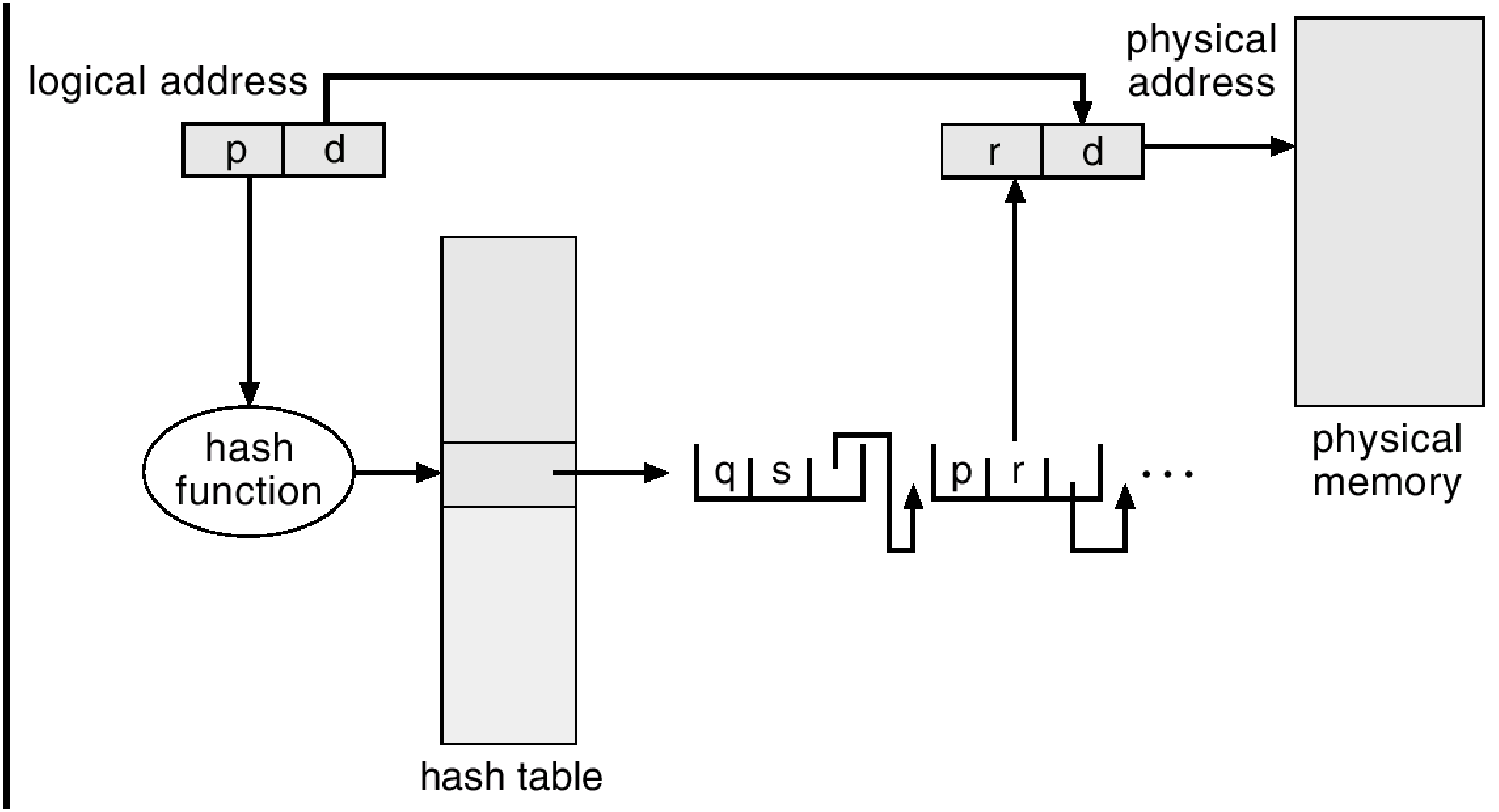
Memory Accesses and Page Faults

- Every memory access translates into three memory accesses
- We can have two page faults:
  - Second-level page table is not in memory
  - (as before:) Content page is not in memory

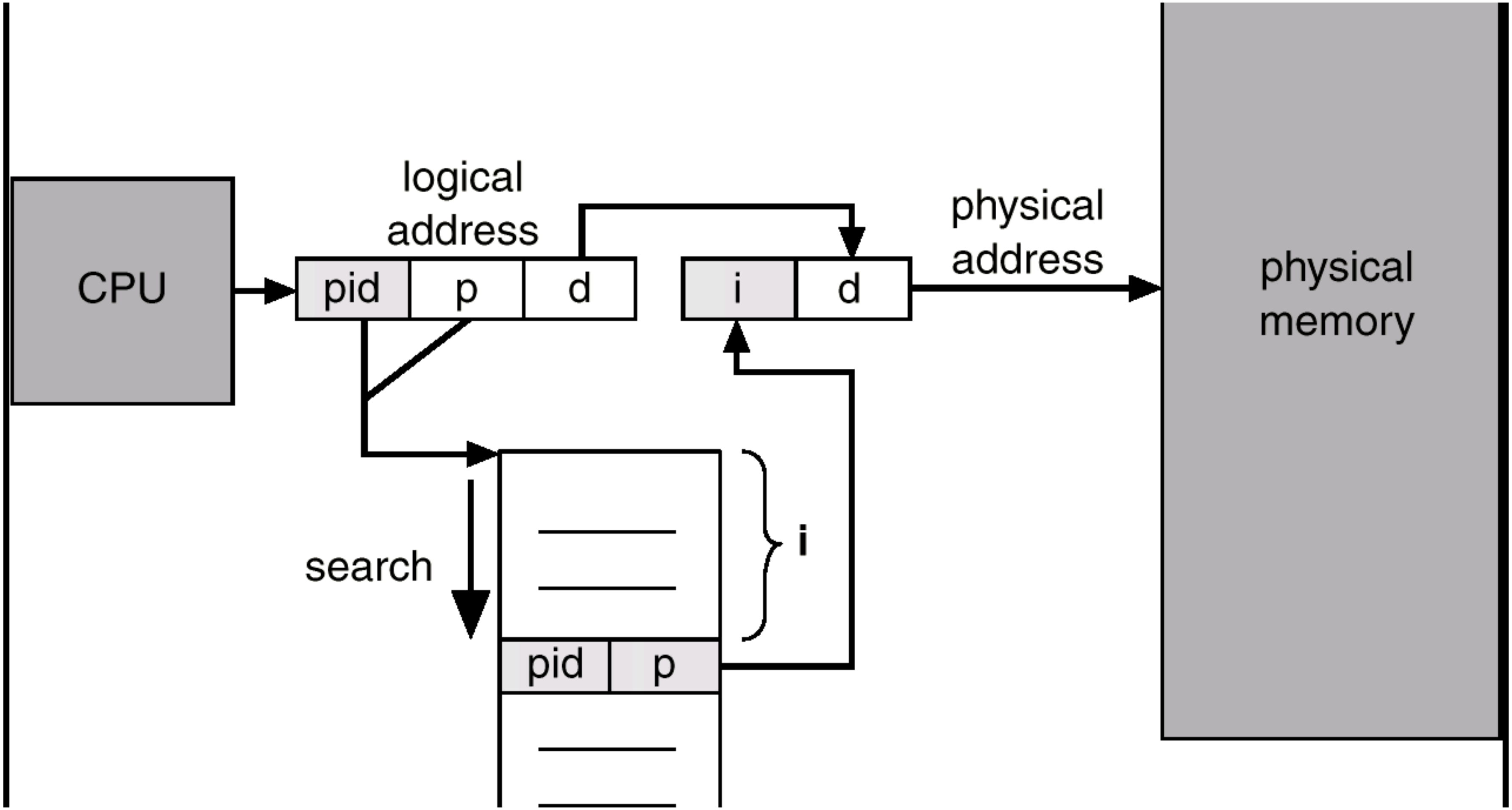
Memory Accesses and Page Faults

- Every memory access translates into three memory accesses
- We can have two page faults:
  - Second-level page table is not in memory
  - (as before:) Content page is not in memory
- Most of the second-level page tables are all invalid
  - No need to store them (anywhere, not even the disk)
  - When page fault to such a table occur, the OS can create all zero frame for that page table; later to be filled with the content we swap into memory

Hashed Page Table



Inverted Page Table



Memory Management - Summary

Problem	Solution
Process view is different than actual view	Address Translation: base+bound, **page tables**
External fragmentation	Compaction, **paging**
Logical memory is larger than physical memory	**Virtual memory + swapping**