Page table implementation

Module 5a self study material

Operating systems 2019

1DT044, 1DT096 and 1DT003

karl.marklund@it.uu.se

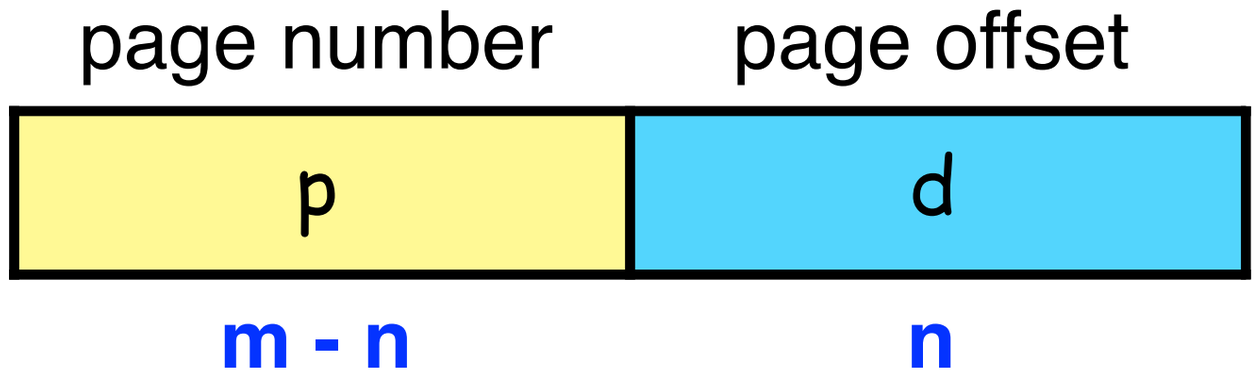
Paged memory management

- **Logical address space** of a process can be **noncontiguous**; process is **allocated physical memory whenever** the latter is **available**.
- Divide **physical memory** into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8,192 bytes).
- Divide **logical memory** into blocks of same size called **pages**.
- Keep track of all **free frames**.
  - To run a program of size n pages, need to find n free frames and load program.
- Set up a **page table** to translate logical to physical addresses.
- **Internal fragmentation** possible.
- No **external fragmentation**.

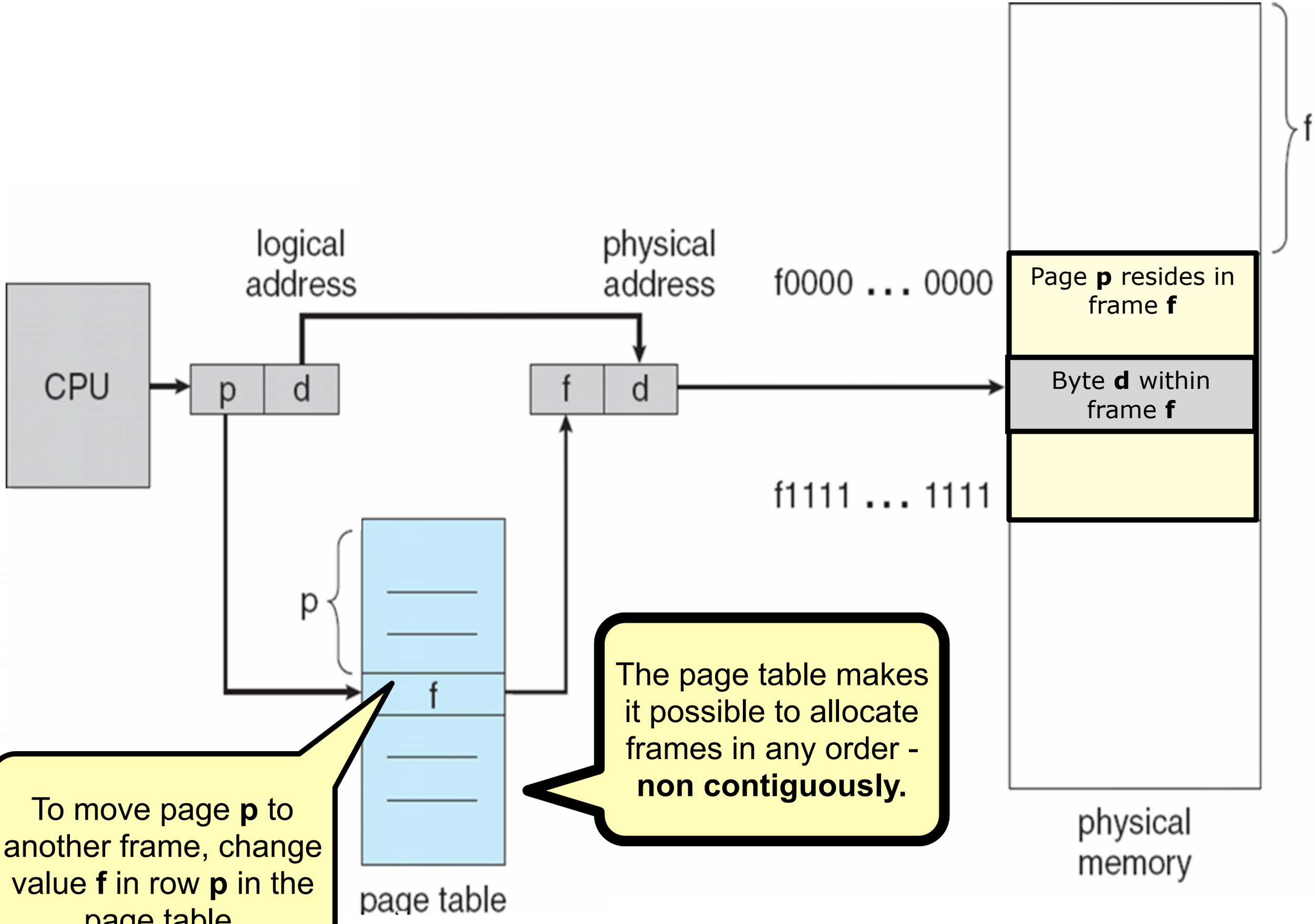
Address translation scheme

Logical address generated by CPU is divided into:

- **Page number** (p) – used as an index into a page table which contains base address of each frame in physical memory
- **Page offset** (d) – combined with base address to define the physical memory address that is sent to the memory unit.



For given logical address space 2^m and page size 2^n


logical address
physical address
CPU
p
d
f
d
f0000 ... 0000
f1111 ... 1111
Page **p** resides in frame **f**
Byte **d** within frame **f**
f
p
f
page table
physical memory
To move page **p** to another frame, change value **f** in row **p** in the page table.
The page table makes it possible to allocate frames in any order - **non contiguously.**

- Page table is kept **in main memory**.
- Page-table **base register** (PTBR) points to the page table.
- Page-table **length register** (PRLR) indicates size of the page table.
- In this scheme every data/instruction access requires two memory accesses.
  - One for the page table.
  - One for the data/instruction.
- Can we do better than this?

Binary prefixes

A binary prefix is a unit prefix for multiples of units in data processing, data transmission, and digital information, notably the bit and the byte, to indicate multiplication by a power of 2.

Binary Prefixes

SI Prefixes are not used in the computer world, things are a bit different:

Commonly used

These prefixes are easily confused with SI Prefixes.

Symbol	Prefix	Factor	Factor
P	peta	2^{50}	1125899906842624
T	tera	2^{40}	1099511627776
G	giga	2^{30}	1073741824
M	Mega	2^{20}	1048576
k	kilo	2^{10}	1024

Symbol	Prefix	Factor
Y	Yotta	10^{24}
Z	Zetta	10^{21}
E	Exa	10^{18}
P	peta	10^{15}
T	tera	10^{12}
G	giga	10^{9}
M	Mega	10^{6}
k	kilo	10^{3}
h	hecto	10^{2}
da	deca	10^{1}
d	deci	10^{-1}
c	centi	10^{-2}
m	milli	10^{-3}
µ	micro	10^{-6}
n	nano	10^{-9}
p	pico	10^{-12}
f	femto	10^{-15}
a	atto	10^{-18}
z	zepto	10^{-21}
y	yokto	10^{-24}

Large address space

Most modern computer systems support a large logical address space, usually with 2^{32} - 2^{64} bytes of addressable memory.

Example

A system uses 32 bit logical memory addresses

Suppose the page size is 4 KB

$$4\ KB = 2^2 x 2^{10} = 2^{12}\ Bytes$$

Number of pages?

$$N = 2^{32} / 2^{12} = 2^{20} = 1048576 \approx 1\ Million\ pages$$

Assume each entry in the page table takes 4 bytes

Size of page table = $4x2^{20}$ = 4 MB.

Observation?

This is quite a lot of memory ... needed by every process!

Symbol	Prefix	Factor	Factor
P	peta	2^{50}	1125899906842624
T	tera	2^{40}	1099511627776
G	giga	2^{30}	1073741824
M	Mega	2^{20}	1048576
k	kilo	2^{10}	1024

If we try to allocate the page table contiguously in memory we will encounter the same problem with fragmentation that we try to solve using page tables in the first place.

Data structures for page tables

To overcome the problem with allocating a large contagious pice of memory for the page table various alternative data structures can be used.

Structure of the page table

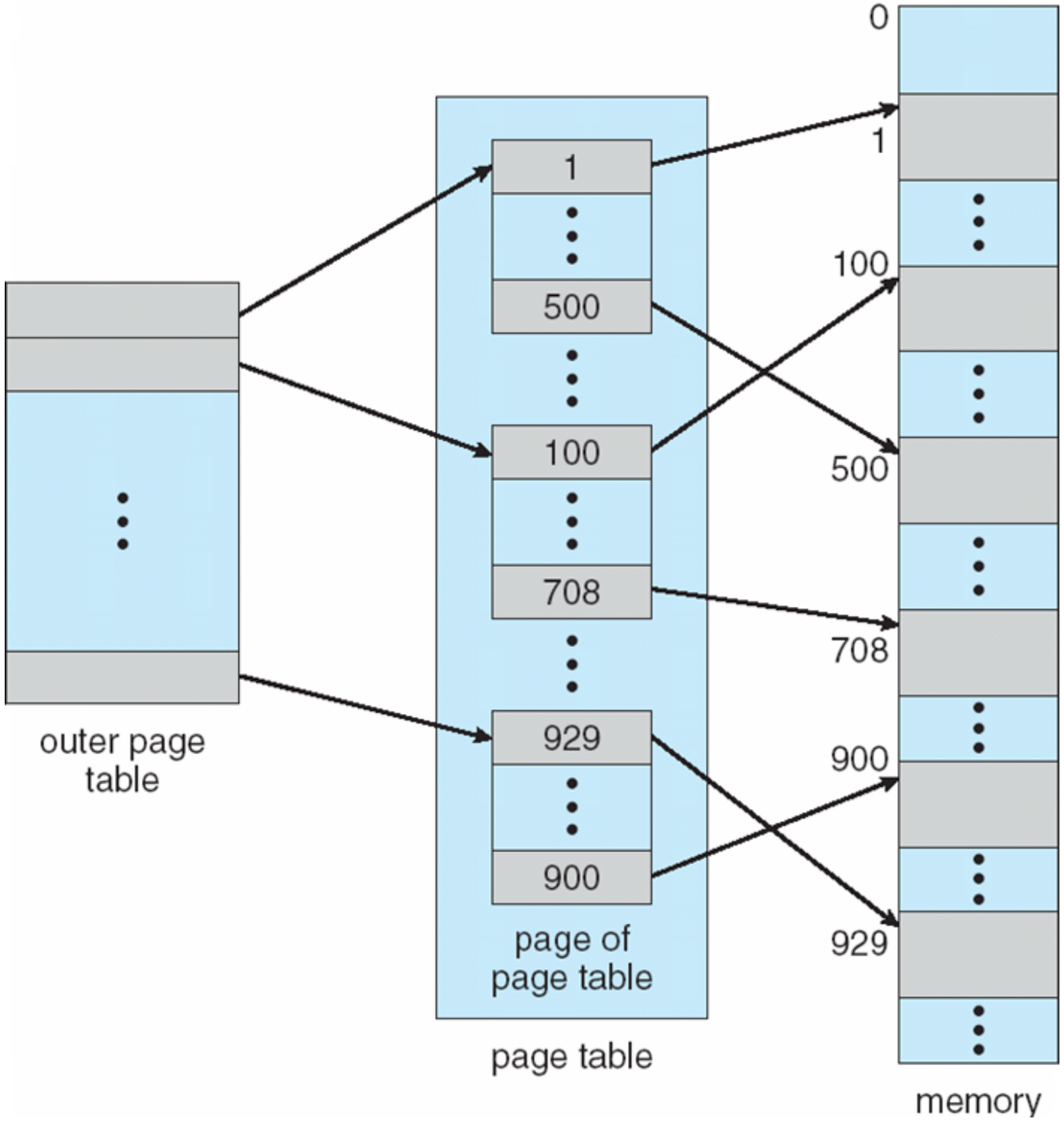
The page table can be structured in various ways.

- Hierarchical paging
- Hashed page tables
- Inverted page tables

Hierarchical page tables

Hierarchical page tables

Break up the logical address space into multiple page tables. A simple technique is a two-level page table.



Two-level paging example

A logical address on a 32-bit machine with 1KB page size is divided into:

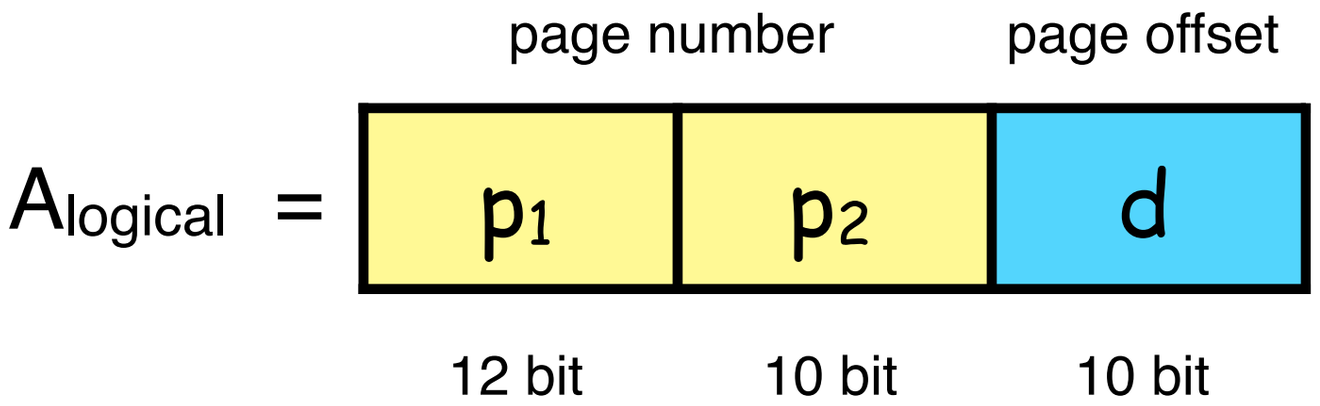
- a page number consisting of 22 bits
- a page offset consisting of 10 bits

Page size = 1KB = 2^{10} Bytes

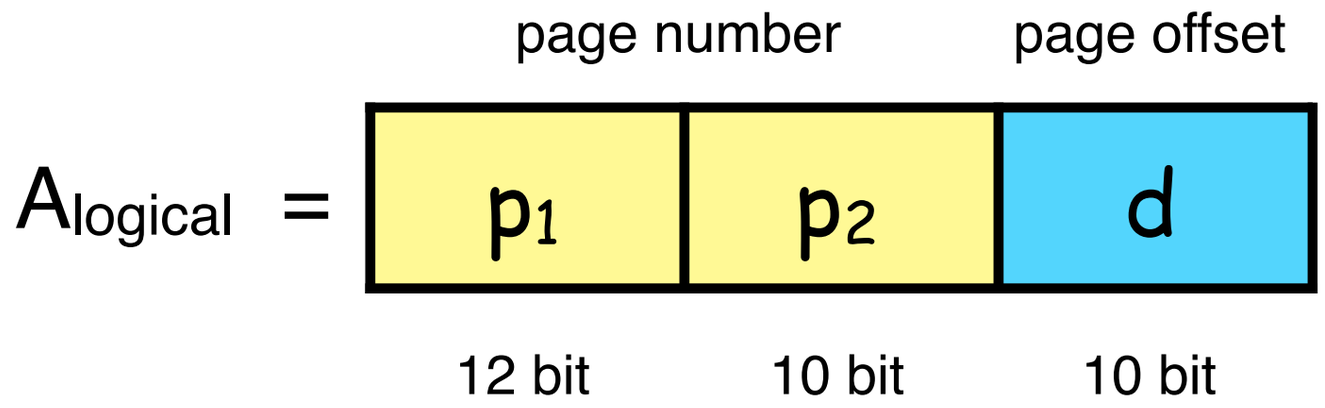
$N_{pages} = 2^{32}/2^{10} = 2^{22}$

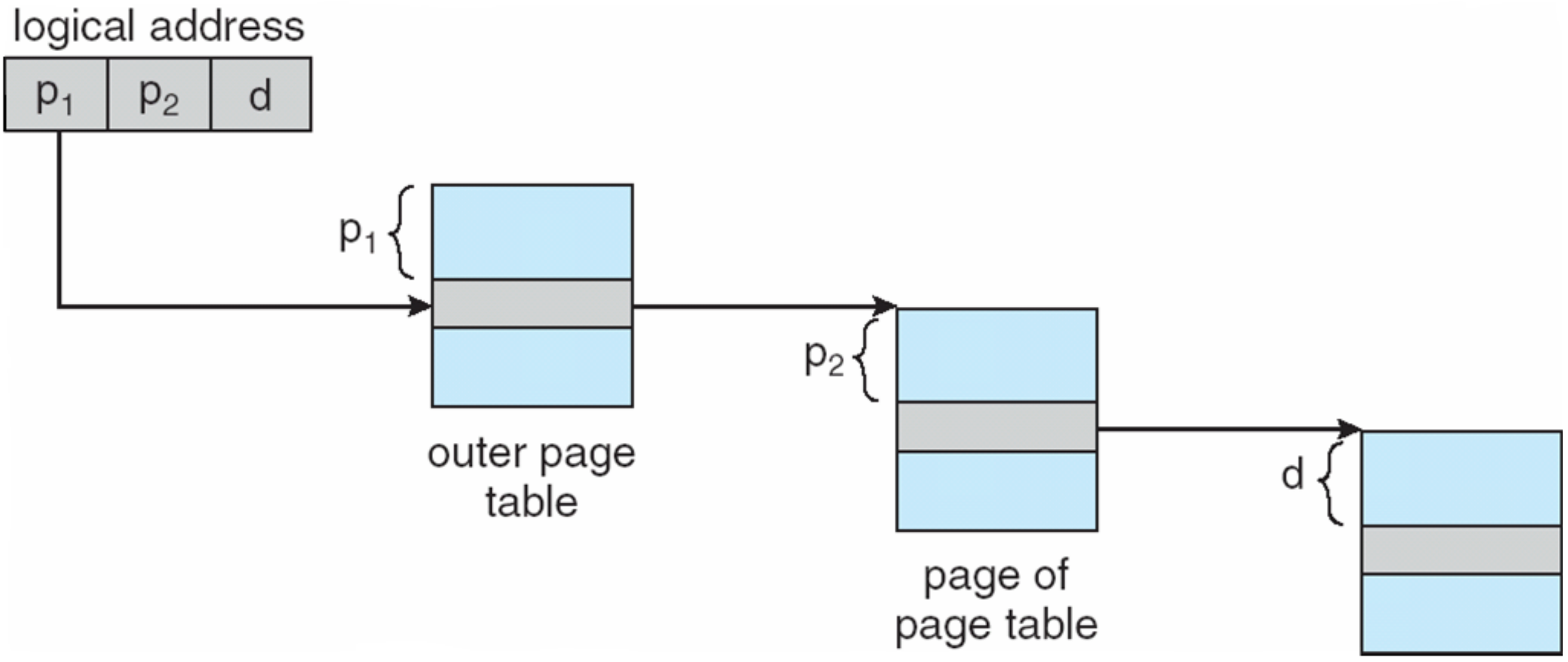
Since the page table is paged, the 22 bit page number is further divided into:

- a 12-bit page number
- a 10-bit page offset



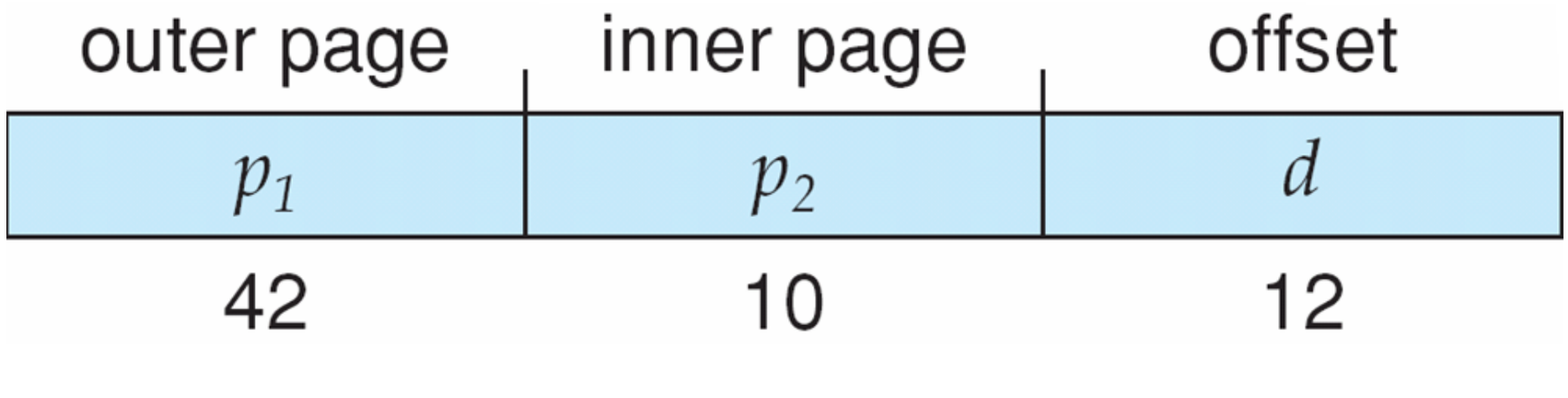
Where $\mathbf{p_1}$ is an index into the outer page table, and $\mathbf{p_2}$ is the displacement within the page of the outer page table


page number
page offset
$A_{logical}$ =
p_1
p_2
d
12 bit
10 bit
10 bit

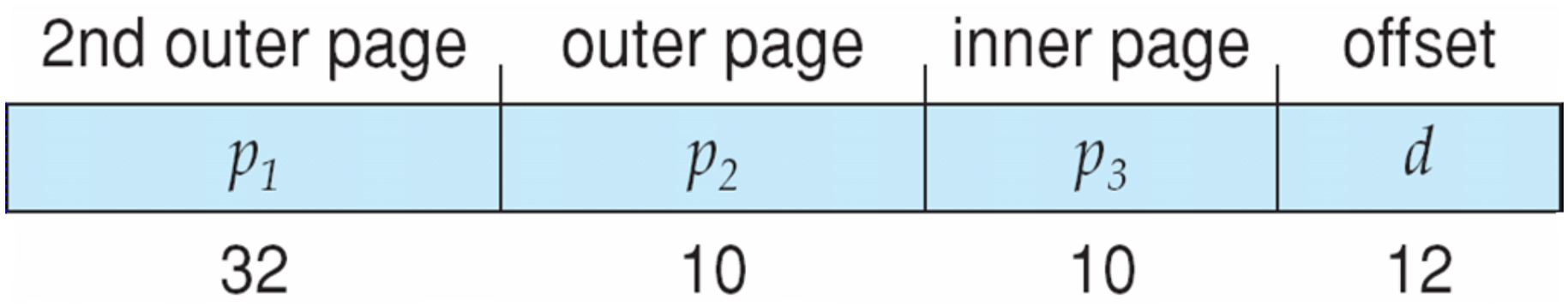

logical address
p_1
p_2
d
p_1
outer page table
p_2
page of page table
d

64 bit address space examples

Two Level Paging



Three Level Paging

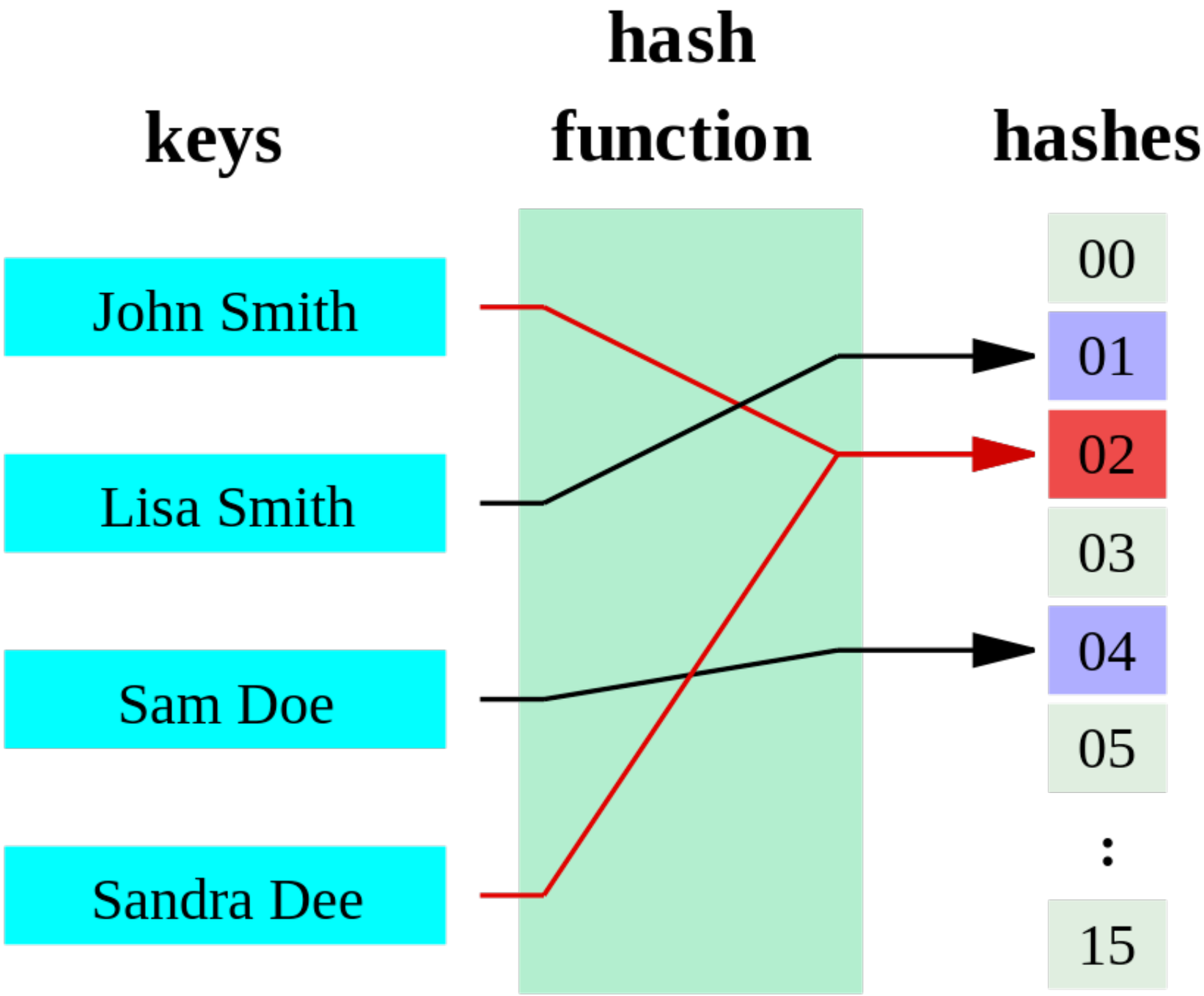


Hash function

A hash function is any function that can be used to map data of arbitrary size to data of fixed size.

The values returned by a hash function are called hash values.

Typically, the domain of a hash function is larger than its range, and so it will map several different keys to the same index which could result in collisions.



A hash function that maps names to integers from 0 to 15. There is a **collision** between keys "John Smith" and "Sandra Dee".

Figure https://en.wikipedia.org/wiki/Hash_function 2018-02-22

Hashed page tables

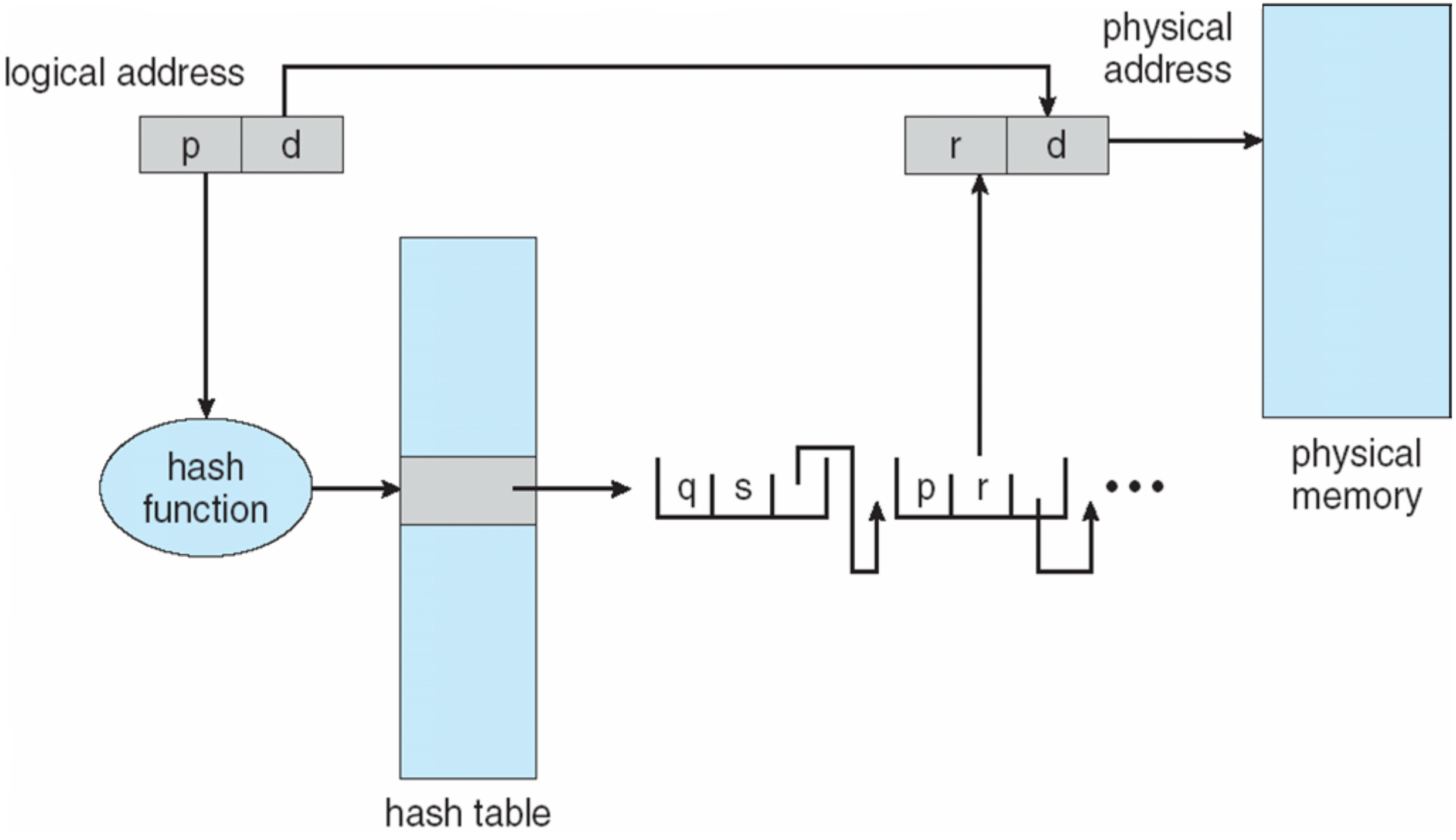
Common in address spaces > 32 bits.

The virtual page number is hashed into a page table.

- This page table contains a chain of elements hashing to the same location.

Virtual page numbers are compared in this chain searching for a match.

If a match is found, the corresponding physical frame is extracted.


logical address
p
d
physical address
r
d
hash function
hash table
q
s
p
r
physical memory

Observations

- ★ Usually, each process has an associated page table.
- ★ The page table has one entry for each page that the process is using.
- ★ The page table is sorted by logical address (seen by the CPU).
- ★ Translation from logical to physical is simple, just find the entry in the page table ...
- ★ Major drawback, page tables can consume lot of memory themselves.

Alternative

- ★ Instead of one page table per process, use one global table.
- ★ One entry for each real frame (ordinary page tables has one entry for each logical page).
- ★ Each entry holds the logical address of a frame.

Inverted page table

Instead of one page table per process, use one global table.

One entry for each real frame.

Each entry holds the logical address of a frame.

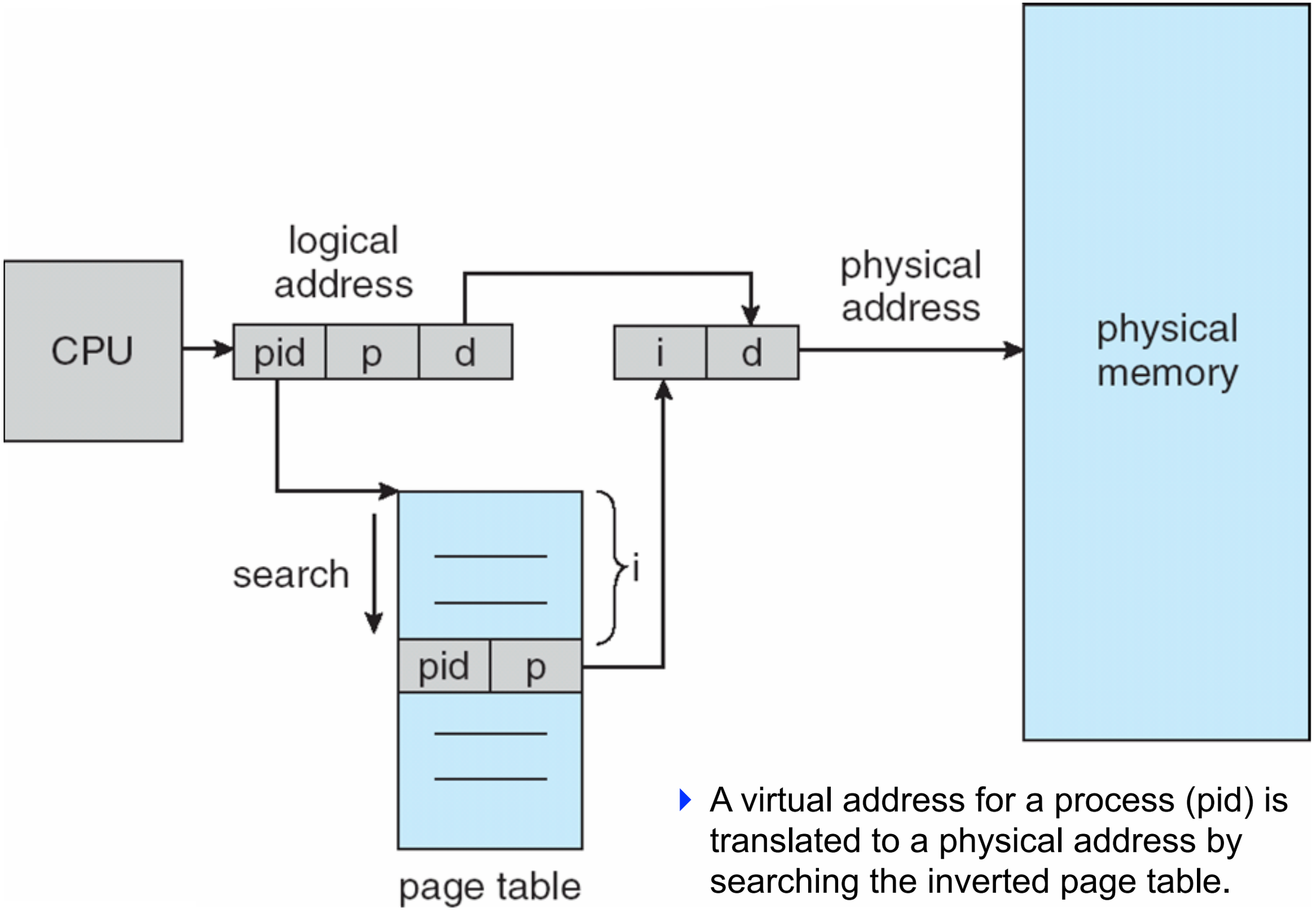
Inverted page table (1)

A single page table for all processes!

One entry for each frame of memory.

Entry consists of a page number (p) and information (pid) about the process that owns that page.

- **Decreases memory needed** to store each page table.
- **Increases time needed** to search the page table when a page reference occurs.



- A virtual address for a process (pid) is translated to a physical address by searching the inverted page table.
- When the entry for the process (pid) and page (p) is found the index (i) in the page table is equal to the physical frame number.

Inverted page table (2)

Use **hash table** to limit the search to one or at most a few page-table entries.