4.3BSD Virtual Memory Management

Speaker: Felix Stahlberg
In context of the proseminar OS Internals of IBDS – System Architecture Group

```
#ifndef BSD4_3
#define BSD4_3 1
#endif /* BSD4_3 */

#ifndef BSD_SYSTEM
#define BSD_SYSTEM 43
#endif /* BSD_SYSTEM */

/* SYSTEM_TYPE should indicate the kind of system you are using.
 It sets the Lisp variable system-type.  */


#define SYSTEM_TYPE "berkeley-unix"

/* nomultiplejobs should be defined if your system's shell
 does not have "job control" (the ability to stop a program,
```

Overview

- History and Goals
- VAX Memory Management Hardware
- Page Tables and PTEs
- Page Replacement

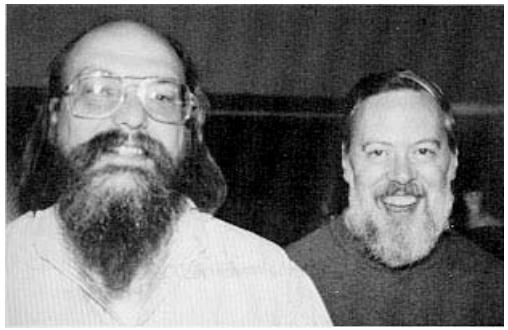
The 4.3BSD Operating System

Evolution of 4.3BSD Memory Management

Memory Management Design Decisions

HISTORY AND GOALS

The 4.3BSD Operating System (1)



- 1969: First version of UNIX
  - Ken Thompson, Dennis Ritchie
- 1979: 3BSD
  - Unix 32V + e.g. demand paging and page replacement
  - Porting to VAX (Virtual Address eXtension)
    - CISC instruction set architecture
    - Early adopter of virtual memory
- 1986: 4.3BSD

The 4.3BSD Operating System (2)

The single Greatest Piece of Software Ever, with the broadest impact on the world, was BSD 4.3. Other Unixes were bigger commercial successes. But as the cumulative accomplishment of the BSD systems, 4.3 represented an unmatched peak of innovation. BSD 4.3 represents the single biggest theoretical undergirder of the Internet. [InformationWeek, 2006]

Evolution of 4.3BSD Memory Management

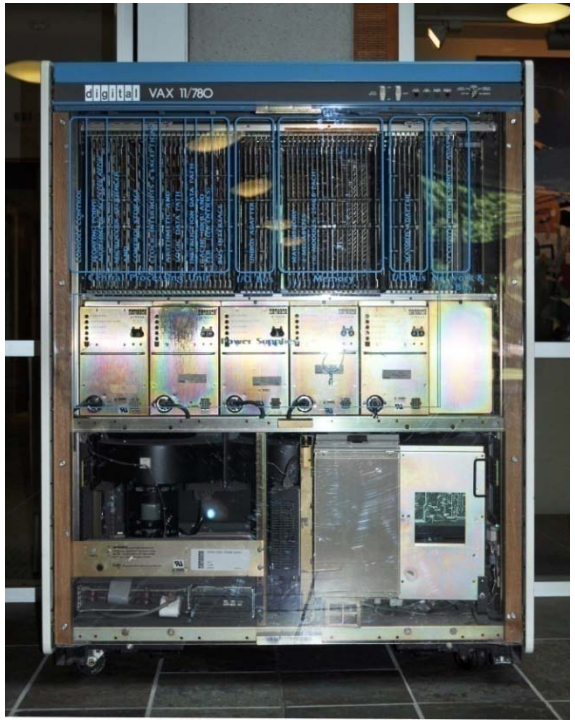
- Version 7 UNIX
  - Segmented memory architecture (8 segments at 8 Kbyte)
  - Processes were fully resident and continguos in memory
- UNIX 32V
  - Tuneable segment size
- 3BSD
  - Demand paging, page-by-page placement policy
- 4.1BSD
  - Page clusters, prepaging

Memory Management Design Decisions (1)

- Mem. Mgmt. of User-Level-Processes [Joy et al., 1986]
  - Three logical segments: *text*, *data*, *stack*
    - *Text* area is read-only and shared, *data* and *stack* areas private to process
  - Highly-developed interface
    - Mapping pages via *mmap()*
    - Page Protection Control via *mprotect()*
    - Synchronization Primitives for Shared Memory
    - No Implementation in 4.3BSD except for *getpagesize()* and *sbrk()*

Memory Management Design Decisions (2)

- Mem. Mgmt. Inside the Kernel
  - 10 different memory allocators
  - 4.3BSD Tahoe: General Purpose Memory Allocator
    [McKusick & Karels, 1988]
    - Small allocations: power-of-two list strategy
    - Larger as 2KByte: paging, first-fit strategy
    - Interface similar to C's *malloc()* and *free()*



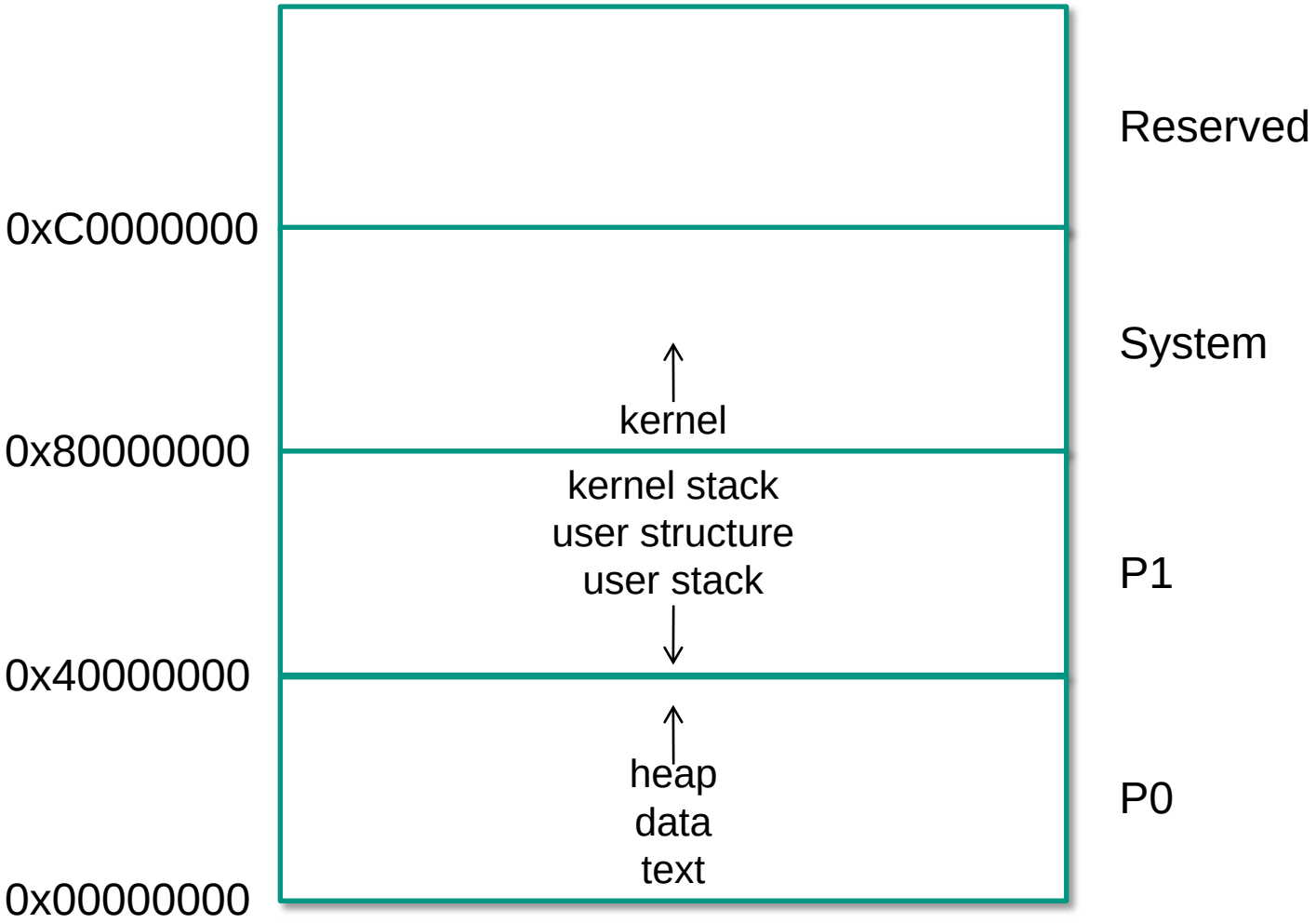
VAX Virtual Address Space

Address Translation

VAX Page Tables

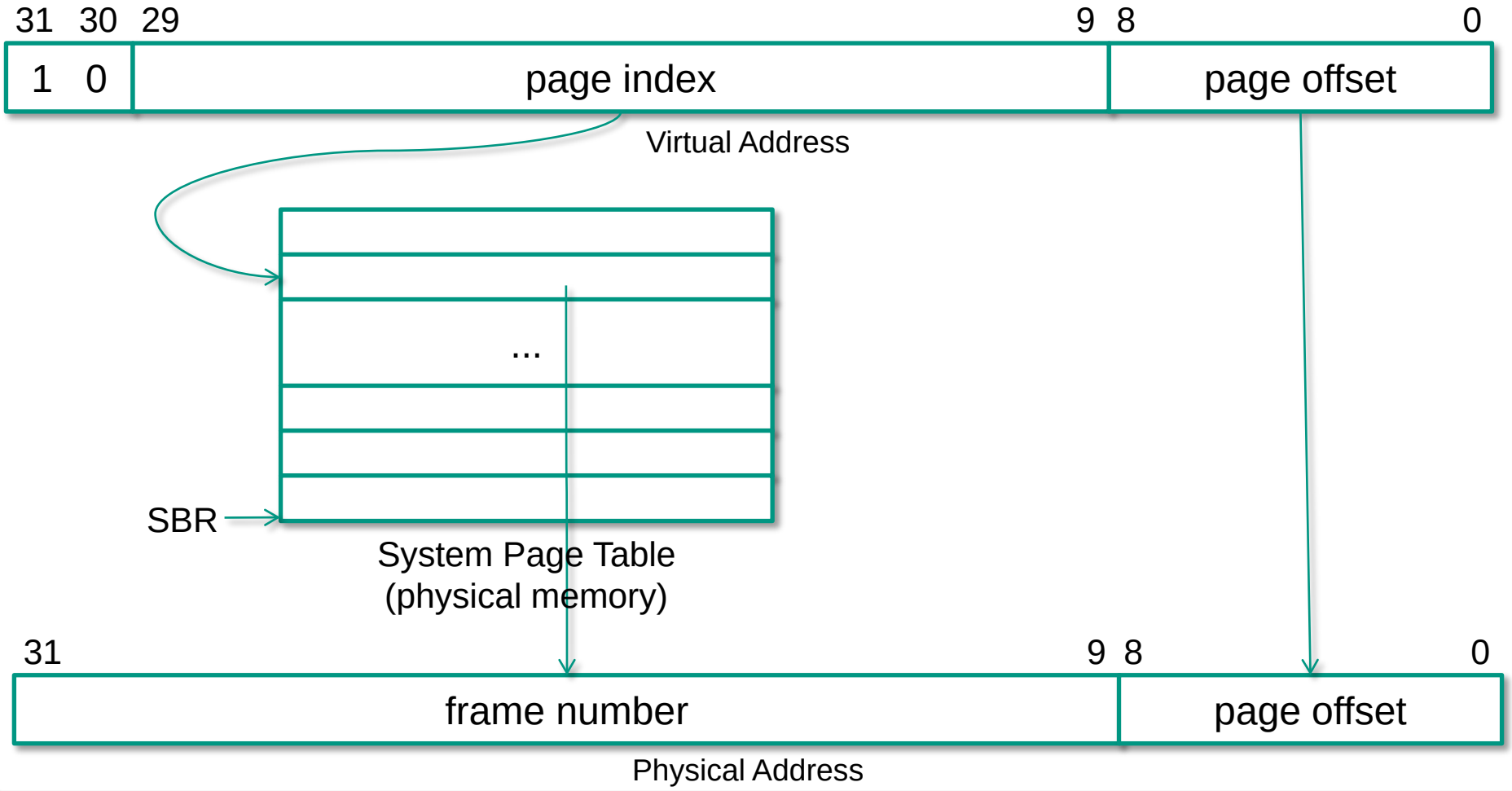
VAX MEMORY MANAGEMENT HARDWARE

VAX Virtual Address Space



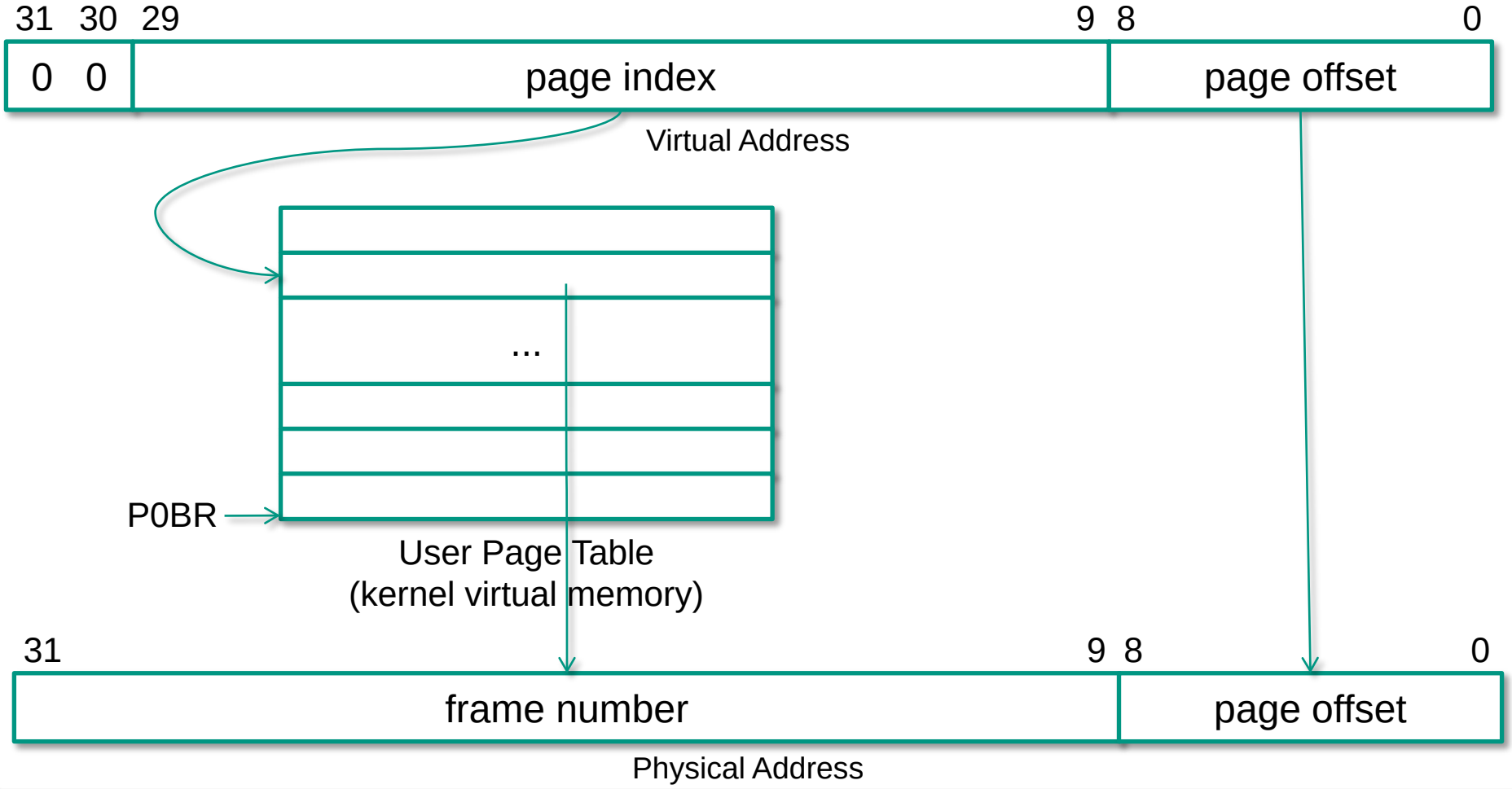
Address Translation (1)

- System-Address Translation



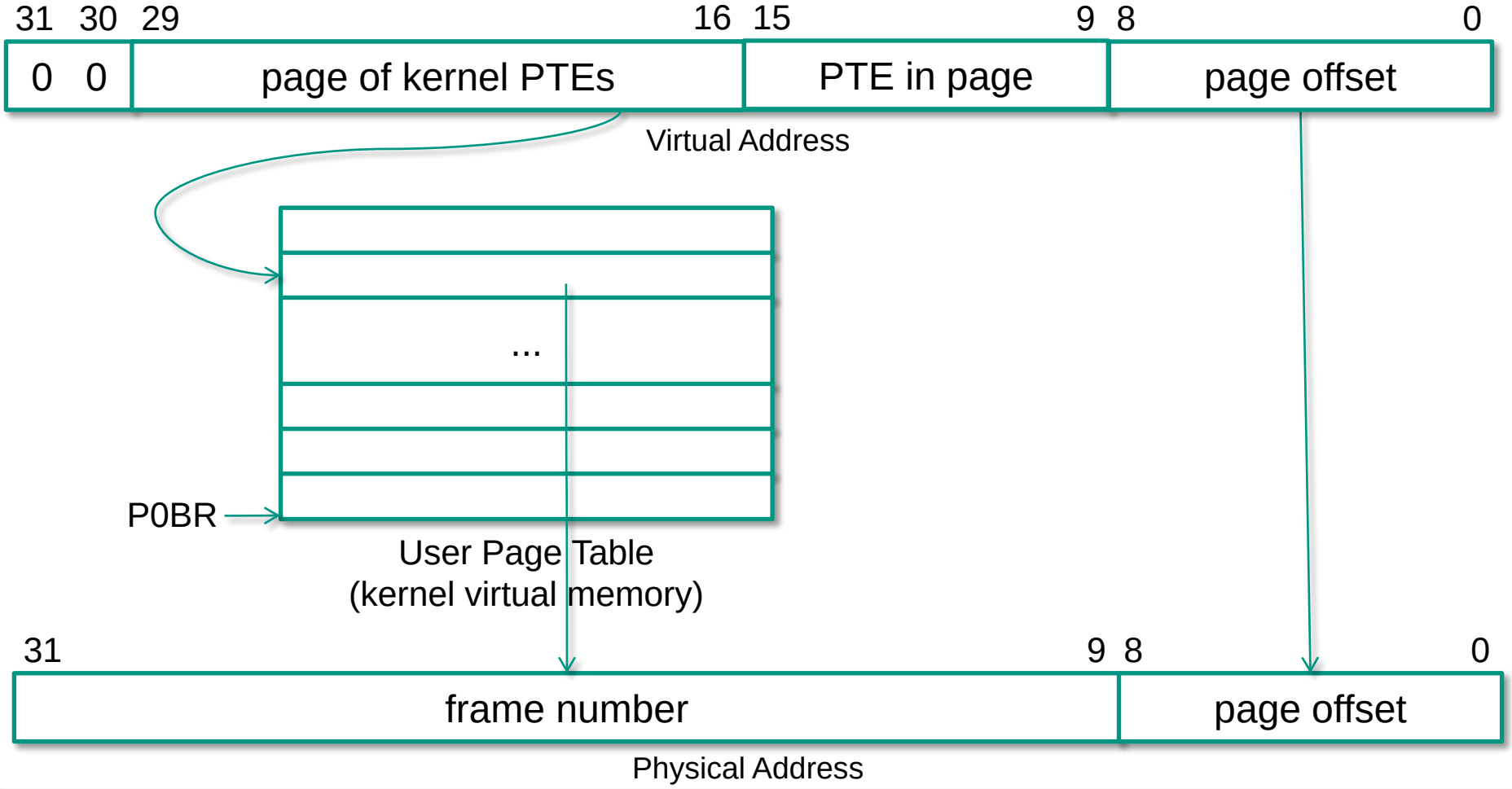
Address Translation (2)

- User-Address Translation



Address Translation (3)

- User-Address Translation (two-level mechanism)



VAX Page Tables

- Format of a VAX page-table entry (PTE)

V: valid bit, *prot*: protection-mode field, *M*: modified bit, *soft*: software defined field

  - 2^{21} page-frame numbers
  - No reference bit

PTE Types

Page Tables

PAGE TABLES AND PTES

PTE Types (1)

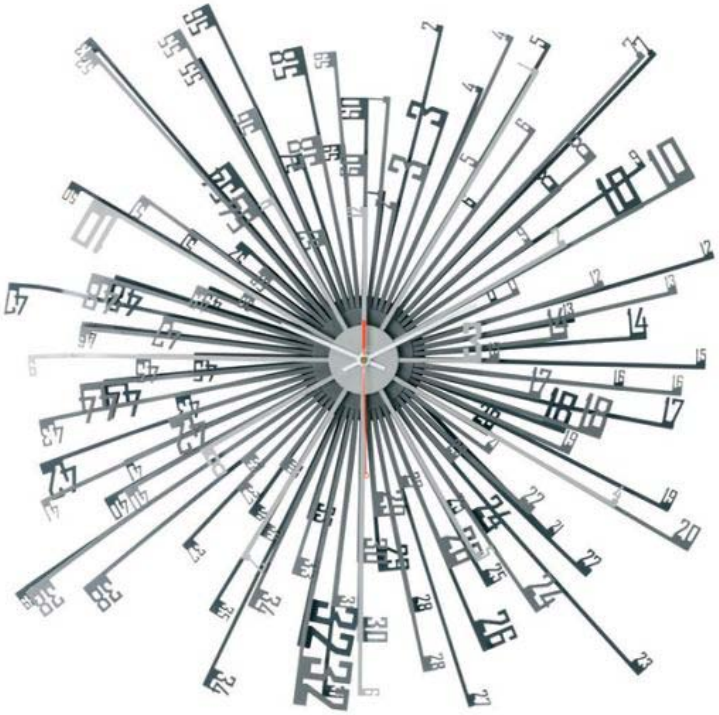
- Normal Page-Table Entry

```
struct pte {
  unsigned int pg_pfnum:21, /* core page frame number or 0 */
               :2,
               pg_vreadm:1, /* modified since vread (or with _m) */
               pg_swapm:1,  /* have to write back to swap */
               pg_fod:1,    /* is fill on demand (=0) */
               pg_m:1,      /* hardware maintained modified bit */
               pg_prot:4,   /* access control */
               pg_v:1;      /* valid bit */
};
```

- Fill-On-Demand Page-Table Entry

```
struct fpte {
  unsigned int pg_blkno:24, /* file system block number */
               pg_fileno:1, /* file mapped from or TEXT or ZERO */
               pg_fod:1,    /* is fill on demand (=1) */
               :1,
               pg_prot:4,
               pg_v:1;
};
```

machine/pte.h



Paging Parameters

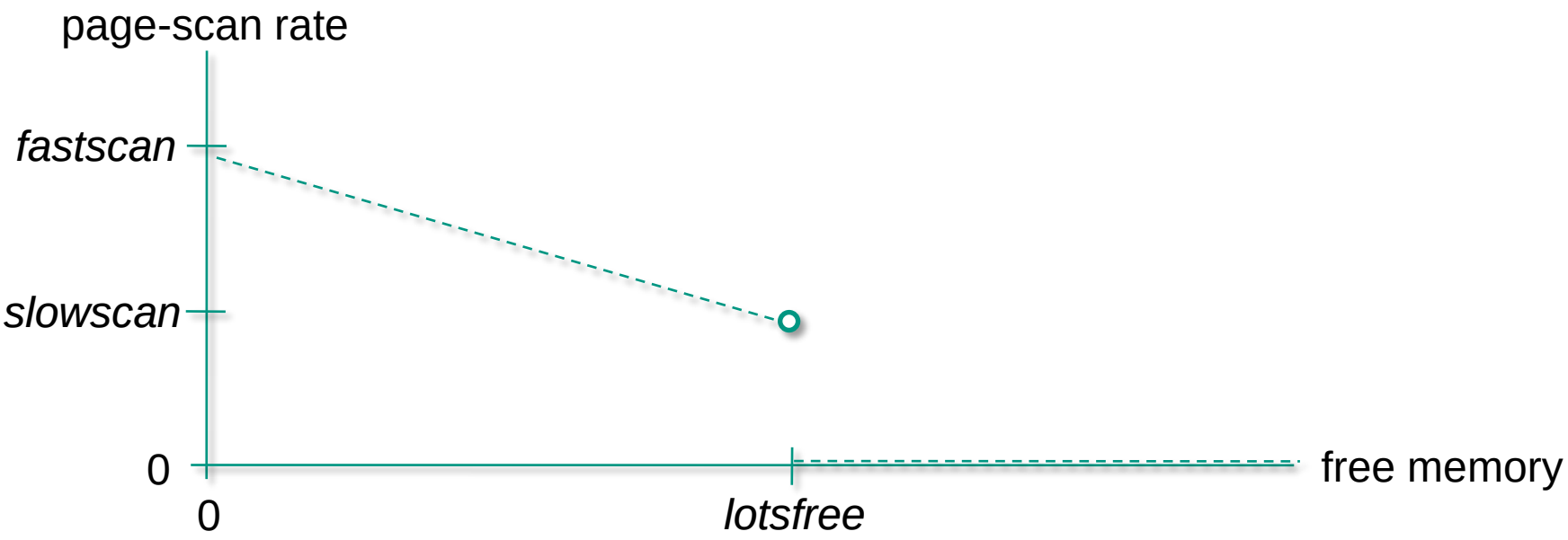
Global CLOCK Algorithm

Two-Handed Clock

Exploring Source Code

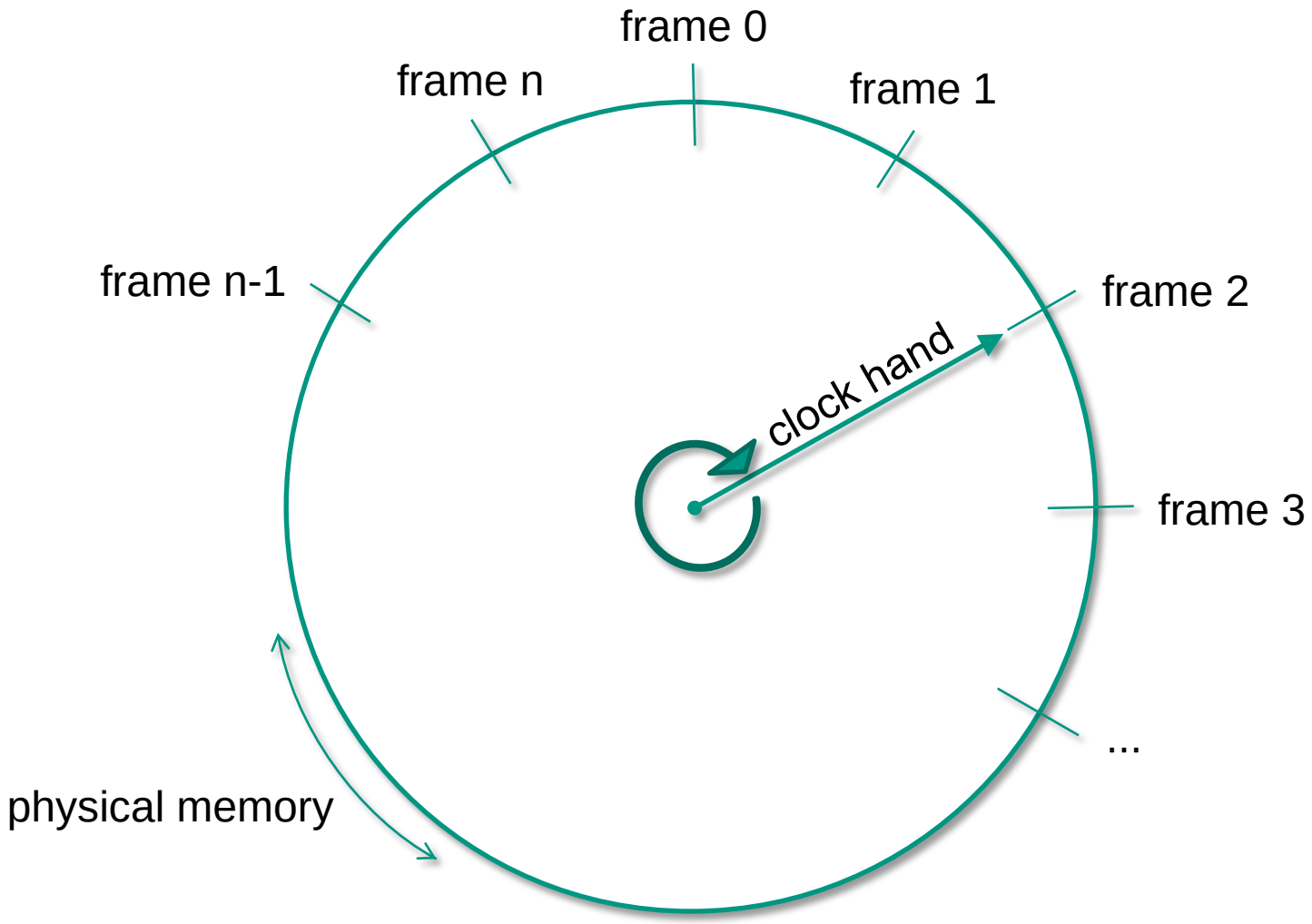
PAGE REPLACEMENT

Paging Parameters



Name	Common Value	Upper Bound	Description
fastscan	200 pages / sec	$^1/_5$ of memory	Fastest page-scan rate
slowscan	100 pages / sec	-	Page-scan rate on *lotsfree*
lotsfree	512 KByte	$^1/_4$ of memory	Threshold for stopping scanning
desfree	200 KByte	$^1/_8$ of memory	Desired amount of free memory
minfree	64 KByte	$^1/_{16}$ of memory	Involuntary swapping begins

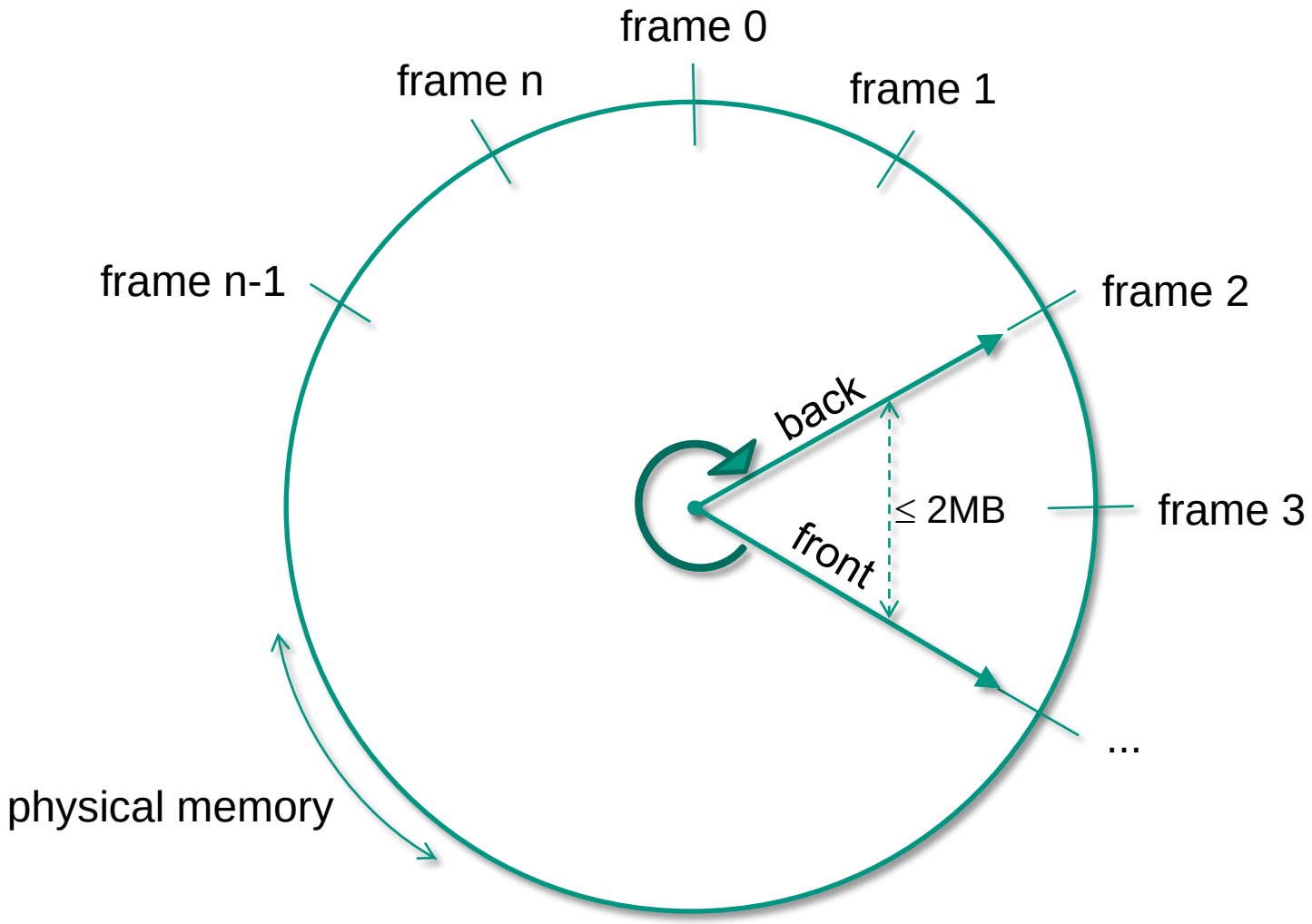
Global CLOCK Algorithm (1)



Global CLOCK Algorithm (2)

- *Global* replacement algorithm
- All processes compete for memory on an equal basis
- No working set model
- Reference bit simulation on VAX
- Performance highly dependent on memory size
- Poor for sudden memory shortfalls

Two-Handed Clock



Exploring S

```
/*
 * An iteration of the clock pointer (hand) around the loop.
 * Look at the page at hand.  If it is a
 * locked (for physical i/o e.g.), system (u., page table)
 * or free, then leave it alone.
 * Otherwise, if we are running the front hand,
 * invalidate the page for simulation of the reference bit.
 * If the proc is over maxrss, we take it.
 * If running the back hand, check whether the page
 * has been reclaimed.  If not, free the page,
 * pushing it to disk first if necessary.
 */
```

```
pageout() {
        register i
        register i
        if (fronth
loop:
        /* (...) *
        sleep((cad
        (void) spl
        count = 0;
        pushes = 0;
        while (nscan < desscan && freemem < lotsfree) {
                if (checkpage(fronthand, FRONT)) count = 0;
                if (checkpage(backhand, BACK)) count = 0;
                cnt.v_scan++;
                nscan++;
                if (++fronthand >= maxhand) {
                        fronthand = 0;
                        cnt.v_rev++;
                        if (count > 2) goto loop;
                        count++;
                }
                if (++backhand >= maxhand) backhand = 0;
        }
        goto loop;
}
```

sys/vm_page.c

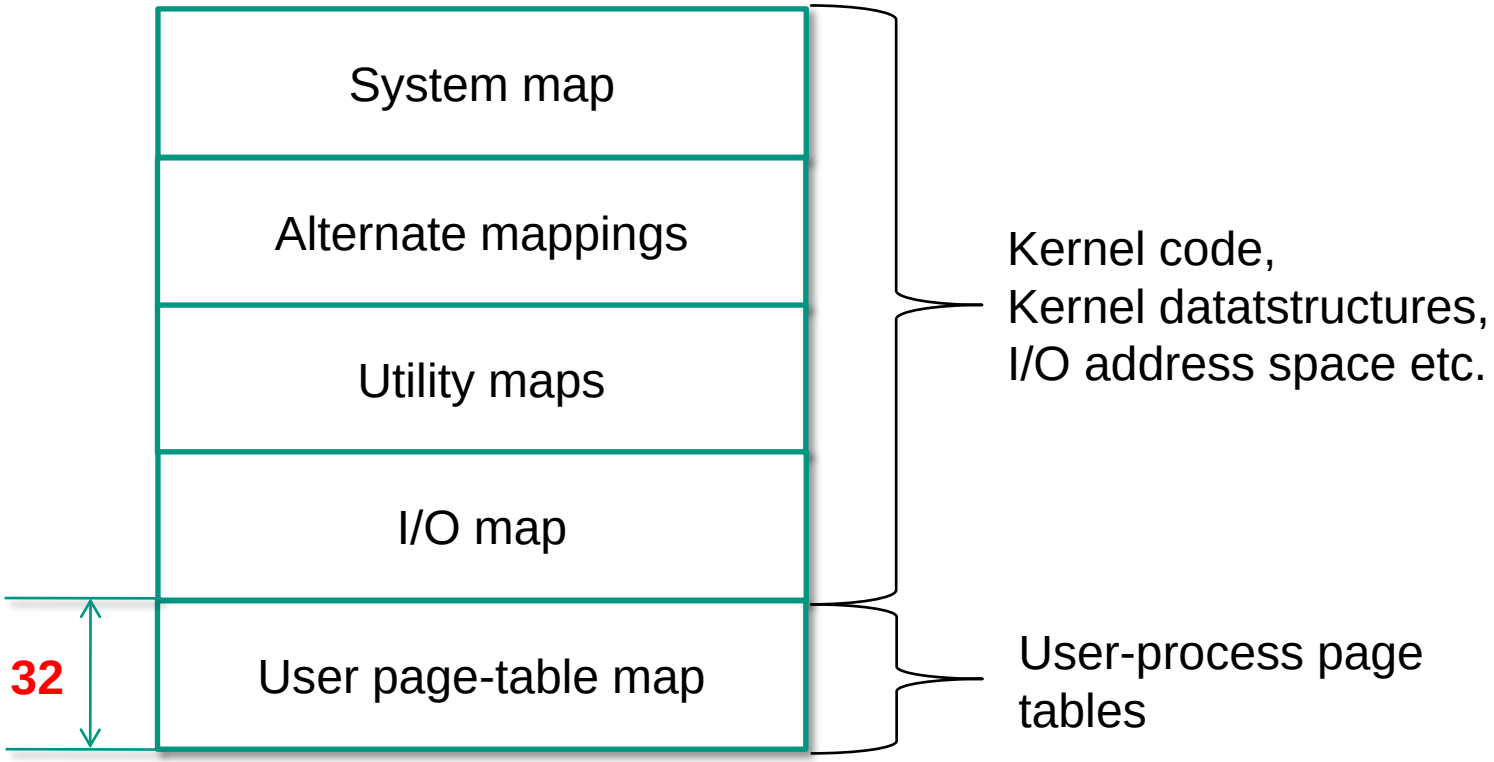
References

- S. J. Leffler, M. K. McKusick, M. J. Karels, J. S. Quaterman „The Design and Implementation of the 4.3BSD UNIX Operating System“
- [Joy et al., 1986]
  W. N. Joy, R. S. Fabry, S. J. Leffler, M. K. McKusick & M. J. Karels „Berkeley Software Architecture Manual, 4.3BSD Edition“ pp. PS1:6-10 – PS1:6-12
- [McKusick & Karels, 1988]
  M. K. McKusick & M. J. Karels, „Design of a General Purpose Memory Allocator for the 4.3BSD UNIX Kernel“ *USENIX Association Conference Proceedings*, pp. 295-303, June 1988.
- [InformationWeek, 2006]
  Babcock, Charles (2006-08-14). "What's The Greatest Software Ever Written?". InformationWeek.
- Wikipedia articles about „UNIX“ and „VAX“, Retrieved 2010-06-14.

Questions

Page Tables

- Layout of the System Page Table



Explorin

```
/*
 * Setup the paging constants for the clock algorithm.
 * Called after the system is initialized and the amount of memory
 * and number of paging devices is known.
 * (...)
 */
```

```
main(firstad
        /* (
        startup(firstaddr);
        /* set up system process 0 (
        /* (...) */
        vminit();
         /* (...) */
/* kick off timeout driven events by
        roundrobin();
        schedcpu();
        schedpaging();
/* set up the root file system, make init process */
        /* (...) */
        /* make page-out daemon (process 2) (...) */
        proc[0].p_szpt = clrnd(ctopt(nswbuf*CLSIZE*KLMAX + UPAGES));
        if (newproc(0)) {
                proc[2].p_flag |= SLOAD|SSYS;
                proc[2].p_dsize = u.u_dsize = nswbuf*CLSIZE*KLMAX;
                pageout();
                /*NOTREACHED*/
        }
        /* enter scheduling loop */
        proc[0].p_szpt = 1;
        sched();
}
```

```
 *      clear and free user core
 *      turn on clock
 *      hand craft 0th process
 *      call all initialization routines
 *      fork - process 0 to schedule
 *           - process 1 execute bootstrap
 *           - process 2 to page out
 */
```

sys/init_main.c

Exploring Source Code (shedpaging)

```
/*
 * Schedule rate for paging.
 * Rate is linear interpolation between
 * slowscan with lotsfree and fastscan when out of memory.
 */
schedpaging()
{
        register int vavail;

        nscan = desscan = 0;
        vavail = freemem - deficit;
        if (vavail < 0)
                vavail = 0;
        if (freemem < lotsfree) {
                desscan = (slowscan * vavail + fastscan * (lotsfree - vavail)) /
                        nz(lotsfree) / RATETOSCHEDPAGING;
                wakeup((caddr_t)&proc[2]);
        }
        timeout(schedpaging, (caddr_t)0, hz / RATETOSCHEDPAGING);
}
```

sys/vm_sched.c