Is it time to replace mmap?

A history of virtual address management (and a proposal for the future)

Brooks Davis
SRI International

BSDTW 2017. Taipei, Taiwan

Approved for public release; distribution is unlimited. This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-10-C-0237. The views, opinions, and/or findings contained in this article/presentation are those of the author(s)/presenter(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

Memory

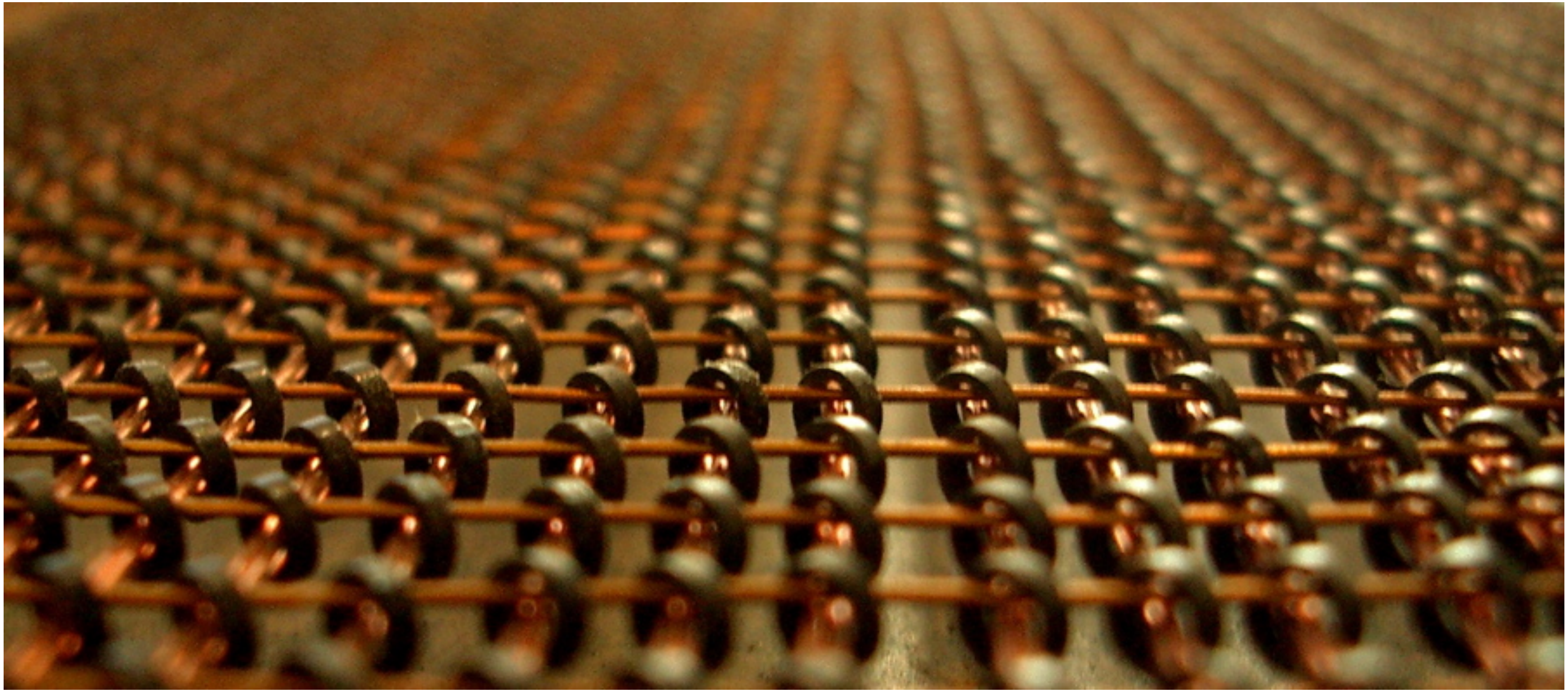
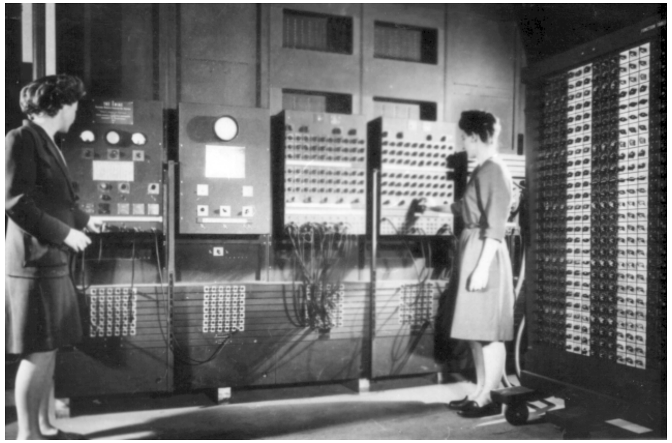


Photo credit: Steve Jurvetson from Menlo Park, USA

A bit of computer history



ENIAC c.1945

EDSAC c.1949

PDP-11 c.1970

Baby c.1948

1386 1985

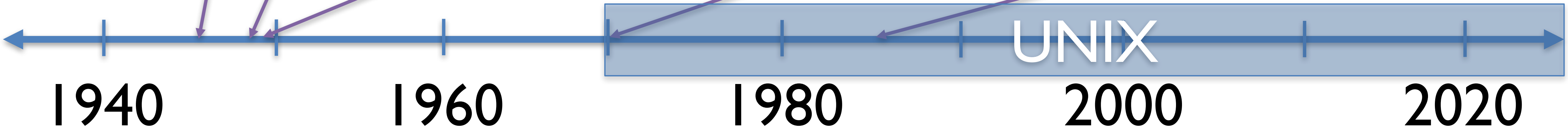
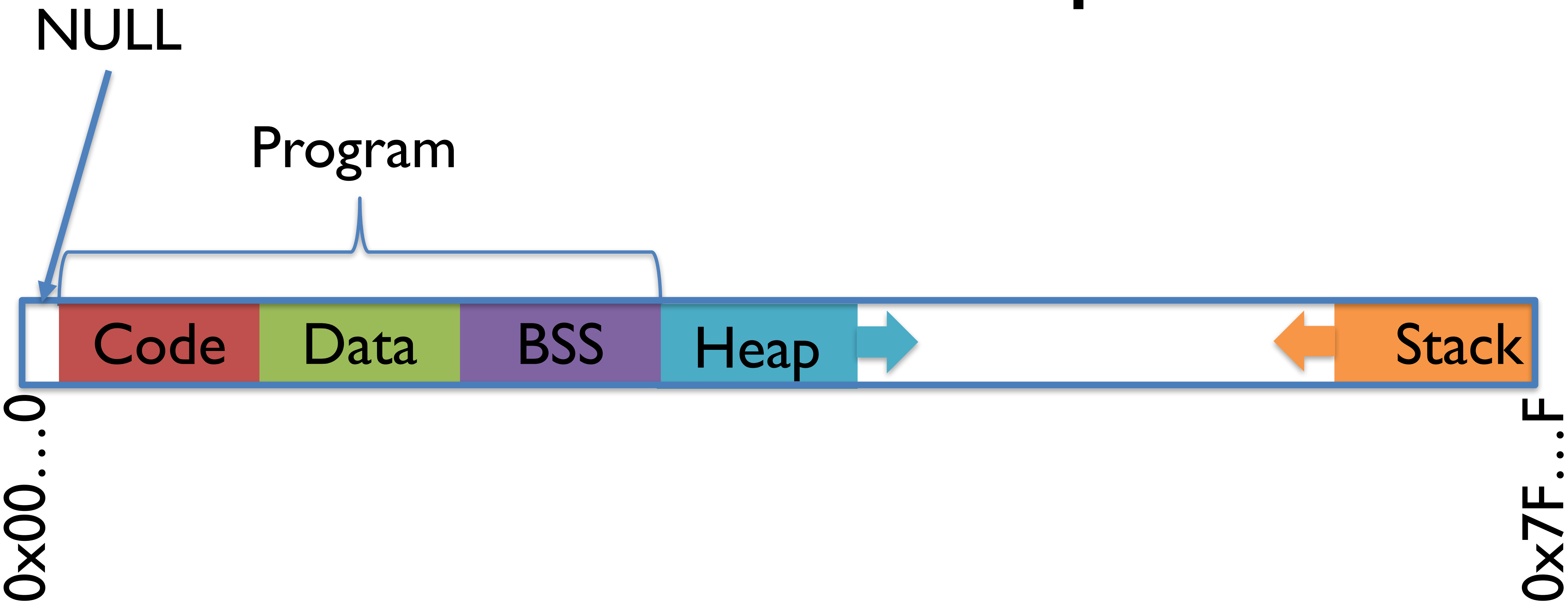
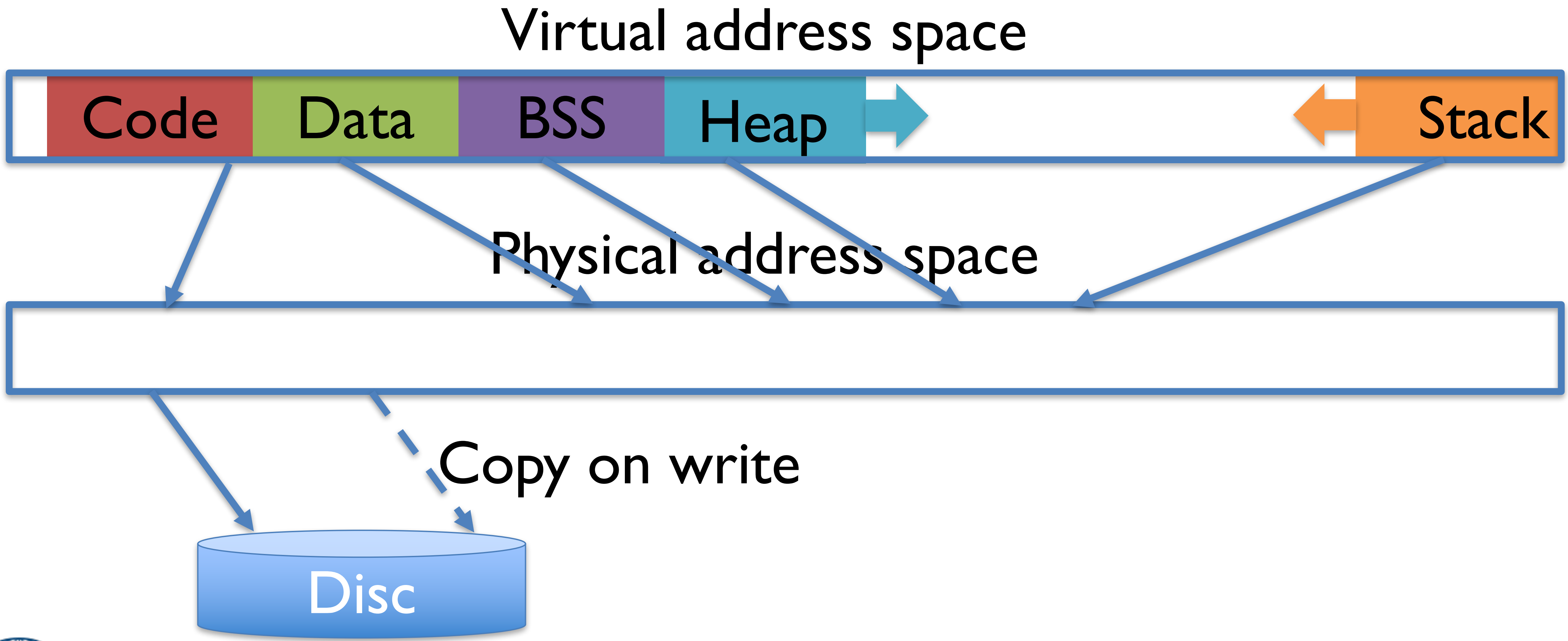


Photo sources:
ENIAC: Two women operating ENIAC - U.S. Army Photo
EDSAC: EDSAC I, R.Hill operating - Copyright Computer Laboratory, University of Cambridge. Reproduced by permission. PDP-11: DEC - PDP-11- Ken Thompson and Dennis Ritchie – Courtesy Computer History Museum

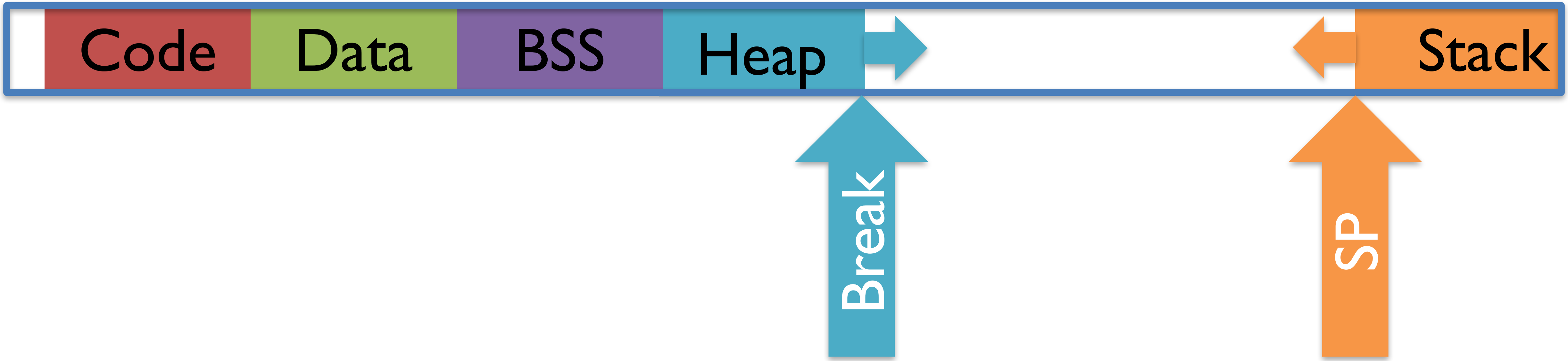
Process address space



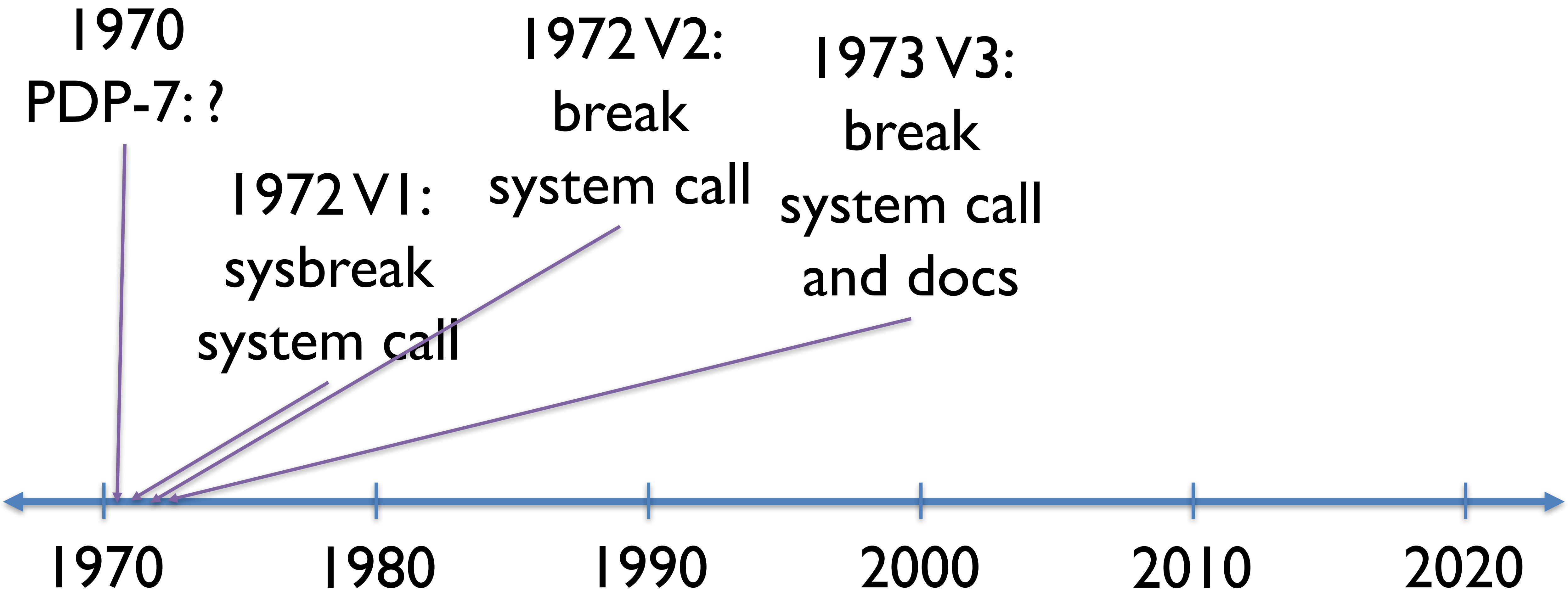
Process address space



Process address space



UNIX and BSD



1970 PDP-7: ?

1972 V1: sysbreak system call

1972 V2: break system call

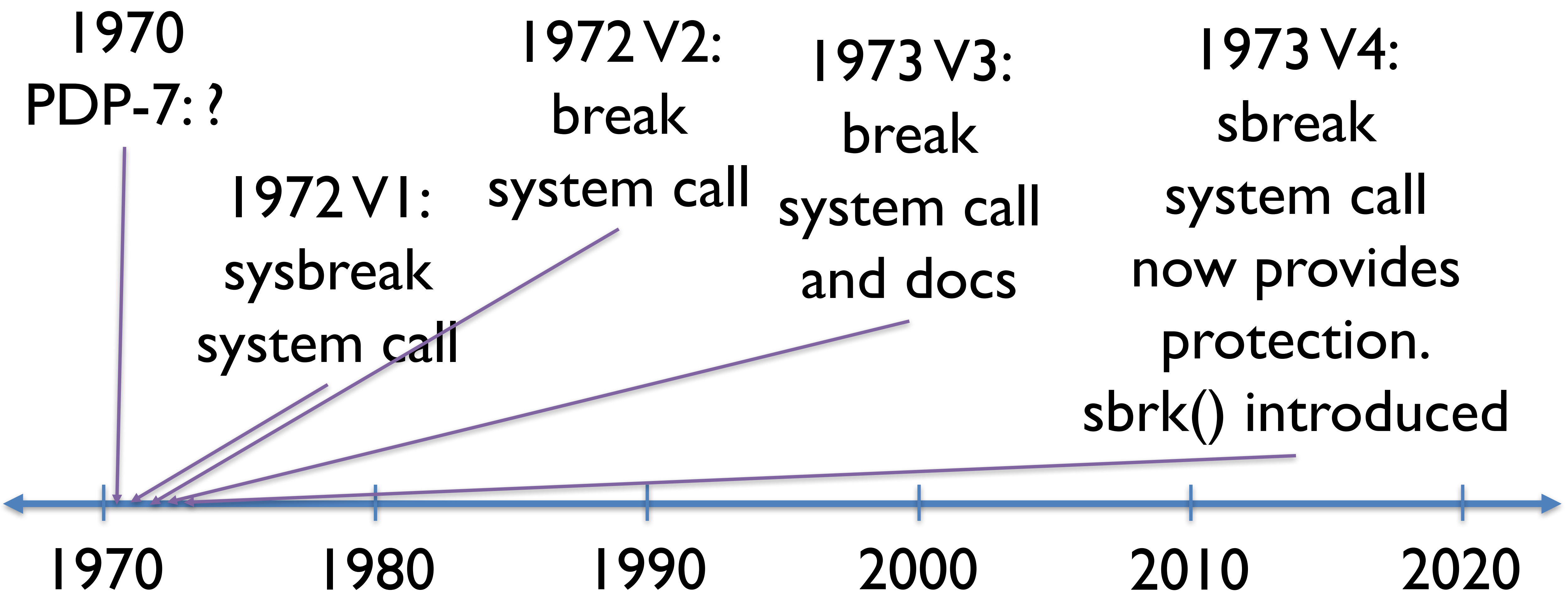
1973 V3: break system call and docs

break.2 (V3 Unix)

break sets the system's idea of the highest location used by the program to addr.

Locations greater than addr and below the stack pointer are not swapped and are thus liable to unexpected modification.

UNIX and BSD



1970 PDP-7: ?

1972 V1: sysbreak system call

1972 V2: break system call

1973 V3: break system call and docs

1973 V4: sbreak system call now provides protection. sbrk() introduced

break.2 (V4 Unix)

Break sets the system's idea of the lowest location not used by the program to *addr* (rounded up to the next multiple of 64 bytes).

Locations not less than *addr* and below the stack pointer are not in the address space and will thus cause a memory violation if accessed.

break.2 (V4 Unix) (cont)

```
char *sbrk(incr)
```

…

From C, the calling sequence is different; *incr* more bytes are added to the program's data space and a pointer to the start of the new area is returned.

break.2 (V4 Unix) (cont)

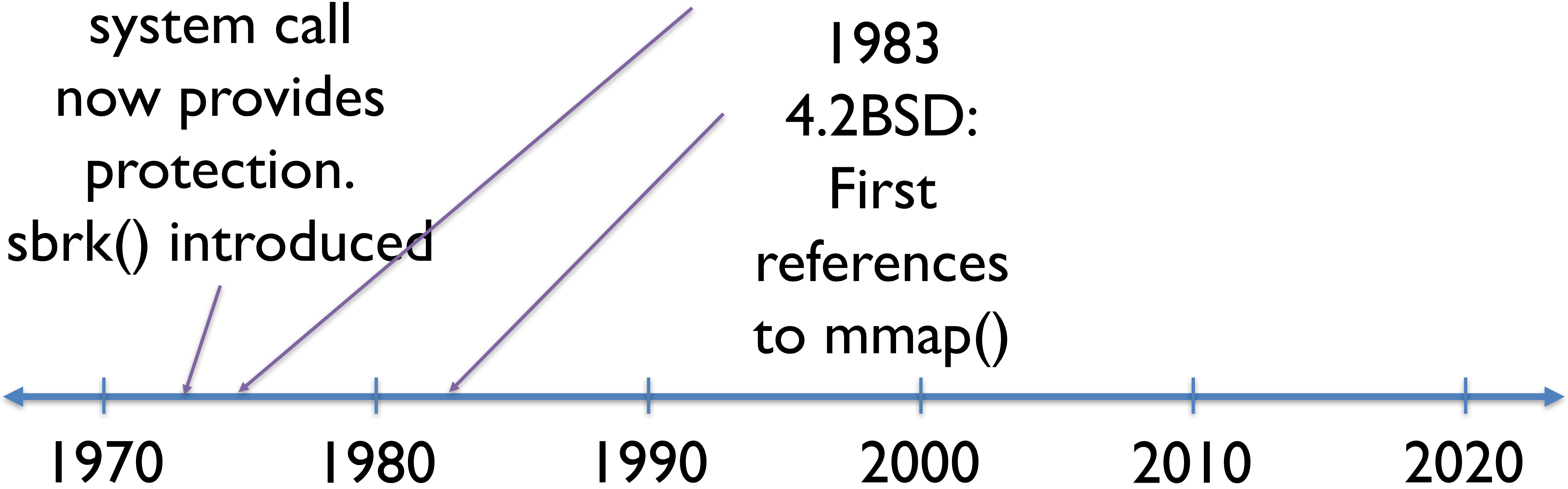
When a program begins execution via *exec* the break is set at the highest location defined by the program and data storage areas. Ordinarily, therefore, only programs with growing data areas need to use *break.*

UNIX and BSD

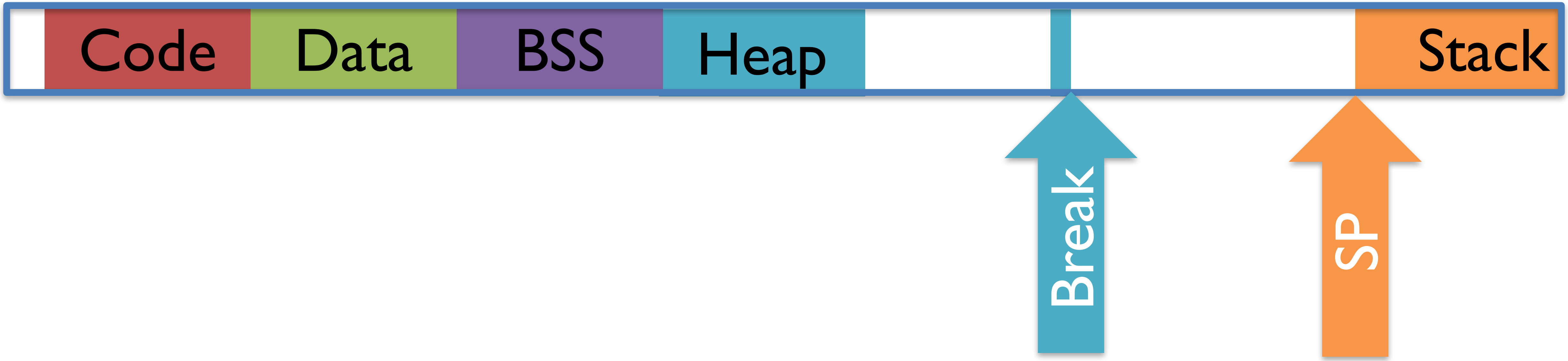
1973 V4:
sbreak
system call
now provides
protection.
sbrk() introduced

1975 V5:
brk() introduced

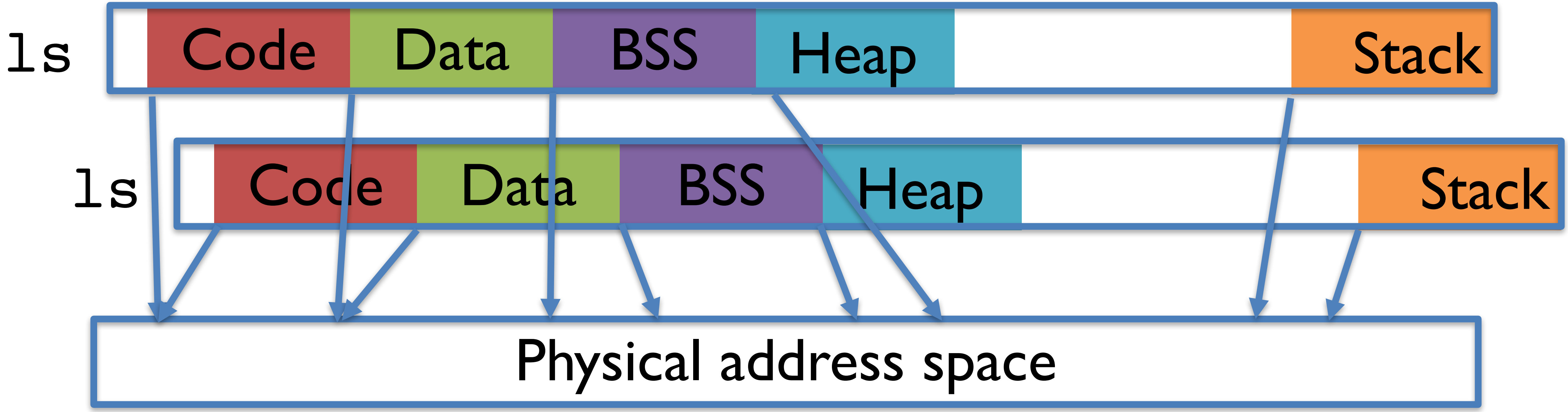
1983
4.2BSD:
First
references
to mmap()



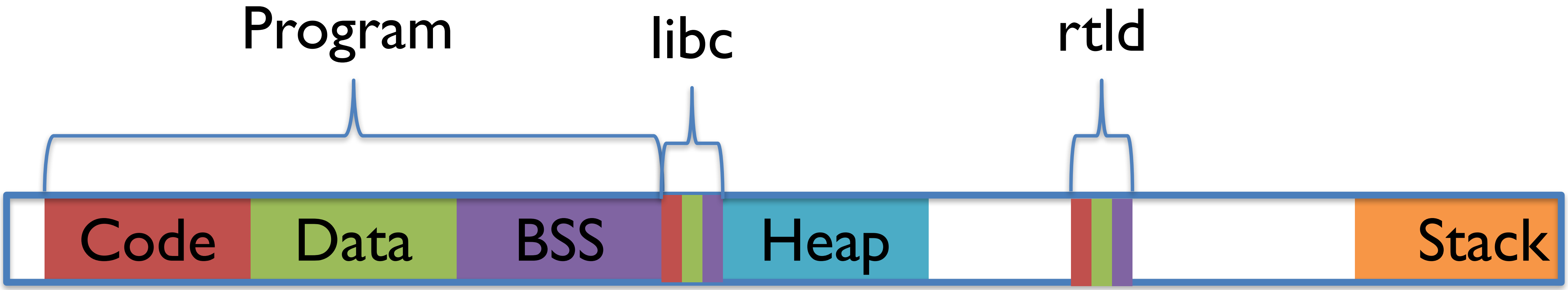
Heap fragmentation



Memory sharing

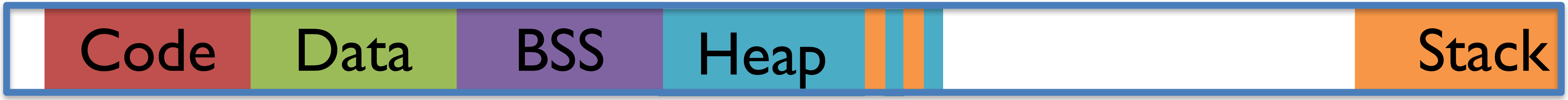


Dynamic linking

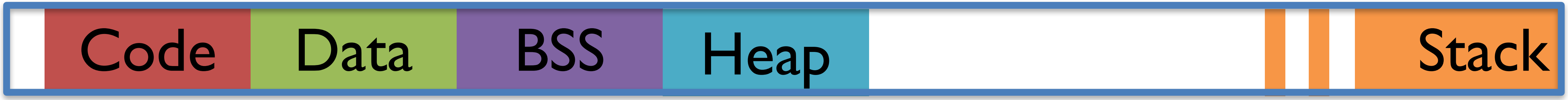


Multi-threaded programs

With sbrk() only



Ideal



4.2BSD memory interfaces

- mmap()
  - Allocate address space
  - Alter backing mappings
- mremap()
  - Relocate or extend mapping
- munmap()
  - Remove backing

4.2BSD memory interfaces

- mprotect()
  - Alter page protections
- madvise()
  - Hint usage to kernel
- mincore()
  - Query backing status
- sbrk()
  - Extend or reduce "break"
- sstk()
  - Extend or reduce stack

4.2BSD memory interfaces

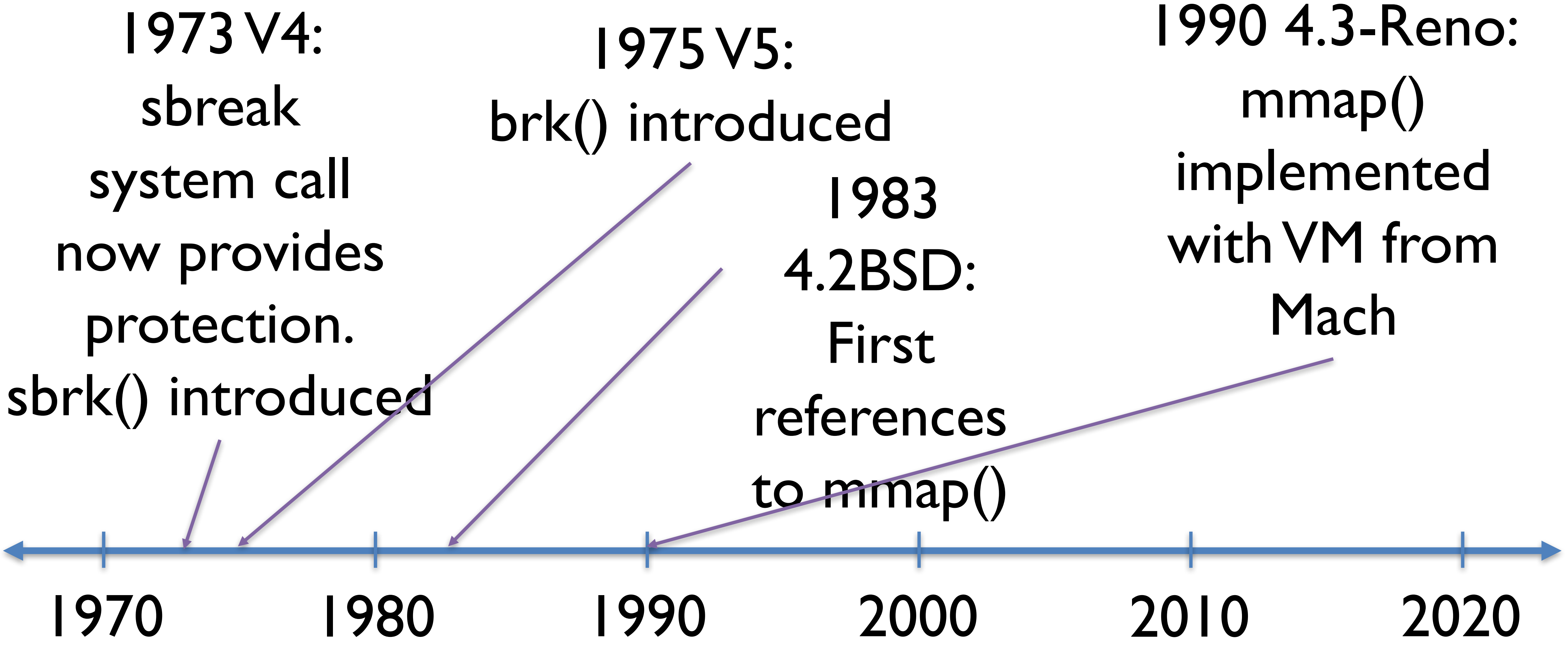
- mprotect()
  - Alter page protect
- madvi
-

  - Query backing status

- sbrk()
- ()
  - Extend or reduce stack

Only sbrk() implemented!

UNIX and BSD



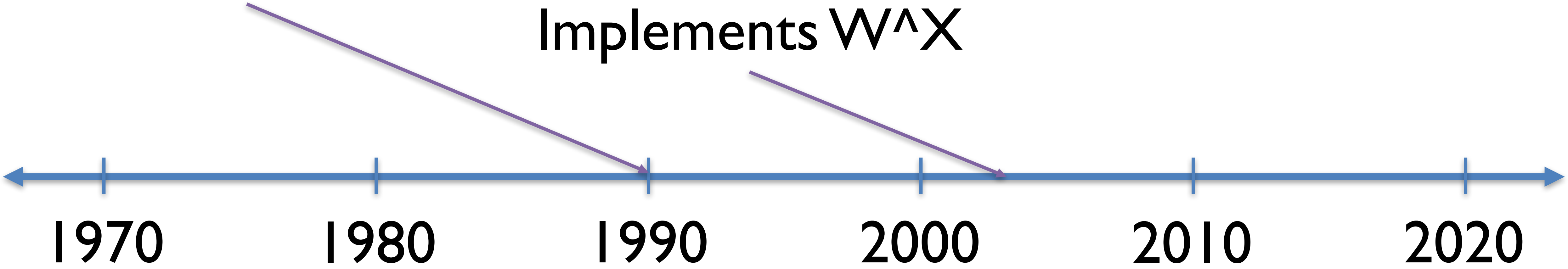
1973 V4:
sbreak
system call
now provides
protection.
sbrk() introduced

1975 V5:
brk() introduced

1983
4.2BSD:
First
references
to mmap()

1990 4.3-Reno:
mmap()
implemented
with VM from
Mach

UNIX and BSD



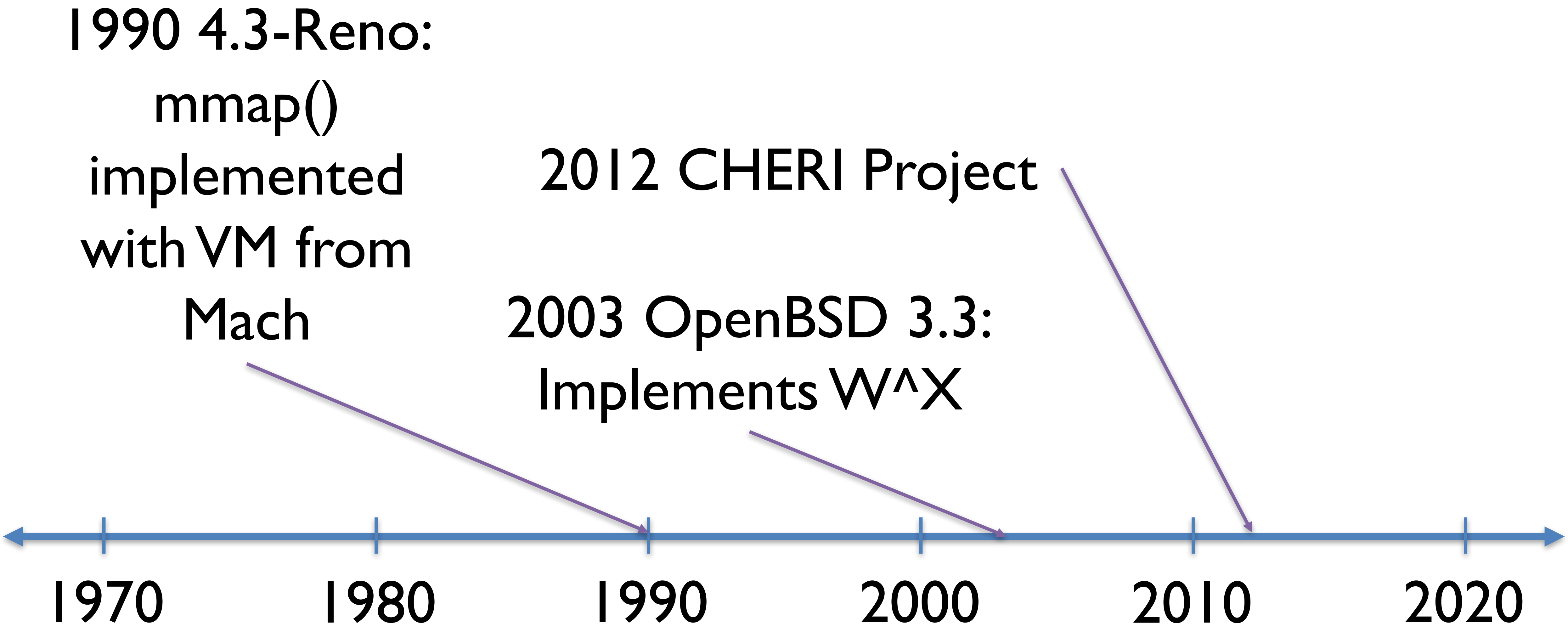
1990 4.3-Reno: mmap() implemented with VM from Mach

2003 OpenBSD 3.3: Implements W^X

W^X and JITs

- Prohibits pages from having both PROT_EXEC and PROT_WRITE simultaneously
- JITs need to write then execute!
- Solution: Map PROT_WRITE then remove PROT_WRITE and add PROT_EXEC
- New problem: most pages should not become executable, but mmap() cannot express this!

UNIX and BSD



1990 4.3-Reno: mmap() implemented with VM from Mach

2012 CHERI Project

2003 OpenBSD 3.3: Implements W^X

CHERI pointers

- Pointers with bounds and permissions
  - With strong monotonicity guarantees
- Want W^X for pointers (in addition to pages)
- API changes required:
  - Should mprotect() return a pointer?
  - Should some other mechanism be used?

mmap() functionality issues

- Interface conflates address reservation and mapping
  - Lack of boundries between reservations leads to bugs: e.g. Stack Clash
- Lack of expressiveness
  - No portable way to express alignment
  - No way to express maximum permission

mmap() API issues

- Too many arguments
  - Can you remember them all?
  - Many calls don’t use them all
- Too many failure modes:
  - FreeBSD 11: 19 documented errors (15 use the same error code)

Other mmap() issues

- No support for mapping more pages than requested
    - Can’t round up to superpage size
    - CHERI bounds compression requires rounding for very large allocations
- No concept of address space ownership
    - Math errors mean changing the wrong region

RFC: cmmap (1/3)

- int cmreserve(cm_t *handlep, size_t length, vaddr_t hint, int prot, cmreq_t *cmr);
  - Reserve a region, optionally mapping.
- int cmgetptr(cm_t handle, void **ptrp);
  - Get a pointer to the region.

RFC: cmmap (2/3)

- int cmap(cm_t handle, cmreq_t *cmr);
  - Replace (part of) a region’s mappings.
- int cmclose(cm_t handle);
  - Close a handle.
- int cmrestrict(cm_t handle, XX ops, XX *oops);
  - Restrict the set of operations on a handle

RFC: cmmap (2/3)

- int cmstat(cm_t handle, size_t index,
  struct cm_stat * cs)
  - Return data on a series of submaps
- cmadvise(), cmincore(), cminherit(), cmsync(), cmunmap()
  - Like mmap() counterparts, but within region

More on map requests

- Request object rather than many arguments
  - cm_request_t following pthread_attr_t model
- Accessor functions to set up request
- Goal: useful defaults
  - Ideally, requests should always be valid

CHERI extensions

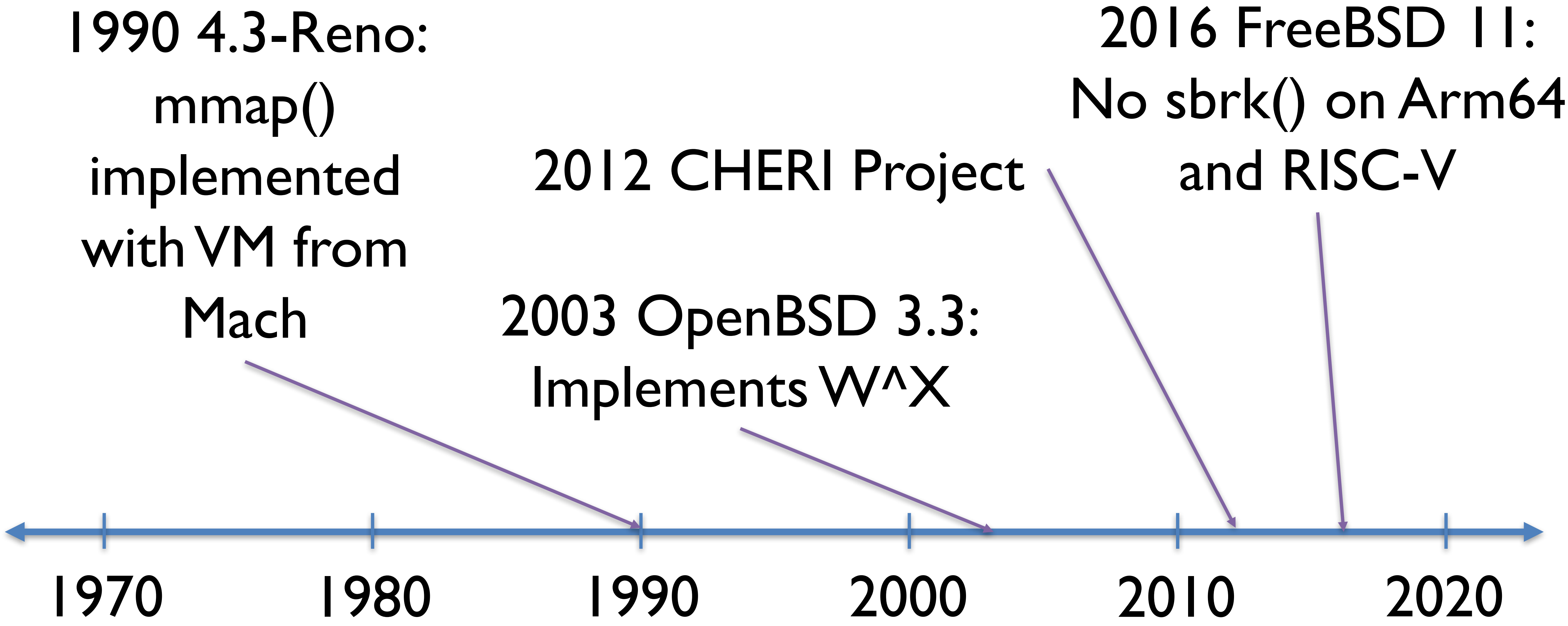
- int cmgetcap(cm_t cookie, void **ptrp, perm_t perms)
  - Get a capability pointer
- int cmandperm(cm_t cookie, perm_t perms, perm_t *operms)
  - Reduce the set of allowed permissions

Should we replace mmap()?

Yes or No?

ON MORE THING

UNIX and BSD



1990 4.3-Reno: mmap() implemented with VM from Mach

2012 CHERI Project

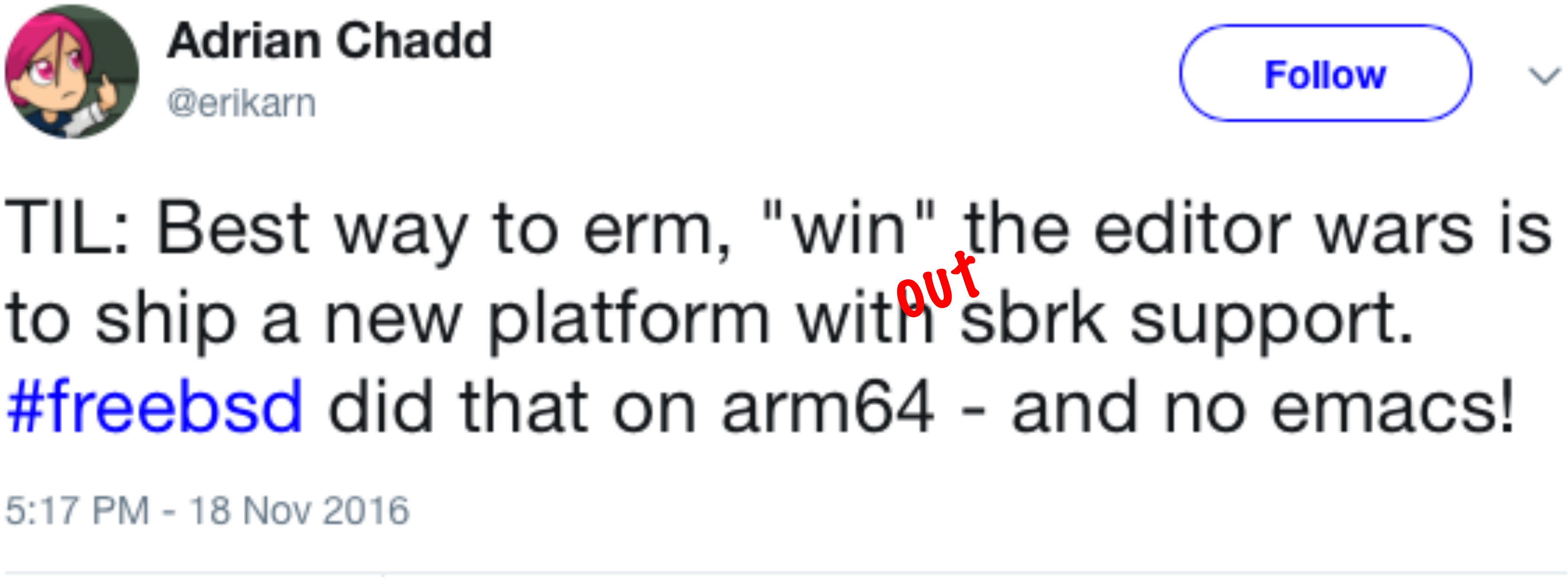
2016 FreeBSD 11: No sbrk() on Arm64 and RISC-V

2003 OpenBSD 3.3: Implements W^X

Removing sbrk()

- Mostly incorrect attempts to measure heap use
  - Usually can be disabled, but some force required
- A few internal allocators
  - Usually can be disabled
- Some LISP interpreters
  - Mostly unpopular ones

Removing sbrk() (cont.)



Adrian Chadd
@erikarn

TIL: Best way to erm, "win" the editor wars is to ship a new platform with out sbrk support. #freebsd did that on arm64 - and no emacs!

5:17 PM - 18 Nov 2016