CTSRD

CRASH-worthy Trustworthy Systems Research and Development

CheriABI

A pure-capability OS process environment

Brooks Davis

Robert N. M. Watson, Simon W. Moore, Peter G. Neumann, Jonathan Woodruff, Jonathan Anderson, Hadrien Barral, Ruslan Bukin, David Chisnall, Nirav Dave, Lawrence Esswood, Khilan Gudka, Alexandre Joannou, Chris Kitching, Ben Laurie, A. Theo Markettos, Alan Mujumdar, Steven J. Murdoch, Robert Norton, Philip Paeps, Alex Richardson, Michael Roe, Colin Rothwell, Hassen Saidi, Stacey Son, Munraj Vadera, Hongyan Xia, and Bjoern Zeeb

University of Cambridge, SRI International

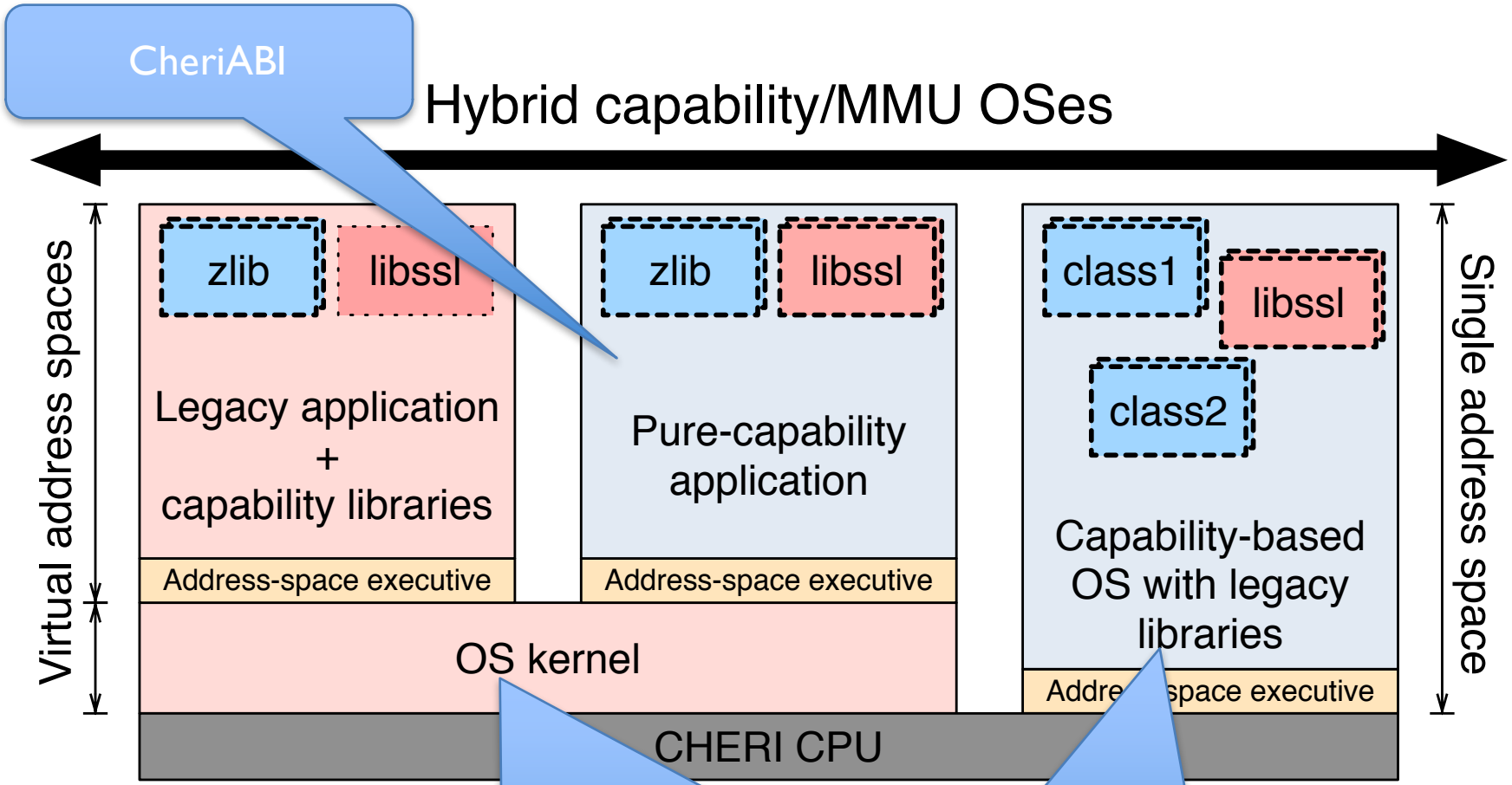
CHERI Microkernel Workshop – 23 April 2016

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 contracts FA8750-10-C-0237 ('CTSRD') and FA8750-11-C-0249 ('MRC2'). 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.

SRI International

University of Cambridge

Software deployment models



Hybrid MMU-capability models: protection and compartmentalization within virtual address spaces

Single-address-space systems are possible but not yet our focus

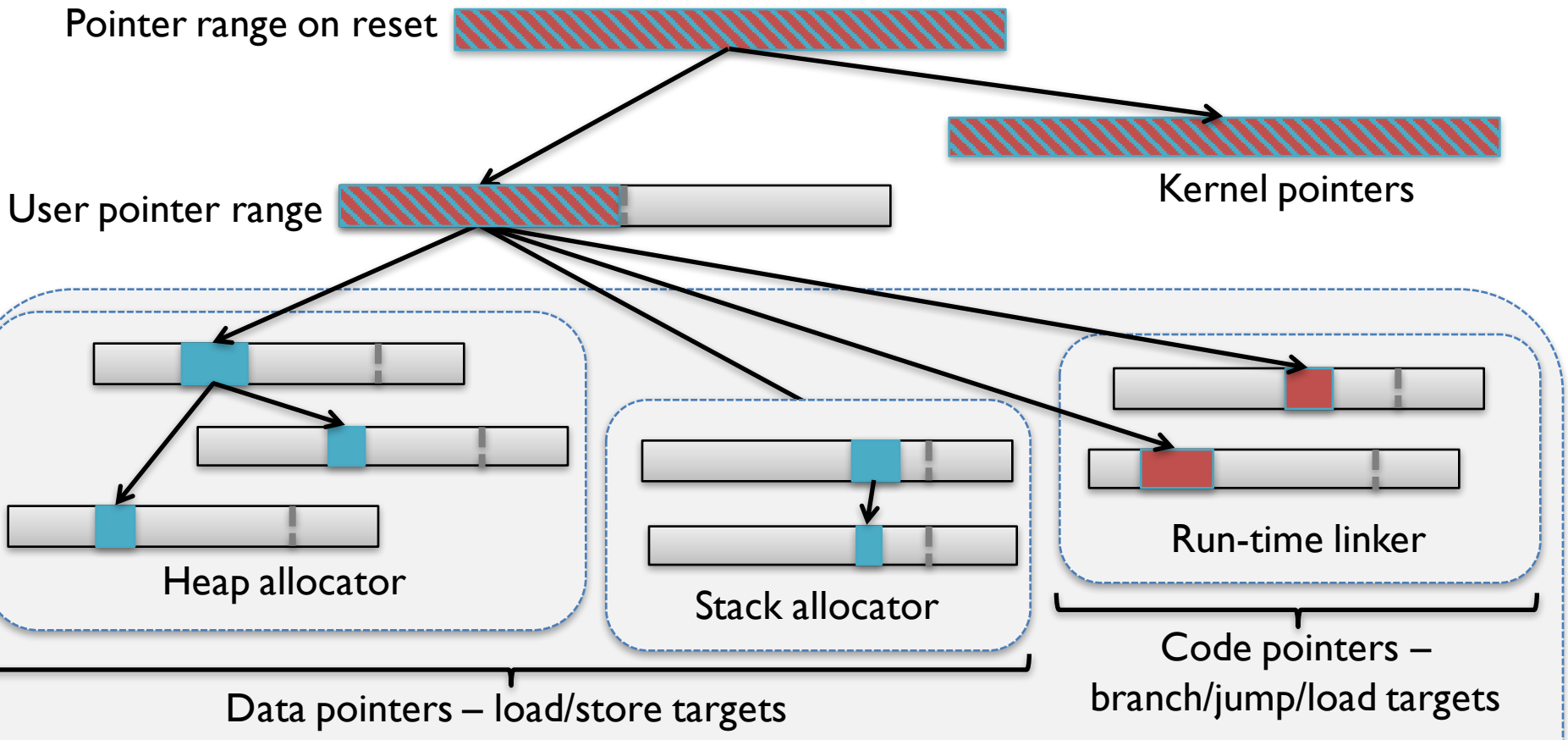
Aim of CheriABI: pure-capability POSIX applications

- **Goals:**
  - Recompile to get memory safety and CFI
  - Explore pure-capability process environment
- **Principles:**
  - Change implementation, not declarations
  - Pointers to C objects are *data* capabilities
    - malloc() allocates data references, never code
  - Page table manipulation syscalls allow (controlled) escape from C memory model

CheriABI challenges

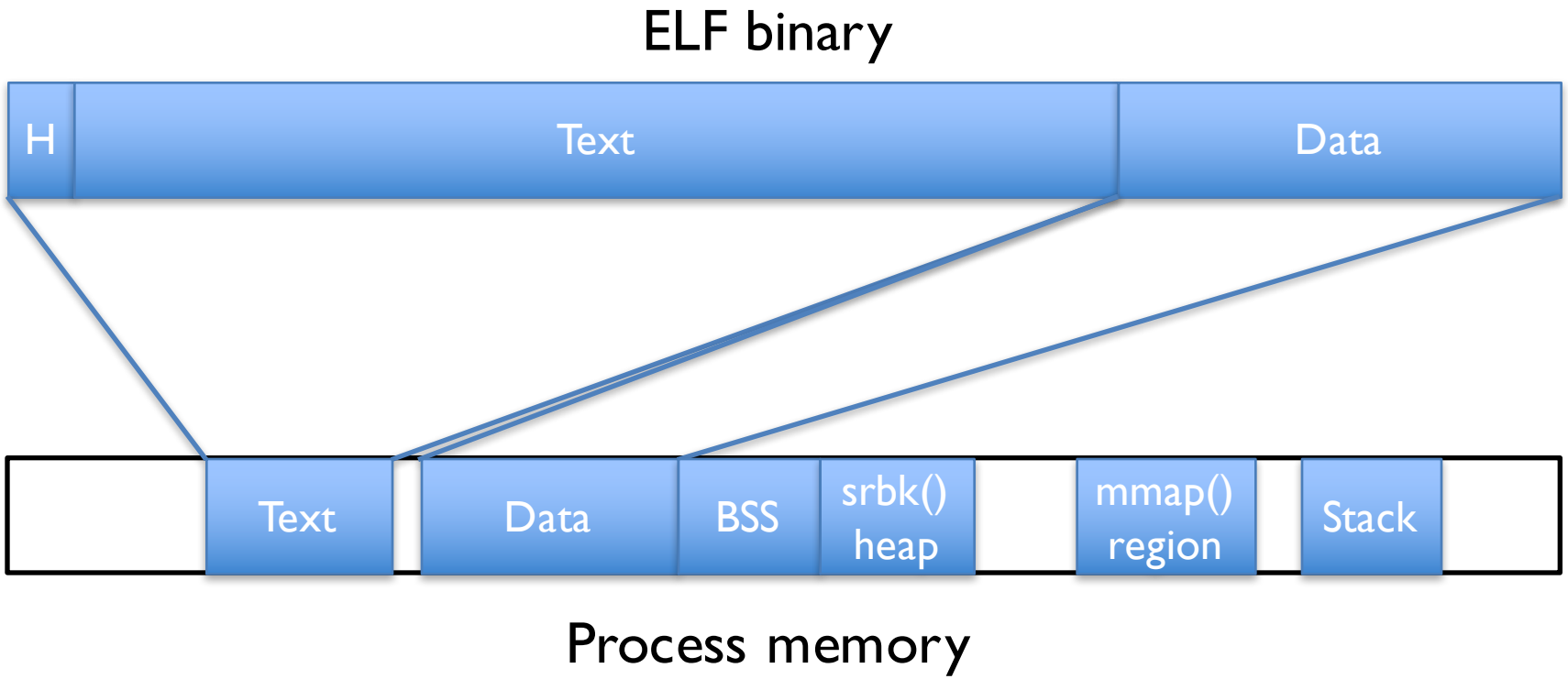
- Managing sources of capabilities
  - Kernel/run-time linker file mappings, argv, environ
  - Language level: malloc(), stack, globals, TLS
  - Page-table mappings
    - POSIX APIs allow any page to be replaced!
- Translating capabilities and capability containing objects into capability-oblivious kernel
  - Validating translations
- Composing defacto-POSIX with memory safety

Enforcing pointer provenance, bounds, permissions, and monotonicity

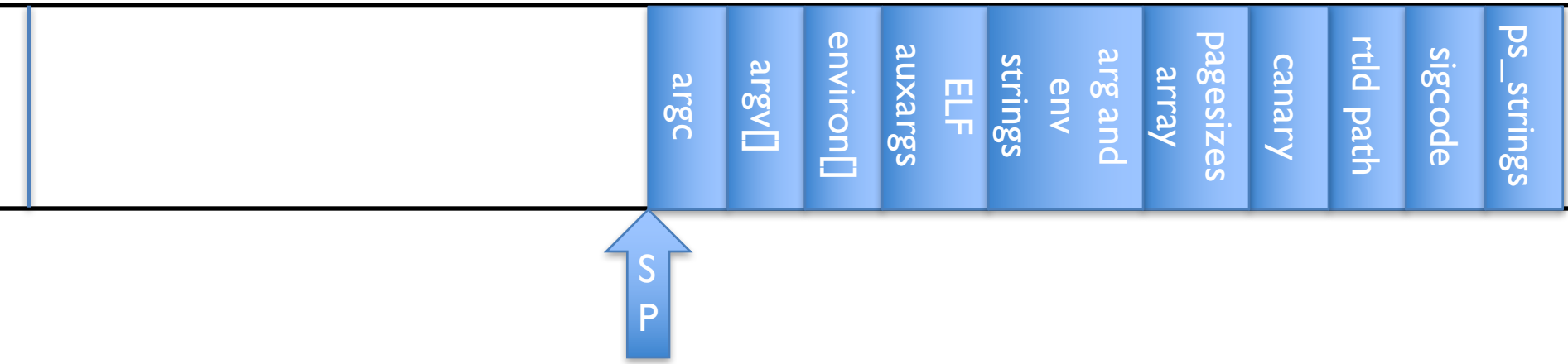


Valid userspace pointer set – pointers not generated using derivation rules are not part of the valid provenance tree and should not be dereferenceable

MIPS process layout



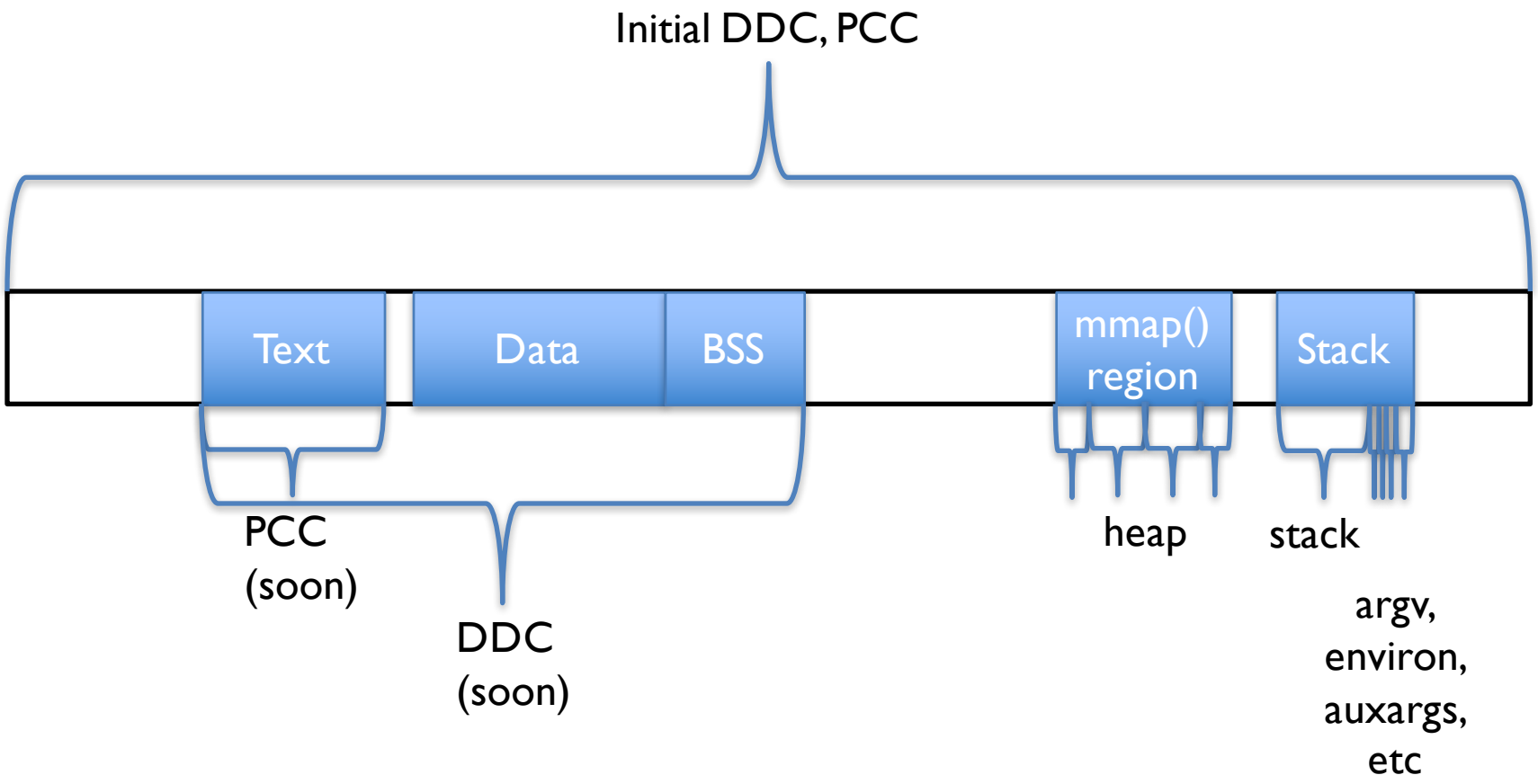
SCO i386 ABI stack



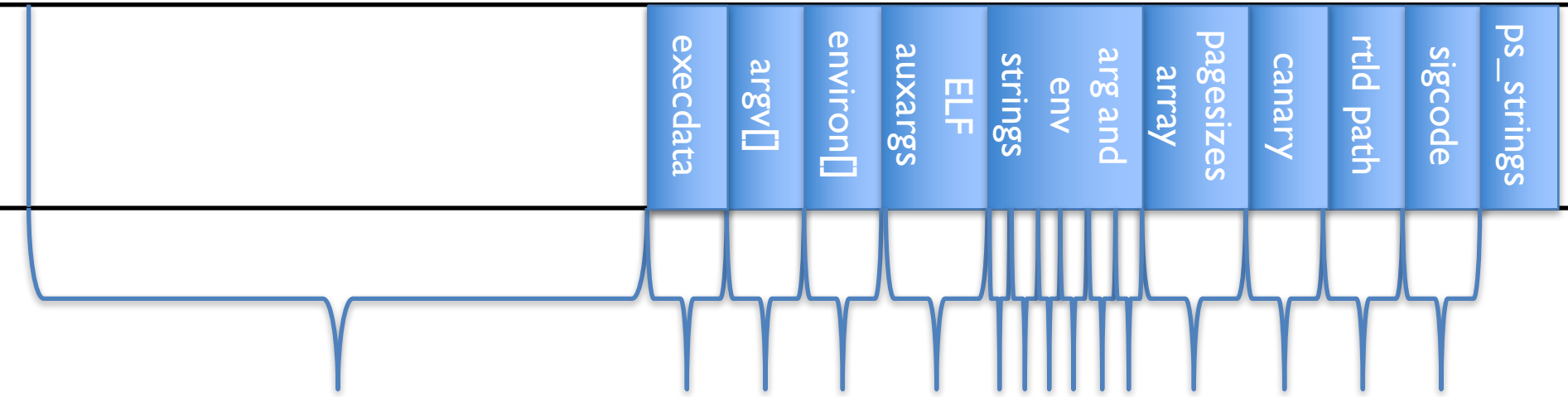

```
_start(char **ap, …) {
    …
      argc = * (long *) ap;
      argv = ap + 1;
      env  = ap + 2 + argc;
    …
```

```
_init_tls() {
    …
    sp = (Elf_Addr *) environ;
    while (*sp++ != 0)
        ;
    aux = (Elf_Auxinfo *) sp;
```

CheriABI process layout



CheriABI stack



- Stack capability excludes strings, etc
- Remove ps_strings
- Capabilities for each object
- argc replaced with cheriabi_execdata
- Capability to execdata passed to _start as argument

```
struct cheriabi_execdata {
        size_t      ce_len;
        int         ce_argc;
        char        **ce_argv;
        char        **ce_envp;
        struct cheriabi_auxarg *ce_auxargs;
        struct ps_strings *ce_ps_strings;
};
```

mmap() in CheriABI

- The mmap() syscall allocates address space
  - Is a direct and indirect source of capabilities
- mmap() also replaces mappings
  - Changes the contents of capabilities
    - Takes capability argument covering page(s)
    - Will require capability permission bit
  - May upgrade permissions on capabilities!

mmap() alignment

- Compressed capabilities require alignment constraints to ensure representability
- New alignment constraints
  - MAP_ALIGNED_CHERI
  - MAP_ALIGNED_CHERI_SEAL
- Flags constrain alignment and assert that the length is representable
  - Usable in CheriABI and native code
- MAP_CHERI_NOSETBOUNDS allows unrepresentable subregions to be mapped within reservation

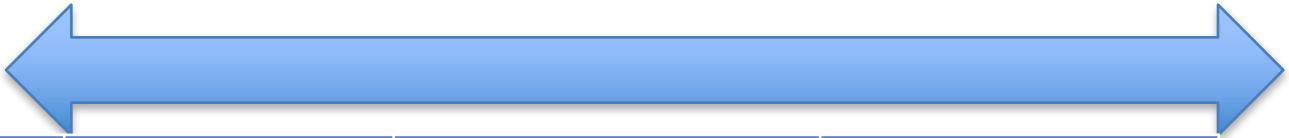
mmap() continued

- Pattern of PROT_NONE reservation followed by mapping sub regions requires two changes:
  - Reservation must have desired permissions for capability followed by mprotect() to remove permissions
  - Use the MAP_CHERI_NOSETBOUNDS flag to avoid potentially unrepresentable sub-regions
- New APIs required

CheriABI syscall translation challenges

- Pointers and integers differ in registers
  - Different registers on MIPS
  - Similar issues with converged register files
- Pointers may point to structures containing pointers – similar to 32-bit/64-bit compat
  - Translation required
- Variadic system calls (fcntl, ioctl, open, etc)
- Multiplexed system calls (ioctl, _umtx)

System-call argument translation spectrum



	Pointer Translation	Capability validation	Capabilities in kernel
Kernel acts via	Virtual address	Virtual address	Capability
Kernel implements user memory model	-	tag, length*, perms	all
Issues	Capabilities bypassed	Annotations must be correct. Subject to confused deputy attacks.*	Need to translate arguments from non-capability ABIs.

* No length on strings or length not bound at time of check

Syscall argument translation vs validation

Translation

```
tag = cheri_gettag(argcap);
if (tag)
        ptr = cheri_ctoprt(argcap, kdc);
else
        ptr = cheri_getbase(argcap) +
            cheri_getoffset(argcap);
```

Validation

```
if (!cheri_gettag(argcap))
        return (EPROT);
if (cheri_getsealed(argcap))
        return (EPROT);
if (cheri_getperms(argcap) &
    reqperms) != reqperms)
        return (EPROT);
if (cheri_getlen(argcap) –
    cheri_getoffset(argcap)  < reqlen)
        return (EPROT);
ptr = cheri_ctoptr(argcap, kdc);
```

Syscall annotations drive validation

```
int open(_In_z_ char *path, int flags, int mode);

int cheriabi_readv(int fd, _Inout_updates_(iovcnt)
    struct iovec_c *iovp, u_int iovcnt);

int minherit(_Pagerange_(len) void *addr,
    size_t len, int inherit);

int accept(int s,
    _Out_writes_bytes_(*anamelen)
    struct sockaddr *name,
    _Inout_opt_ socklen_t *anamelen);
```

CheriABI results

- CheriBSD tests run and largely pass
- Variety of applications work without modification
  - From echo to ssh
  - Spatial memory safety for "free"
- Progressing toward multi threaded, dynamically linked applications
  - Using modern, thread-aware malloc()
  - Starting work on runtime linker

CheriABI conclusions

- CheriABI runs (mostly) unmodified POSIX applications with memory safety and CFI
- We are exploring transition paths:
  - From a monolithic, capability-oblivious kernel to a capability-aware kernel.
  - From traditional C to memory-safe C programming environment

BACKUP SLIDES

Kernel Changes

Component	File	Lines +	Lines -
Headers	19	1424	11
CHERI initialization	2	49	4
Context management	2	392	10
Exception handling	3	574	90
Memory copying	2	122	0
Virtual memory	5	398	27
Object capabilities	2	883	0
System calls	2	76	0
CheriABI	6	2855	0
Signal delivery	3	327	71
Process monitoring/debugging	3	298	0
Kernel debugger	2	264	0