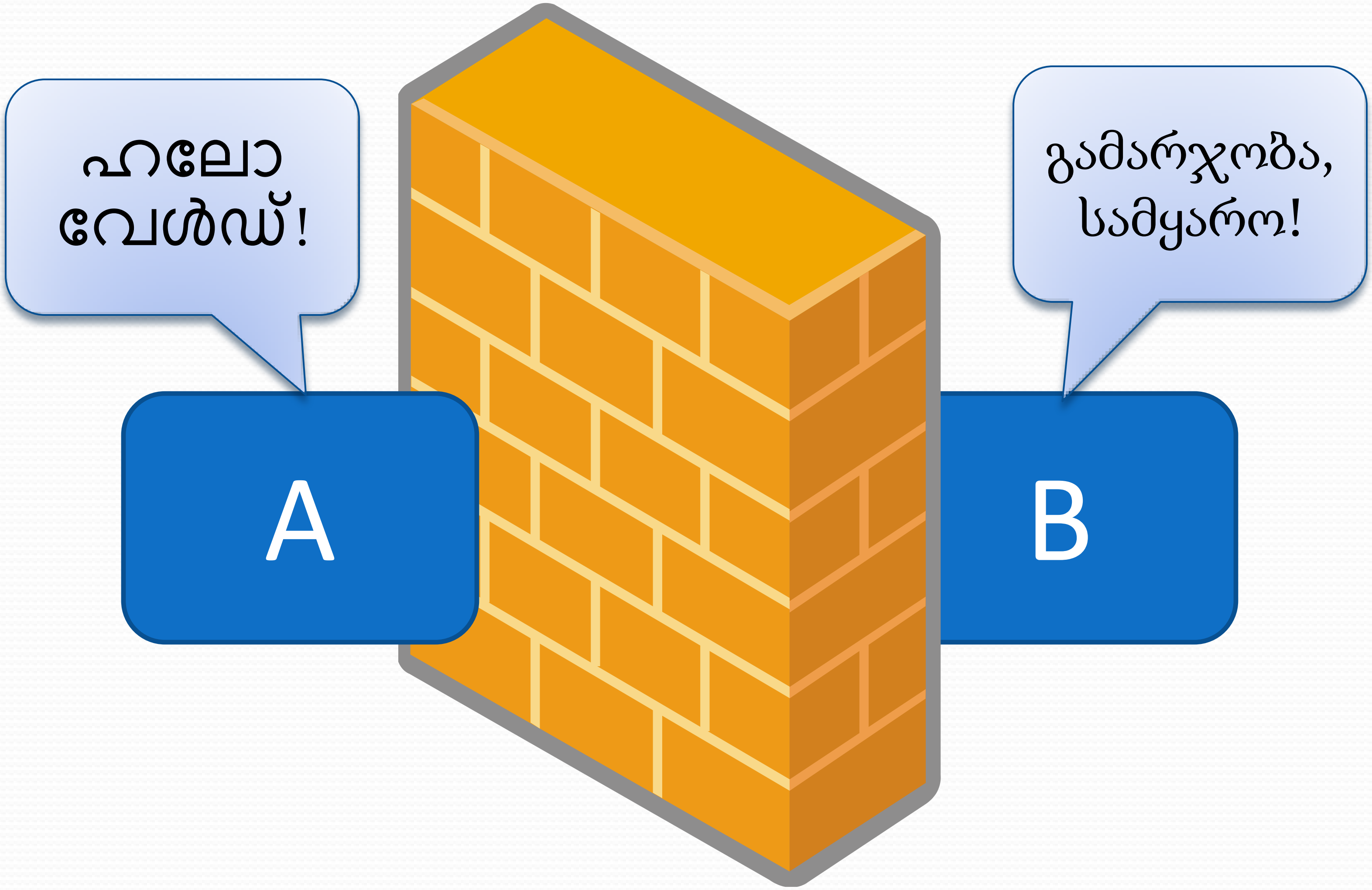
Chapter 17: Another Level of Indirection

李羽修

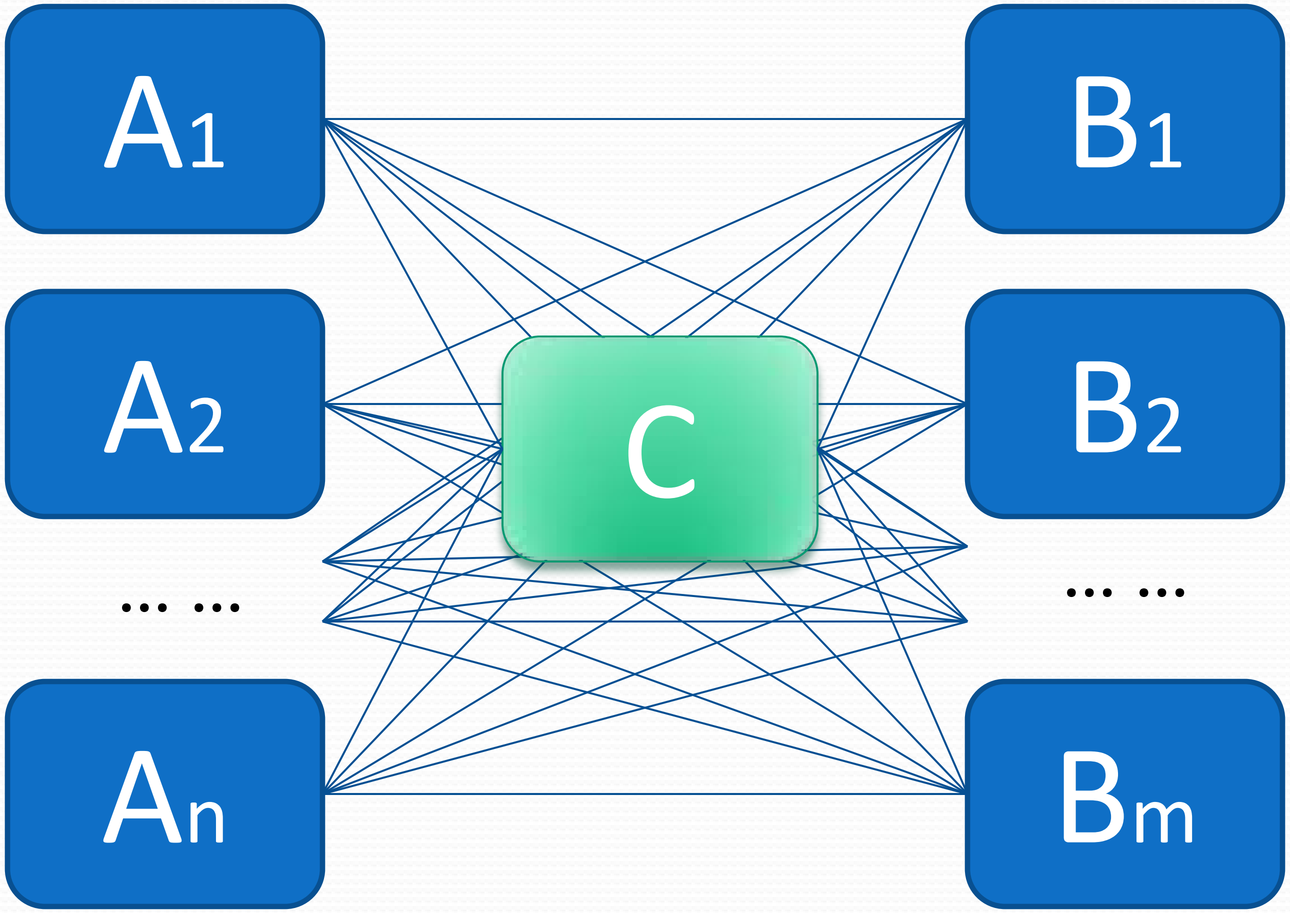
“All problems in computer science can be solved by another level of indirection.”

-- David Wheeler

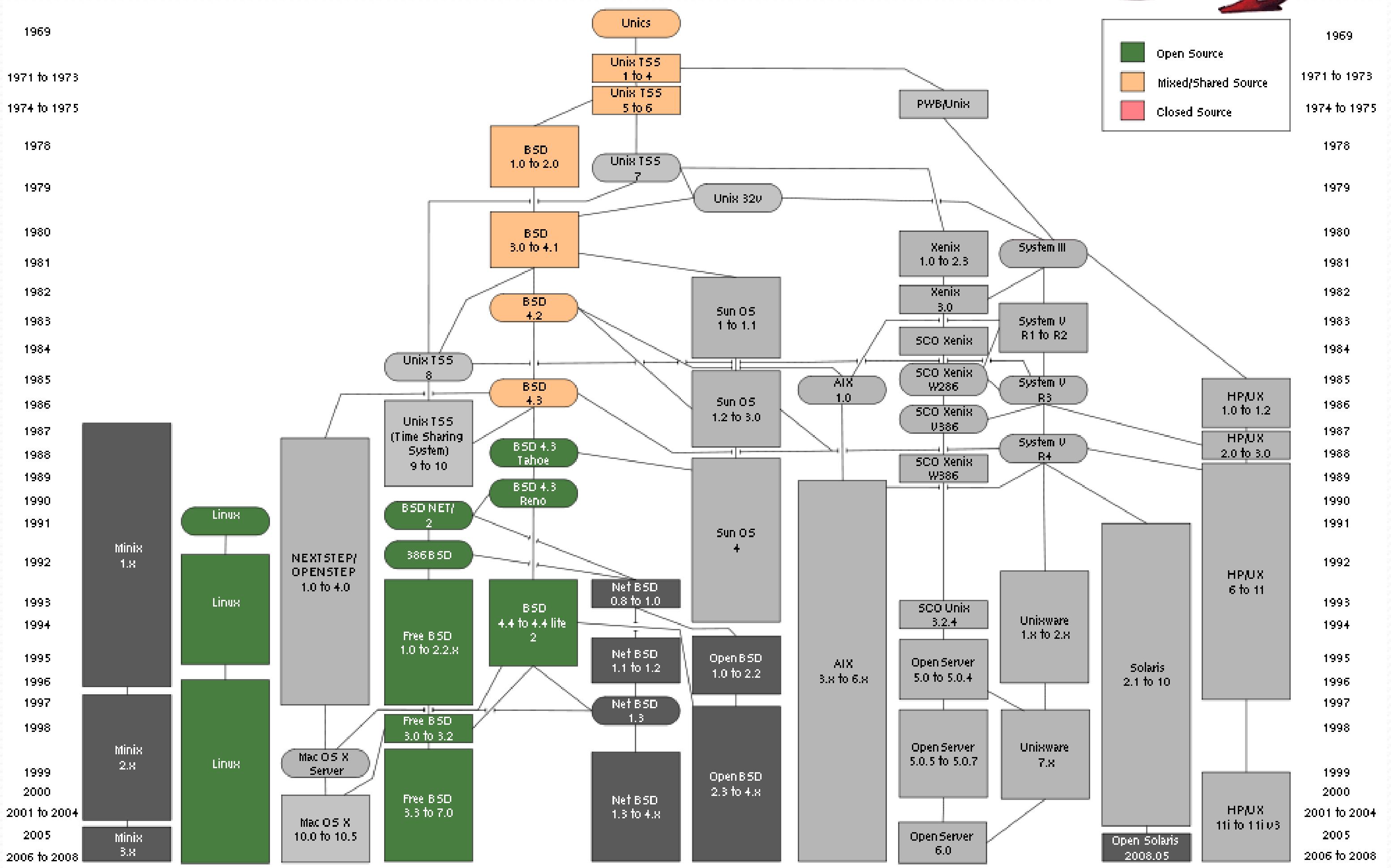
What is indirection?



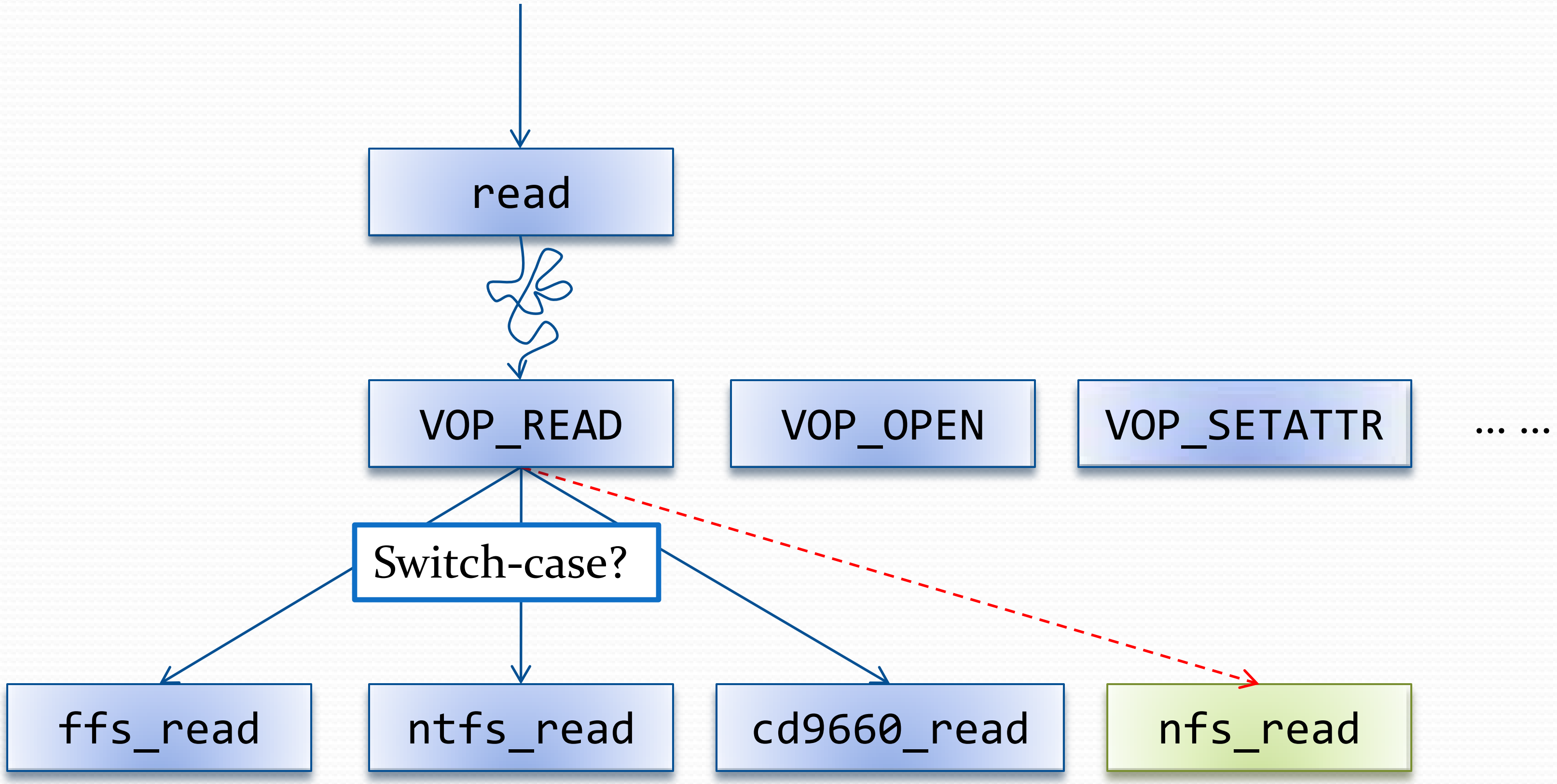
What is indirection?



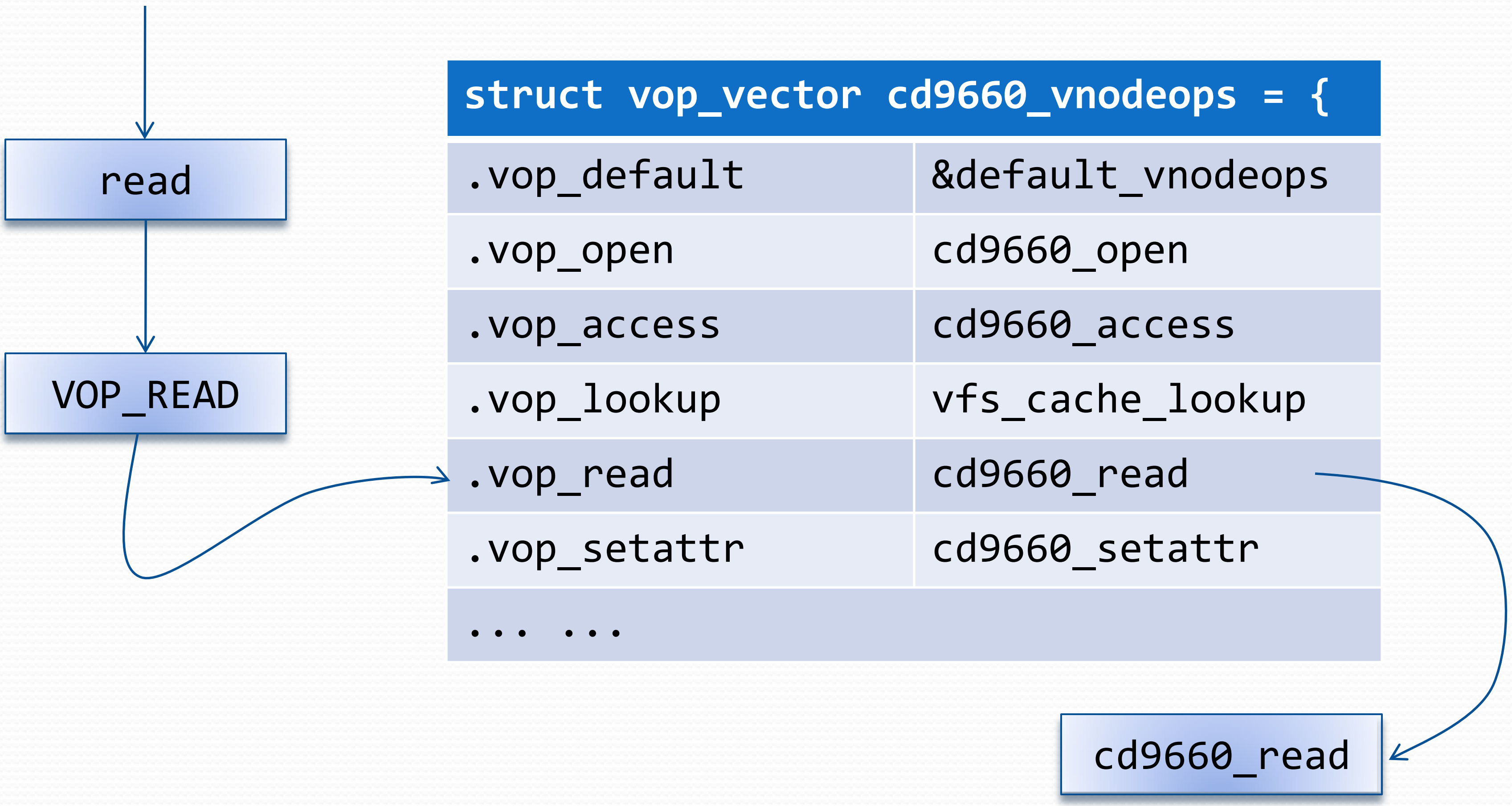
The story of FreeBSD



Handling Various File Systems



A Dynamic Way



struct vop_vector cd9660_vnodeops = {	
.vop_default	&default_vnodeops
.vop_open	cd9660_open
.vop_access	cd9660_access
.vop_lookup	vfs_cache_lookup
.vop_read	cd9660_read
.vop_setattr	cd9660_setattr
... ...	

Layered File System

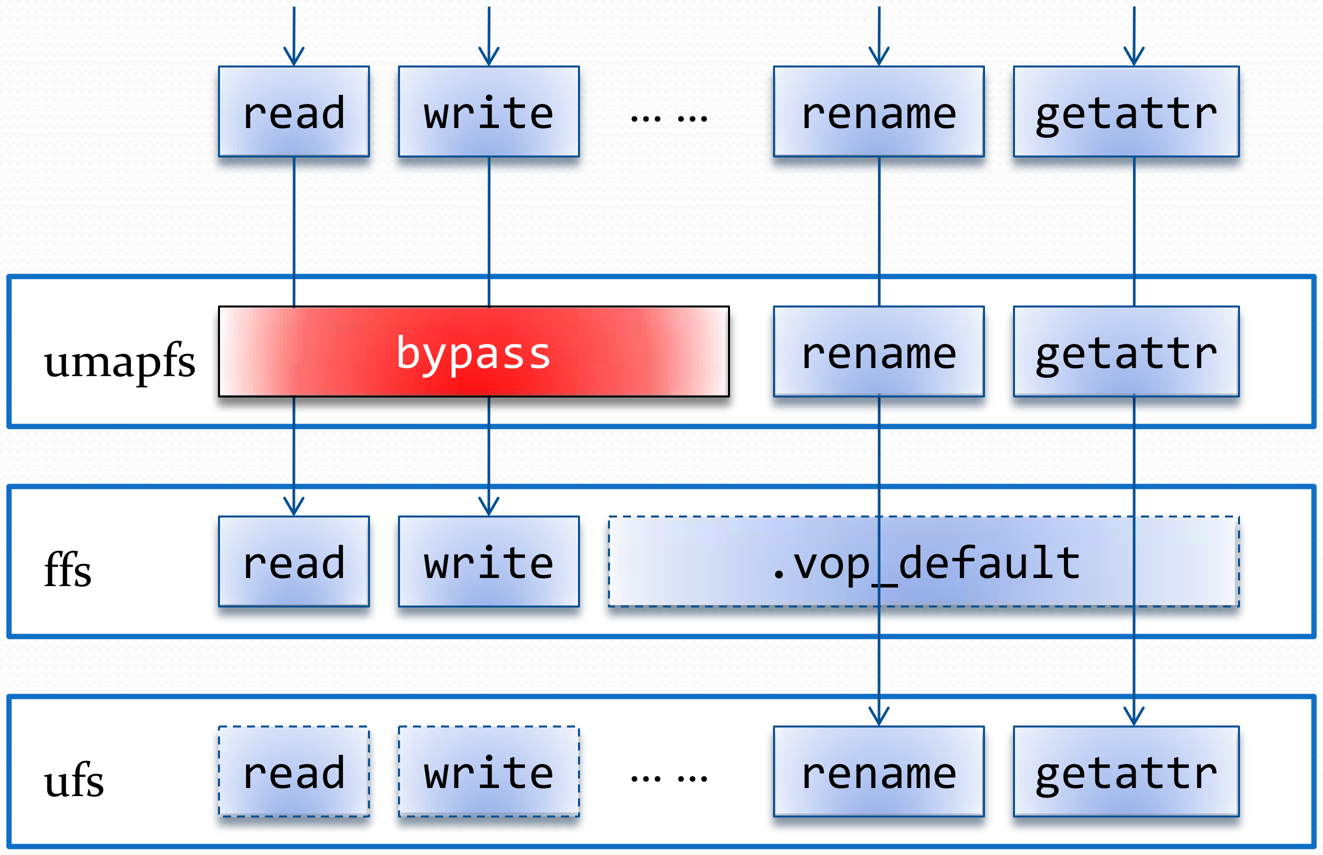
… …

umapfs

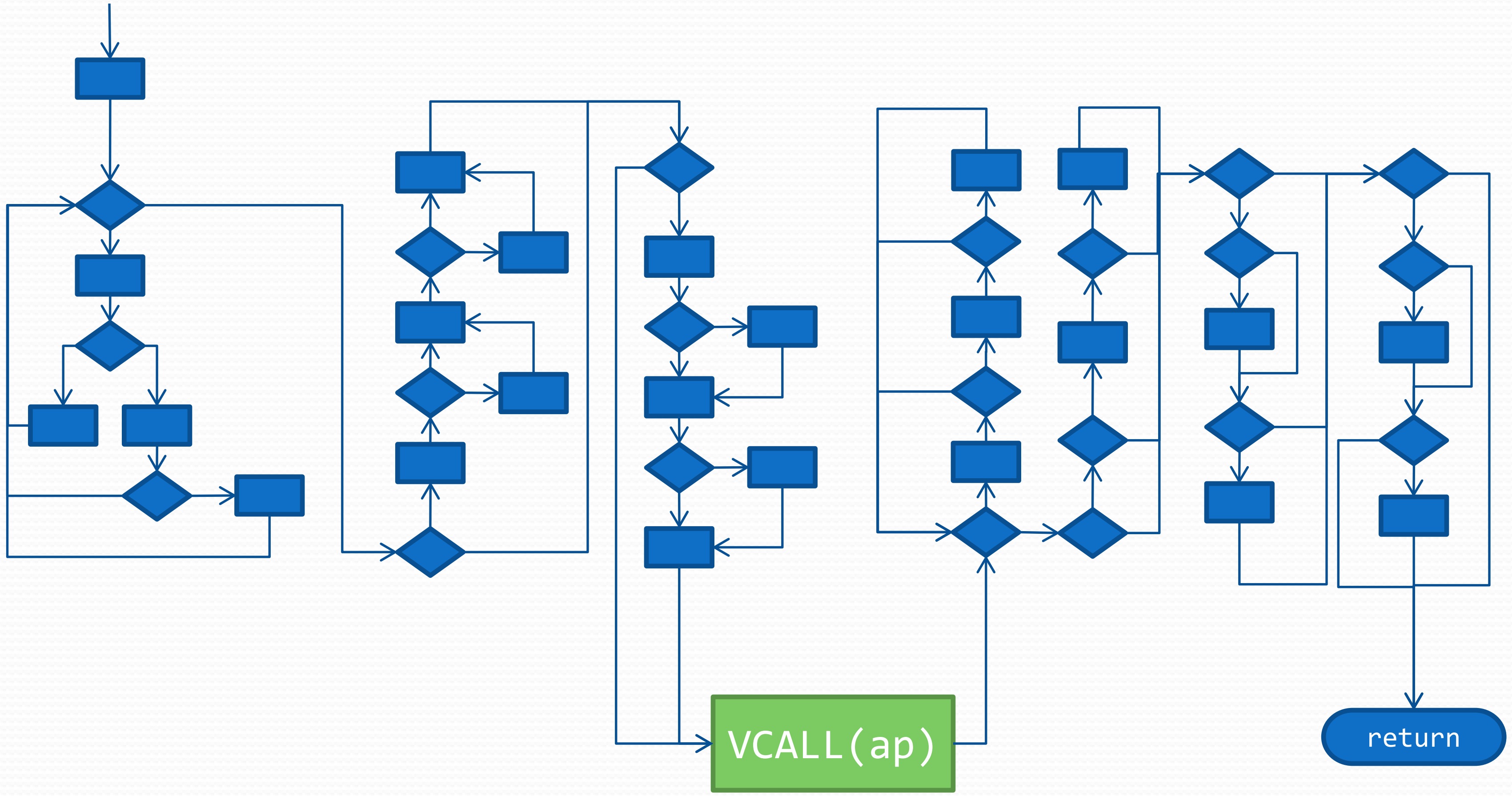
ffs

ufs (BSD 4.2)

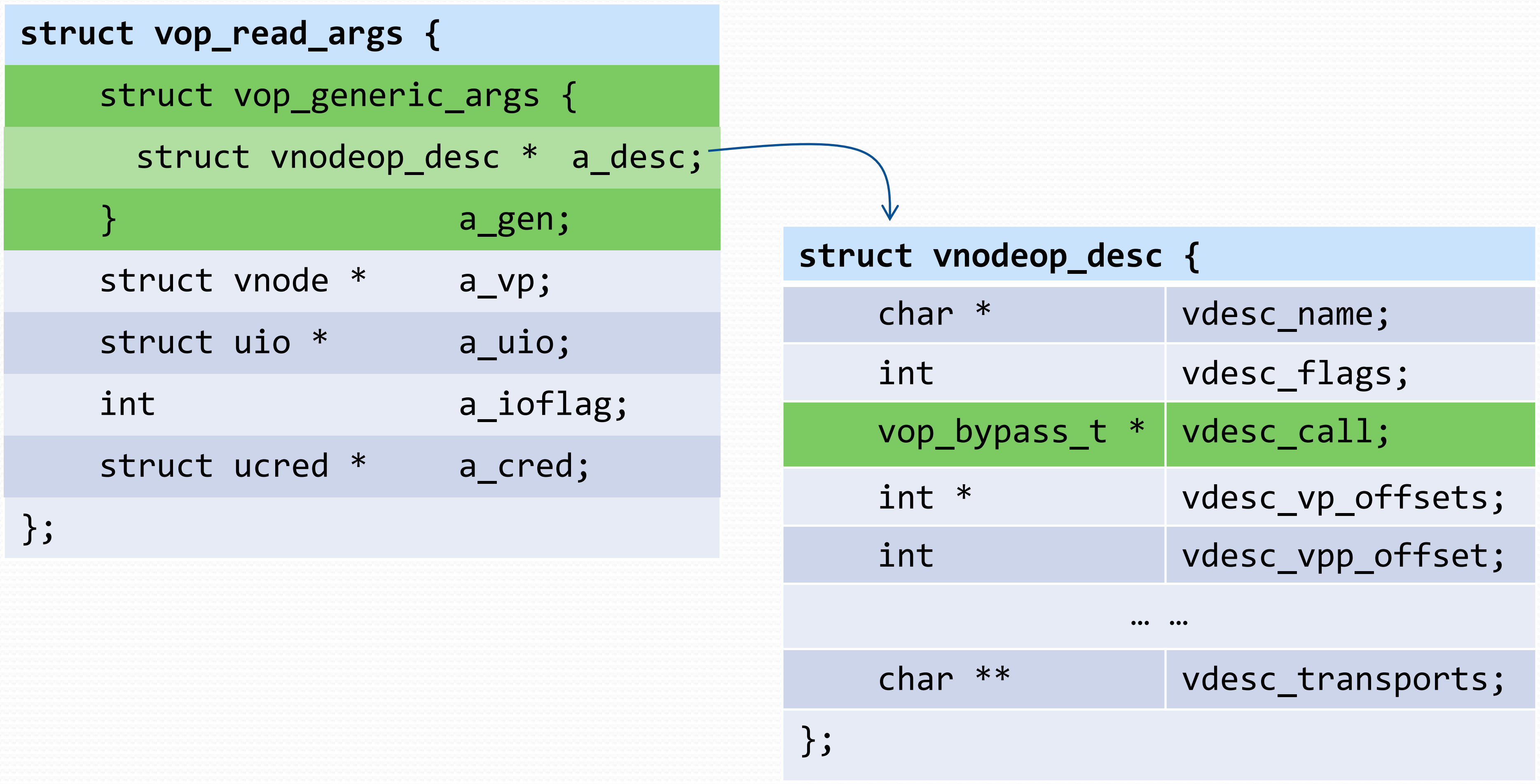
Bypassing Calls



What Does ‘bypass’ Do



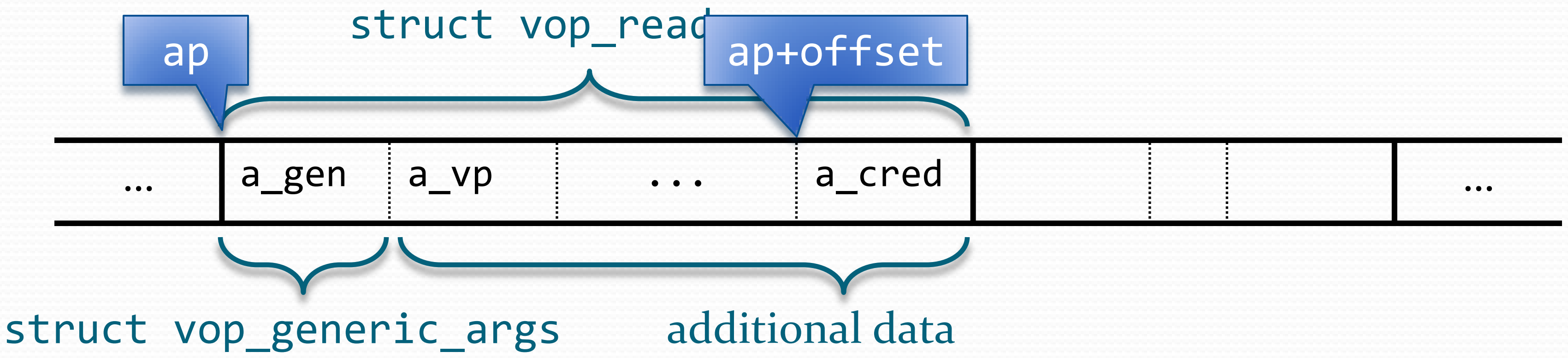
How The Arguments Are Packed



```
struct vop_read_args {
    struct vop_generic_args {
      struct vnodeop_desc *  a_desc;
    }                        a_gen;
    struct vnode *           a_vp;
    struct uio *             a_uio;
    int                      a_ioflag;
    struct ucred *           a_cred;
};
```

```
struct vnodeop_desc {
    char *           vdesc_name;
    int              vdesc_flags;
    vop_bypass_t *   vdesc_call;
    int *            vdesc_vp_offsets;
    int              vdesc_vpp_offset;
                 … …
    char **          vdesc_transports;
};
```

The Offset Trick



```
struct vop_read_args *a;
// ...
vop_bypass((struct vop_generic_args *) a);
```

How The Calls Are Dispatched

```
int VOP_READ_APV(struct vop_vector *vop, struct vop_read_args *a) {
    // [...]
    // Drill down the filesystem layers to find one
    // that implements the function or a bypass
    while (vop != NULL && vop->vop_read == NULL && vop->vop_bypass == NULL)
        vop = vop->vop_default;

    // [...]
    // Call the function or the bypass
    if (vop->vop_read != NULL)
        rc = vop->vop_read(a);
    else
        rc = vop->vop_bypass(&a->a_gen);

    // [...]
```

How The Calls Are Dispatched

```
static __inline int VOP_READ(struct vnode *vp, struct uio *uio,
                             int ioflag, struct ucred *cred) {
    struct vop_read_args a;

    a.a_gen.a_desc = &vop_read_desc;
    a.a_vp = vp;
    a.a_uio = uio;
    a.a_ioflag = ioflag;
    a.a_cred = cred;

    return (VOP_READ_APV(vp->v_op, &a));
}
```

How The Calls Are Dispatched

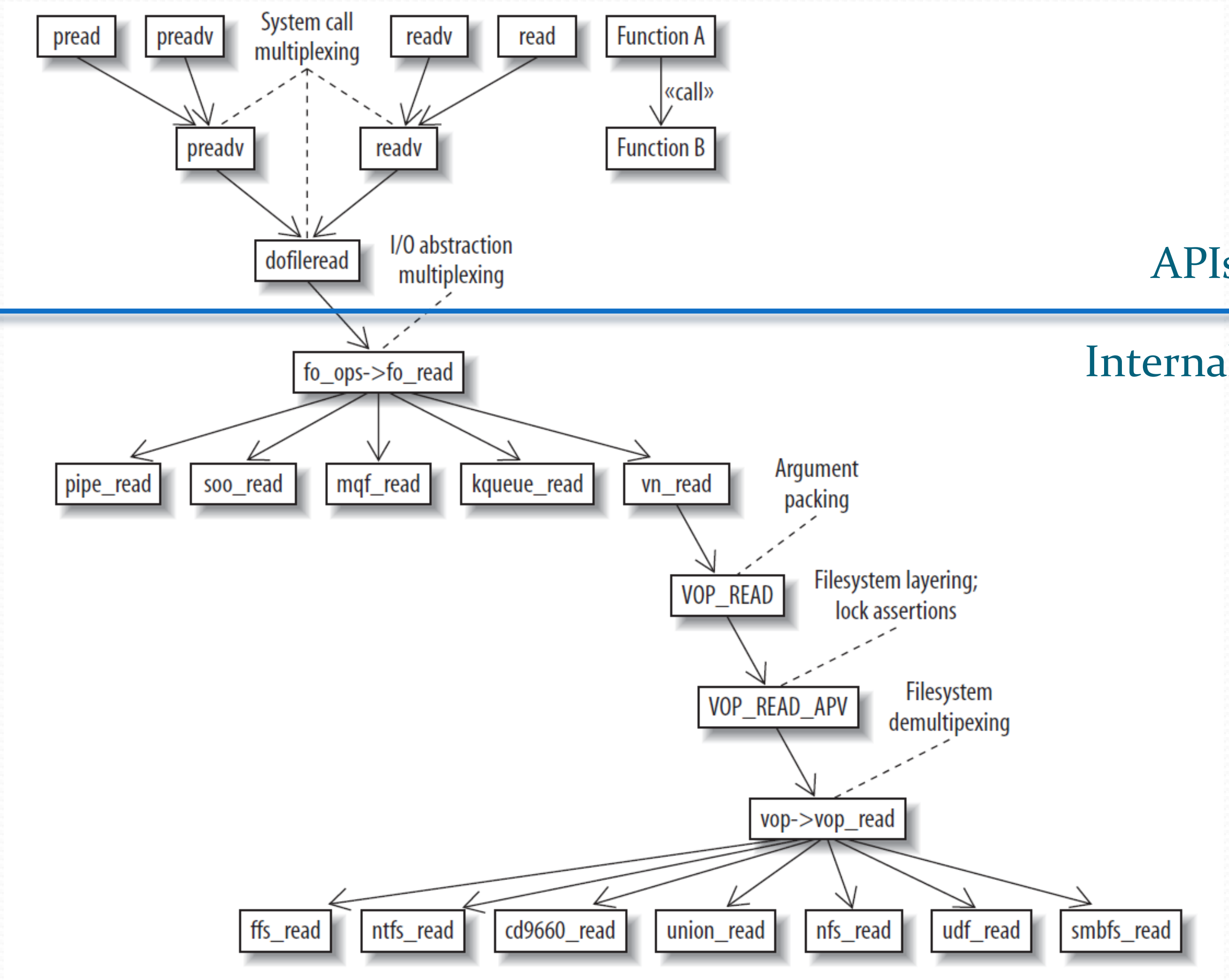
```
struct vnodeop_desc vop_read_desc = {
    "vop_read",
    0,
    (void*)(uintptr_t)VOP_READ_AP,
    vop_read_vp_offsets,
    VDESC_NO_OFFSET,
    VOPARG_OFFSETOF(struct vop_read_args,a_cred),
    VDESC_NO_OFFSET,
    VDESC_NO_OFFSET,
    NULL,
};
```

Eventually We Have To Duplicate

- Generate C code from Domain-Specific Language.

```
#
#% read vp L L L
#
vop_read {
    IN struct vnode *vp;
    INOUT struct uio *uio;
    IN int ioflag;
    IN struct ucred *cred;
};
```

More Indirections Upstream



“All problems in computer science can be solved by another level of indirection. But that usually will create another problem.”

-- David Wheeler

More Resources

- FreeBSD Code Repository (stable-6)
  http://svn.freebsd.org/viewvc/base/stable/6/
- ufs & ffs: /sys/ufs
  Read '*_vnops.c'
- umapfs, ntfs & nullfs: /sys/fs
- DSL Related:
  /sys/tools/vnode_if.awk /sys/kern/vnode_if.src [-c|-h|-p|-q]

Q & A

- Why not auto-generate `bypass_read`, `bypass_write`, etc?

Thank You All!

(Or I probably should thank the country first.)