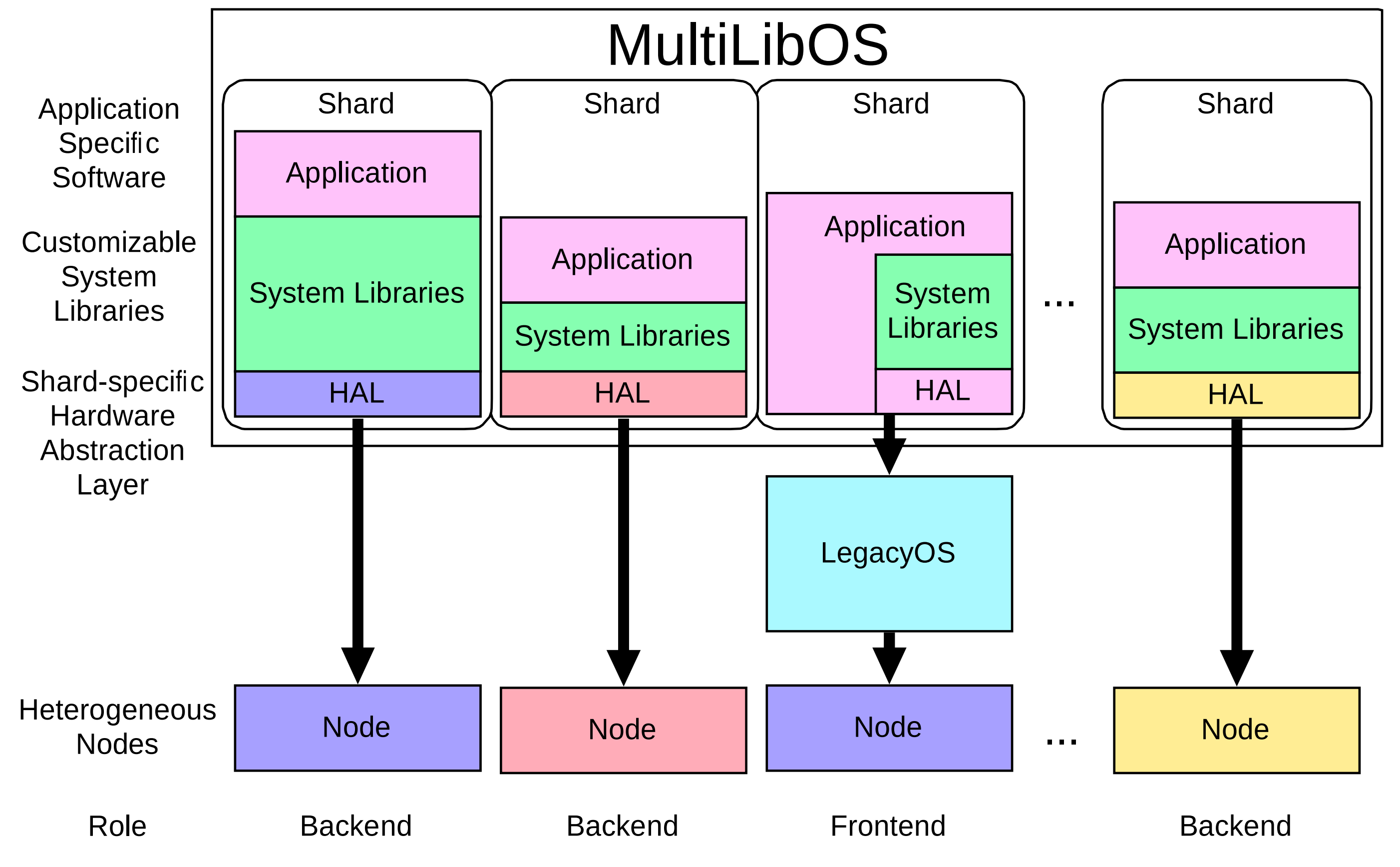
A LibraryOS for Cloud Computing

James Cadden, Dan Schatzberg, Orran Krieger, Jonathan Appavoo

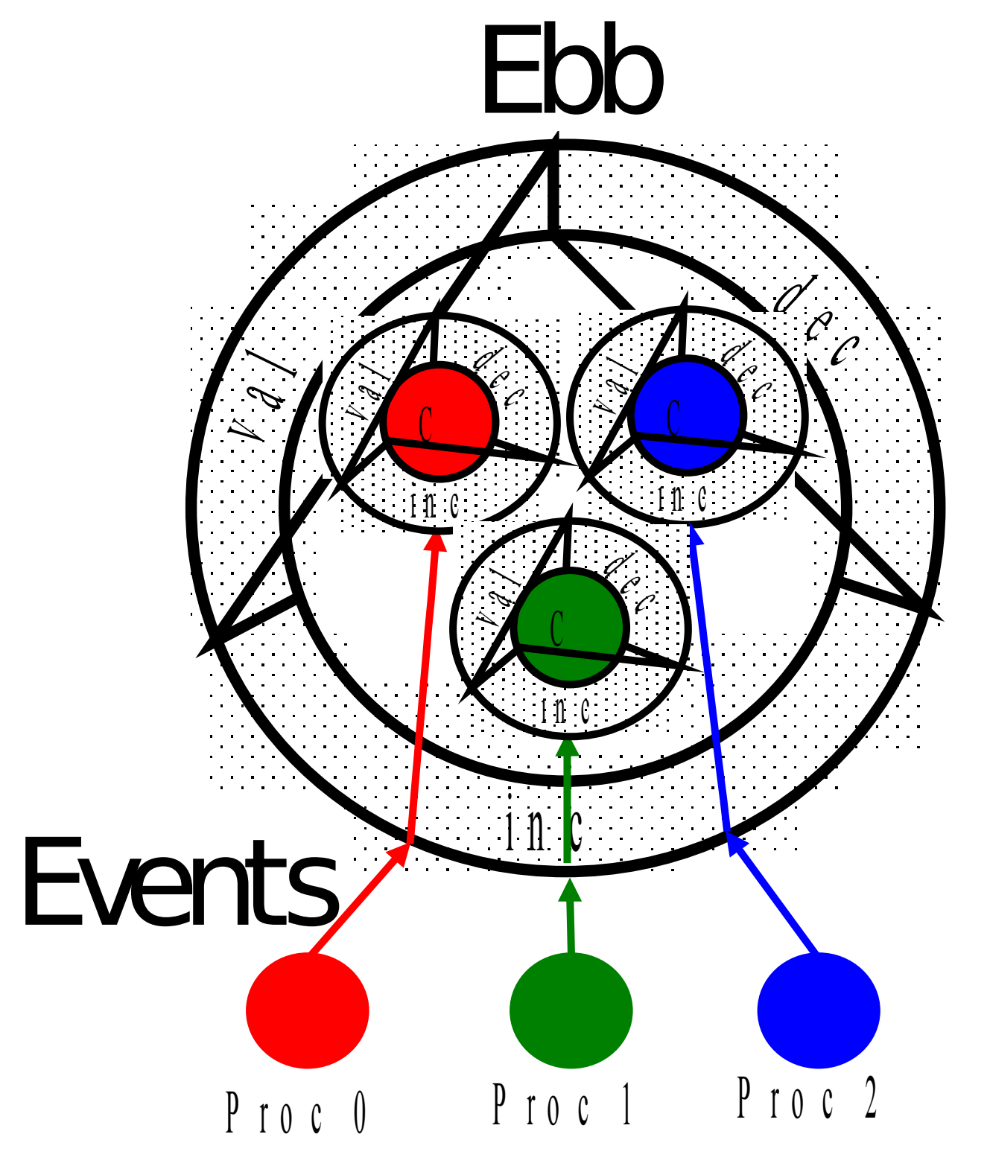
Computer Science Department, Boston University

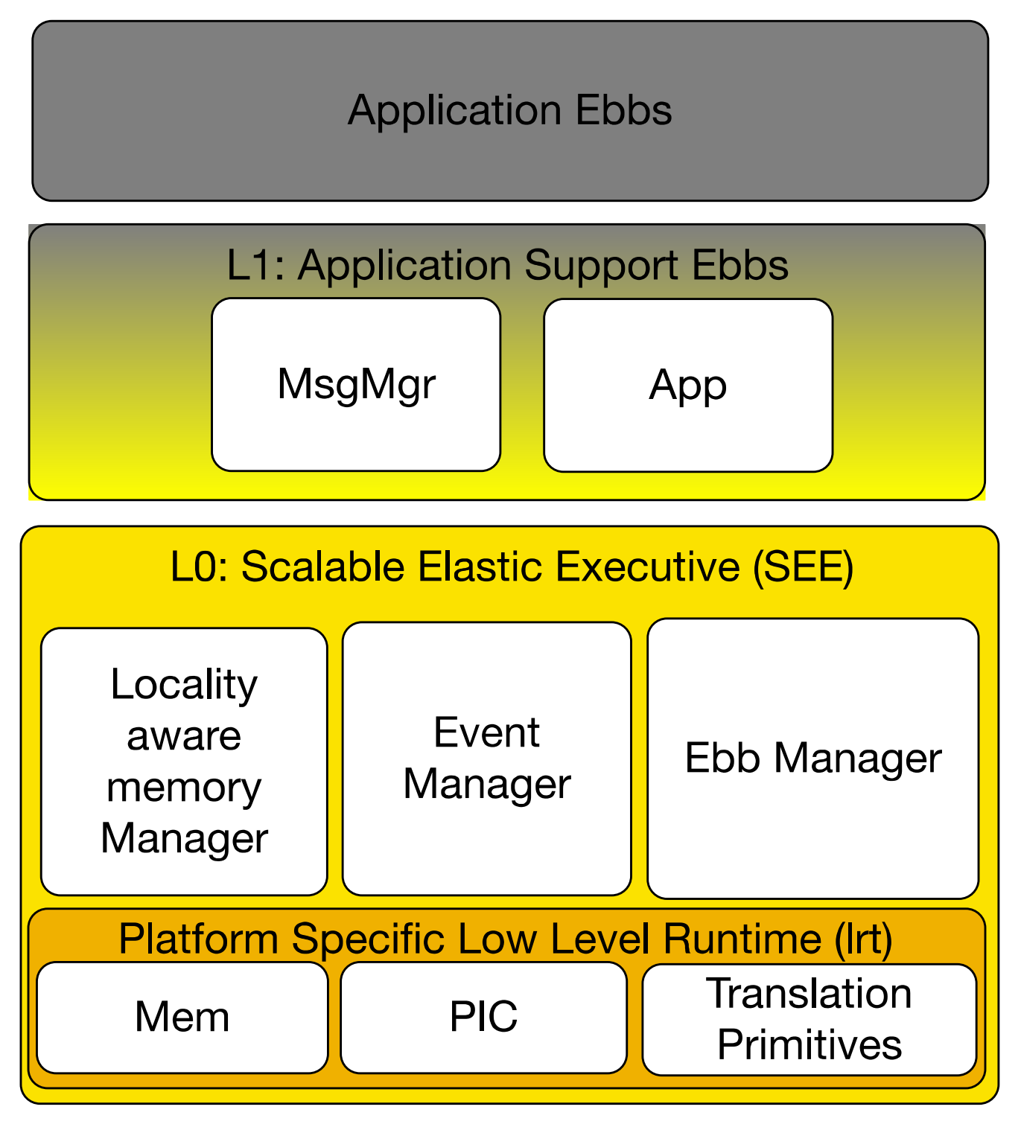
MultiLibOS: A **single-tenant, single-process** distributed OS composed of library OS instances that run across many heterogeneous nodes.

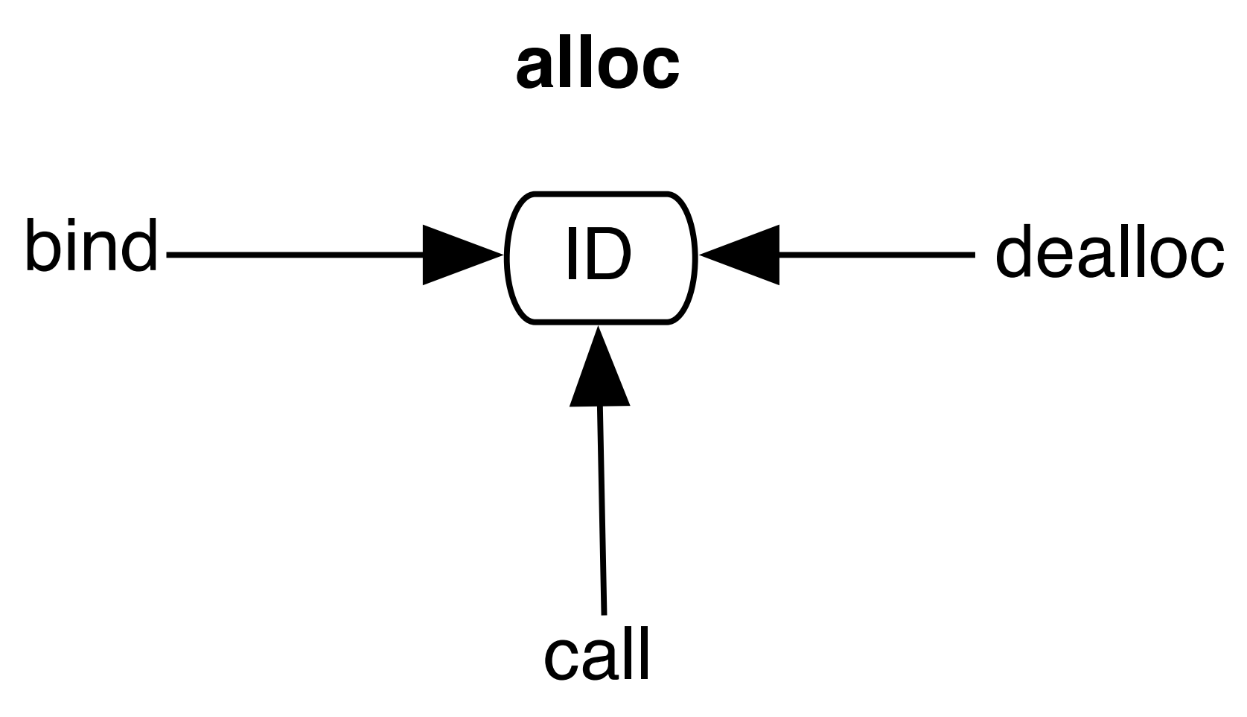

MultiLibOS
Application Specific Software
Customizable System Libraries
Shard-specific Hardware Abstraction Layer
Shard
Application
System Libraries
HAL
...
LegacyOS
Heterogeneous Nodes
Node
Role
Backend
Frontend

- MultiLibOS combines legacy OS compatibility with highly-optimized efficiency on hardware
- Asymmetric 'library' framework distributes single application across many heterogeneous nodes
- No need for traditional OS level functionality on majority of nodes

EbbOS: A MultiLibOS on which applications are constructed as Elastic Building Blocks. EbbOS combines **distributed system objects** and an **event driven programming** infrastructure.




Application Ebbs
L1: Application Support Ebbs
MsgMgr
App
L0: Scalable Elastic Executive (SEE)
Locality aware memory Manager
Event Manager
Ebb Manager
Platform Specific Low Level Runtime (lrt)
Mem
PIC
Translation Primitives



- Software is constructed as a collection Ebb instances that are bound to Ebb identifiers (ID's).
- Ebb instances may have a distributed implementation, with multiple "reps" local to different nodes and cores.
- Indirection is exploited to enable elastic behavior to be programmable.

Find us on GitHub! https://github.com/SESA/EBBlib

Elastic building block (Ebb) the core programming abstraction

Classic Object-Like Interface

```
COBJ_EBBType(EBBCtr) {
  EBBRC (*inc)  (EBBCtrRef _self);
  EBBRC (*dec)  (EBBCtrRef _self);
  EBBRC (*val)  (EBBCtrRef _self, uintptr_t *v);
};
```

Internals defined as represent objects that combine interface and data members

```
CObject(EBBCtrDistributed) {
  COBJ_EBBFuncTbl(EBBCtr);
  uintptr_t localValue;
  CObjEBBRootMultiRef theRoot;
};
```

Methods can be invoked via function **call** or direct target of an **event** (see bottom)

```
static EBBRC
EBBCtrDistributed_inc(EBBCtrRef _self)
{
  EBBCtrDistributedRef self = (EBBCtrDistributedRef)_self;
  __sync_fetch_and_add(&self->localValue,1);
  return EBBRC_OK;
}
```

Internals of object may be decomposed into a dynamic set of distributed representatives. Methods may operate across the set.

```
static EBBRC
EBBCtrDistributed_val(EBBCtrRef _self, uintptr_t *v)
{
  EBBCtrDistributedRef self = (EBBCtrDistributedRef)_self;
  uintptr_t val = 0;
  RepListNode *node;
  EBBRep *rep = NULL;
  for (node = self->theRoot->ft->nextRep(self->theRoot, 0, &rep);
       node;
       node = self->theRoot->ft->nextRep(self->theRoot, node, &rep)) {
    val += ((EBBCtrDistributedRef)rep)->localValue;
  }
  *v = val;
  return EBBRC_OK;
}
```

Every Ebb specifies behavior for lazy/dynamic creation on first access in a new location combining with event bind is key to elastic programming.

```
static EBBRep *
EBBCtrDistributed_createRep(CObjEBBRootMultiRef _self) {
  EBBCtrDistributedRef repRef;
  EBBMalloc(sizeof(EBBCtrDistributed), &repRef, EBB_MEM_DEFAULT);
  EBBCtrDistributedSetFT(repRef);
  repRef->theRoot = _self;
  repRef->localValue = 0;
  return (EBBRep *)repRef;
}
```

EBBAllocId: Allocates an id (initially bound to NULLEbb) CObjEBBBind: Binds an allocated id to instance

```
EBBRC
EBBCtrDistributedCreate(EBBCtrId *id)
{
  CObjEBBRootMultiImpRef rootRef;
  CObjEBBRootMultiImpCreate(&rootRef, EBBCtrDistributed_createRep);
  EBBAllocId((EBBId *)id);
  CObjEBBBind((EBBId)*id, rootRef);
  return EBBRC_OK;
}
```

Call can trigger elastic/lazy behavior

```
EBBCtrDistributedCreate(&cid);
EBBCALL(cid, inc);
```

Ebb Id and method can be bound directly to an Event

```
EBBCALL(theEvtMgrId, allocEventNo, &en);
EBBCALL(theEvtMgrId, bindEvent, en, cid, FUNCNUM(cid, inc));
```