Cooperative Data Sharing (CDS)

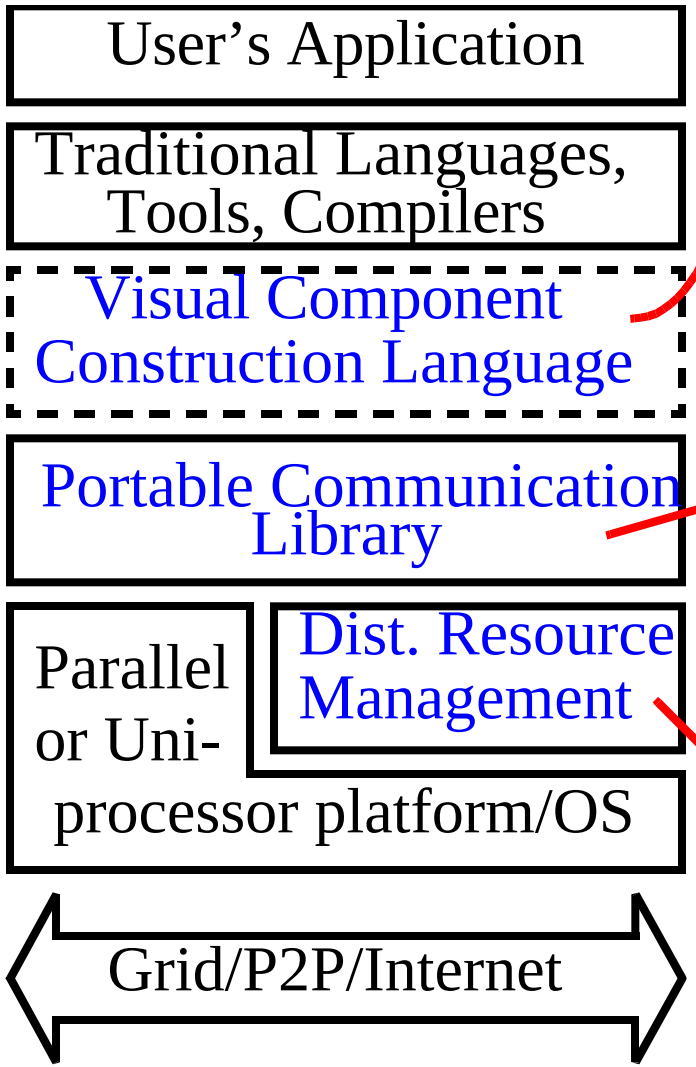
Overview

David DiNucci, PhD

Elepar: Working Together Independently

`dave@elepar.com` `www.elepar.com`

Elepar's Three Layers



Software Cabling (SC): Visual OO component coordination methodology for building & analyzing portable, distributable apps from modules implemented in traditional langs

Cooperative Data Sharing (CDS): Efficiently supports messaging (push), blackboard/shared (pull), & hybrid styles on variety of architectures

People, Instruments, Computers, and Archives (PICA): Rules and protocols for finding, bargaining for, and scheduling distributed, independently-controlled resources of all kinds

CDS: Cooperative Data Sharing

History: **Early musings at OGI, prototype implemented and published at NASA Ames, now under development at Elepar**

Approach: **Determine common features of shared memory and message passing, build subroutine interface around those features, include other expected features (process control, active messages, conversion/marshalling).**

Result: **Compared to other communication layers (e.g. MPI, sockets, DSM), it is:**

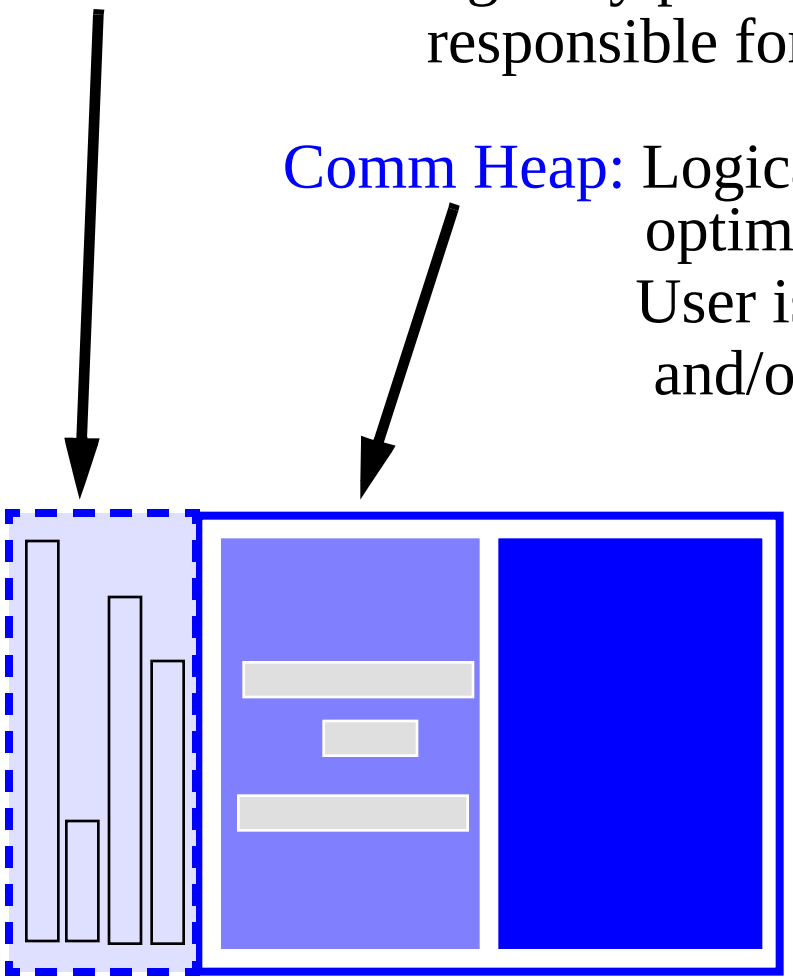
- **Relatively simple/Small**
- **Expressive/Powerful**
- **Very portable to different uniprocessor & parallel architectures**

CDS: Anatomy of a CDS Process

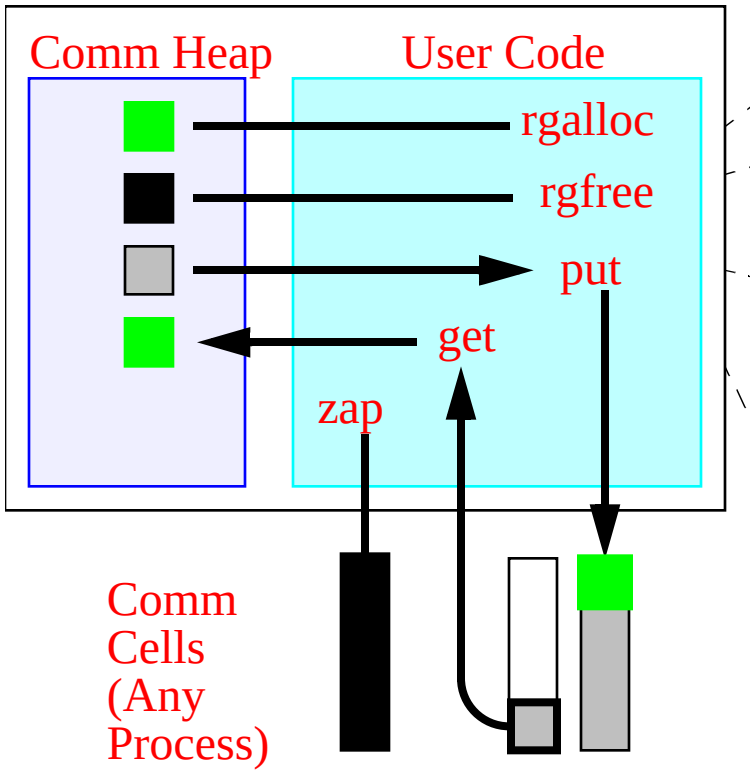
Comm Cells: Logically public set of queues. User is responsible for creating and deleting.

Comm Heap: Logically private heap. Data is optimized for communication. User is responsible for enlarging and/or shrinking.

User code & data: Standard Unix process.



CDS Basic Communication Operations



Allocates a region in the local comm heap

Frees up a region in the local comm heap

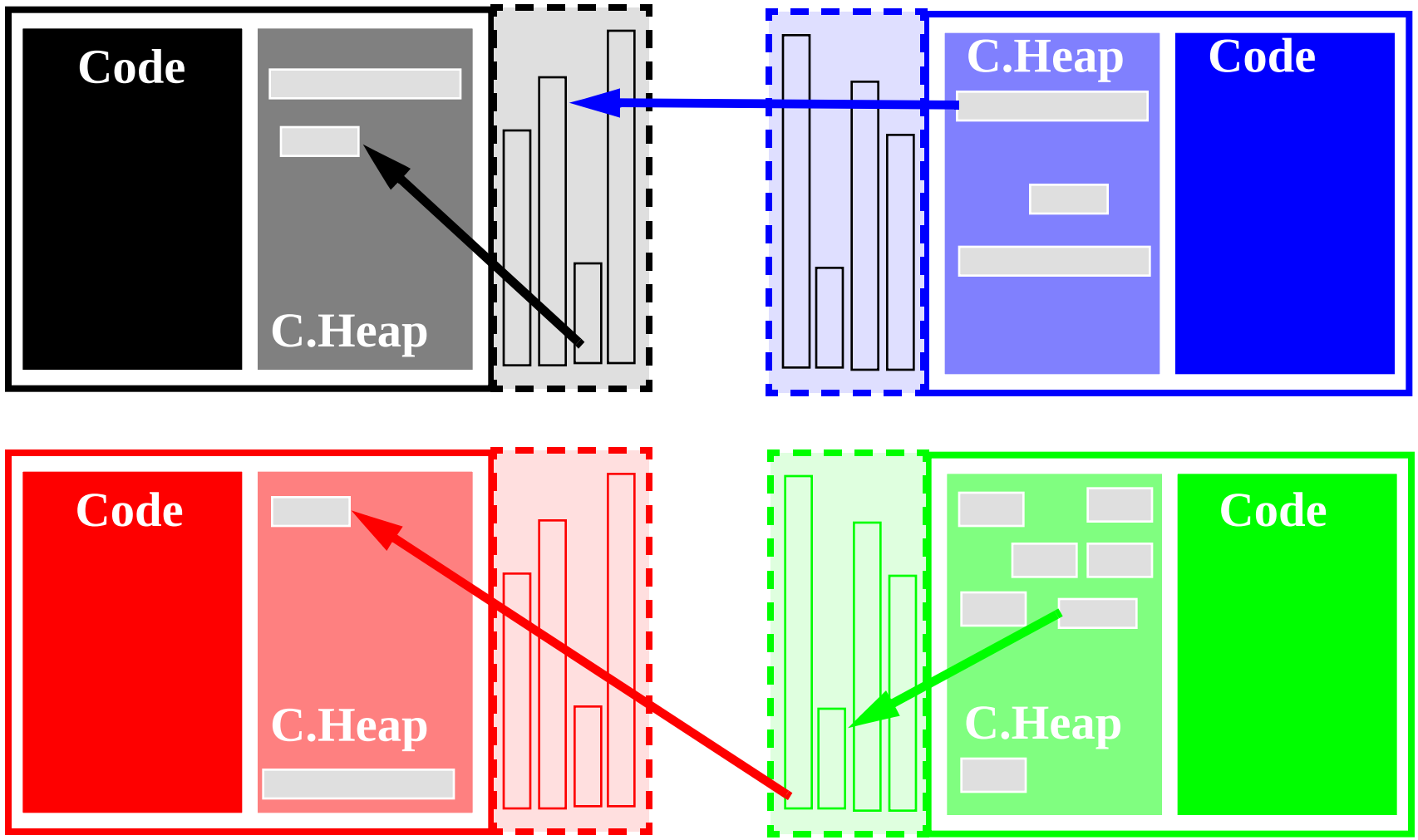
"Copies" region from local comm heap to end of any cell, optionally freeing region from heap and/or zapping cell before depositing new region. AKA "write" if cell zapped, "enq" if not. bput same, but blocks until cell empty and a get is waiting

"Copies" region from beginning of any cell to local comm heap, optionally removing it from cell after. AKA "deq" if removed, "read" if not.

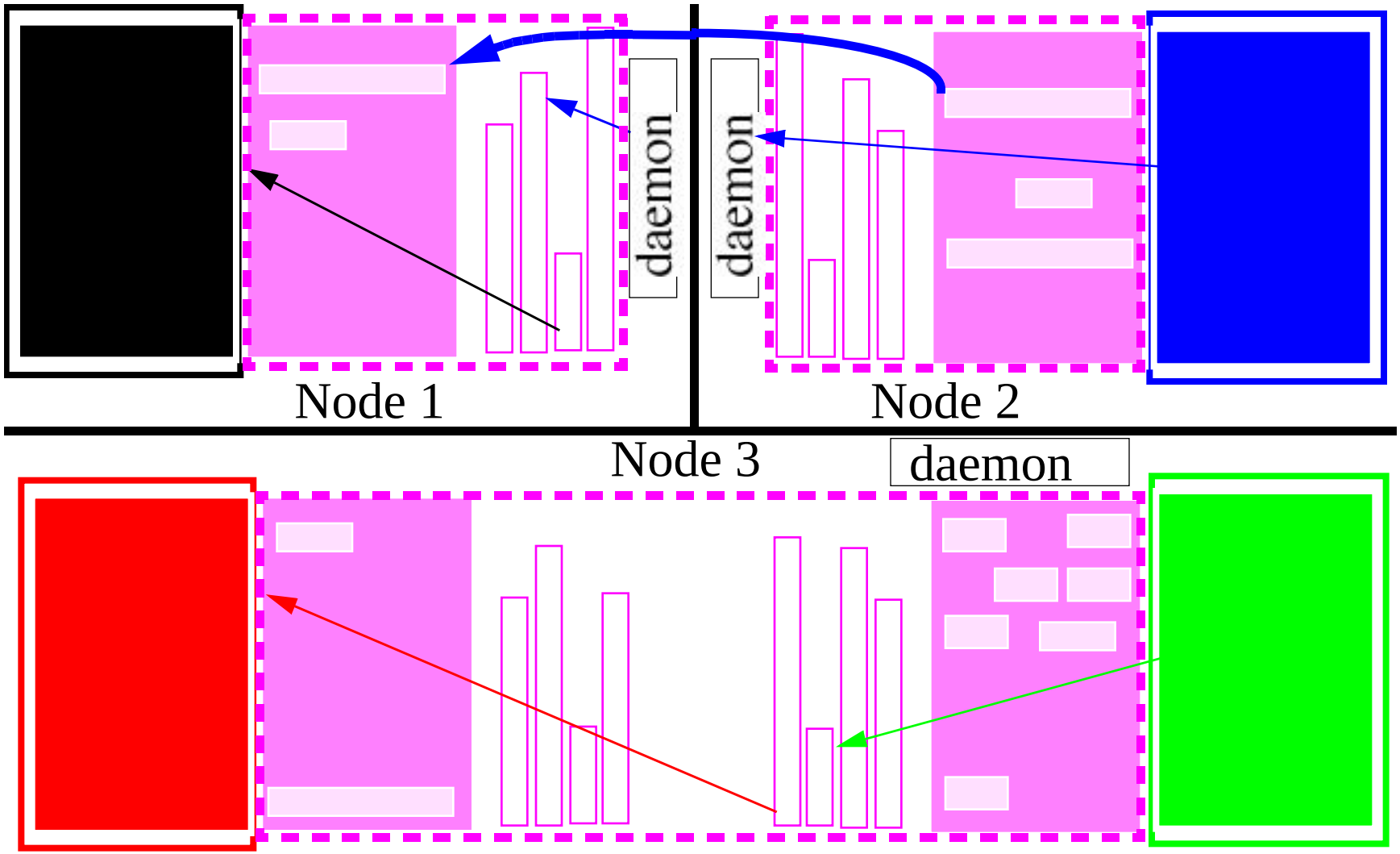
All ops that can block (i.e. `bput`, `get`, `deq` and `read`) take a time-out value, and also "i" versions (`ibput`, `iget`, `ideq`, and `iread`, respectively), resolved with a `wait` op.

"Copy" operation is virtual (i.e. usually copy on write), so these are usually just pointer ops. For portability, `rgmod` must be called before modifying any potentially-shared rgn.

CDS: Logical View



CDS: Physical View on 3 Nodes



CDS: Other Functionality

Process Initiation/Active Messages ("Handlers")

High and low water marks can be set on each cell

A "handler" function can be chosed to be invoked each time that watermark is exceeded.

Copying/Marshalling/Conversion

Although process can access regions in comm heap directly, "`copyfm`", "`copyto`" routines exist to pack, unpack, and/or convert data as it is being moved to or from region, based on internally-supported conversion tables.

CDS Shared Mem & Msg Passing "Macros"

enqing reqion ~= releasing a lock, deqing region ~= acquiring a lock.

"Macro"	Meaning	Translates Into
`acqwl`	Acquire write lock	`deq, rgmod`
`rlswl`	Release write lock	`write, rgfree`
`acqrl`	Acquire read lock	`read`
`rlsrl`	Release read lock	`rgfree`
`wl2rl`	Write lock -> read lock	`write`

Msg passing includes copy to/from comm heap, can be optimized out.

"Macro"	Meaning	Semantically identical to
`send`	Send message	`rgalloc, copyto, enq, rgfree`
`recv`	Receive message	`deq, copyfm, rgfree`
`sendx`	Destructive send	`rgalloc, copyto, write, rgfree`
`recvx`	Non-destructive receive	`read, copyfm, rgfree`
`bsend`	Synchro or ready send	`rgalloc, copyto, bput, rgfree`

Corresponding "i" ops: `iacqrl`, `iacqwl`, `irecv`, `irecvx`, `ibsend`

Comparing CDS Featureset

Features	CDS	DSM	MPI	SOCK	LINDA
Some data can be traded/shared in place (true 0 copy!)	x	x			
Consumer can pull (get) data from passive producer	x	x	2		x
Consumer can prefetch/prepull data to hide latency	x	?	2		
Producer can push (send) data to passive consumer	x		x	x	?
Data can be queued at producer waiting for pull	x		x	x	?
Pushed data can be made to overwrite previous value	x	x			x
Producer can retain access rights to comm'd data	x		2		x
Producer can relinq access rights to comm'd data	x	x	x		x
Dynamic memory allocation for shared memory	x	?			
Consumer can specify timeout for waiting	x	?			
Supports heterogeneous platforms	x		x		
Simplicity (~number of function + macro interfaces)	51	20	!!!	13	5

The CDS Interface

Managing comm heap and contexts/cells

`rgalloc rgmod rgfree rgsize rgrealloc`

`addcntxt delcntxt grwcntxt`

Communication Primitives

`read deq benq enq write zap enqm writem`

`iread ideq ibenq wait waitm ienqm benqm`

Copying and Translation

`copyto copyfm copytofm transtab`

Composite functions (shared mem and msg passing)

`recv bsend recvx send sendx sendm sendxm`

`acqrl acqwl rlsrl rlswl wl2rl`

`irecv ibsend irecvx iacqrl iacqwl`

Process and thread control

`enlist init myinfo hdlr prior`