# CDS1 Conclusion

Other functionality:

- Data translation, for heterogeneous processing
- Dynamic process creation
- Message Handlers with some simple thread mgmt

Current Status:

- Prototype implemented on Davinci cluster, better performance than optimized MPI on shared-bus in some cases (due to lack of copy).
- Currently unfunded. If not re-funded, looking for a private company or standards committee to adopt.

# Comparison of Semantic Options

| | MPI | Linda | RK | (D)SM | CDS |
|---|---|---|---|---|---|
| Non-destructive write (e.g. enq) | X | X | X | | X |
| Destructive write (i.e. overwrite) | | | | X | X |
| Destructive read (e.g. deq) | X | X | X | | X |
| Non-destructive read (i.e. read) | | X | | X | X |
| Keep copy of comm’d data | X | X | | | X |
| Don’t ““““ | | | X | X | X |
| Identify consumer | X | * | X | | X |
| Don’t ““““ | | X | | X | X |
| Identify producer | X | * | | | X |
| Don’t ““““ | X | X | X | X | X |

*Linda expects consumer and/or producer to be recognized, in some cases, by globally pre-processing source code

# CDS: Cooperative Data Sharing

**Objective:** Provide a single, simple, and efficient interface that allows the user to specify the semantics required of each communication, so that it can run as efficiently as possible on the architecture available

**Approach:** Provide two layers of API

- CDS1 - Kernel level. Objectives are minimality, orthogonality, portability, efficiency, utility.
- CDS2 - User level. Objective is a “nice” user interface.

# Architecture-Independent Models?

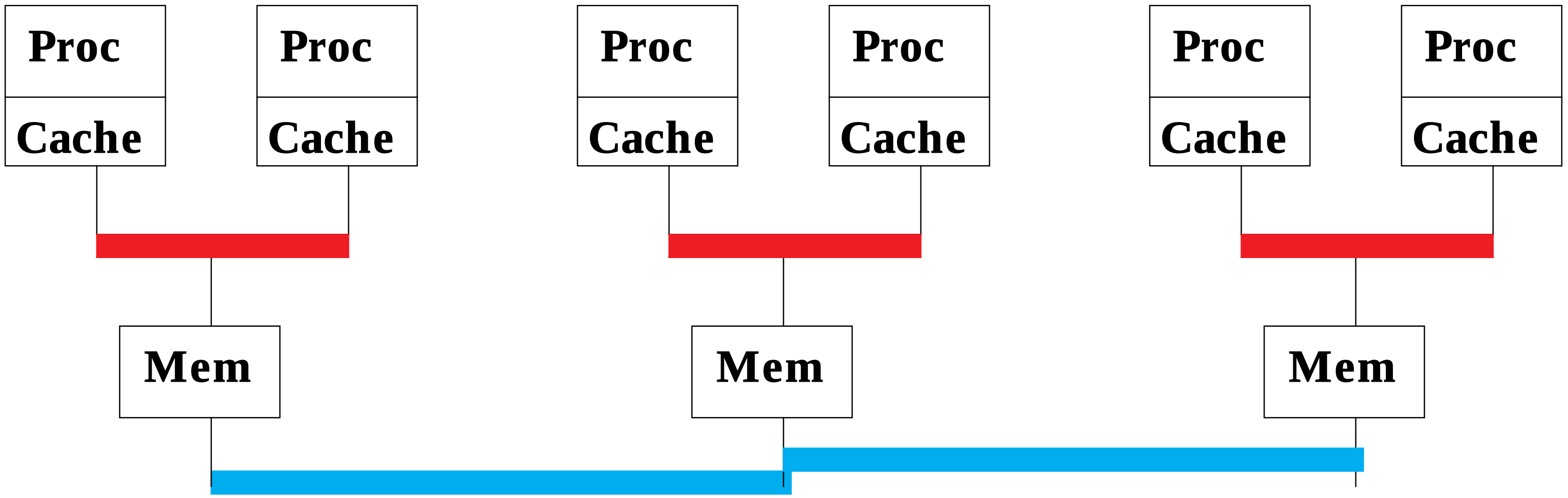
Examples: Linda, RK, Distributed Shared Memory

Each dictates pre-determined answers to one or more of these questions:

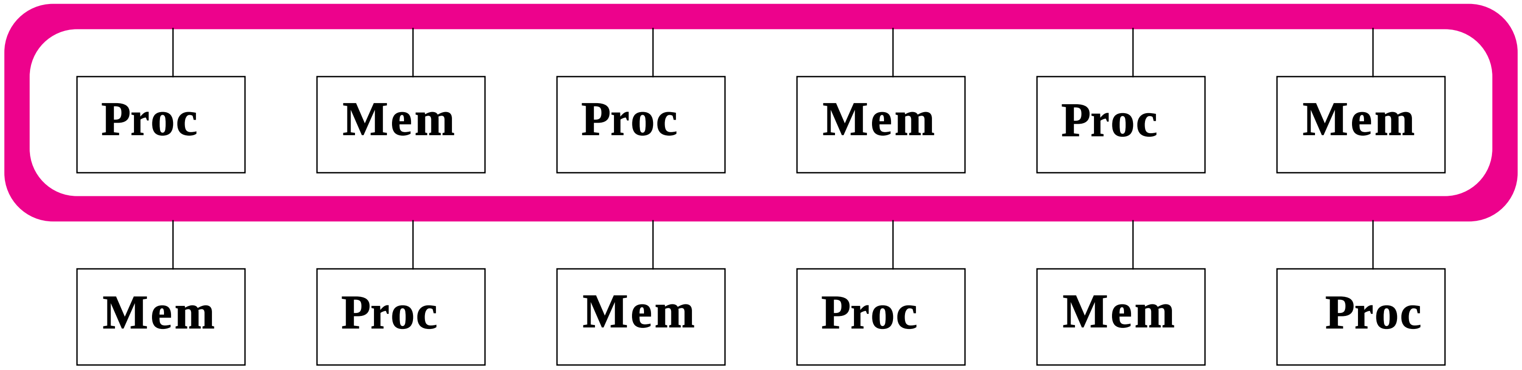
- Does communication require that a copy be made?
- Must the “producer” know the “consumer”?
- Must the “consumer” know the “producer”?
- Does newly produced data over-write older data (or is the data collected)?
- Does consumed data remain to be re-consumed (or is the data destroyed when consumed)?

These should all be up to the app, not the model or language!

# Parallel Architecture 102: Hybrids



OR



# Use of Shared-Memory Semantics

Shared-Memory semantics are usually used for Shared-bus architectures because:

- No data movement is required, only coordination

Shared-Memory semantics are usually not used for distributed-memory architectures because:

- Cannot move data toward next processor before it is requested (to hide latency), even if previous process knows where it will be needed next
- Requests are always made in small granularity, so multiple requests must be made to move much data, and each experiences latency of interconnect twice

# Use of Message-Passing Semantics

Distributed Memory architectures are often programmed with Message-Passing semantics because:

- Copying data to local memory decreases costly accesses over slow interconnect
- Initiation of copy by source before destination needs data decreases lag caused by interconnect latency

Shared-Bus architectures are often not programmed with Message-Passing semantics because:

- Copying data serves no purpose in many cases, just increases latency, decreases bandwidth
- Initiating copy before both sides are ready requires buffering, which serves no purpose in many cases

# Parallel Programming Semantics 101

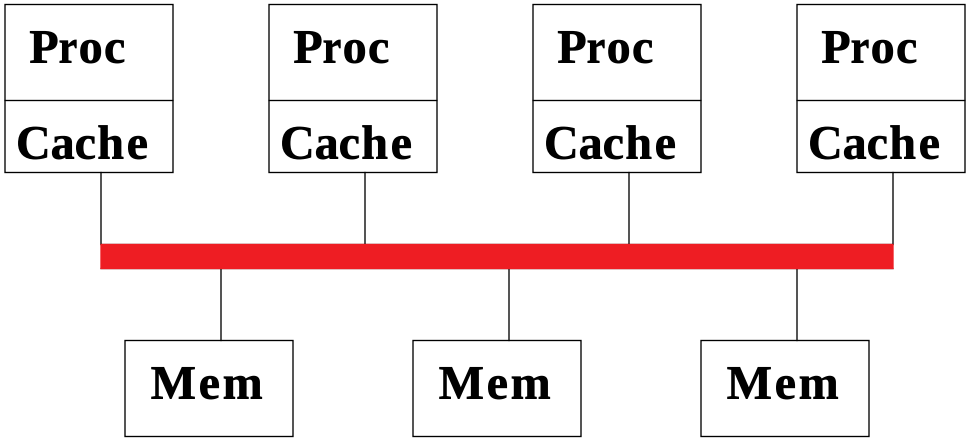
## Message-Passing (aka Distributed Copying)

- Data is copied, usually between processes.
- Each process specifies one address -- i.e. the source or destination, usually in its address space
- Any necessary synchronization is performed automatically, by buffering data and/or delaying copy in either source or destination process

## Shared-Memory (aka Remote Memory Access)

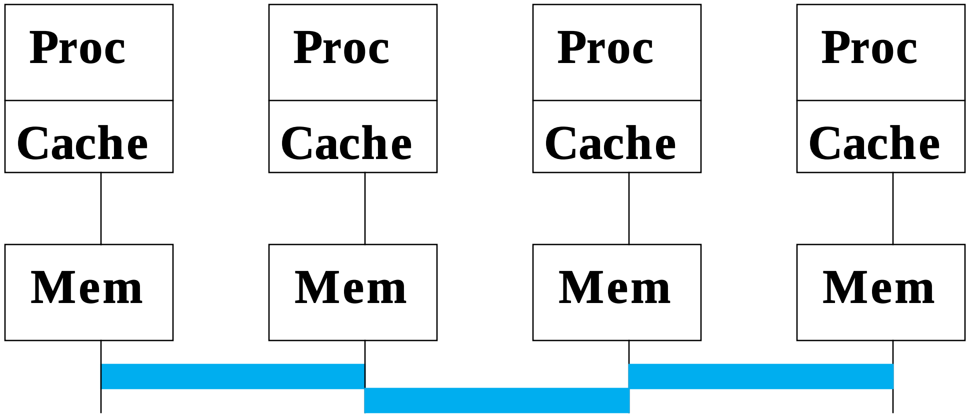
- Data (which may be regarded as residing outside of a process's address space) is accessed in situ
- Synchronization primitives help individual processes coordinate access

# Parallel Architectures 101



## Shared-Bus (aka SMP, or “Dance-Hall”)

**Very fast bus connects all processors to all memory, but all processes and memory share bandwidth, so not very scalable**



## Distributed-Memory (aka MPP, or Scalable)

**Interconnect between any 2 processors relatively high latency, perhaps low BW, but each link is relatively or completely independent of others, so more scalable**

# Cooperative Data Sharing: An Architecture-Independent Interface for Implementing Parallel CFD Applications

(and other stuff)

**David C. DiNucci**
**MRJ, Inc.**
**NAS, NASA Ames Research Center**
**Parallel Tools Team**