Distributed Resource Collectives (P2P & Grids)
Simplicity + Portability = Success

David C. DiNucci, PhD

dave@elepar.com www.elepar.com

Overview

Overview/History

Big Picture

- **Social and Technical Challenges**
- **Concepts**
- **Architecture**

Architectural Layers

- **Distributed Resource Management: PICA**
- **Portable Communication and Threads: CDS**
- **Component Assembly and Scope Management: SC**

Conclusion

History

1985-91: PhD at OGI, "F-Nets", formal model for architecture-independent parallel software engineering", F-nets, refinement/formalization of advisor Babb's LGDF

91: Created Software Cabling based on F-nets, postdoc at LLNL NERSC

91-96: Parallel tools group at NASA Ames NAS (prototyped CDS)

96-98: Planning for NASA Info Power Grid, led "Distributed Architectures and Scheduling" team (members included Fran Berman, Andrew Grimshaw, Ian Foster, Carl Kesselman, Bill Nitzberg, see PICA page for more)

98-present: Elepar commercializing this work (Beaverton in '99)

elepar

Technical Challenges: P2P and Grids

- **Dynamic topology -> Resource discovery/reservation/scheduling**
- **Heterogeneous speed/archit. -> Portability/Variable granularity**
- **Local or distributed comm -> Latency tolerance/Low ovhd/QoS**
- **Many decentralized components -> Fault tolerance**
- **Complex & concurrent -> Formal analysis, debugging rules**
- **App still #1 -> Leveraging existing tools, languages, techniques**
- **Utilizing untrusted resources -> Privacy/Security/Anonymity**
- **ROI -> Revenue models, pricing, bidding, accounting**
- **Connectivity -> Firewalls, NATs, dropped lines**
- **Intellectual property -> digital watermarking & rights mgmt**
- **Multiple administrative domains -> flexible policies**

Elepar's summary: "Performance, Portability, Programmability"
(After solving these, standardization.)

Social Challenges: Peer-to-Peer vs. Grids

	Peer-to-Peer	Grid
People	Sales, marketing, analysts, hobbyists	Researchers, scientists, engineers, designers
Archives	Documents, sales, music, game state	Scientific, design, historical databases
Compute	PCs, PDAs	Parallel servers, supers
Peripherals	GUIs, personal devices, printers	CAD, immersive VR, sensory, robotic devices
Economy	Cheaper (conserve in existing practice)	Bigger (enable new capability / approach)
Motivators	Data access, privacy, autonomy, indep.	Capacity availability, scalability, efficiency
Dynamics	Assemble, disband	Utility, grow & shrink

Solutions: Elepar's Approach

One fully-supported set of architectural layers (3)

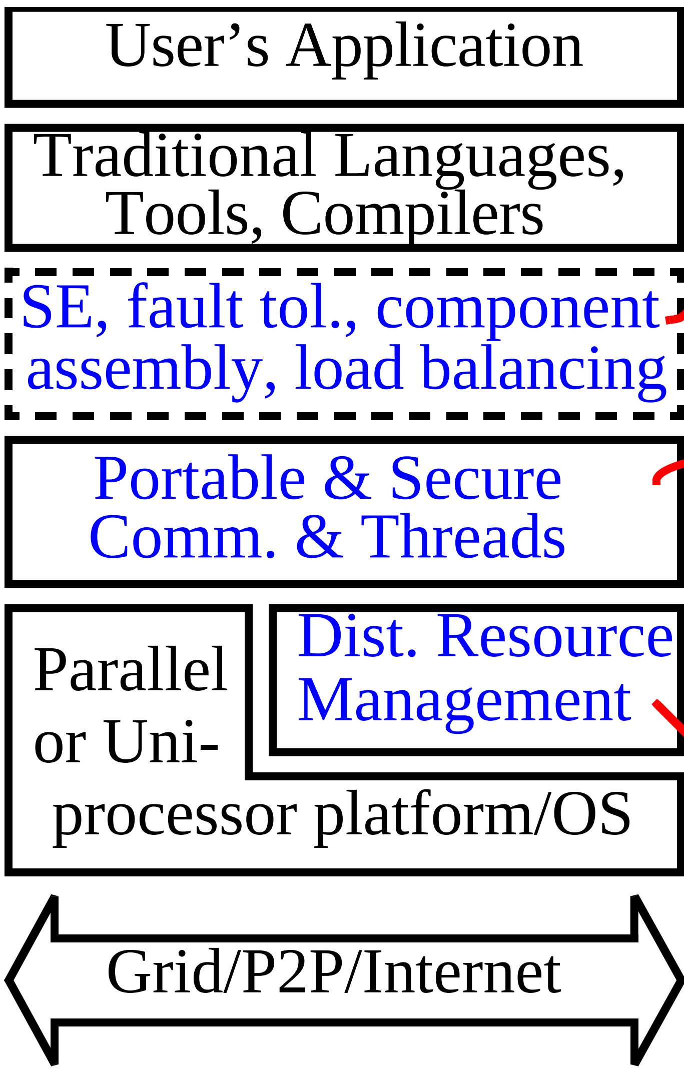
- **Each tackles different parts of the overall problem**
- **Independent, but built to work well together**
- **Can be individually replaced or omitted**

Manage complexity, allow *technical* objectives to guide the design.

Important concepts:

- **Abstract Resources (people, instruments, computers, archives)**
- **Resource Companies (entities which deal in those resources)**
- **Portable threads (cooperative data sharing)**
- **Scope management (who needs to know what, who doesn't)**
- **Compupackets (dataflow), resource cloud, atomic transactions**
- **Modules as objects, programs as modules (e.g. multi-disciplinary)**

Solutions: Architectural Layers



Software Cabling (SC): Visual OO component coordination methodology for building & analyzing dataflow/"indep thread" apps from modules implemented in trad'l langs

Cooperative Data Sharing (CDS): Efficient runtime support for portable threads and communication between them, 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

Distributed Resource Mgmt Layer: PICA

PICA stands for **P**eople, **I**nstruments, **C**omputers, and **A**rchives

PICA is a set of protocols & guidelines for how distributed resources look & act with respect to discovery, bidding, (co)scheduling, reservation, acquisition, and relinquishment.

Four principle components:

- **Resources:** Standard way to specify complex resources (P, I, C, A)
- **Resource Keys** (i.e. capabilities): How resources are referenced, and how those references are passed from place to place
- **Resource Companies:** Standard way to request, bid for, and/or provide access to resources (symbolized as resource keys)
- **Resource Supply Chains:** Fan-in/Fan-out of complex resources and payments between suppliers (i.e. resource companies) & end users

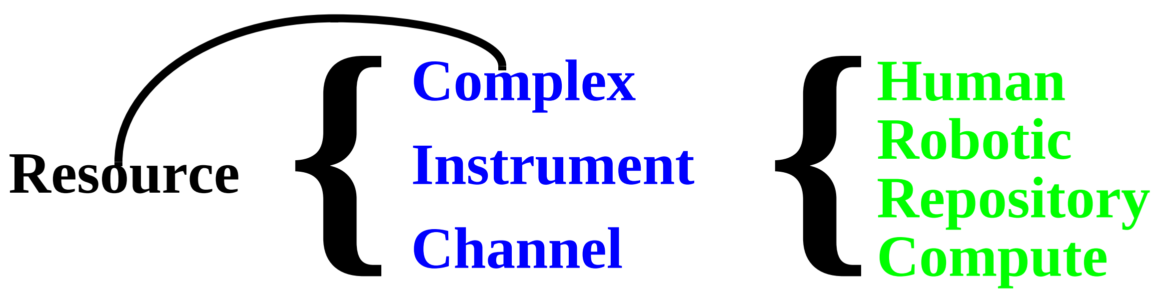
PICA: Resource Classes

There are three kinds of resources:

1. **Instrument**: Lies in a particular region of spacetime; creates, consumes, or transforms data/information
2. **Channel**: Moves data/info in space and/or time
3. **Complex**: Collection of instruments, usually networks of above

Instruments, in turn, are either:

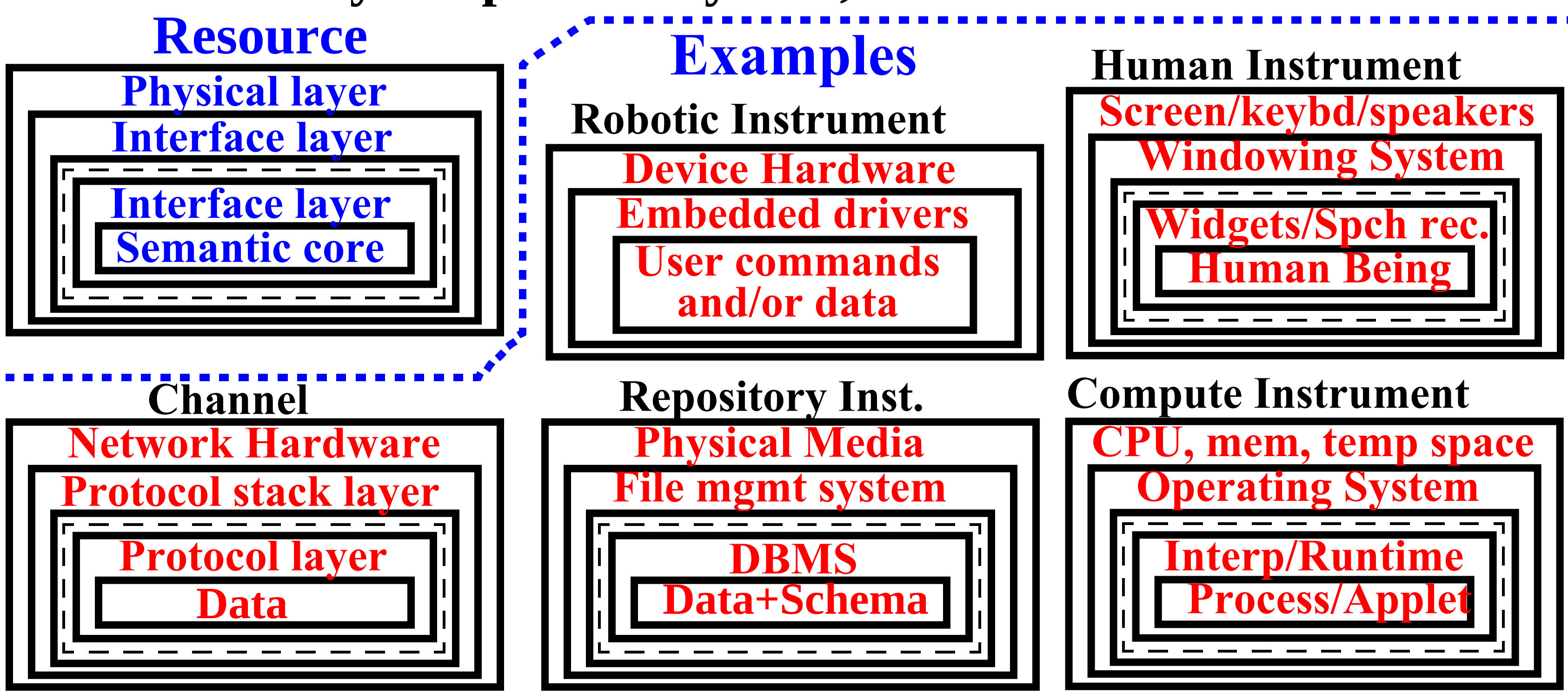
1. **Human**: Interface to human intelligence
2. **Robotic**: A peripheral, meant to sense/alter "the outside world"
3. **Repository**: Preserves and/or retrieves data/information
4. **Compute**: Automata to create &/or transform data/information



PICA: Instruments, Channels, and Tasks

Instruments and channels are layered: A physical layer; interface (or abstraction) layers provide virtualization; innermost semantic core is ultimate (abstract) behavior ("program", or data channeled).

Different instruments can share some outer layers. "Task" is defined as those inner layers specified by user, after resource allocation.

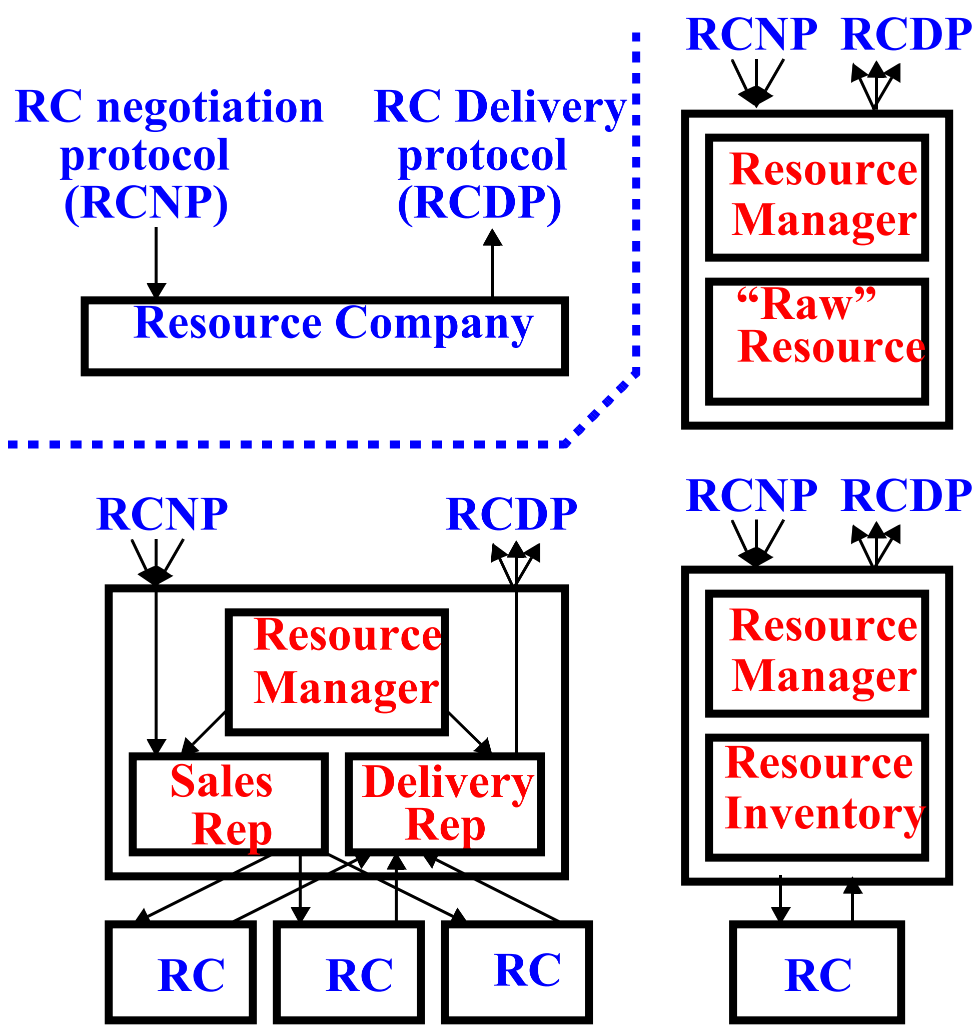


PICA: Resource Companies

To get a resource, one goes to a Resource Company (RC). An RC may simply provide a "raw resource", or may compose simpler resources it obtains (from other RCs) into complex, or may just optimize resource flow from other RCs by acting as "retailer" or "broker".

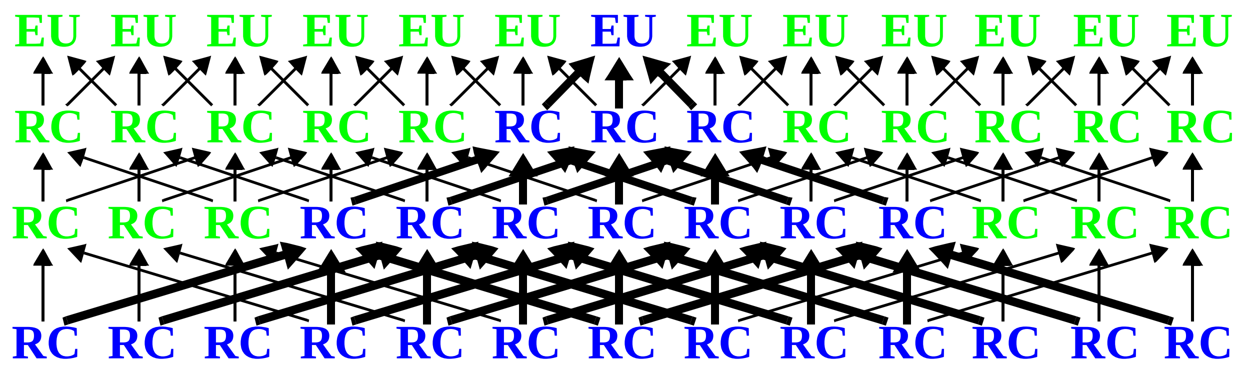
"RC Negotiation Protocol" standard way to discover & request resources from RCs.

Co-scheduling is an automatic byproduct.

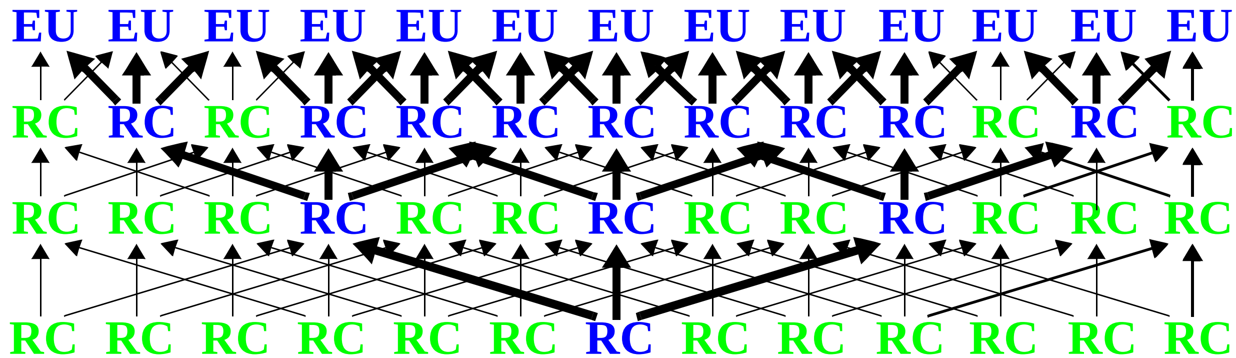


PICA: Supply Chain

Even limited RC connectivity allows a single end-user to obtain resources containing “parts” from many different “suppliers”...

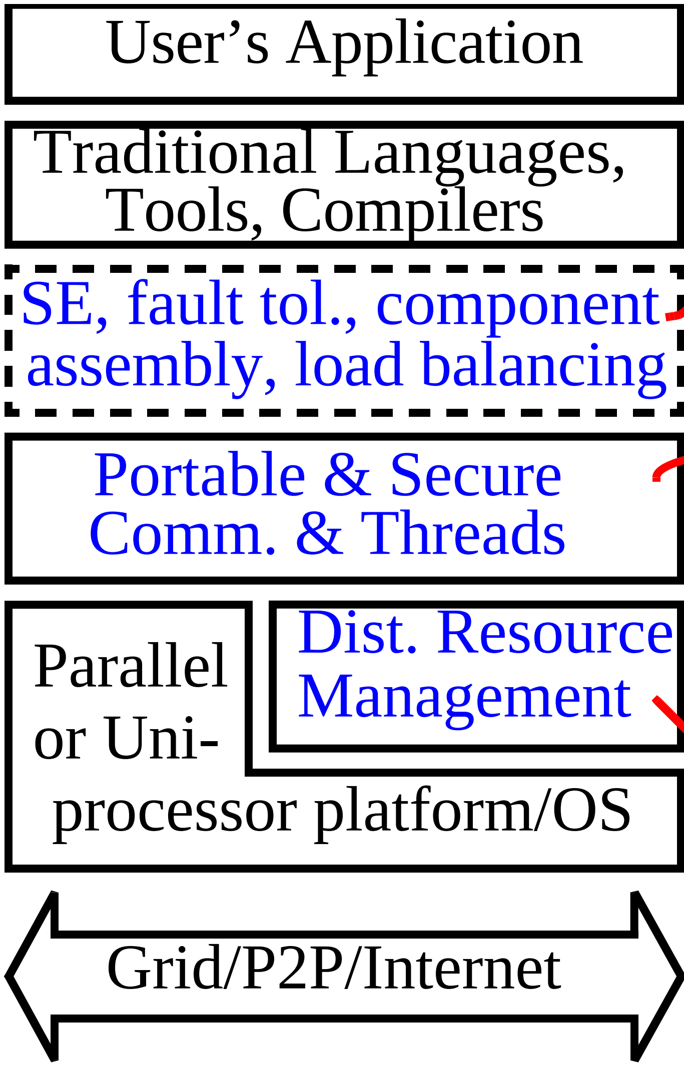


while many end-users obtain them w/parts from a single supplier.



Reverse arrows and it’s payments & billing without micropayments.

Solutions: Architectural Layers

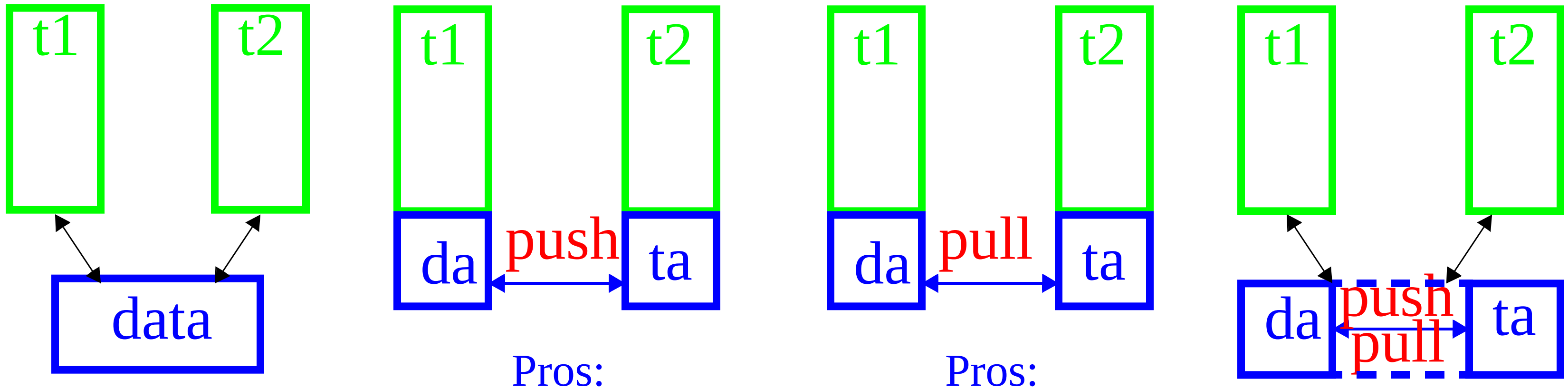


Software Cabling (SC): Visual OO component coordination methodology for building & analyzing dataflow/"indep thread" apps from modules implemented in trad'l langs

Cooperative Data Sharing (CDS): Efficient runtime support for portable threads and communication between them, 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

Portable Comm & Threads Layer: CDS



Threads

Pros:
Low overhead

Cons:
All data must be held in common or shared memory

Msg passing

Pros:
Latency hiding
Queuing
Data translation

Cons:
Copy overhead always suffered

DSM

Pros:
No extra copy

Cons:
No latency hdg
No queuing
No data translation
Mem mgmt req'd?

CDS

Pros:
Latency hiding
Queuing
Data translation
No extra copy
No mem mgmt

CDS (Cooperative Data Sharing) blurs distinction between proc & thread, blends the semantics & advantages of MP and DSM, includes support for process control, active messages, conversion/marshalling.

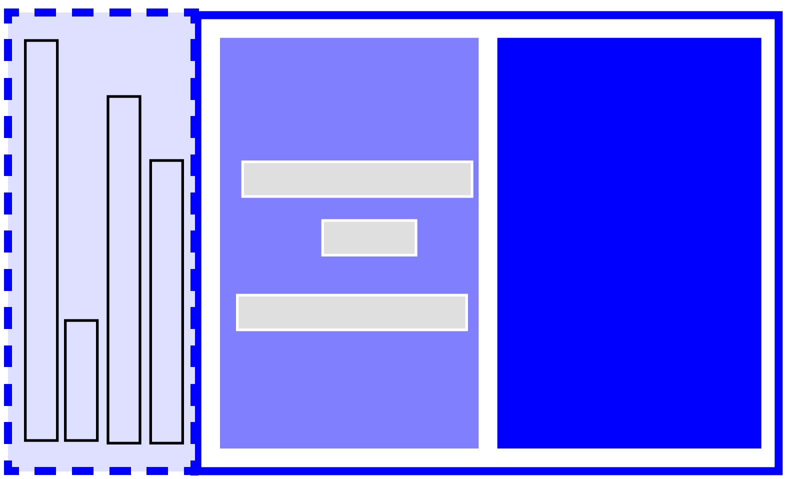
CDS: Featureset

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

CDS: Compute Entity (CCE) Anatomy

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

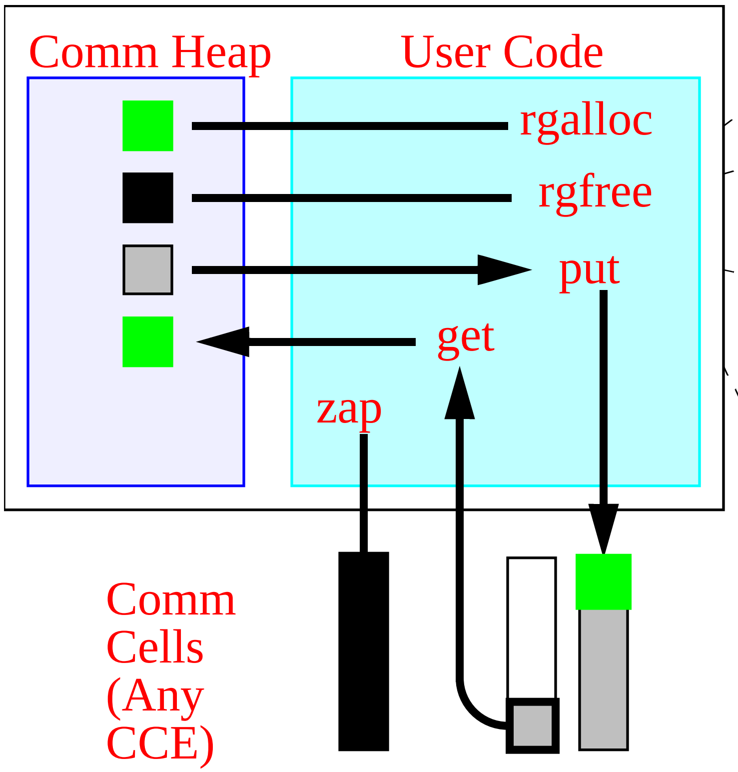
Comm Heap: Can be treated like standard heap: i.e. **`malloc`**, **`free`**. Holds data on its way to or from a Comm Cell. User is responsible for enlarging and/or shrinking.



User code & data: Standard Unix process (or thread!)

`get` and **`put`** move data between comm heap & any comm cell.

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: As General-Purpose API

CDS offers very general concurrent programming support, addressing many current challenges in parallel and distributed computing—e.g.

- Programming heterogeneous architectures (e.g. clusters of SMPs)
- Making applications more portable between distributed and/or shared memory and/or uniprocessor architectures
- Providing a much simpler programming interface than MPI-2 while offering similar (or greater, in some cases) functionality
- Providing a common API capable of leveraging the power of newer transport protocols like VIA and InfiniBand

Elepar's current CDS product is called "BCR", built upon SysV shared memory segments, UDP/IP, and custom locking protocols

CDS: The Interface (C Binding)

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`

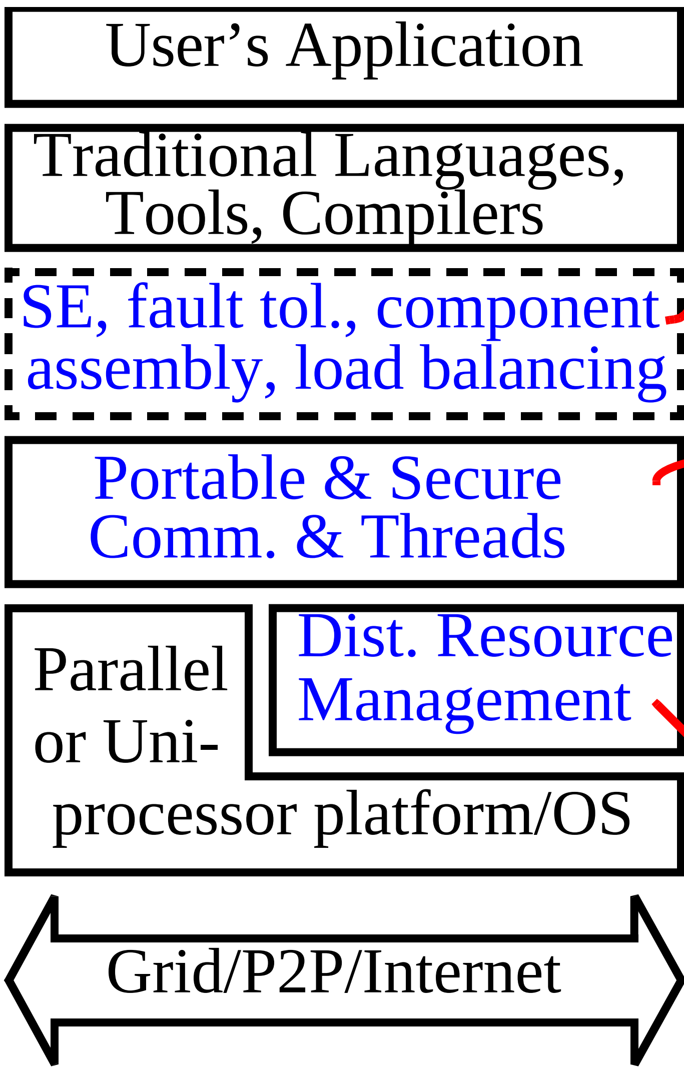
`acqrl acqwl rlsrl rlswl wl2rl`

`irecv ibsend irecvx iacqrl iacqwl`

Process and thread control

`enlist init myinfo hdlr prior`

Solutions: Architectural Layers



Software Cabling (SC): Visual OO component coordination methodology for building & analyzing dataflow/"indep thread" apps from modules implemented in trad'l langs

Cooperative Data Sharing (CDS): Efficient runtime support for portable threads and communication between them, 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

SE/LB/FT Layer: Software Cabling (SC)

Software Engineering goals:

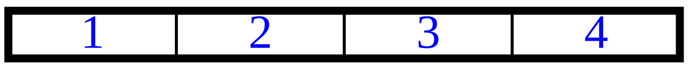
- Manage (i.e. determine/dictate scope of) data, events, side-effects
- Provide straightforward semantics (declarative or functional) that can be used to reason abstractly in dynamic, hetero env.
- Components, templates, & OO: To manage complexity and ease construction, from simple tasks to multi-disciplinary systems

Execution-based goals:

- Fault tolerance, through atomic transactions
- Efficient resource use by commoditizing usage—i.e. load balancing generic transactions
- Latency hiding & bandwidth preservation via split transactions, understanding potential producers & consumers of information

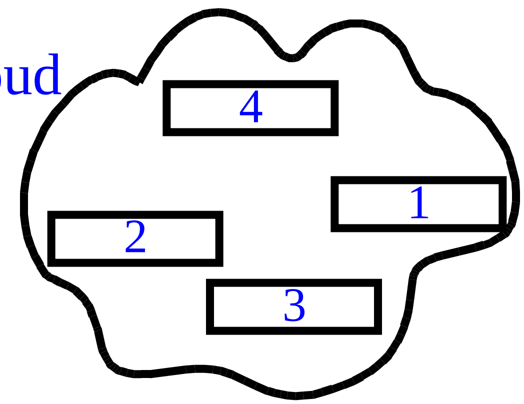
SC: "CompuPackets" (Dataflow)

Traditional Packet Switching

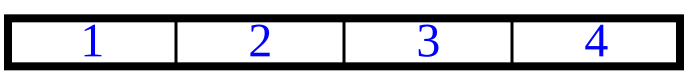


Message is indep packets

BW cloud

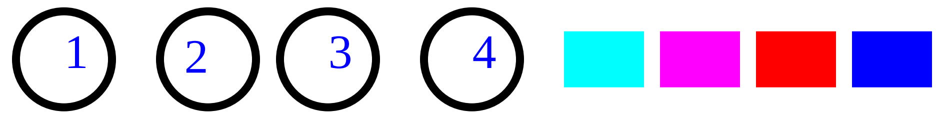


Packets independently use BW to get to destination



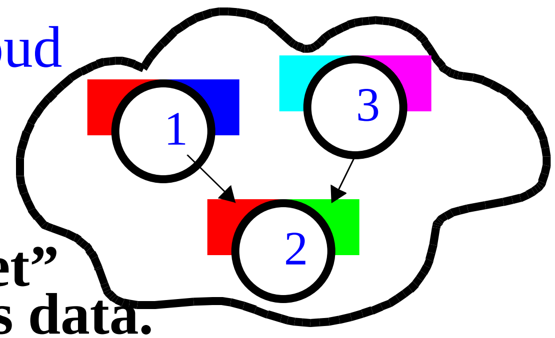
Re-interpret packets as msg

"Compu-Packet Switching"



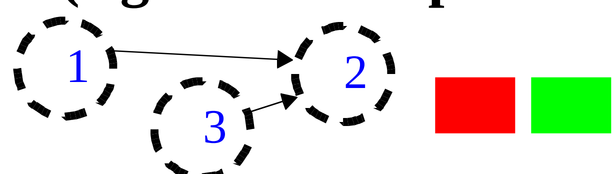
Program is "indep" threads, init data

Resource cloud
(BW+cycles
+...)



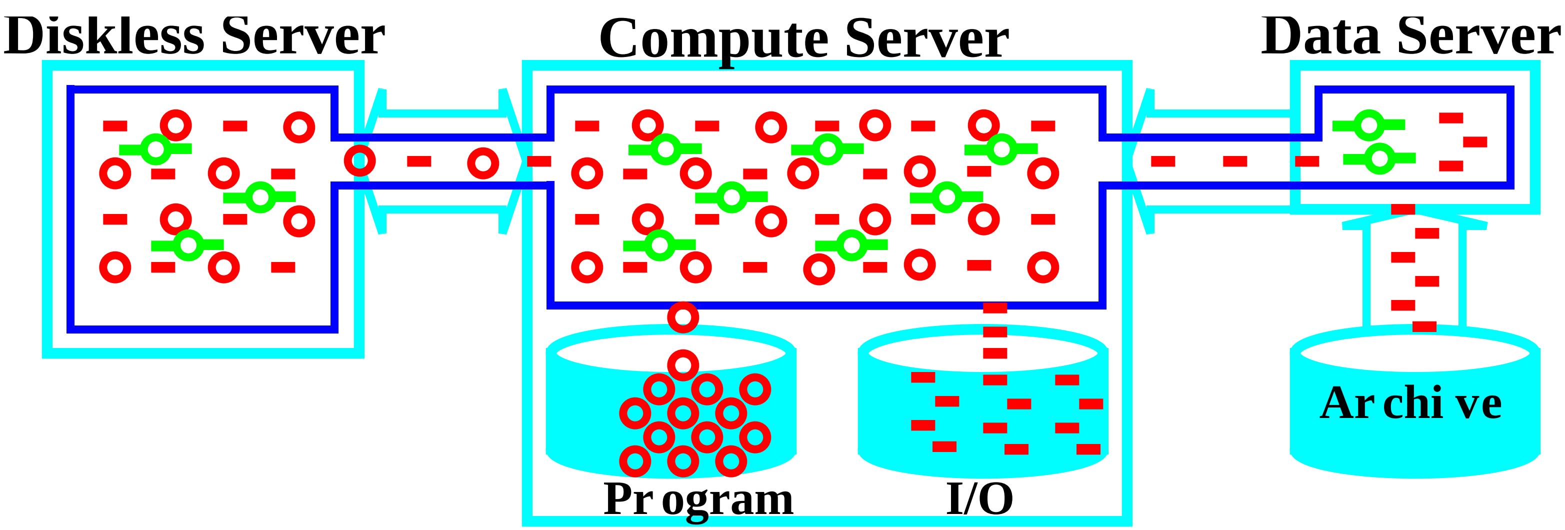
"Compupacket"
= thread+its data.

Compupackets independently use cycles to run, create new data (e.g. for new c'packets).



Interpret as computation DAG+results

SC: The Three Layers Revisited



Tool	Notation	Role
PICA		Raw resources (Machines, disks, networks, etc.)
CDS		Common execution/communication/security env.
Software Cabling		Stateless (non-executing) program fragment
		State (i.e. data item or region)
		Executing (or execute-ready) "compupacket" (Program + Data = atomic transaction)

SC: Advantages/Challenges of Dataflow

Advantages:

- Compackets indep, either run or don't: No waiting for each other
- Compupackets fill up processors like pebbles into bucket, efficiently using whatever cycles are available
- Compupackets, as atomic transactions, facilitate fault tolerance
- Each compupacket is functional, easy to specify & reason about

Challenges:

ready compupackets should be large (> # available processors)

- Need methods to build programs in this form (w/existing langs)
- Binding data to compupacket and initiating it must be low ov'h'd
- Link latency must be hidden or avoided when possible
- Need strategies for compupacket binding, processor assignment

SC: From Program to (Portable) Threads

Making multi-threaded apps or parallelizing compilers is tough, BUT

- **virtually all programs are built from smaller components (i.e. functions, subroutines, methods, etc.)**
- **if side-effect free, they already act like compupackets—i.e. they begin with their input data, run to completion creating results**
- **so, facilitate concurrent composition, manage scope of side-effects**

SC: Concurrent Composition, Scoping

Software Cabling uses CDS-style comm to compose modules written in one or more traditional languages (e.g. C, C++, Fortran, Java).

- **Programmer explicitly manages of scope of side-effects (e.g. data access/flow) between modules using the concept of "cabling"**
- **Execution order is purely event driven, facilitating concurrency**
- **To capture all potential module interactions, a schematic-like visual syntax (and accompanying CASE tools) supported**
- **Constructs facilitate hierarchical composition, templates, OO, data parallelism, distributed mem alloc, distributable arrays**

So, SC good for single app, multi-disciplinary, & mission-critical SE:

- **Formal, functional specifications provide leverage for program spec, verification, and powerful debugging techiques & replay**
- **Fault tolerance (because compupackets are atomic transactions!)**

SC: 20,000ft View

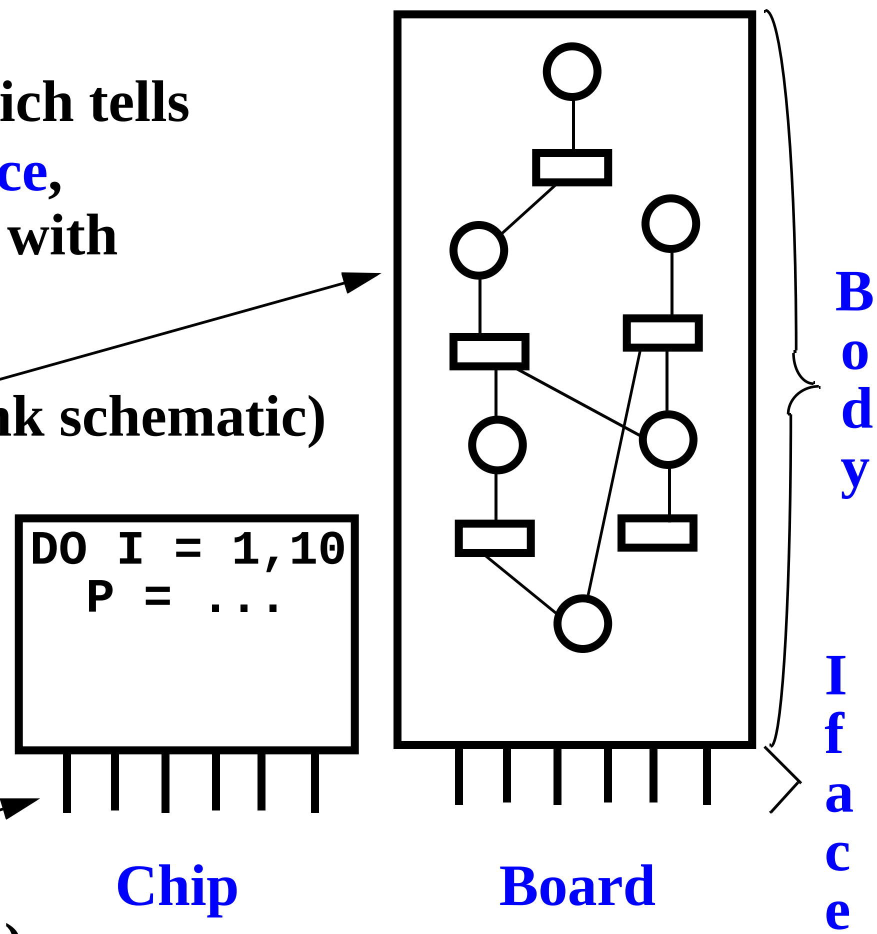
Two kinds of modules:

Both consist of a body, which tells what to do, and an interface, which allows it to interact with its environment

Board body is graphical (think schematic)

Chip body is written in your favorite language (C++, Java...Fortran?)

Interface consists of pins which carry data and signals. (Correspond closely to CDS regions, cells.)



SC: Chip Body

A chip's body is a subprogram or function, and the pins in the chip's interface are its arguments. The only syntax extensions are:

1. An interface definition language (IDL) to define interface

2. A "signal" statement, effectively a "return" that allows one argument at a time to be returned—roughly of the form:

```
post signame to pinname
```

Unlike locking primitives or message receipts, it does not change the local behavior of the code*. So, the programmer uses a purely sequential, traditional mindset when implementing components.

***except for optionally making the pin argument inaccessible**

Summary

Elepar breaks the problems of distributed resource collectives (e.g. P2P & Grids) down into:

- **Distributed Resource Management: PICA**
- **Portable threads and safe and efficient communication: CDS**
- **Component/OO-based SW construction and fault tolerance: SC**

Each is based on a sound technical approach. Each is made to work with the others, but can be modularly replaced. CDS is available now in demo form (on Elepar website), info on other tech also there.

Elepar is happy to discuss consulting, collaboration, and investment arrangements to help others leverage this technology.