(SBCL)Lisp w/Hadoop Streaming

&

distributed(CLOS)objects/code w/CLIPS&PVM

Mike.Bobak@gmail.com

Streaming is very flexible

Hadoop Streaming

Hadoop provides an API to MapReduce that allows you to write your map and reduce functions in languages other than Java. *Hadoop Streaming* uses Unix standard streams as the interface between Hadoop and your program, so you can use any language that can read standard input and write to standard output to write your MapReduce program.

Streaming is naturally suited for text processing (although, as of version 0.21.0, it can handle binary streams, too), and when used in text mode, it has a line-oriented view of data. Map input data is passed over standard input to your map function, which pro- cesses it line by line and writes lines to standard output. A map output key-value pair is written as a single tab-delimited line. Input to the reduce function is in the same format—a tab-separated key-value pair—passed over standard input. The reduce func- tion reads lines from standard input, which the framework guarantees are sorted by key, and writes its results to standard output.

Let’s illustrate this by rewriting our MapReduce program for finding maximum tem- peratures by year in Streaming.

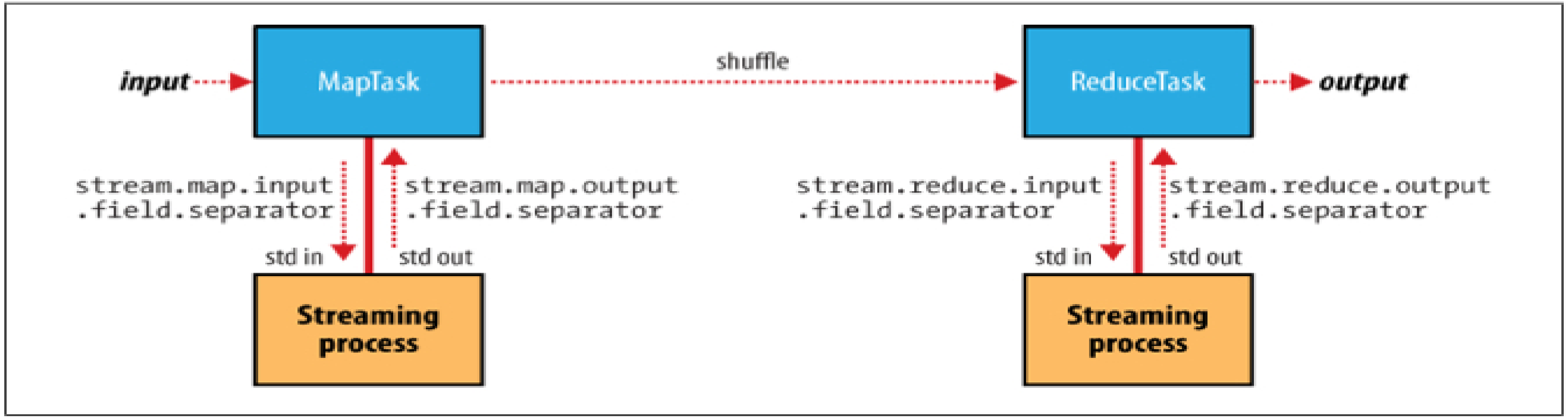

input
MapTask
shuffle
ReduceTask
output
stream.map.input .field.separator
std in
stream.map.output .field.separator
std out
Streaming process
stream.reduce.input .field.separator
std in
stream.reduce.output .field.separator
std out
Streaming process

Figure 7-1. Where separators are used in a Streaming MapReduce job

examples of setup files & processing

```
hadoop fs -rmr streaming/results
hadoop jar /usr/lib/hadoop/contrib/streaming/hadoop-streaming-0.20.2+320.jar \
    -verbose \
    -input streaming/urls \
    -output streaming/results \
        -mapper  "sbcl --core km.core --load mapper.cl" \
        -reducer "sbcl --core km.core --load reducer.cl" \
    -file mapper.cl \
    -file reducer.cl
```

Can also interact w/AWS ..

Can get to S3 storage w/: http://www.xach.com/lisp/zs3/
get-string/put-string of http://common-lisp.net/project/cl-json/ strs

or

Simple Command-Line Access to Amazon EC2 and Amazon S3, via:
http://common-lisp.net/project/trivial-shell/ to http://aws.amazon.com/developertools/739 AWS shell tool.

For fast front-end use could put the json to http://www.cliki.net/cl-mongo &can make use of: http://cliki.net/cl-redis too.

Had considered being able to take avro (self-describing) inputs.
It has a json header that describes the binary encoded stream.

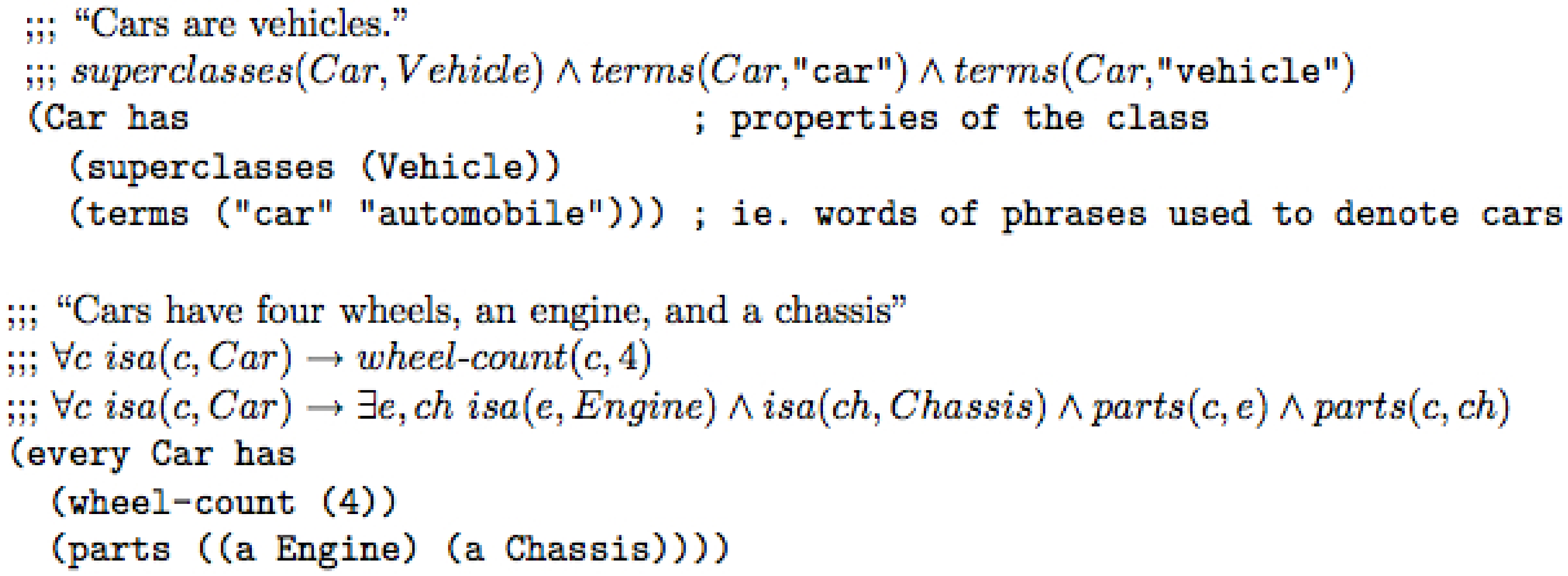
can even mix w/hadoop-streaming, but be sure that it is faster than a multi-step process, which could be organized&run by oozie

Though for some tasks you might just be able to use *cascalog*, in clojure which allows for tighter Hadoop/Java integration,
&for more real-time streaming work a clojure hadoop like system called *storm* which uses MQ, though possibly rewritable in Lisp.

Possible app: hadoop(like) graph-store

BBN's SHARD triple-store uses flat HDFS files w/great benchmarks. I think this could be done w/Lisp, and have considered doing some of the reasoning w/KnowledgeMachine, which was used in the HALO, then CALO project which turned into Siri.

SHARD http://www.lotico.com/slides/20110208/Rohloff_Meetup_02_08_2011.pdf
KM http://www.cs.utexas.edu/~mfkb/km/

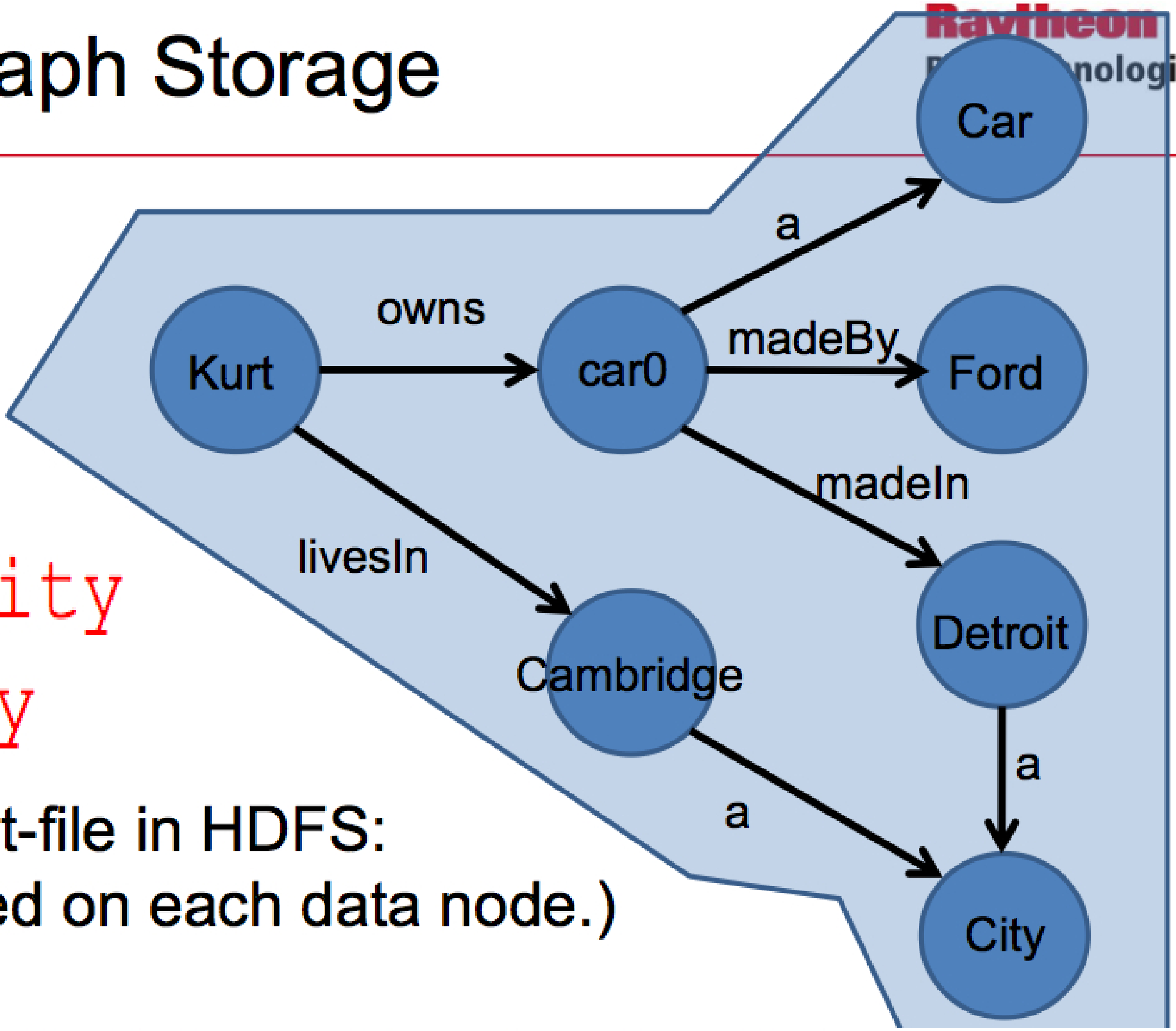


```
;;; “Cars are vehicles.”
;;; superclasses(Car, Vehicle) ∧ terms(Car,"car") ∧ terms(Car,"vehicle")
(Car has                              ; properties of the class
  (superclasses (Vehicle))
  (terms ("car" "automobile"))) ; ie. words of phrases used to denote cars

;;; “Cars have four wheels, an engine, and a chassis”
;;; ∀c isa(c, Car) → wheel-count(c, 4)
;;; ∀c isa(c, Car) → ∃e, ch isa(e, Engine) ∧ isa(ch, Chassis) ∧ parts(c, e) ∧ parts(c, ch)
(every Car has
  (wheel-count (4))
  (parts ((a Engine) (a Chassis))))
```

SHARD: Big(Semantic)Data

HDFS Graph Storage



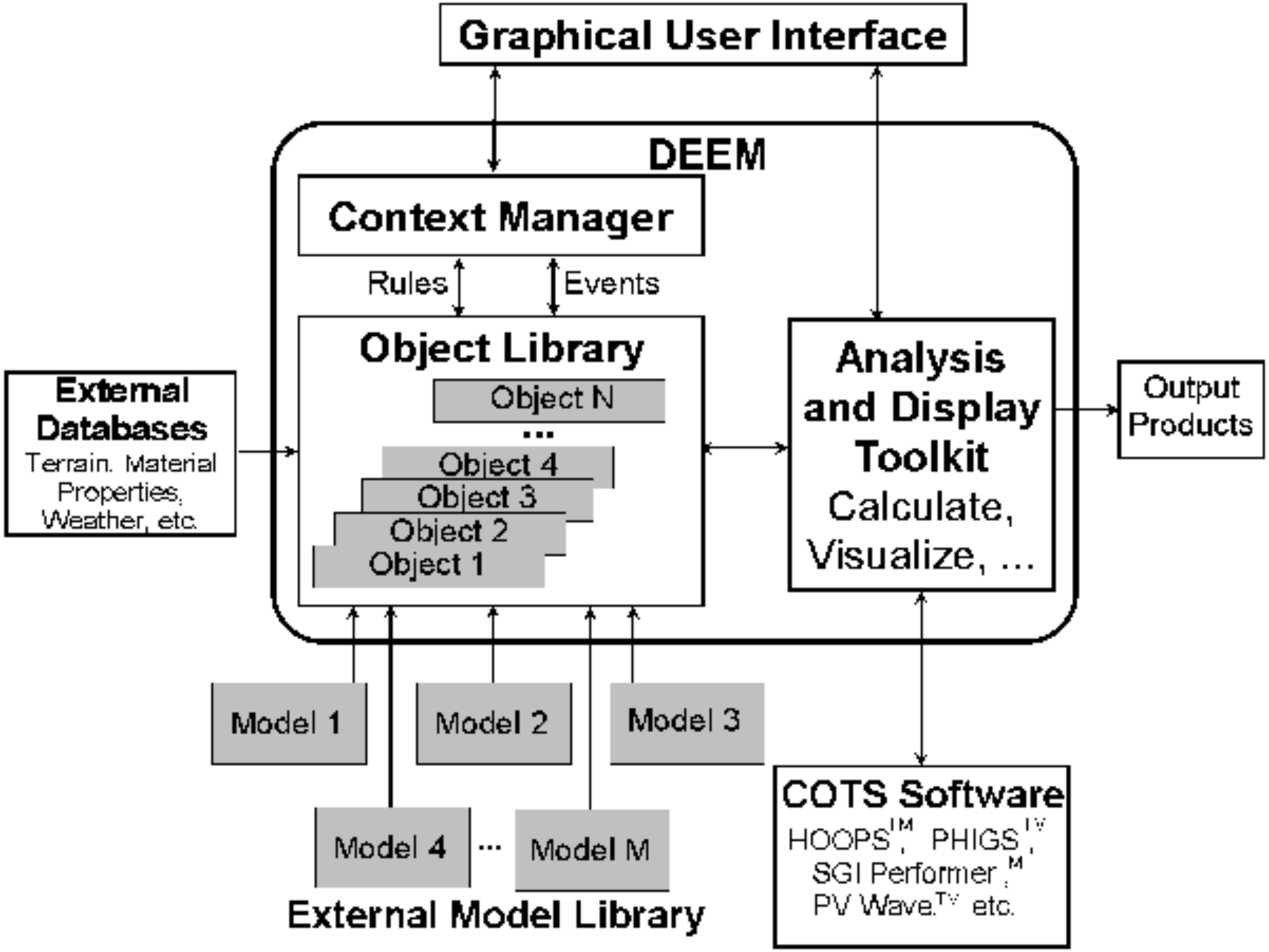
```
Cambridge a City
Detroit a City
```

Graphs saved as flat-file in HDFS:
(Portions of file saved on each data node.)

```
Kurt owns car0 livesIn Cambridge
Car0 a Car madeBy Ford madeIn Detroit
```

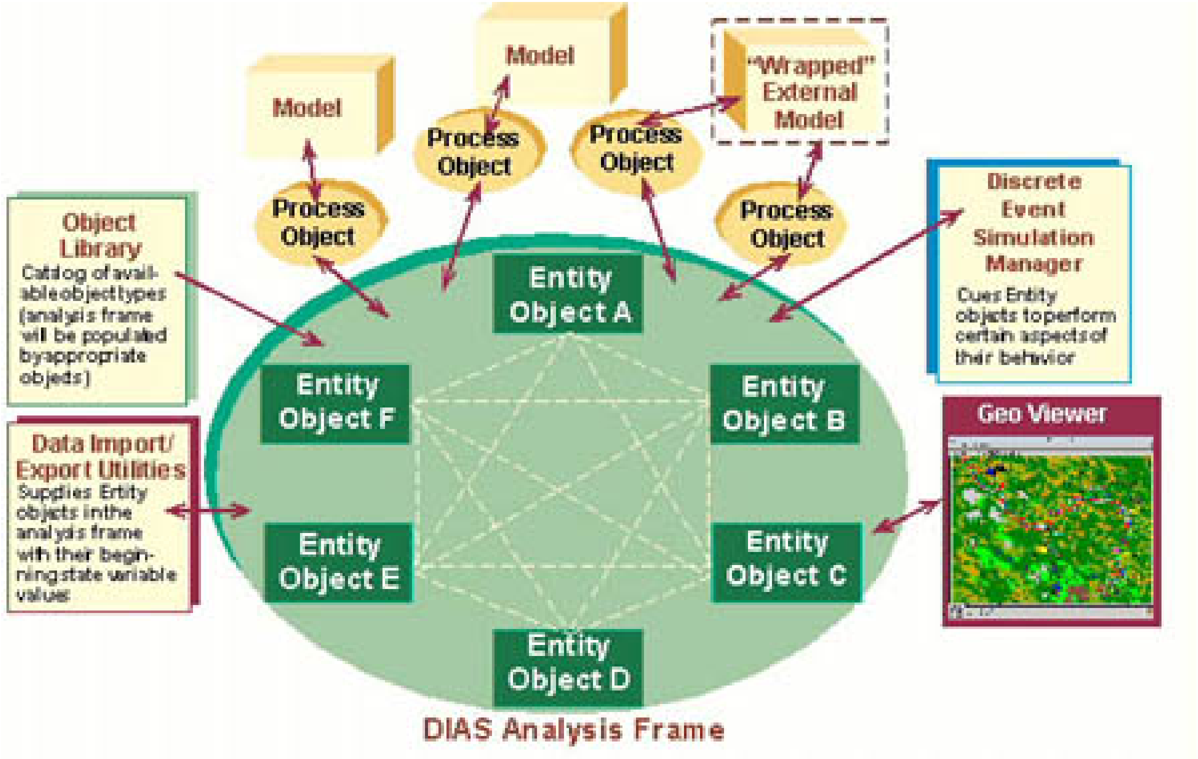
CLIPS w/PVM to run distrib-simulations

in CM



A live,data/goal driven distrib-simulation

latter used in:



Catalog of available object types (analysis frame will be populated by appropriate objects)

Cues Entity objects to perform certain aspects of their behavior

Supplies Entity objects in the analysis frame with their beginning state variable values

prototype comm for CM/SimMngr at: https://github.com/MBcode/CLIPSmsc/tree/master/clp-pvm
cl-pvm also available, so I would write similar methods, so they could interoperate.

MsgPassing:PVM (a more simple MPI)

We are seeing HPC msg-passing libs like MPI being brought into new Hadoop like packages; to allow for orders of magnitude improvements (via tigher-loops) for applications like Machine-Learning

Having distributed objects in Lisp is very easy, as you have the ability to easily reflect during runtime; slot-names &associated msg-packing routines for all their slot-value types makes it easy. The only trick was to start the obj-packed msg, w/a str, that could be eval'd to create an instance on the other side. This includes msg-unpacking methods for each of the slot/val pairs. This can be a nice way to quickly share huge chunks of data between several different dynamic languages.

Even tighter coupling like in rcl&cl-octave can benefit from msg-passing
I would like to share data w/xlispstat's vista, Lush,& Clojure.

Clojure (cascalog&storm)

Look great, but I'd rather build up in Lisp, and move it toward what Clojure can do, vs. not building on the rich set of Lisp Libs.

Could probably use cl-zmq as a starting place for a storm-like.

I wish java interop was easier, Rich H tried it before writing Clojure.
I think external-program, streaming,&msg-passing can get around some of the/se tighter/(more radical) ways of getting interop.

a few refs:

http://www.sbcl.org/ or http://ccl.clozure.com/ & http://www.quicklisp.org/ ;http://franz.com/agraph/
;http://protege.stanford.edu/

http://www.cs.utexas.edu/~mfkb/km/ KnRepr&Reasoning/Qry
http://www.dist-systems.bbn.com/people/krohloff/shard.shtml

Data: http://code.google.com/p/cl-protobuf/ or http://avro.apache.org/ in lisp w/cl-json &a decent cl binary pkg
Store: http://www.hdfgroup.org/HDF5/ git://gitorious.org/dh-misc/hdf5.git https://github.com/filonenko-mikhail/cl-scidb.git

I'd like semantics for petabytes, to help guide it's use.

http://www.csm.ornl.gov/pvm/PVMvsMPI.ps MsgPassing

some Lisp(likes):
http://en.wikipedia.org/wiki/CLIPS (has CLOS) RuleBased-Shell

http://en.wikipedia.org/wiki/XLispStat or in Lisp: https://github.com/blindglobe/common-lisp-stat http://www.visualstats.org/

http://common-lisp.net/project/rcl/ to R which has
R+pvm: http://cran.r-project.org/web/packages/rpvm/index.html

http://lush.sf.net

http://common-lisp.net/project/cl-octave/
to http://octave.sf.net w/ http://www.netlib.org/pvm3/ pkg

https://github.com/nathanmarz cascalog storm in Clojure
Cascalog is a replacement for tools like Pig, Hive, and Cascading. Storm is a distributed, reliable,&fault-tolerant stream processing system.