THE CLOUD GOES BOOM

DATA-CENTRIC PROGRAMMING FOR DATACENTERS

JOSEPH M HELLERSTEIN
U C BERKELEY

JOINT WORK

- Peter
  Tyson
  Neil
  William

ALVARO
CONDIE
CONWAY
MARCZAK

- Khaled
  Rusty SEARS

ELMELEEGY

YAHOO! RESEARCH

TODAY

- data-centric cloud programming
- declarative language status
- experiment: BOOM analytics
- directions

THE FUTURE'S SO CLOUDY

- a new software dev/deploy *platform*
  - shared, dynamic, evolving
  - spanning multiple machines over time
  - data and session-centric

WHAT DRIVES A NEW PLATFORM?

http://en.wikipedia.org/wiki/IBM_PC

http://en.wikipedia.org/wiki/Macintosh

http://en.wikipedia.org/wiki/Iphone

http://en.wikipedia.org/wiki/Wii

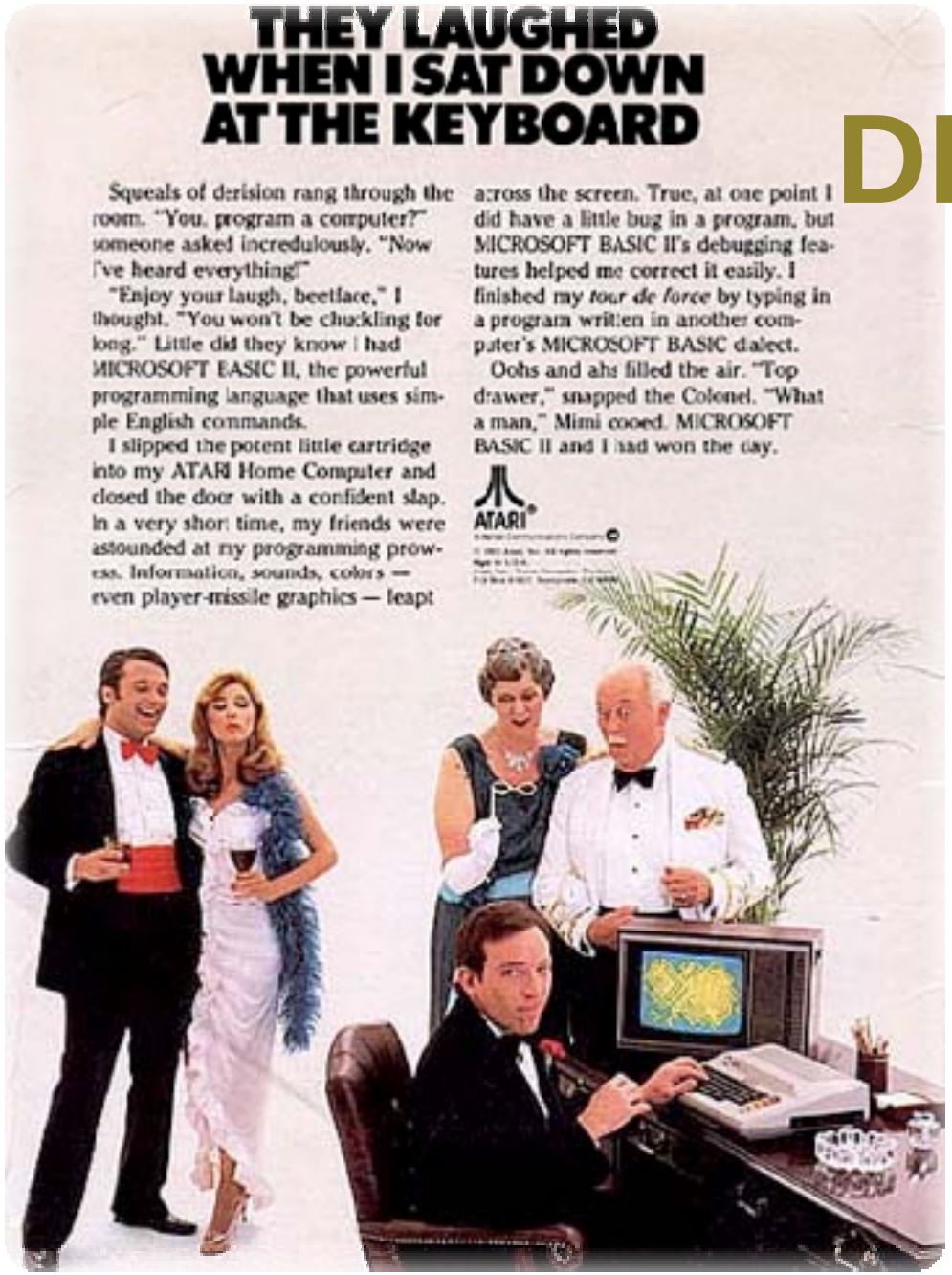
http://en.wikipedia.org/wiki/Connection_Machine

facebook

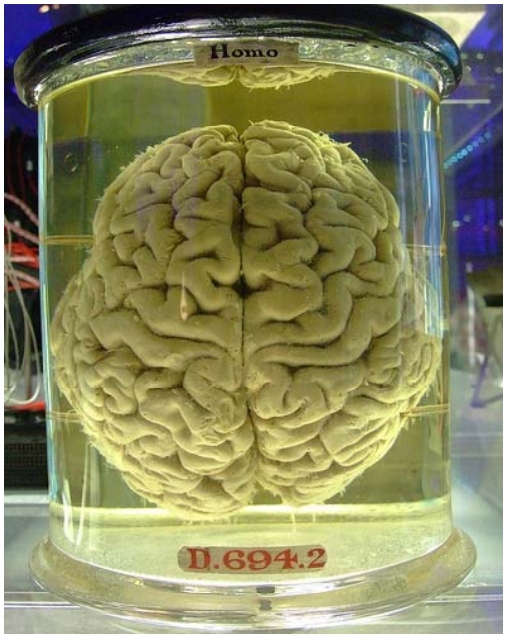
http://en.wikipedia.org/wiki/Facebook

http://www.flickr.com/photos/kky/7040567911/

DEVELOPERS!



http://www.flickr.com/photos/nicoll/150272557/



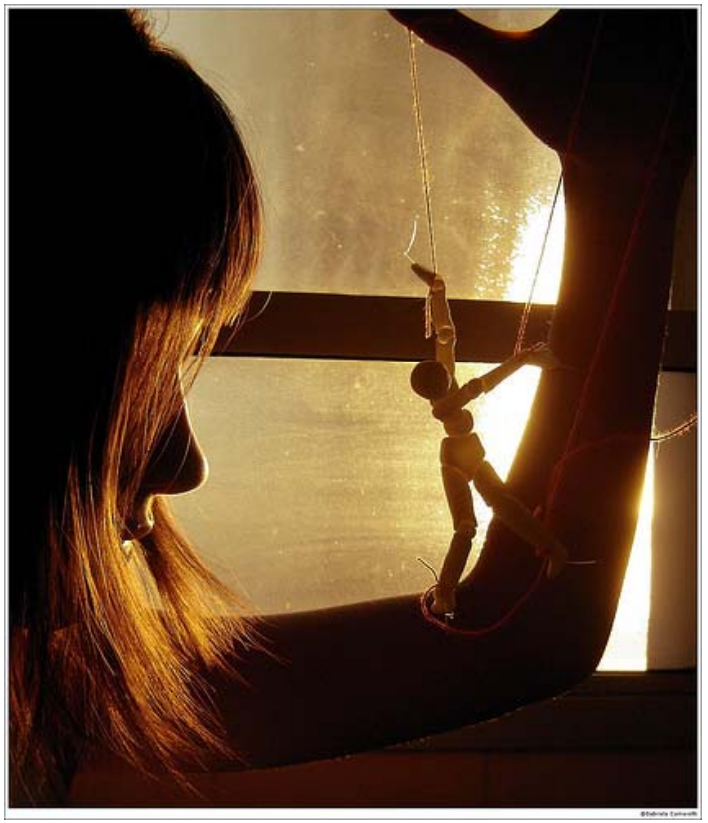
http://www.flickr.com/photos/gaetanlee/4219491 67/

CLOUD DEVELOPMENT

- the ultimate challenge?
  - parallel
  - distributed
  - shared resources, shared state
  - elastic/minimally managed

WHO'S THE BOSS

- it's all about the (distributed) *state*
  - session state
  - coordination state
  - system state
  - protocol state
  - permissions state
  - .. and the mission critical stuff



http://www.flickr.com/photos/face_it/2178362181/

- and deriving/updating that state!

http://www.flickr.com/photos/pshan427/2331162310/

WINNING STRATEGY

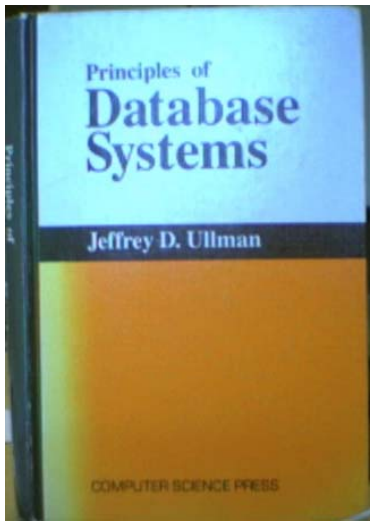
- *reify* state as data
  - system state is 1st-class data.
  - model. evolve. react.
- data-centric programming
  - instead of threads & cores
  - programs are *declarative* specs. *safety & liveness*
  - distribution, parallelism, synchronization *fall out* naturally

GRAND ENOUGH FOR YOU?

- automatic programming ... Gray's Turing lecture
  - "the problem is too hard ... Perhaps the domain can be limited ... In some domains, declarative programming works." (Lampson, JACM 50'th)
- can cloud be one of those domains?
- how many before we emend Lampson?

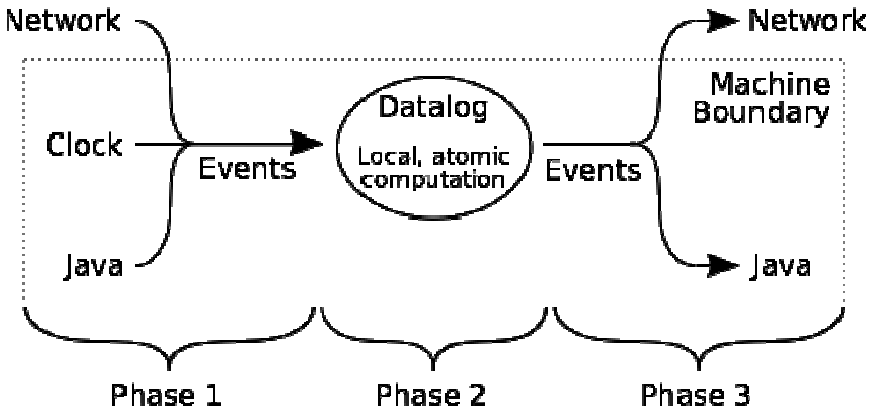
TODAY

- data-centric cloud programming
- declarative language status
- experiment: BOOM analytics
- directions



DATA-CENTRIC LANGUAGES

- decades of theory
  - logic programming, dataflow
- but: recent groundswell of applied research
  - security, networking, statistical machine learning, multiplayer games, robotics, natural language processing, compiler analysis...
  - see CCC Blog: http://www.cccblog.org/2008/10/20/the-data-centric-gambit/



OVERLOG IS...

- Datalog + Stratified Negation/Aggs
- + Horizontally partitioned tables
- + “Event” tables (clock/net/host)
- Self-timed Logic
  - iterated fixpoints
  - “state update” between fixpoints
    - recent formalism captures this in timed logic

SOME SIMPLE OVERLOG

- Asynch Service:

```
msg(Client, @Server, Svc, X) :-
   request(@Client, Server, Svc, X).

response(@Client, Server, Svc, X, Y) :-
     msg(Client, @Server, Svc, X),
     service(@Server, Svc, X, Y).
```

- Timeout:

```
timer(t, physical, 1000, infinity, 0).

waits(@C,S,Sv,X¸cnt<f_rand()>) :- t(_,_,_),
     request(@C,S,Sv,X),
     !response(@C,S,Sv,X,_).
late(@C,S,Sv,X) :- waits(@C,S,Sv,X,Delay), Delay > 1.
```

SOME SIMPLE OVERLOG

- Multicast:

```
msg(@Dest, Payload) :- xmission(@Src, Payload),
                       group(@Src, Dest).
```

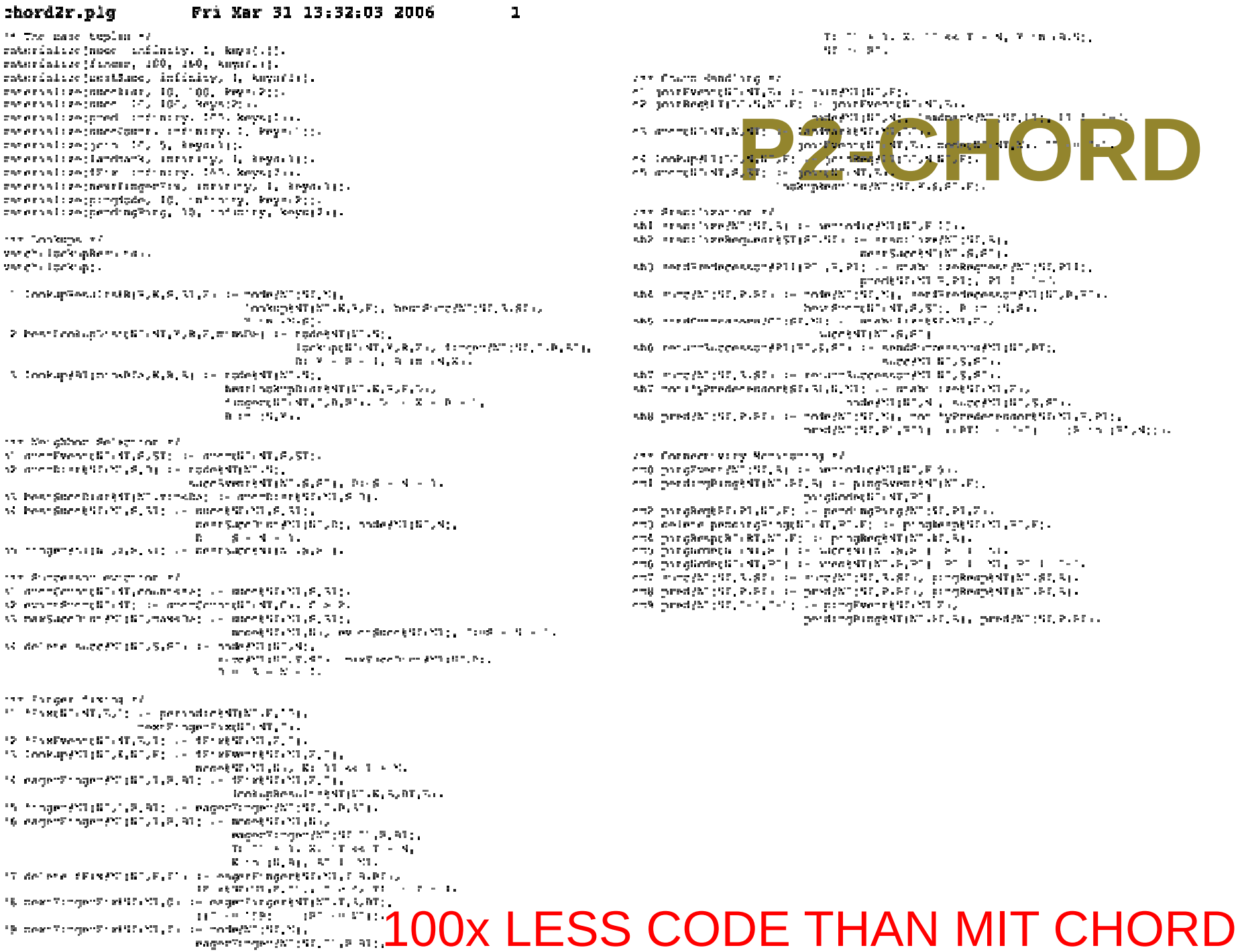
- NW Routes:

```
path(@Src, Dest, Dest, Cost) :-
    link(@Src, Dest, Cost).
path(@Src, Dest, Hop, C1+C2) :-
    link(@Src, Hop, C1),
    path(@Hop, Dest, N, C2).
bestcost(@Src, Dest, min<Cost>) :-
    path(@Src, Dest, Hop, Cost).
bestpath(@Src, Dest, Hop, Cost) :-
    path(@Src, Dest, Hop, Cost),
    bestcost(@Src, Dest, Cost).
```

http://www.flickr.com/photos/28682144@N02/2765974909/

P2-CHORD

- chord *distributed hash table*
  - Internet overlay for content-based routing
- high-function implementation
  - all the research bells and whistles
- 48 rules, 13 table definitions


chord2r.pig Fri Mar 31 13:32:03 2006 1
P2CHORD
100x LESS CODE THAN MIT CHORD

TODAY

- data-centric cloud programming
- declarative language status
- experiment: BOOM analytics
- directions

THE CLOUD GOES BOOM!

- Berkeley Orders Of Magnitude
  - OOM bigger systems
  - OOM less code
- done for network protocols, time to generalize
  - and make attractive to developers
  - Lincoln

HADOOP GOES BOOM!

- experiment: build a Big Data cloud stack in Overlog
  - goal 1: convince ourselves we're on track
  - goal 2: inform the design of a better language for the cloud
    - and multicore?
  - goal 3: pull off some feats of derring-do
- metrics
  - not just LOCs
  - flexibility, malleability, performance

EVOLUTION SCENARIO

- prototype: basic Hadoop functionality
- subsequent revisions (Hadoop fast-forward)
  - availability rev: hot-standby FS masters
    - live multipaxos replication in 45 rules
  - scalability rev: scale-out FS master state via parallelism
    - added in 1 day!
  - monitoring rev: invariant checks, logging via metaprogramming
- 9 months, 4 grad student developers
  - most work done in a 3-month span

CURRENT CODE BASE

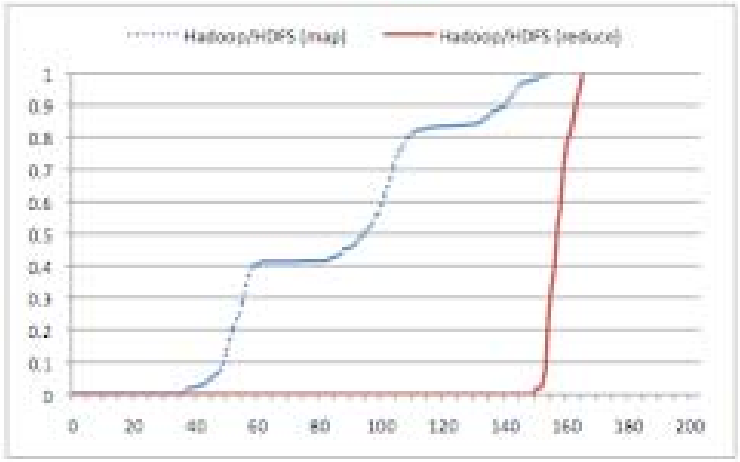
- Filesystem

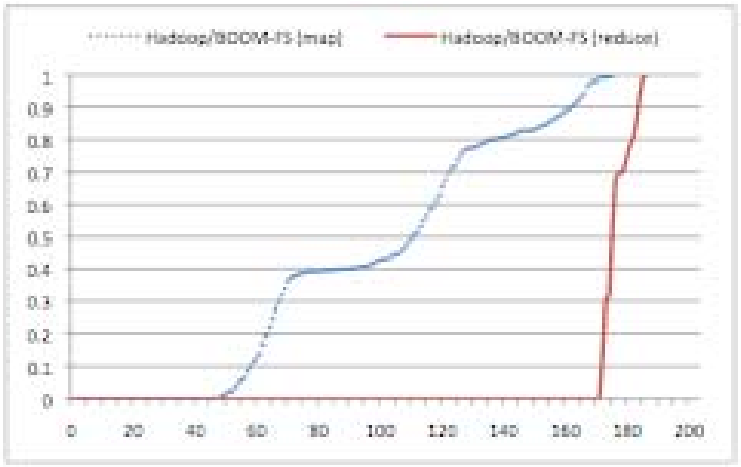
	Lines of Java	Lines of Overlog
HDFS	21,700	0
BOOM-FS	1,431	469

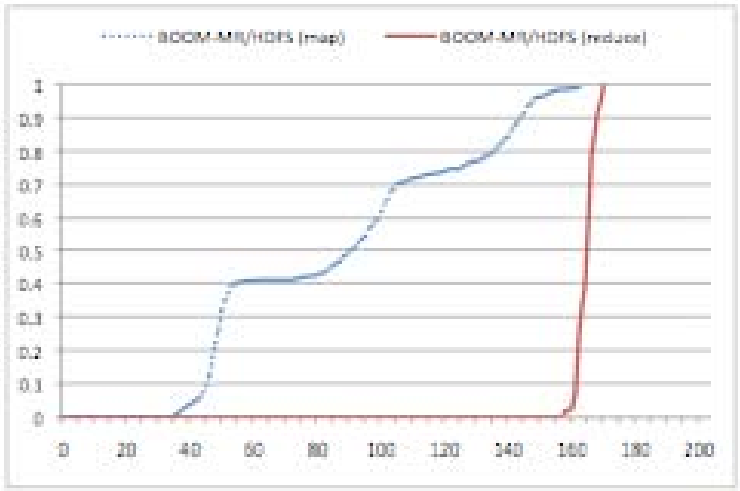
- MapReduce

	Lines of Java	Lines of Overlog
Hadoop	88,864	0
BOOM-FS	82,291	396

THE $3,500 SLIDE







AVAILABILITY REV

- HDFS has single point of failure at master
  - we found JIRA proposals for warm-standby
- but we went for a hot-standby scheme
  - had wanted to do serious Paxos all along as a stress test
  - Paxos: 50 Overlog rules (Stasis for persistence)
  - basic Paxos vs. serious multiPaxos

BASIC PAXOS

1. Priest p chooses a new ballot number b greater than lastTried [p], sets lastTried [p] to b, and sends a NextBallot (b) message to some set of priests.

2. Upon receipt of a NextBallot (b) message from p with b > nextBal [q], priest q sets nextBal [q] to b and sends a LastVote (b, v) message to p, where v equals prevVote [q]. (A NextBallot (b) message is ignored if b < nextBal [q].)

3. After receiving a LastVote (b, v) message from every priest in some majority set Q, where b = lastTried [p], priest p initiates a new ballot with number b, quorum Q, and decree d, where d is chosen to satisfy B3. He then sends a BeginBallot (b, d) message to every priest in Q.

4. Upon receipt of a BeginBallot (b,d) message with b = nextBal [q], priest q casts his vote in ballot number b, sets prevVote [q] to this vote, and sends a Voted (b, q) message to p. (A BeginBallot (b, d) message is ignored if b = nextBal [q].)

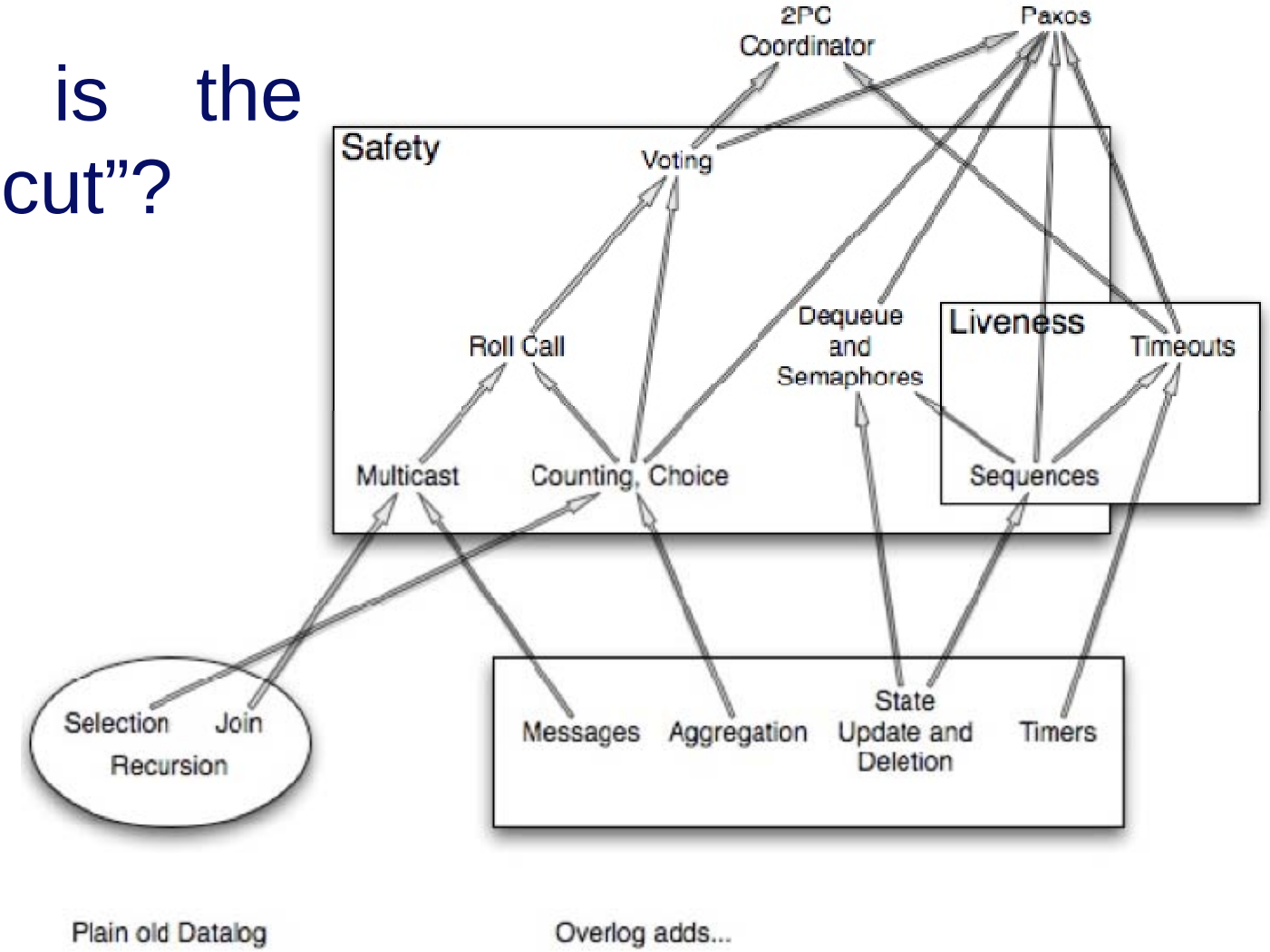
5. If p has received a Voted (b, q) message from every priest q in Q (the quorum for ballot number b), where b = lastTried [p], then he writes d (the decree of that ballot) in his ledger and sends a Success (d) message to every priest.

```
lastTried(Priest,Bnum) :- lastTried(Priest,Old), nextBallot(Priest,Bnum,Decree), Bnum>=Old;

nextBallot(Priest,Ballot,Decree) :- decreeRequest(Priest,Decree), lastTried(Priest,Old), priestCnt(Priest,Cnt),
Ballot:=Old+Cnt;

sendNextBallot(@Peer,Ballot,Decree,Priest) :- nextBallot(@Priest,Ballot,Decree), parliament(@Priest,Peer);

nextBal(Priest,Ballot) :- nextBal(Priest,Old), lastVote(Priest,Ballot,OldBallot,Decree), Ballot>=Old;

lastVote(Priest,Ballot,OldBallot,OldDecree,Peer) :- sendNextBallot(Priest,Ballot,Decree,Peer),
prevVote(Priest,OldBallot,OldDecree), Ballot>=OldBallot;

sendLastVote(@Lord,Ballot,OldBallot,Decree,Priest) :- lastVote(@Priest,Ballot,OldBallot,Decree,Lord);

priestCnt(Lord,count<*>) :- parliament(Lord,Priest);

lastVoteCnt(Lord,Ballot,count<Priest>) :- sendLastVote(Lord,Ballot,Foo,Bar,Priest);

maxPrevBallot(Lord,max<OldBallot>) :- sendLastVote(Lord,Ballot,OldBallot,Decree,Priest);

quorum(Lord,Ballot) :- priestCnt(Lord,Pcnt), lastVoteCnt(Lord,Ballot,Vcnt), Vcnt>(Pcnt/2);

beginBallot(Lord,Ballot,OldDecree) :- quorum(Lord,Ballot), maxPrevBallot(Lord,MaxB),
nextBallot(Lord,Ballot,Decree), sendLastVote(Lord,Ballot,MaxB,OldDecree,Priest), MaxB!=-1;

beginBallot(Lord,Ballot,Decree) :- quorum(Lord,Ballot), maxPrevBallot(Lord,MaxB),
sendLastVote(Lord,Ballot,MaxB,OldDecree,Priest), nextBallot(Lord,Ballot,Decree), MaxB==-1;

sendBeginBallot(@Priest,Ballot,Decree,Lord) :- beginBallot(@Lord,Ballot,Decree), parliament(@Lord,Priest);

vote(Priest,Ballot,Decree) :- sendBeginBallot(Priest,Ballot,Decree,Lord), nextBal(Priest,OldB), Ballot==OldB;

prevVote(Priest,Ballot,Decree) :- prevVote(Priest,Old), lastVote(Priest,Ballot,OldBallot,Decree),
vote(Priest,Ballot,Decree), Ballot>=Old;

sendVote(@Lord,Ballot,Decree,Priest) :- vote(@Priest,Ballot,Decree),
sendBeginBallot(@Priest,Ballot,Decree,Lord);


voteCnt(Lord,Ballot,count<Priest>) :- sendVote(Lord,Ballot,Decree,Priest);

decree(Lord,Ballot,Decree) :- lastTried(Lord,Ballot), voteCnt(Lord,Ballot,Votes),
lastVoteCnt(Lord,Ballot,Votes), beginBallot(Lord,Ballot,Decree);
```

MULTIPAXOS IN OVERLOG

"I Do Declare...", Alvaro, et al. NetDB 09

- what is the right "cut"?



MONITORING REV

- invariant checking easy to add
  - messages are data; just query that messages match protocol
  - we validated Paxos message counts
- tracing/logging via metaprogramming
  - code is data: can write “queries” to generate more code
  - we built a code coverage tool in a day (17 rules + a java driver)
- system telemetry, logging/querying
  - sampled /proc into tuples
  - easily wrote real-time in-network monitoring in Overlog

LESSONS 1

- everything is data
  - persistent stuff (e.g. FS metadata)
  - runtime state (e.g. Hadoop bookkeeping)
  - summary stats (e.g. LATE metrics)
  - in-flight msgs and system events
  - even parsed code

LESSONS 2

- because everything is data...
  - easy to design scale-out
  - *interposition* (classic OS goal) easy via dataflow
  - simpler concurrency?
    - data derivation vs. locks on object updates
    - dataflow typing vs. state/event combinatorics
- all this applies to dataflow programming
  - e.g. mapreduce++
  - potentially sacrifice code analysis

LESSONS 3

- overlog woes
  - datalog syntax: hard to write, *really* hard to read
  - operational semantics of unique keys and updates tricky
- thematic issues
  - distributed systems psuedocode often written as automata
    - hence a lot of our Overlog looks like automata, rather than invariants
    - fixable? automatically?
  - java/overlog boundaries hard to pick, affect performance

TODAY

- data-centric cloud programming
- declarative language status
- experiment: BOOM analytics
- directions

NEXT

Lincoln

- a cloud *programmer's* language, distilled from Overlog
  - demystify syntax
  - address semantics of space, time, and state
  - integrate with practical host language(s)
  - unimpeachable performance

BOOM

- build out a cloud infrastructure stack
  - extended Analytics
  - consistent write-oriented store
  - cloud configuration, monitoring, and management

QUERIES?

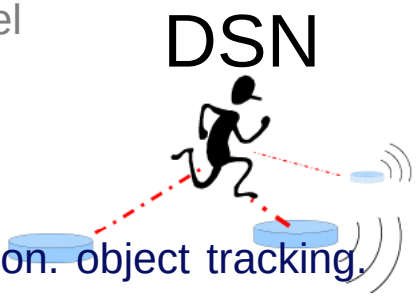
http://www.declarativity.net

DATA-CENTRIC LANGUAGES

- lots of academic experimentation
  - largely domain-specific
- substantial success
  - wide variety of domains
- still a largely disconnected field

DECLARATIVE NETWORKING @ BERKELEY/INTEL, ETC.

- textbook routing protocols
  - internet-style and wireless SIGCOMM 05, Berkeley/Wisconsin
- distributed hash tables
  - chord overlay network SOSP 05, Berkeley/Intel
- distributed debugging
  - watchpoints, snapshots EuroSys 06, Intel/Rice/MPI
- metacompilation Evita Raced VLDB 08, Berkeley/Intel
- wireless sensornets DSN
  - link estimation. geo routing. data collection. code dissemination. object tracking. localization. SenSys 07, IPSN 09, Berkeley



DECLARATIVE NETS: EXTERNAL

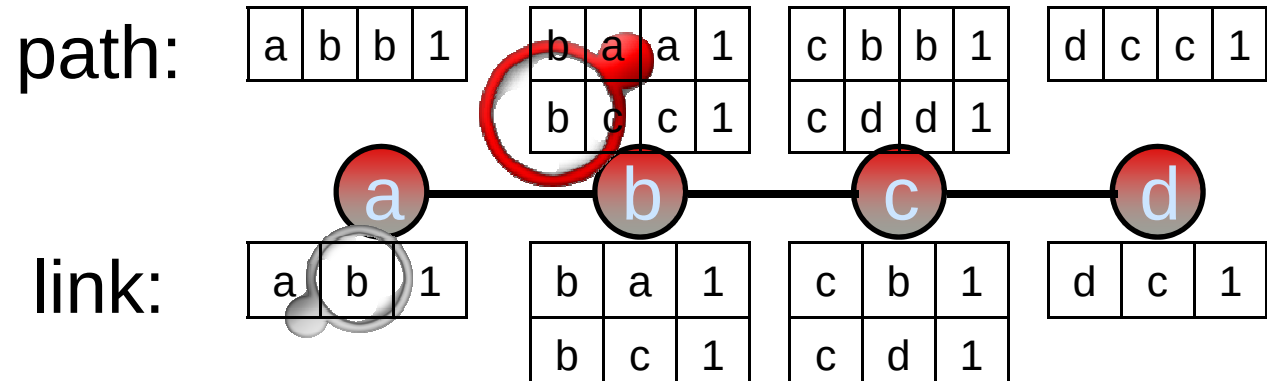
- simple paxos in overlog 44 lines, Harvard, 2006
- secure networking SeNDLog. NetDB07, MSR/Penn
- flexible replication in overlog PADRE/PADS SOSP07, NSDI09, Texas
- overlog semantics & analysis MPII 09, AT&T 09
- network transport BU 09
- distributed ML inference CMU/Berkeley 08

OTHERS

- compiler analysis (bddbddb) Stanford
- nlp (dyna) Johns Hopkins
- modular robotics (meld) CMU
- video games (sgl) Cornell
- 3-tier apps (hilda, xquery) Cornell, ETH, Oracle
- trust management (lbtrust) Penn/LogicBlox
- security protocols (pcl) Stanford

PARTITION SPECS INDUCE COMMUNICATION

- link(@X,Y,C)
- path(@X,Y,Y,C) :- link(@X,Y,C)
- path(@X,Z,Y,C+D) :- link(@X,Y,C), path(@Y,Z,N,D)



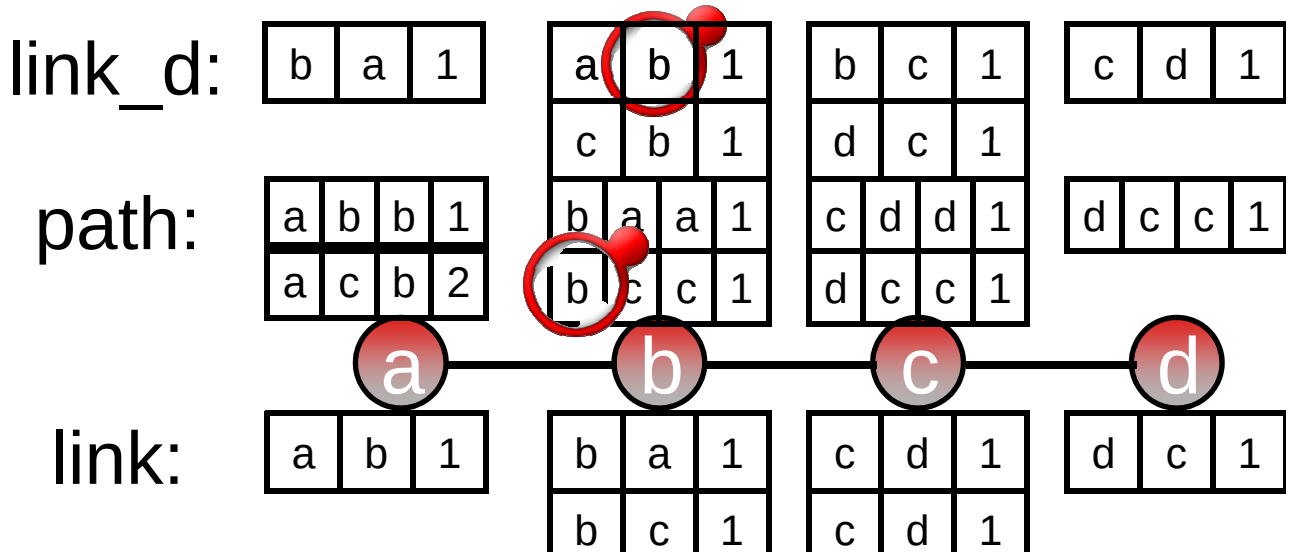
PARTITION SPECS INDUCE COMMUNICATION

- link(@X,Y)

Localization Rewrite

- path(@X,Y,Y,C) :- link(@X,Y,C)
  - link_d(X,@Y,C) :- link(@X,Y,C)
  - path(@X,Z,Y,C+D) :- link_d(X,@Y,C), path(@Y,Z,N,D)

THIS IS DISTANCE VECTOR



FLEXIBLE M.R. SCHEDULING

- Konwinski/Zaharia's LATE protocol:
  - 3 lines pseudocode, 5 rules in Overlog
  - vs. 800-line patchfile
    - ~200 lines implement LATE
    - other ~600 lines modify 42 Java files
  - comparable results