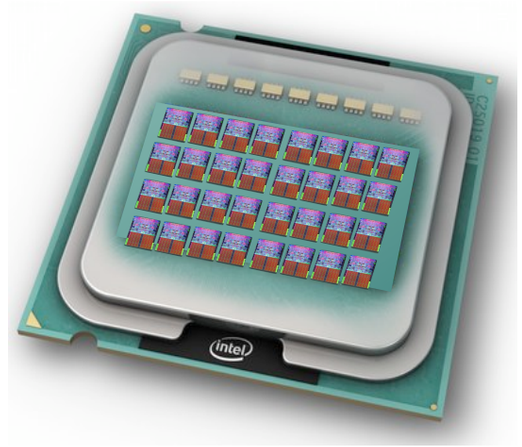
CHALLENGE 12

DECLARATIVE LANGUAGES

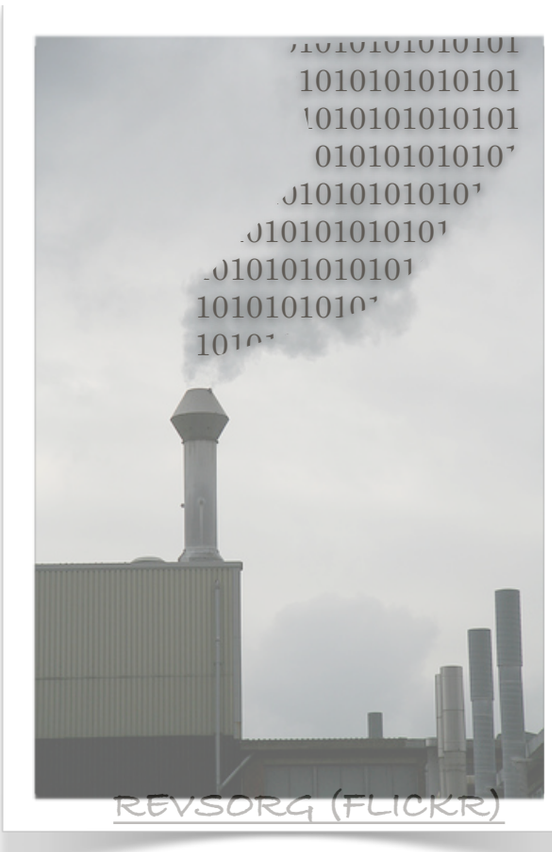
PLAY THE FIELD

JOSEPH M HELLERSTEIN

CHANGE AFOOT



- datacenters/virtualization
- manycore
- big data (industrial revolution!)



- => ready to revisit the whole SW stack

JIM GRAY CHALLENGE #12

JIM GRAY CHALLENGE #12

Automatic Programming

Do What I Mean (not 100$ Line of code!, no programming bugs)

The holy grail of programming languages & systems

12. Devise a specification language or UI
    1. That is easy for people to express designs (1,000x easier),
    2. That computers can compile, and
    3. That can describe all applications (is complete).

- System should "reason" about application
  - Ask about exception cases.
  - Ask about incomplete specification.
  - But not be onerous.
- This already exists in domain-specific areas.
  (i.e. 2 out of 3 already exists)
- An imitation game for a programming staff.

JIM GRAY CHALLENGE #12

Automatic Programming

Do What I Mean (not 100$ Line of code!, no programming bugs)

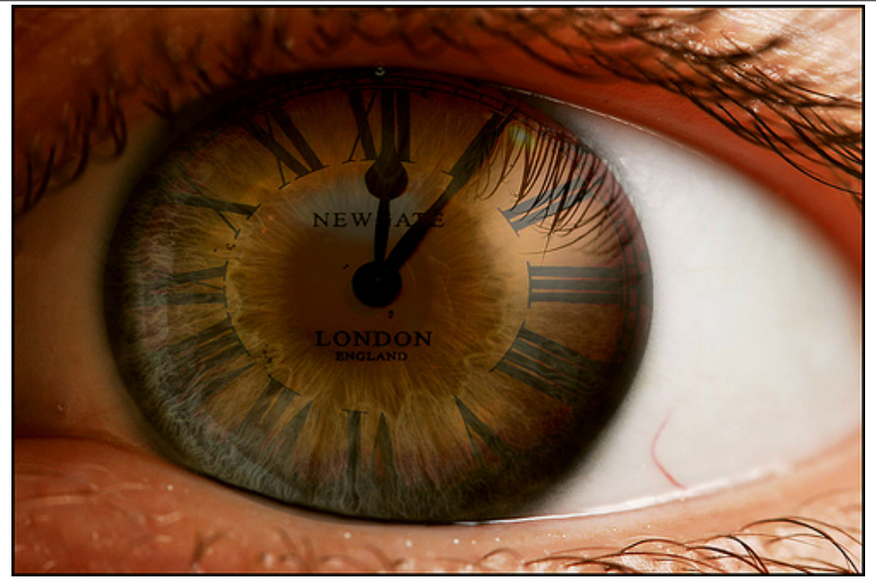
The holy grail of programming languages & systems

12. Devise a specification language or UI
    1. That is easy for people to express designs (1,000x easier),
    2. That computers can compile, and
    3. That can describe all applications (is complete).

- System should "reason" about application
  - Ask about exception cases.
  - Ask about incomplete specification
  - Build bridges
- This already exists in domain-specific areas.
  (i.e. 2 out of 3 already exists)
- An imitation game for a programming staff.

44

- now add auto-parallelization



TIMING!

- CS ready to revisit programming
- data + parallel + distributed
- we have their attention!
- and things are already happening...

DATALOG? OMG!

- been getting around in last 5 years!
  - despite god-awful syntax
  - a “domain-specific” language that’s playing the field
- our work on *declarative networking*, plus:
  - compilers (bddbddb, stanford ’05)
  - natural language processing (dyna, hopkins, ’05)
  - security protocols (AT&T, Stanford, ’01)
  - information extraction (wisc/yahoo ’07)
  - modular robotics (meld, cmu, ’07)
  - machine learning (berkeley/cmu ’08)

P2: DECLARATIVE INTERNET OVERLAYS

- chord routing, with:
  - multiple successors
  - stabilization
  - optimized finger maintenance
  - failure detection
- 48 “overlog” rules

```
/* The base tuples */
materialize(node, infinity, 1, keys(1)).
materialize(finger, 180, 160, keys(2)).
materialize(bestSucc, infinity, 1, keys(1)).
materialize(succDist, 10, 100, keys(2)).
materialize(succ, 10, 100, keys(2)).
materialize(pred, infinity, 100, keys(1)).
materialize(succCount, infinity, 1, keys(1)).
materialize(join, 10, 5, keys(1)).
materialize(landmark, infinity, 1, keys(1)).
materialize(fFix, infinity, 160, keys(2)).
materialize(nextFingerFix, infinity, 1, keys(1)).
materialize(pingNode, 10, infinity, keys(2)).
materialize(pendingPing, 10, infinity, keys(2)).

/** Lookups */
watch(lookupResults).
watch(lookup).

l1 lookupResults@R(R,K,S,SI,E) :- node@NI(NI,N),
                                  lookup@NI(NI,K,R,E), bestSucc@NI(NI,S,SI),
                                  K in (N,S].
l2 bestLookupDist@NI(NI,K,R,E,min<D>) :- node@NI(NI,N),
                                         lookup@NI(NI,K,R,E), finger@NI(NI,I,B,BI),
                                         D:=K - B - 1, B in (N,K).
l3 lookup@BI(min<BI>,K,R,E) :- node@NI(NI,N),
                               bestLookupDist@NI(NI,K,R,E,D),
                               finger@NI(NI,I,B,BI), D == K - B - 1,
                               B in (N,K).

/** Neighbor Selection */
n1 succEvent@NI(NI,S,SI) :- succ@NI(NI,S,SI).
n2 succDist@NI(NI,S,D) :- node@NI(NI,N),
                          succEvent@NI(NI,S,SI), D:=S - N - 1.
n3 bestSuccDist@NI(NI,min<D>) :- succDist@NI(NI,S,D).
n4 bestSucc@NI(NI,S,SI) :- succ@NI(NI,S,SI),
                           bestSuccDist@NI(NI,D), node@NI(NI,N),
                           D == S - N - 1.
n5 finger@NI(NI,0,S,SI) :- bestSucc@NI(NI,S,SI).

/** Successor eviction */
s1 succCount@NI(NI,count<*>) :- succ@NI(NI,S,SI).
s2 evictSucc@NI(NI) :- succCount@NI(NI,C), C > 2.
s3 maxSuccDist@NI(NI,max<D>) :- succ@NI(NI,S,SI),
                                node@NI(NI,N), evictSucc@NI(NI), D:=S - N - 1.
s4 delete succ@NI(NI,S,SI) :- node@NI(NI,N),
                              succ@NI(NI,S,SI), maxSuccDist@NI(NI,D),
                              D == S - N - 1.

/** Finger fixing */
f1 fFix@NI(NI,E,I) :- periodic@NI(NI,E,10),
                      nextFingerFix@NI(NI,I).
f2 fFixEvent@NI(NI,E,I) :- fFix@NI(NI,E,I).
f3 lookup@NI(NI,K,NI,E) :- fFixEvent@NI(NI,E,I),
                           node@NI(NI,N), K:=1I << I + N.
f4 eagerFinger@NI(NI,I,B,BI) :- fFix@NI(NI,E,I),
                                lookupResults@NI(NI,K,B,BI,E).
f5 finger@NI(NI,I,B,BI) :- eagerFinger@NI(NI,I,B,BI).
f6 eagerFinger@NI(NI,I,B,BI) :- node@NI(NI,N),
                                eagerFinger@NI(NI,I1,B,BI),
                                I:=I1 + 1, K:=1I << I + N,
                                K in (N,B), BI != NI.
f7 delete fFix@NI(NI,E,I1) :- eagerFinger@NI(NI,I,B,BI),
                              fFix@NI(NI,E,I1), I > 0, I1 == I - 1.
f8 nextFingerFix@NI(NI,0) :- eagerFinger@NI(NI,I,B,BI),
                             ((I == 159) || (BI == NI)).
f9 nextFingerFix@NI(NI,I) :- node@NI(NI,N),
                             eagerFinger@NI(NI,I1,B,BI),
                             I:=I1 + 1, K:=1I << I + N, K in (B,N),
                             NI != BI.

/** Churn Handling */
c1 joinEvent@NI(NI,E) :- join@NI(NI,E).
c2 joinReq@LI(LI,N,NI,E) :- joinEvent@NI(NI,E),
                            node@NI(NI,N), landmark@NI(NI,LI), LI != "-".
c3 succ@NI(NI,N,NI) :- landmark@NI(NI,LI),
                       joinEvent@NI(NI,E), node@NI(NI,N), LI == "-".
c4 lookup@LI(LI,N,NI,E) :- joinReq@LI(LI,N,NI,E).
c5 succ@NI(NI,S,SI) :- join@NI(NI,E),
                       lookupResults@NI(NI,K,S,SI,E).

/** Stabilization */
sb1 stabilize@NI(NI,E) :- periodic@NI(NI,E,15).
sb2 stabilizeRequest@SI(SI,NI) :- stabilize@NI(NI,E),
                                  bestSucc@NI(NI,S,SI).
sb3 sendPredecessor@PI1(PI1,P,PI) :- stabilizeRequest@NI(NI,PI1),
                                     pred@NI(NI,P,PI), PI != "-".
sb4 succ@NI(NI,P,PI) :- node@NI(NI,N), sendPredecessor@NI(NI,P,PI),
                        bestSucc@NI(NI,S,SI), P in (N,S).
sb5 sendSuccessors@SI(SI,NI) :- stabilize@NI(NI,E),
                                succ@NI(NI,S,SI).
sb6 returnSuccessor@PI(PI,S,SI) :- sendSuccessors@NI(NI,PI),
                                   succ@NI(NI,S,SI).
sb7 succ@NI(NI,S,SI) :- returnSuccessor@NI(NI,S,SI).
sb7 notifyPredecessor@SI(SI,N,NI) :- stabilize@NI(NI,E),
                                     node@NI(NI,N), succ@NI(NI,S,SI).
sb8 pred@NI(NI,P,PI) :- node@NI(NI,N), notifyPredecessor@NI(NI,P,PI),
                        pred@NI(NI,P1,PI1), ((PI1 == "-") || (P in (P1,N))).

/** Connectivity Monitoring */
cm0 pingEvent@NI(NI,E) :- periodic@NI(NI,E,5).
cm1 pendingPing@NI(NI,PI,E) :- pingEvent@NI(NI,E),
                               pingNode@NI(NI,PI).
cm2 pingReq@PI(PI,NI,E) :- pendingPing@NI(NI,PI,E).
cm3 delete pendingPing@NI(NI,PI,E) :- pingResp@NI(NI,PI,E).
cm4 pingResp@RI(RI,NI,E) :- pingReq@NI(NI,RI,E).
cm5 pingNode@NI(NI,SI) :- succ@NI(NI,S,SI), SI != NI.
cm6 pingNode@NI(NI,PI) :- pred@NI(NI,P,PI), PI != NI, PI != "-".
cm7 succ@NI(NI,S,SI) :- succ@NI(NI,S,SI), pingResp@NI(NI,SI,E).
cm8 pred@NI(NI,P,PI) :- pred@NI(NI,P,PI), pingResp@NI(NI,PI,E).
cm9 pred@NI(NI,"-","-") :- pingEvent@NI(NI,E),
                           pendingPing@NI(NI,PI,E), pred@NI(NI,P,PI).
```

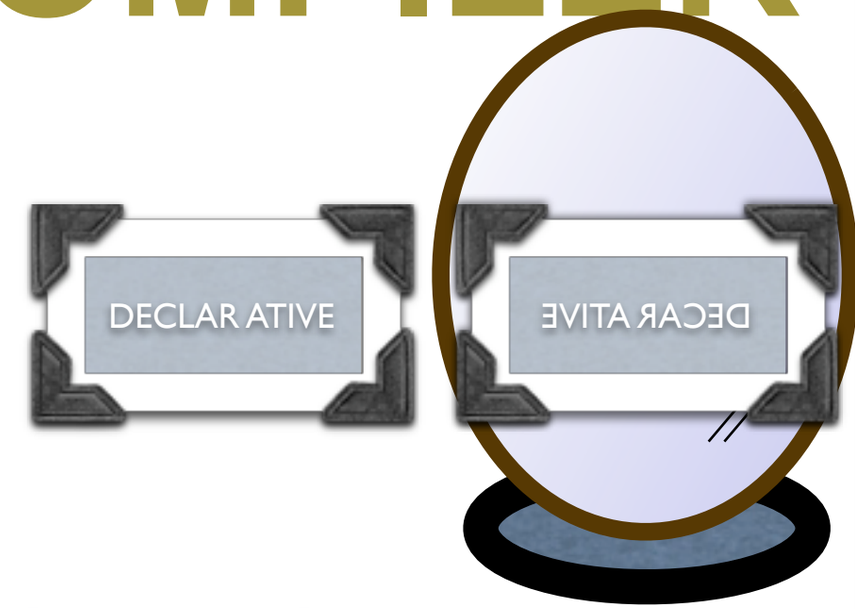
Boon Loo, et al, SOSP ‘05

```
/* The base tuples */
materialize(node, infinity, 1, keys(1)).
materialize(finger, 180, 160, keys(2)).
materialize(bestSucc, infinity, 1, keys(1)).
materialize(succDist, 10, 100, keys(2)).
materialize(succ, 10, 100, keys(2)).
materialize(pred, infinity, 100, keys(1)).
materialize(succCount, infinity, 1, keys(1)).
materialize(join, 10, 5, keys(1)).
materialize(landmark, infinity, 1, keys(1)).
materialize(fFix, infinity, 160, keys(2)).
materialize(nextFingerFix, infinity, 1, keys(1)).
materialize(pingNode, 10, infinity, keys(2)).
materialize(pendingPing, 10, infinity, keys(2)).

/** Lookups */
watch(lookupResults).
watch(lookup).

l1 lookupResults@R(R,K,S,SI,E) :- node@NI(NI,N),
                                  lookup@NI(NI,K,R,E), bestSucc@NI(NI,S,SI),
                                  K in (N,S].
l2 bestLookupDist@NI(NI,K,R,E,min<D>) :- node@NI(NI,N),
                                         lookup@NI(NI,K,R,E), finger@NI(NI,I,B,BI),
                                         D:=K - B - 1, B in (N,K).
l3 lookup@BI(min<BI>,K,R,E) :- node@NI(NI,N),
                               bestLookupDist@NI(NI,K,R,E,D),
                               finger@NI(NI,I,B,BI), D == K - B - 1,
                               B in (N,K).

/** Neighbor Selection */
n1 succEvent@NI(NI,S,SI) :- succ@NI(NI,S,SI).
n2 succDist@NI(NI,S,D) :- node@NI(NI,N),
                          succEvent@NI(NI,S,SI), D:=S - N - 1.
n3 bestSuccDist@NI(NI,min<D>) :- succDist@NI(NI,S,D).
n4 bestSucc@NI(NI,S,SI) :- succ@NI(NI,S,SI),
                           bestSuccDist@NI(NI,D), node@NI(NI,N),
                           D == S - N - 1.
n5 finger@NI(NI,0,S,SI) :- bestSucc@NI(NI,S,SI).

/** Successor eviction */
s1 succCount@NI(NI,count<*>) :- succ@NI(NI,S,SI).
s2 evictSucc@NI(NI) :- succCount@NI(NI,C), C > 2.
s3 maxSuccDist@NI(NI,max<D>) :- succ@NI(NI,S,SI),
                                node@NI(NI,N), evictSucc@NI(NI), D:=S - N - 1.
s4 delete succ@NI(NI,S,SI) :- node@NI(NI,N),
                              succ@NI(NI,S,SI), maxSuccDist@NI(NI,D),
                              D == S - N - 1.

/** Finger fixing */
f1 fFix@NI(NI,E,I) :- periodic@NI(NI,E,10),
                      nextFingerFix@NI(NI,I).
f2 fFixEvent@NI(NI,E,I) :- fFix@NI(NI,E,I).
f3 lookup@NI(NI,K,NI,E) :- fFixEvent@NI(NI,E,I),
                           node@NI(NI,N), K:=1I << I + N.
f4 eagerFinger@NI(NI,I,B,BI) :- fFix@NI(NI,E,I),
                                lookupResults@NI(NI,K,B,BI,E).
f5 finger@NI(NI,I,B,BI) :- eagerFinger@NI(NI,I,B,BI).
f6 eagerFinger@NI(NI,I,B,BI) :- node@NI(NI,N),
                                eagerFinger@NI(NI,I1,B,BI),
                                I:=I1 + 1, K:=1I << I + N,
                                K in (N,B), BI != NI.
f7 delete fFix@NI(NI,E,I1) :- eagerFinger@NI(NI,I,B,BI),
                              fFix@NI(NI,E,I1), I > 0, I1 == I - 1.
f8 nextFingerFix@NI(NI,0) :- eagerFinger@NI(NI,I,B,BI),
                             ((I == 159) || (BI == NI)).
f9 nextFingerFix@NI(NI,I) :- node@NI(NI,N),
                             eagerFinger@NI(NI,I1,B,BI),
                                I:=I1 + 1, K:=1I << I + N, K in (B,N),
                                NI != BI.

/** Churn Handling */
c1 joinEvent@NI(NI,E) :- join@NI(NI,E).
c2 joinReq@LI(LI,N,NI,E) :- joinEvent@NI(NI,E),
                            node@NI(NI,N), landmark@NI(NI,LI), LI != "-".
c3 succ@NI(NI,N,NI) :- landmark@NI(NI,LI),
                       joinEvent@NI(NI,E), node@NI(NI,N), LI == "-".
c4 lookup@LI(LI,N,NI,E) :- joinReq@LI(LI,N,NI,E).
c5 succ@NI(NI,S,SI) :- join@NI(NI,E),
                       lookupResults@NI(NI,K,S,SI,E).

/** Stabilization */
sb1 stabilize@NI(NI,E) :- periodic@NI(NI,E,15).
sb2 stabilizeRequest@SI(SI,NI) :- stabilize@NI(NI,E),
                                  bestSucc@NI(NI,S,SI).
sb3 sendPredecessor@PI1(PI1,P,PI) :- stabilizeRequest@NI(NI,PI1),
                                     pred@NI(NI,P,PI), PI != "-".
sb4 succ@NI(NI,P,PI) :- node@NI(NI,N), sendPredecessor@NI(NI,P,PI),
                        bestSucc@NI(NI,S,SI), P in (N,S).
sb5 sendSuccessors@SI(SI,NI) :- stabilize@NI(NI,E),
                                succ@NI(NI,S,SI).
sb6 returnSuccessor@PI(PI,S,SI) :- sendSuccessors@NI(NI,PI),
                                   succ@NI(NI,S,SI).
sb7 succ@NI(NI,S,SI) :- returnSuccessor@NI(NI,S,SI).
sb7 notifyPredecessor@SI(SI,N,NI) :- stabilize@NI(NI,E),
                                     node@NI(NI,N), succ@NI(NI,S,SI).
sb8 pred@NI(NI,P,PI) :- node@NI(NI,N), notifyPredecessor@NI(NI,P,PI),
                        pred@NI(NI,P1,PI1), ((PI1 == "-") || (P in (P1,N))).

/** Connectivity Monitoring */
cm0 pingEvent@NI(NI,E) :- periodic@NI(NI,E,5).
cm1 pendingPing@NI(NI,PI,E) :- pingEvent@NI(NI,E),
                               pingNode@NI(NI,PI).
cm2 pingReq@PI(PI,NI,E) :- pendingPing@NI(NI,PI,E).
cm3 delete pendingPing@NI(NI,PI,E) :- pingResp@NI(NI,PI,E).
cm4 pingResp@RI(RI,NI,E) :- pingReq@NI(NI,RI,E).
cm5 pingNode@NI(NI,SI) :- succ@NI(NI,S,SI), SI != NI.
cm6 pingNode@NI(NI,PI) :- pred@NI(NI,P,PI), PI != NI, PI != "-".
cm7 succ@NI(NI,S,SI) :- succ@NI(NI,S,SI), pingResp@NI(NI,SI,E).
cm8 pred@NI(NI,P,PI) :- pred@NI(NI,P,PI), pingResp@NI(NI,PI,E).
cm9 pred@NI(NI,"-","-") :- pingEvent@NI(NI,E),
                           pendingPing@NI(NI,PI,E), pred@NI(NI,P,PI).
```

Boon Loo, et al, SOSP ‘05

100x LESS CODE THAN MIT CHORD

EVITA RACED: OVERLOG METACOMPILER



- go to the mirror, boy
  - would *you* build a system with that?
- overlog optimizer in ~150 overlog rules
  - system r + histograms + magic + stratification checks
  - exercise: system r => cascades?
    - 24 hours, 27 rules
  - porting to Pig@yahoo

Tyson Condie, et al.
VLDB '08

OUR LEVERAGE?

- we actually can auto-parallelize datalog
  - “just” dataflow, but well beyond embarrassing

- good for decentralized (datacenter) kernels
  - paxos & commit, DHTs, multicast, ...

- obviously good for queries, etc.
  - and we’re tackling fancy machine learning

GETTING THERE

- reality check: everyone hates datalog
  - so toss is out!
- design a new language, for hackers
  - attractive syntax, tools
  - get the complexity in a box
  - integrate with the rest of the universe
  - go fast
  - exploit/sell what we know
- remember:
  - the language train only leaves once a decade!
  - our turn to drive

CHALLENGE 12

you may say i’m a dreamer

... but i’m not the only one