DECLARATIVE NETWORKING:

WHAT IS NEXT.

JOSEPH M HELLERSTEIN
U C BERKELEY

JOINT WORK

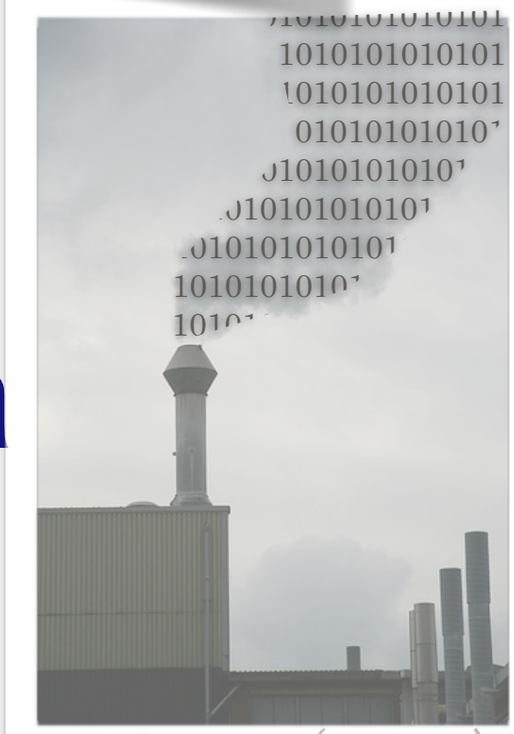
- David CHU Tyson CONDIE Lucian Popa Arsalan Tavakoli Scott Shenker Ion STOICA Berkeley
- Boon Thau LOO, UPenn
- David GAY Petros MANIATIS intel
- Timothy ROSCOE, ETH Zurich
- Raghu RAMAKRISHNAN, Minos GAROFALAKIS Yahoo!
- Carlos GUESTRIN, CMU
- Philip LEVIS, Stanford

TWO SOURCES OF FLUX

- internet revolution

  GENI, wireless sensors, overlay nets, datacenters

- industrial revolution of data



HOW DO WE HARNESS THE FLUX?

IN THIS TIME OF FLUX, WHAT CAN HARNESS AND ACCELERATE THE ENERGY AND INNOVATION.

WHAT IS THE FUTURE: DECLARATIVE NETWORKING

- WHAT: beyond the network *how*
  - topology specification. routing constraints. addressing by content
- who where why when
  - authentication. geolocation. consensus. forensics.
- NW data, reasoning and control
  - search. query. inference. movement.

THE EVOLUTION OF WHAT

- query (in) the network
- networks VIA queries
- queries, networks and uncertainty

- **WATCH THE SYNTHESIS**

TODAY

- WHY WHAT?
- SAY WHAT
- WHAT: HOW
- WHAT IS NEXT?
- WHAT’S IT TO YOU

WHY WHAT?

- ease, insight:
  - rapid prototyping & customization
  - fitness to many distributed tasks
  - uplevel ideas and their synergies
- towards safety
  - static checks
  - (synthesized) runtime checks

WHAT FIRST

- textbook routing protocols
  - internet-style and wireless (SIGCOMM 05, Berkeley/Wisconsin)
- distributed hash tables
  - chord overlay network (SOSP 05, Berkeley/Intel)

- distributed debugging
  - watchpoints, chandy-lamport snapshots (EuroSys 06, Intel/Rice/MPI)
- consensus
  - paxos (44 lines, TR 06, Harvard)

WHAT NOW

- wireless sensornets **dsn**
  - radio link estimation. geo routing. data collection. code dissemination. object tracking. localization. SNLog. (SenSys 07, Berkeley)
- secure networking SeNDLog. (NetDB07, MSR/Penn)
- flexible data replication PADRE (SOSP07 poster, Texas)
- mobile networks (MobiArch07, Penn)
- modular robotics MELD (IROS 07, CMU)

WHAT NEXT

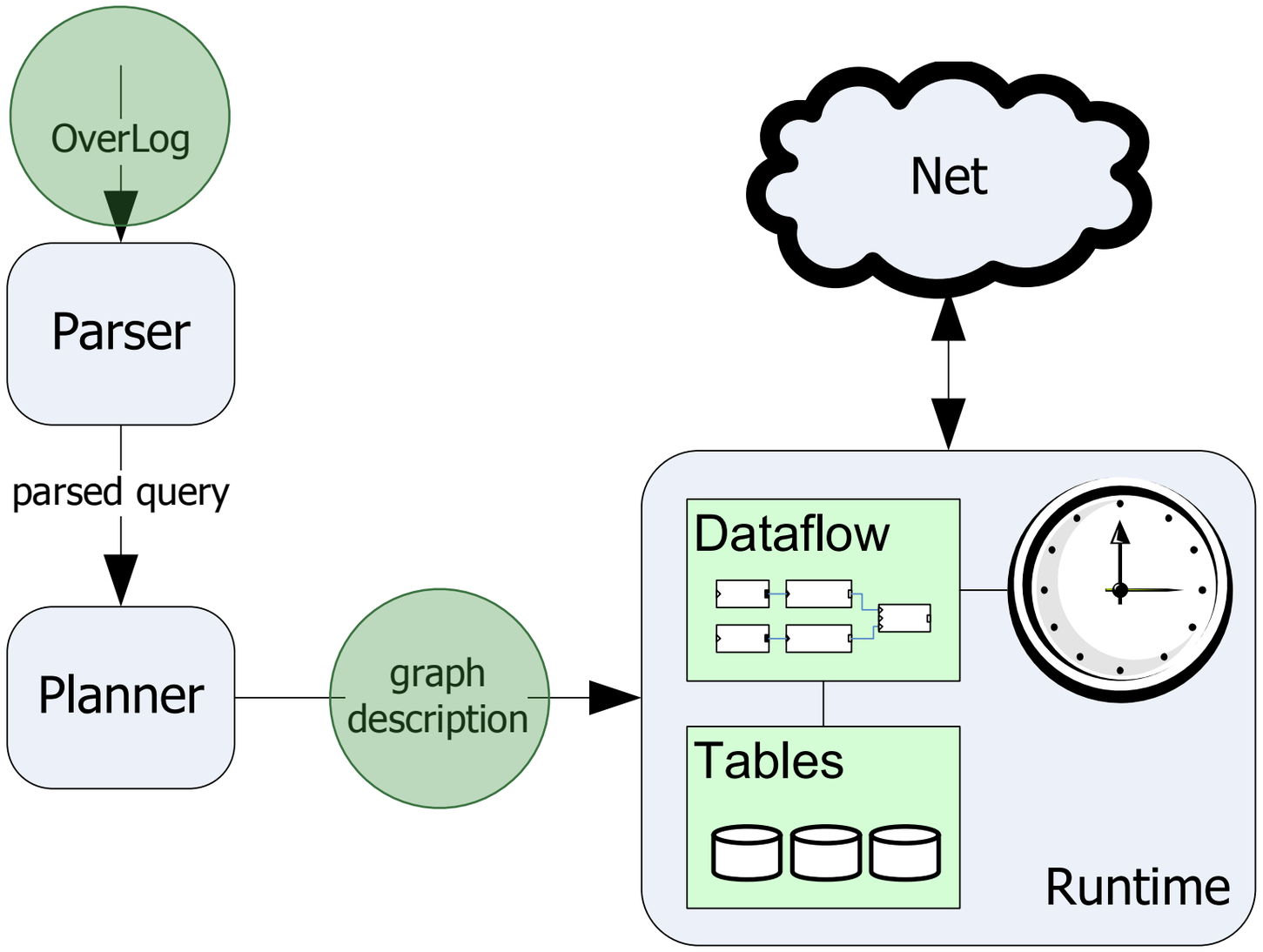
- metacompilation
  - declarative compilers for declarative languages (Berkeley/Intel)
- distributed inference
  - junction trees and loopy belief propagation (Berkeley/CMU)

TODAY

- WHY WHAT?
- SAY WHAT

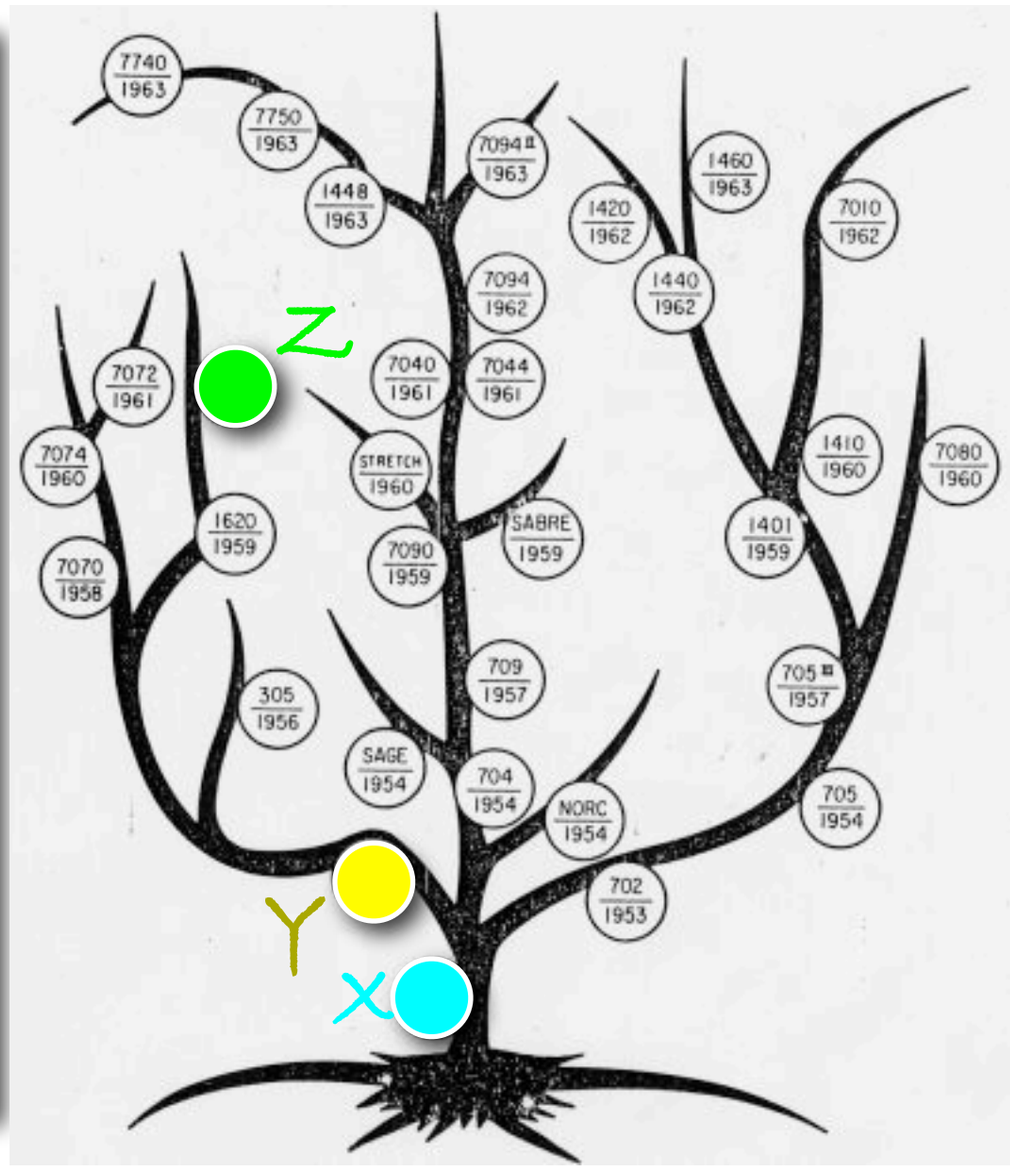
- WHAT: HOW
- WHAT IS NEXT?
- WHAT’S IT TO YOU

P2 @ 10,000 FEET



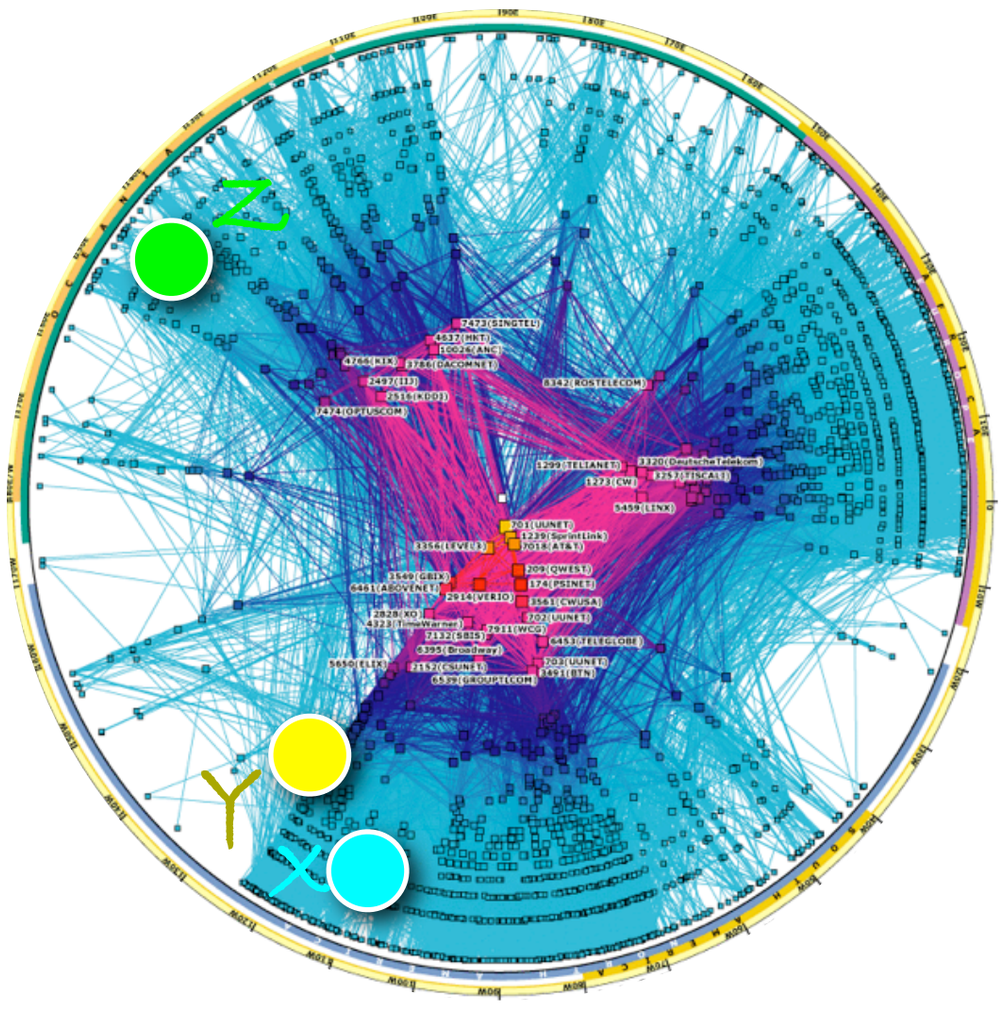
DUSTY OLD DATALOG

- parent(X,Y).
- anc(X,Y) :- parent(X,Y).
- anc(X,Z) :- parent(X,Y),
  anc(Y,Z).
- anc(X, s)?



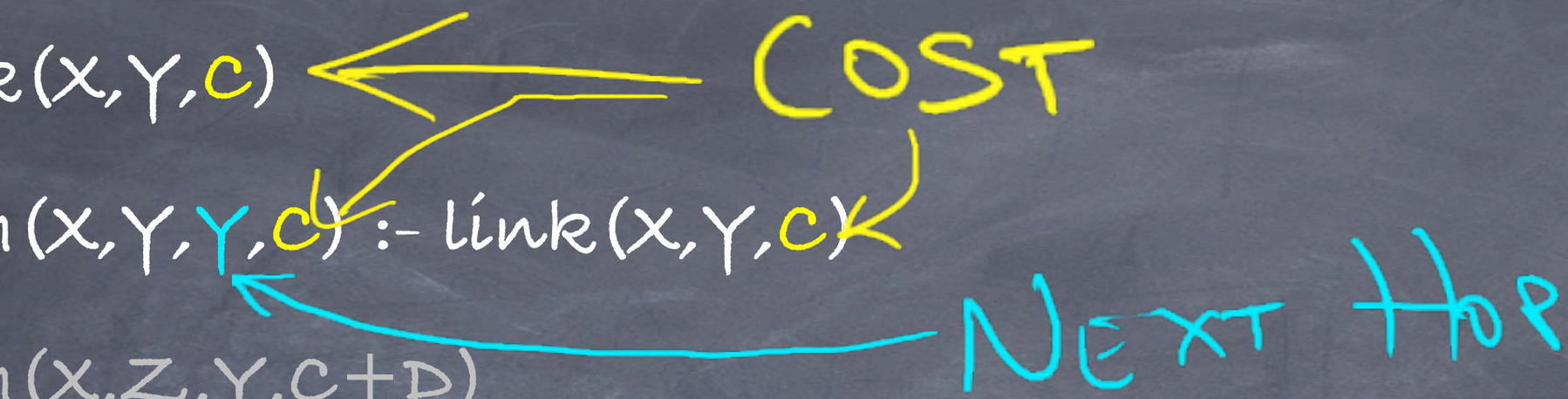
THE INTERNET CHANGES EVERYTHING?

- link(X,Y).
- path(X,Y) :- link(X,Y).
- path(X,Z) :- link(X,Y),
  path(Y,Z).
- path(X, s)?



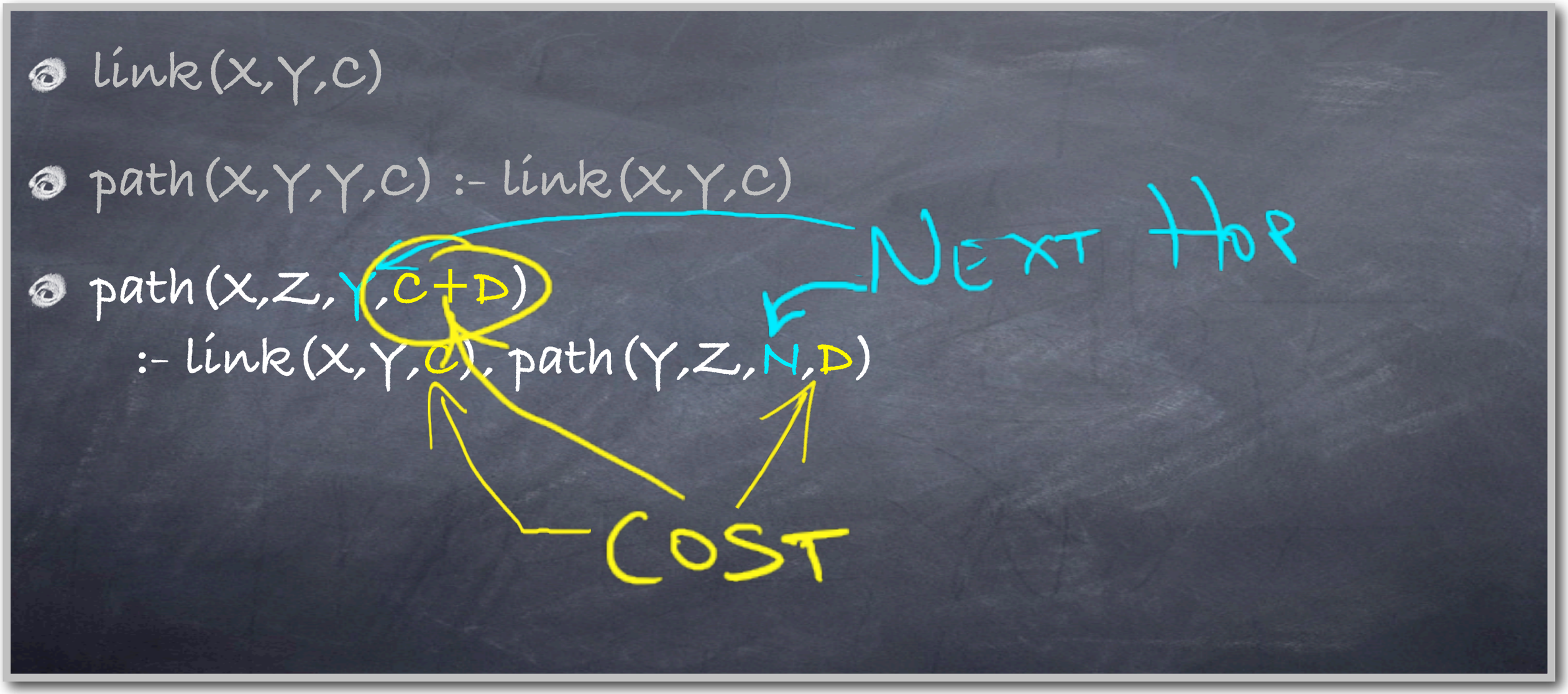
FORMING PATHS

- 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)



FORMING PATHS

- 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)



BEST PATHS

- link(X,Y)
- path(X,Y,Y,C) :- link(X,Y,C)
- path(X,Z,Y,C+D) :- link(X,Y,C), path(Y,Z,N,D)
- mincost(X,Z,min<C>) :- path(X,Z,Y,C)
- bestpath(X,Z,Y,C) :- path(X,Z,Y,C), mincost(X,Z,C)
- bestpath(src,D,Y,C)?

SO FAR...

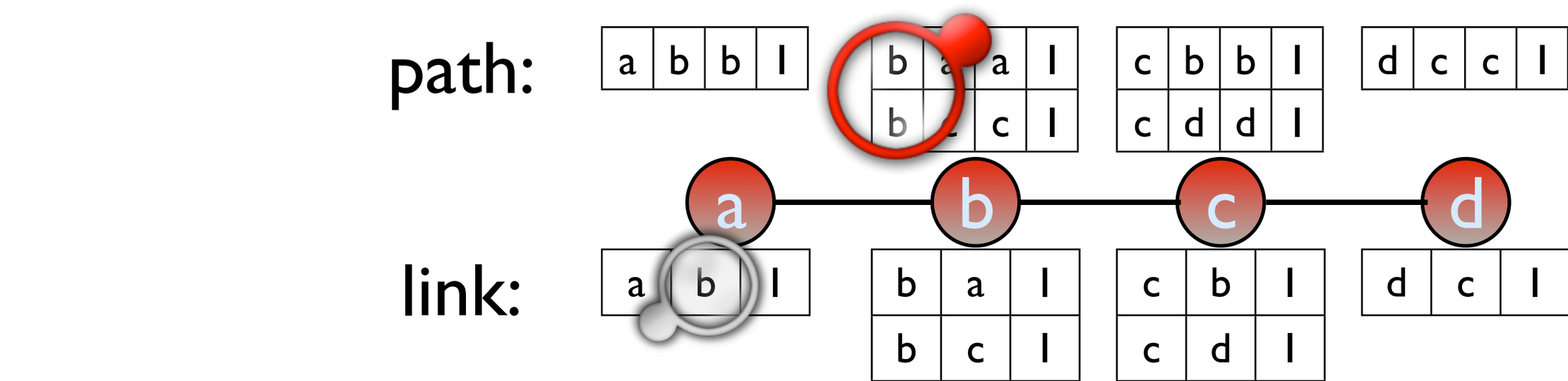
- logic for path-finding
- on the link DB in the sky
- but can this lead to protocols?

TOWARD DISTRIBUTION: DATA PARTITIONING

- logically global tables
- horizontally partitioned
- an address field per table
  - *location specifier: @*
  - data placement based on loc.spec.

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)

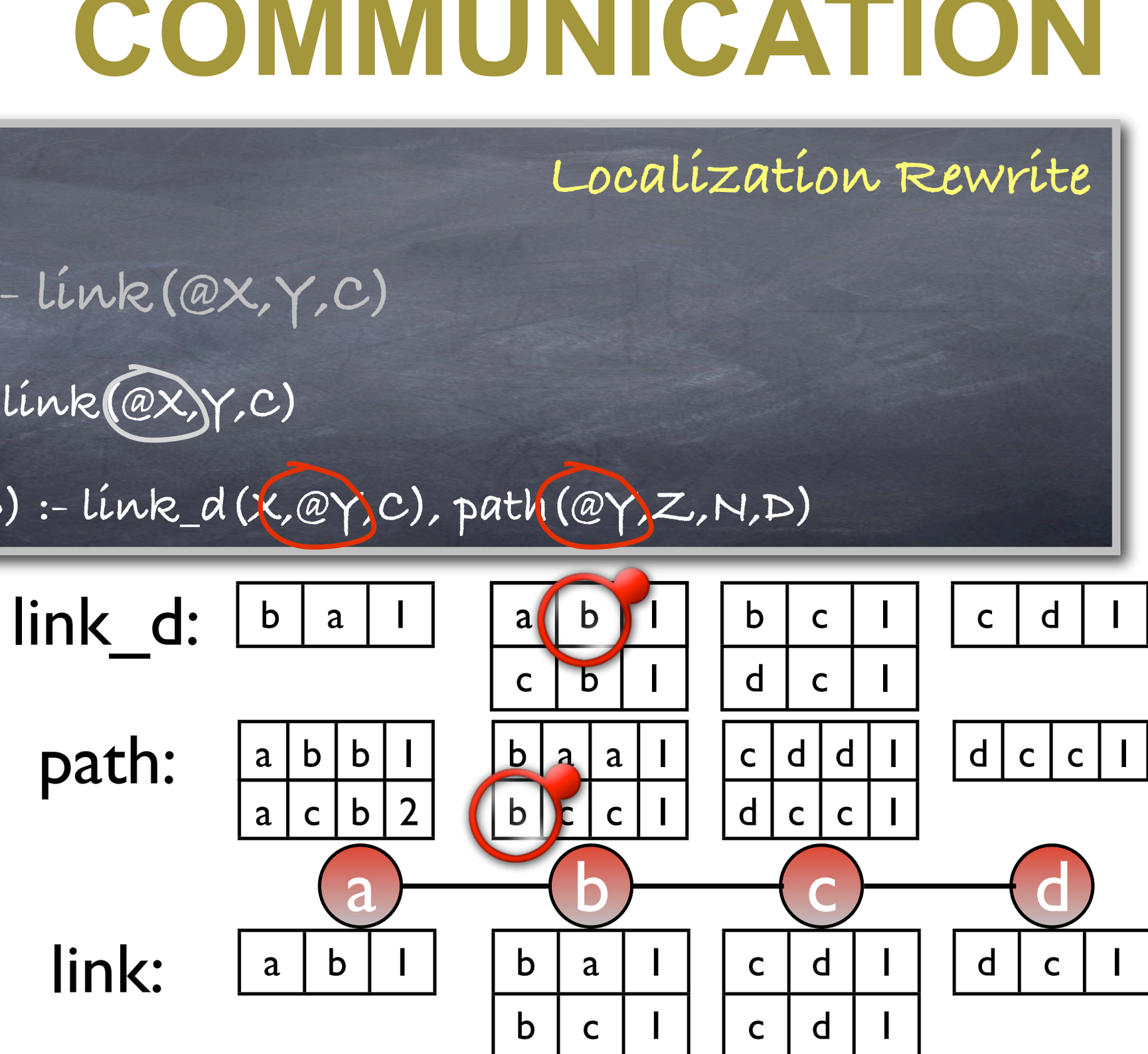


PARTITION SPECS INDUCE COMMUNICATION

Localization Rewrite

- link(@X,Y)
- 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)

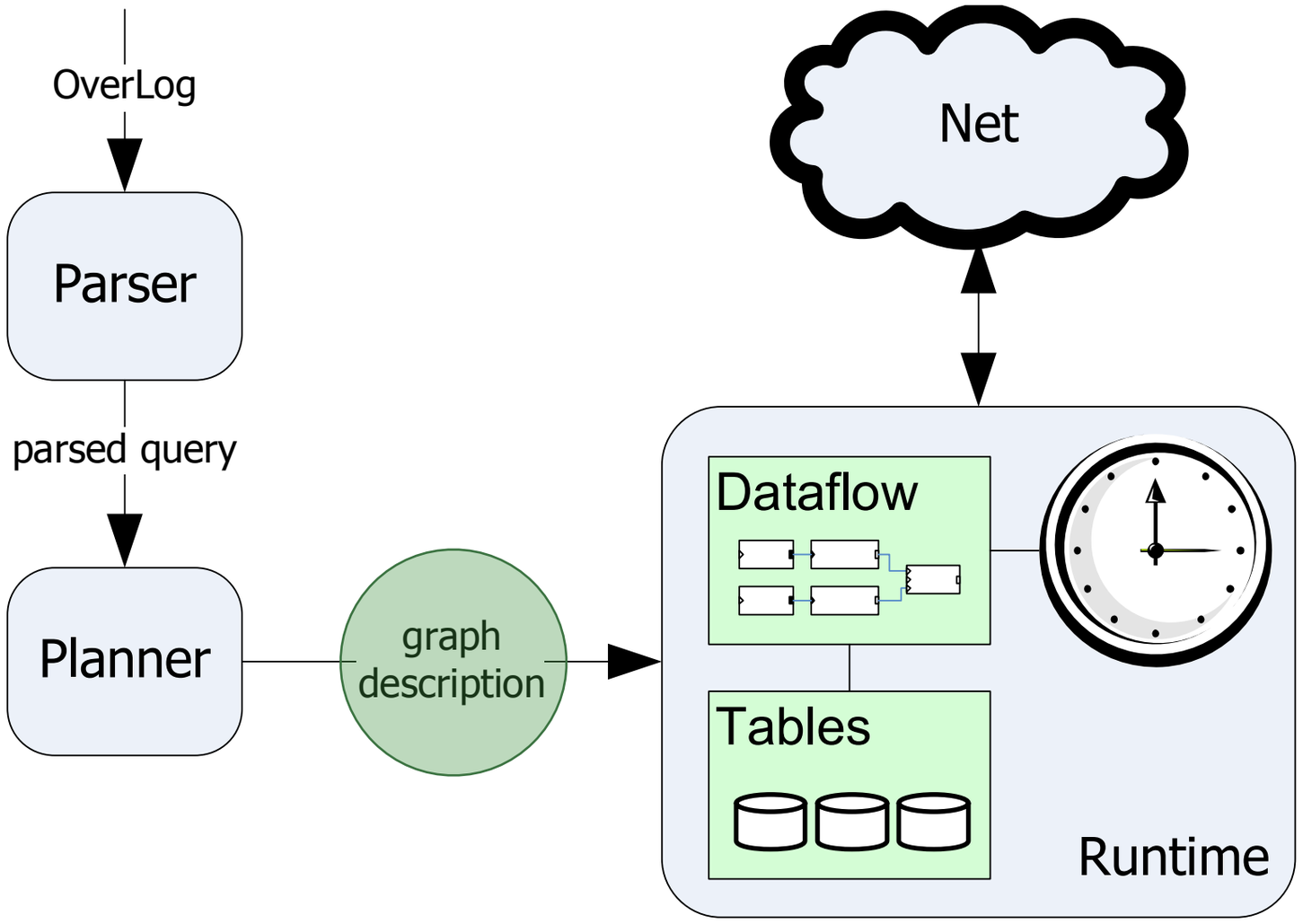
THIS IS DISTANCE VECTOR



TODAY

- WHY WHAT?
- SAY WHAT
- WHAT: HOW
- WHAT IS NEXT?
- WHAT’S IT TO YOU

P2 @ 10,000 FEET



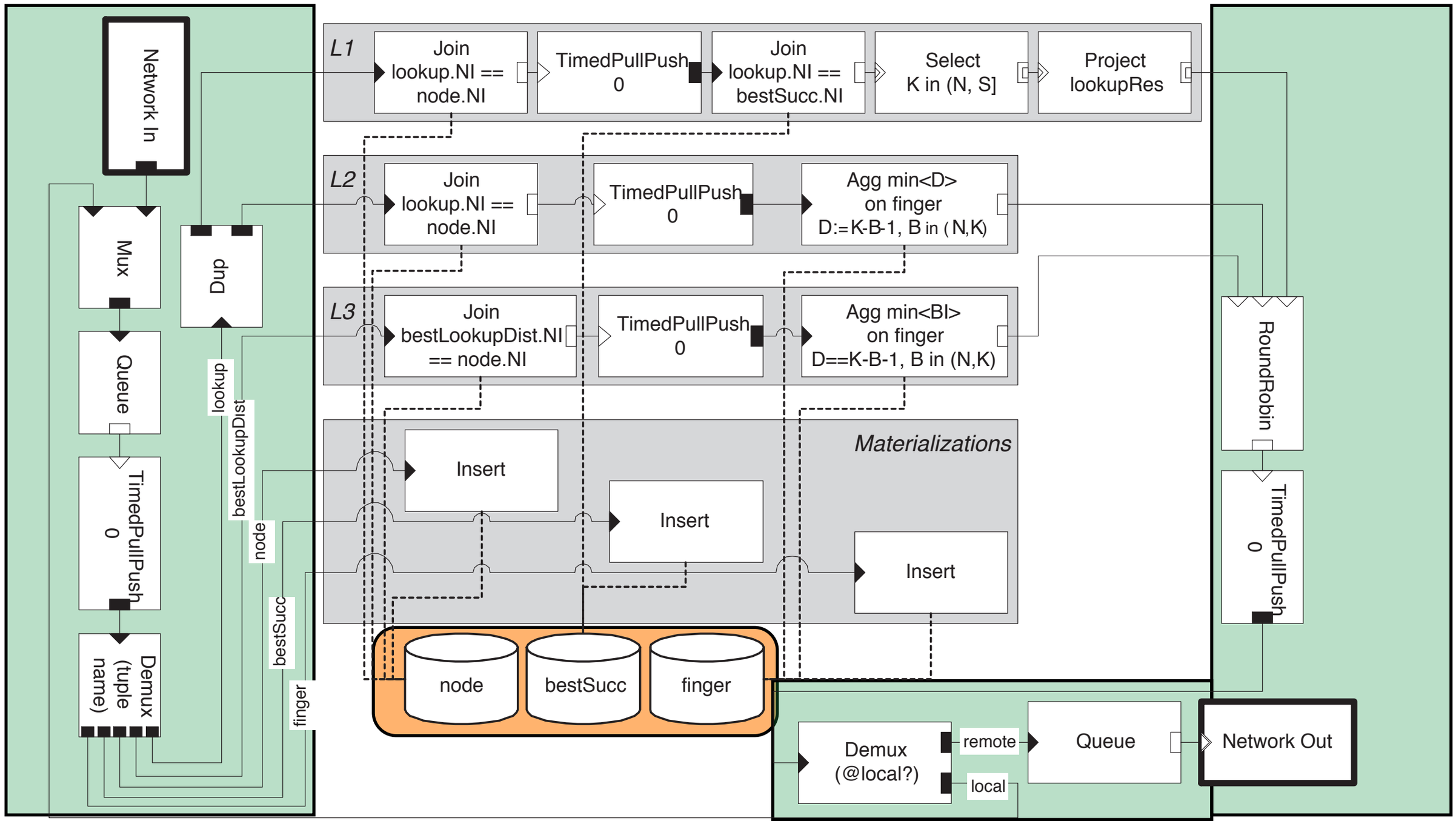
DATAFLOW EXAMPLE IN P2

- L1 lookupResults(@R,K,S,SI,E) :- node(@NI,N), lookup(@NI,K,R,E),
  bestSucc(@NI,S,SI),
  K in (N, S].

- L2 bestLookupDist(@NI,K,R,E,min<D>) :- node(@NI,N),
  lookup(@NI,K,R,E),
  finger(@NI,I,B,BI),
  D:=K-B-1, B in (N,K)

- L3 lookup(@min<BI>,K,R,E) :- node(@NI,N),
  bestLookupDist(@NI,K,R,E,D),
  finger(@NI,I,B,BI), D==K-B-1,
  B in (N,K).

DATAFLOW EXAMPLE IN P2

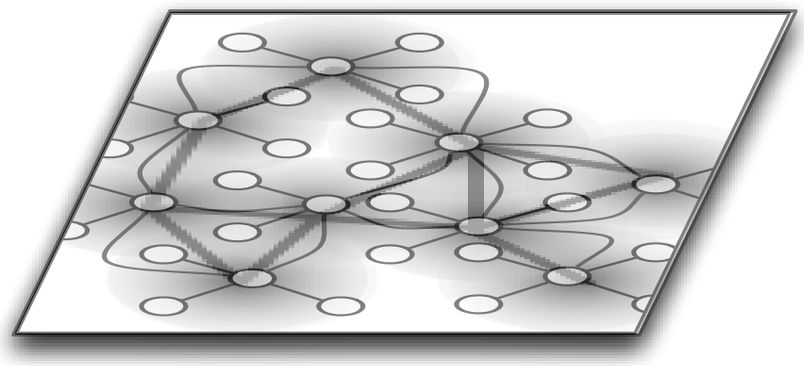


SOME LIKE IT HOW

```
macro _TCP(ip, port) {
  let q     := Queue("Source Data Q", size);
  let udp   := Udp("Test Udp", port);
  let cct   := CCT("Congestion Control Transmit", 1, 2048);
  let ccr   := CCR("Congestion Control Receive", 2048);
  let order := Order("Ordered delivery");

  input q; output order;

  q[0] -> Route(ip) -> Sequence("Sequence", 1) -> RDelivery("Reliable Delivery") ->
  cct -> [1]MarshalField("marshal data", 1)[0] -> udp;

  udp -> UnmarshalField("Unpack", 1) -> ccr -> order;
}

let b     := TimedPushSource("Data Generator", .01);
let tcp   := _TCP(129.0.0.1, 80);

b[0] -> tcp                          -> Queue("Q", 1);
```

OVERLAY NETWORKS

- distributed apps on the network
  - the game: track...
  - subset of participating nodes
  - names for participating nodes
  - multi-hop routing via other nodes



- many examples
  - VPNs, P2P, MS Exchange, Distributed Hash Tables...

DECLARATIVE OVERLAYS

- more challenging than simple routing
- must generate/maintain overlay topology
- message delivery, acks, failure detection, timeouts, periodic probes, etc…
- timer-based “built-in” event predicates:

```
ping(@D,S) :- periodic(@S,10), link(@S,D)
```

chord2r.plg Fri Mar 31 13:32:03 2006 1

```
/* 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,SI).
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,BI), 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).
```

P2-CHORD

chord routing, with:

multiple successors

stabilization

optimized finger maintenance

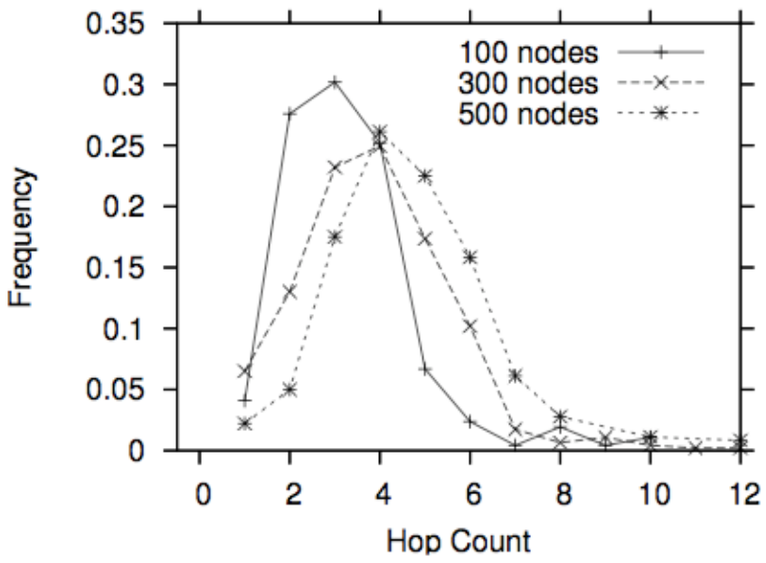
failure detection

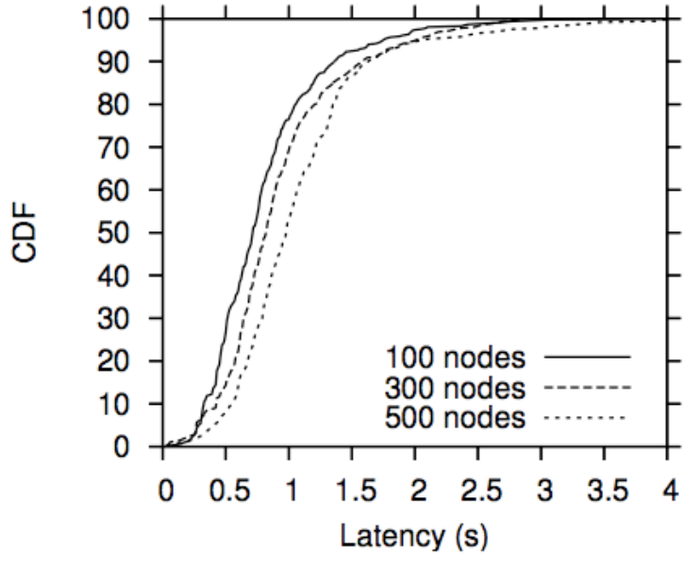
48 rules

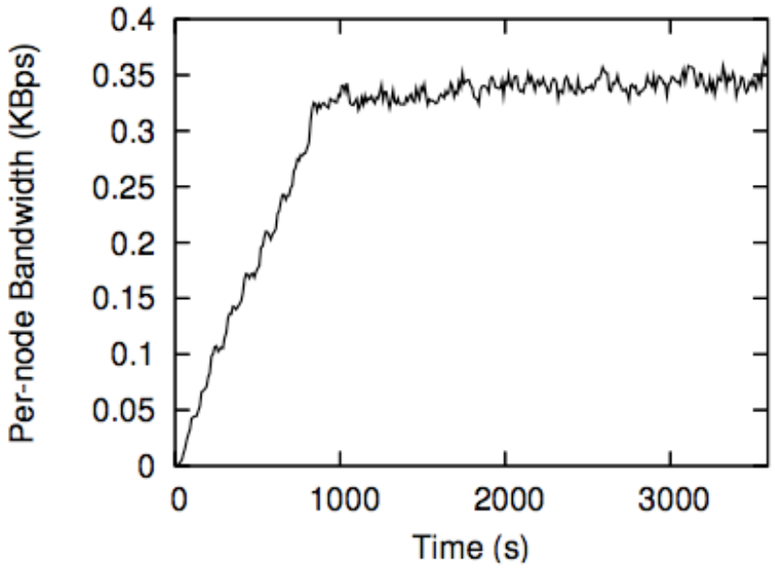
100x LESS CODE THAN MIT CHORD

P2-CHORD EVALUATION

- P2 nodes running Chord on 100 Emulab nodes:
  - Logarithmic lookup hop-count and state (“correct”)
  - Median lookup latency: 1-1.5s
  - BW-efficient: 300 bytes/s/node







CHURN PERFORMANCE

- P2-Chord:
  - P2-Chord@90mins: 99% consistency
  - P2-Chord@47mins: 96% consistency
  - P2-Chord@16min: 95% consistency
  - P2-Chord@8min: 79% consistency
- C++ Chord:
  - MIT-Chord@47mins: 99.9% consistency

DSN-TRICKLE

Levis, et al., Sensys 2004

Event	Action
τ Expires	Double τ, up to τ_h. Reset c, pick a new t.
t Expires	If $c < k$, transmit.
Receive same metadata	Increment c.
Receive newer metadata	Set τ to τ_l. Reset c, pick a new t.
Receive newer code	Set τ to τ_l. Reset c, pick a new t.
Receive older metadata	Send updates.

t is picked from the range $[\frac{\tau}{2}, \tau]$

Figure 12: Trickle Pseudocode.

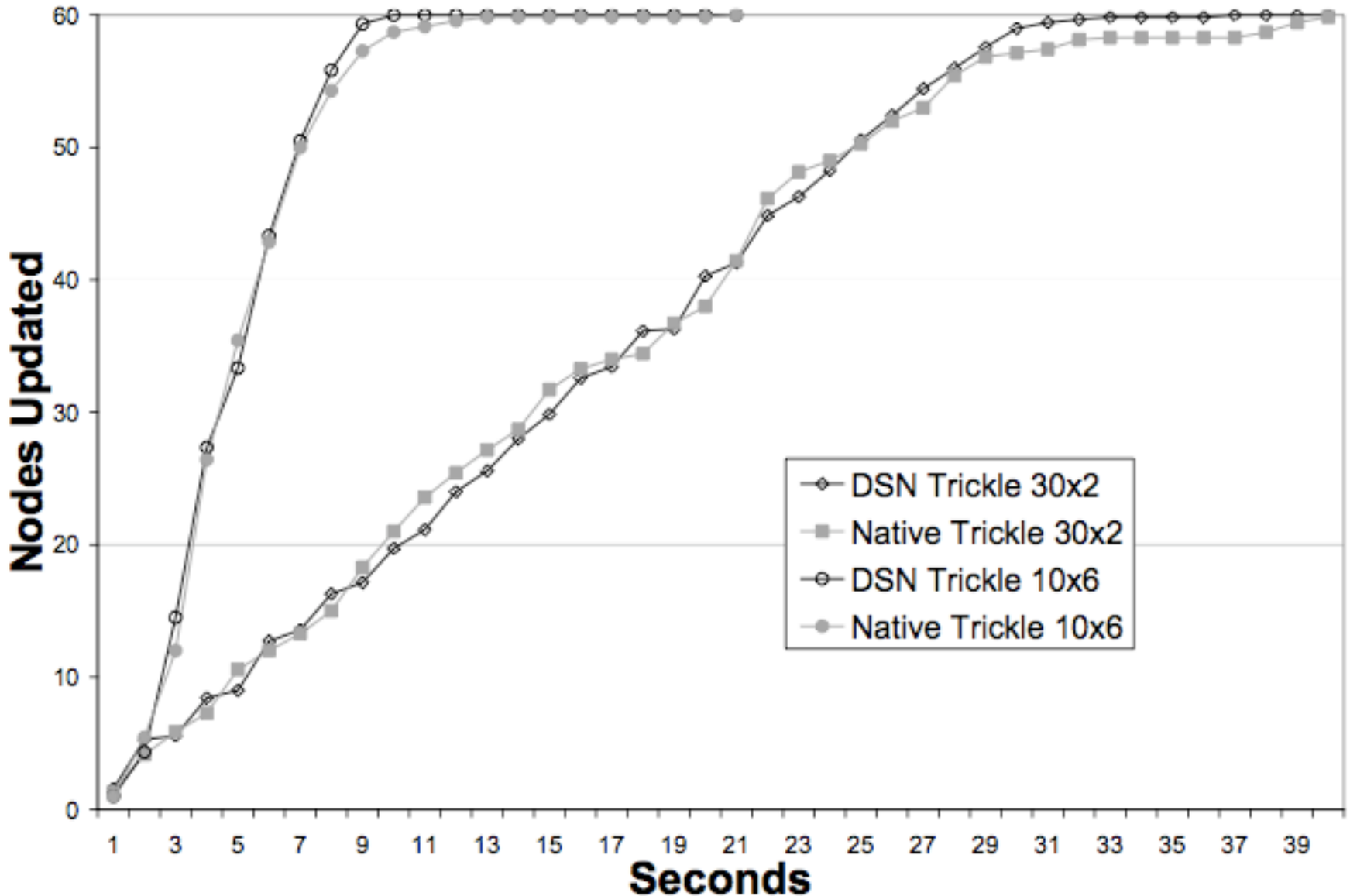
Chu, et al., Sensys 2007

```
1 % Tau expires:
2 % Double Tau up to tauHi. Reset C, pick a new T.
3 tauVal(@X,Tau*2) :- timer(@X,tauTimer,Tau), Tau*2 < tauHi.
4 tauVal(@X,tauHi) :- timer(@X,tauTimer,Tau), Tau*2 >= tauHi.
5 timer(@X,tTimer,T) :- tauVal(@X,TauVal), T =
      rand(TauVal/2,TauVal).
6 timer(@X,tauTimer,TauVal) :- tauVal(@X,TauVal).
7 msgCnt(@X,0) :- tauVal(@X,TauVal).
8
9 % T expires: If C< k, transmit.
10 msgVer(@*,Y,Oid,Ver) :- ver(@Y,Oid,Ver), timer(@Y,tTimer,_),
      msgCnt(@Y,C), C < k.
11
12 % Receive same metadata: Increment C.
13 msgCnt(@X,C++) :- msgVer(@X,Y,Oid,CurVer), ver(@X,Oid,CurVer),
      msgCnt(@X,C).
14
15 % Receive newer metadata:
16 % Set Tau to tauLow. Reset C, pick a new T.
17 tauVal(@X,tauLow) :- msgVer(@X,Y,Oid,NewVer),
      ver(@X,Oid,OldVer), NewVer > OldVer.
18
19 % Receive newer data:
20 % Set Tau to tauLow. Reset C, pick a new T.
21 tauVal(@X,tauLow) :- msgStore(@X,Y,Oid,NewVer,Obj),
      ver(@X,Oid,OldVer), NewVer > OldVer.
22
23 % Receive older metadata: Send updates.
24 msgStore(@*,X,Oid,NewVer,Obj) :- msgVer(@X,Y,Oid,OldVer),
      ver(@X,Oid,NewVer), NewVer > OldVer,
      store(@X,Oid,NewVer,Obj).
25
26 % Update version upon successfully receiving store
27 store(@X,Oid,NewVer,Obj) :- msgStore(@X,Y,Oid,NewVer,Obj).
      store(@X,Oid,OldVer,Obj), NewVer > OldVer.
28 ver(@X,Oid,NewVer,Obj) :- store(@X,Oid,NewVer,Obj).
```

Listing 3. Trickle Version Coherency

DSN vs NATIVE TRICKLE

	Native	DSN
LOC	560 (NesC)	13 rules, 25 lines

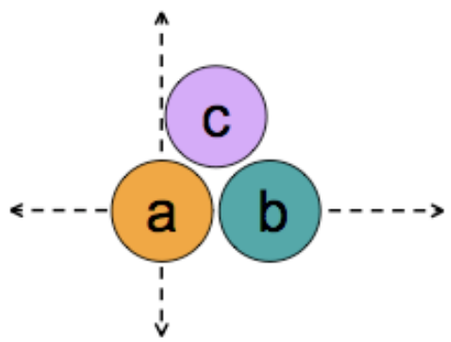


MOVING CATOMS IN MELD

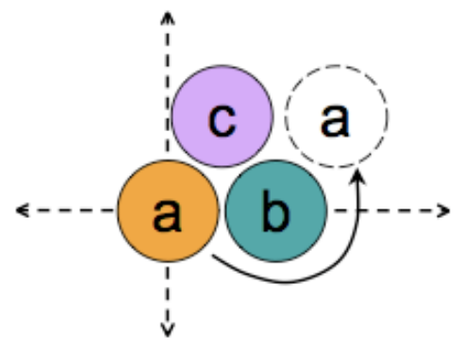
RULE1: `Dist(S,D):- At(S,P),`
P_d = destination(),
D = |P − P_d|,
D > *robot radius*.

RULE2: `Farther(S,T):-` Neighbor(S,T),
Dist(S,D_S),
Dist(T,D_T),
$D_S \geq D_T$.

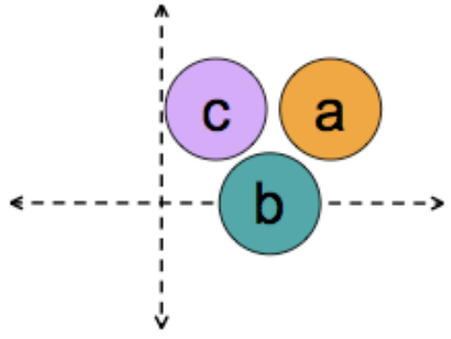
RULE3: `MoveAround(S,T,U):-` Farther(S,T),
Farther(S,U),
U ≠ T.



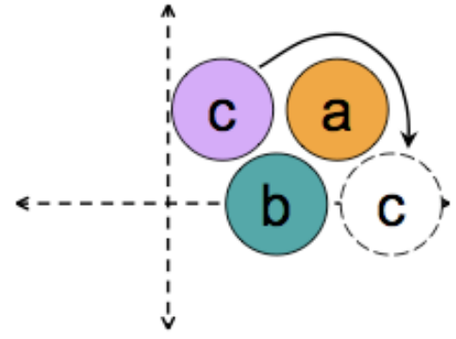
(i) starting location at origin



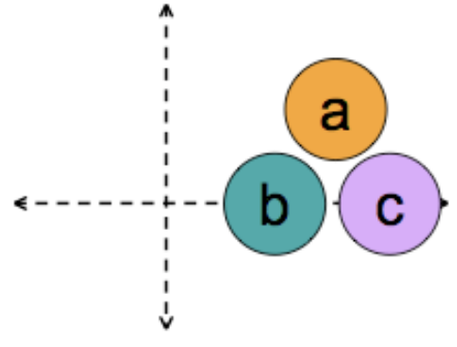
(ii) robot a moves around b



(iii) robot a finishes moving



iv) robot c now moves around a



(v) robot c finishes moving

TODAY

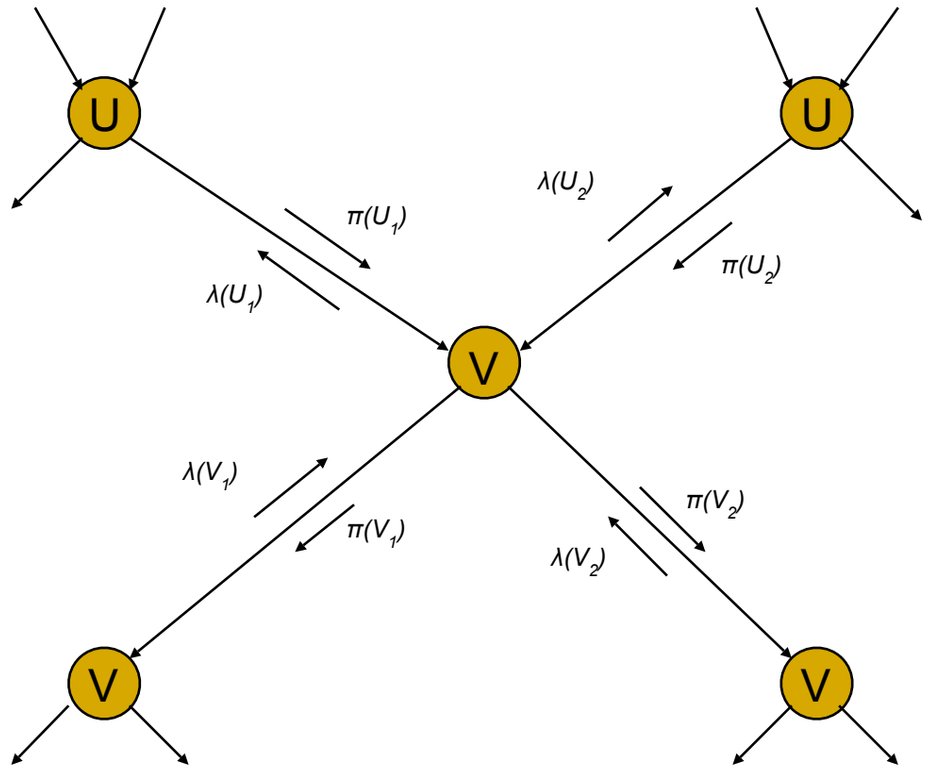
- WHY WHAT?
- SAY WHAT
- WHAT: HOW
- WHAT IS NEXT?
- WHAT’S IT TO YOU

DISTRIBUTED INFERENCE

- industrial revolution in data
  - data. networks. uncertainty.
- challenge: real-time information
  - despite uncertainty and acquisition cost
- applications
  - internet security, building control, disaster response, robotics. ANY distributed query.

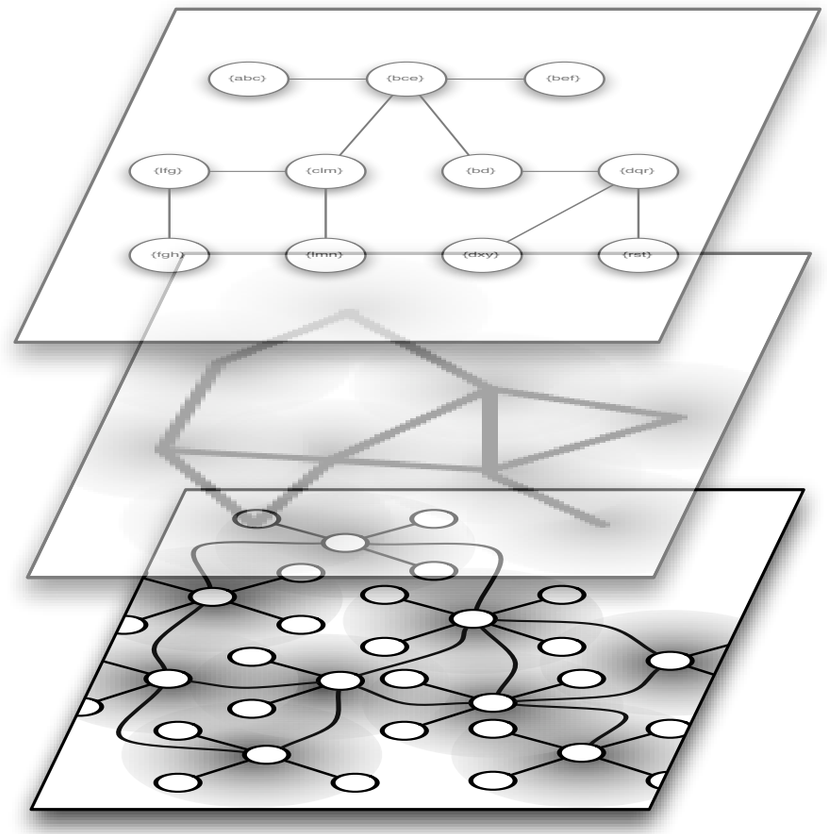
INFERENCE (CENTRALIZED)

- given:
  - a graphical model
    - node: random variables
    - edge: correlation
  - evidence (data)
- find probabilities
- tactic: belief propagation
  - a “message passing” algorithm



DISTRIBUTED INFERENCE

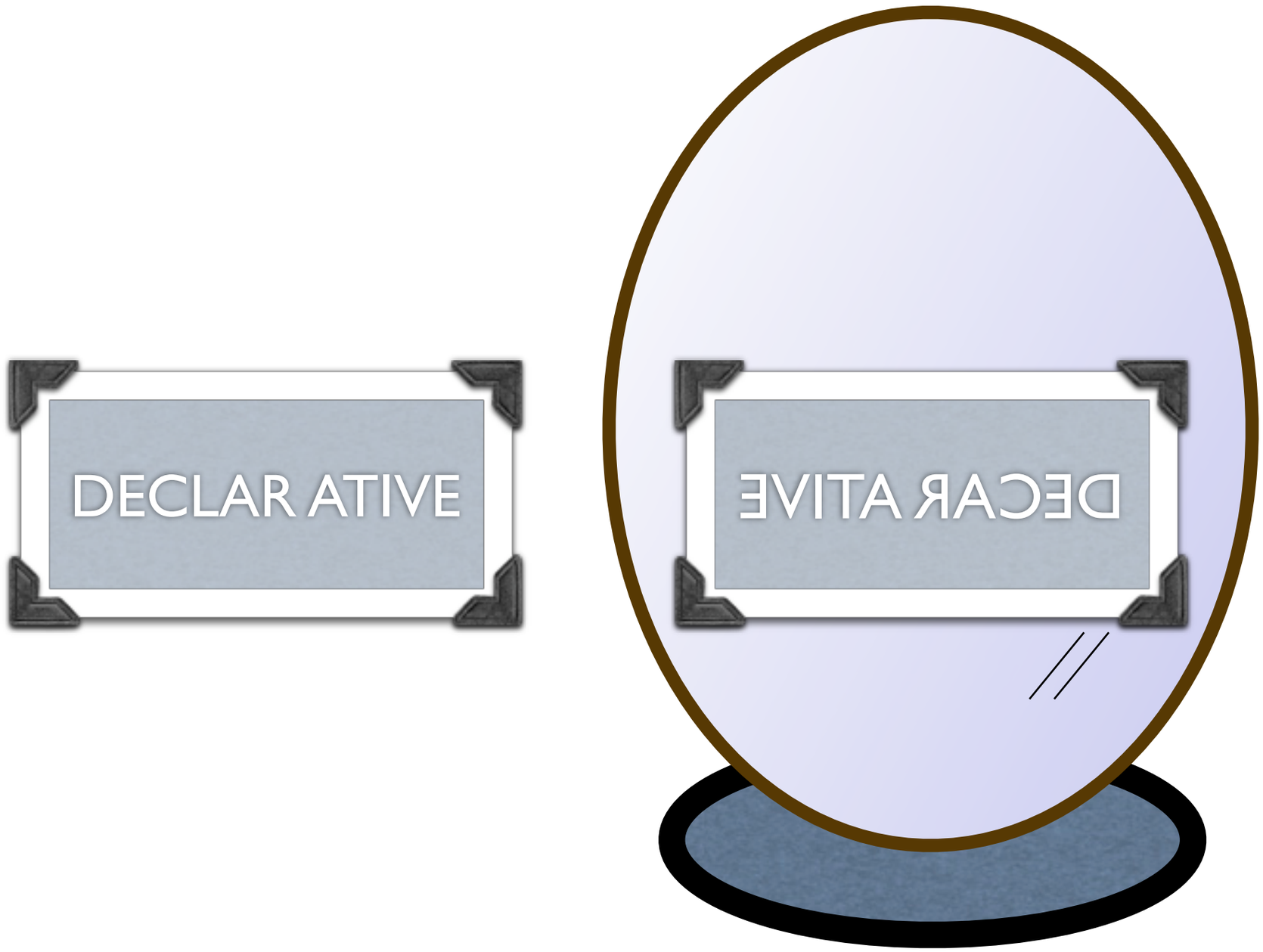
- graphs upon graphs
  - hard to build
  - challenging cross-layer optimization



DECLARATIVE DISTRIBUTED INFERENCE

- overlay is easy (handful of lines)
- hypothesis: even fancy belief propagation is not bad
  - junction tree implementation in progress
- optimization across layers?
  - synthesis of custom Inference Overlay Networks (IONs)?
  - network-aware inference algorithms (NAIAs)?
- proposed applications?
  - network monitoring, disaster response

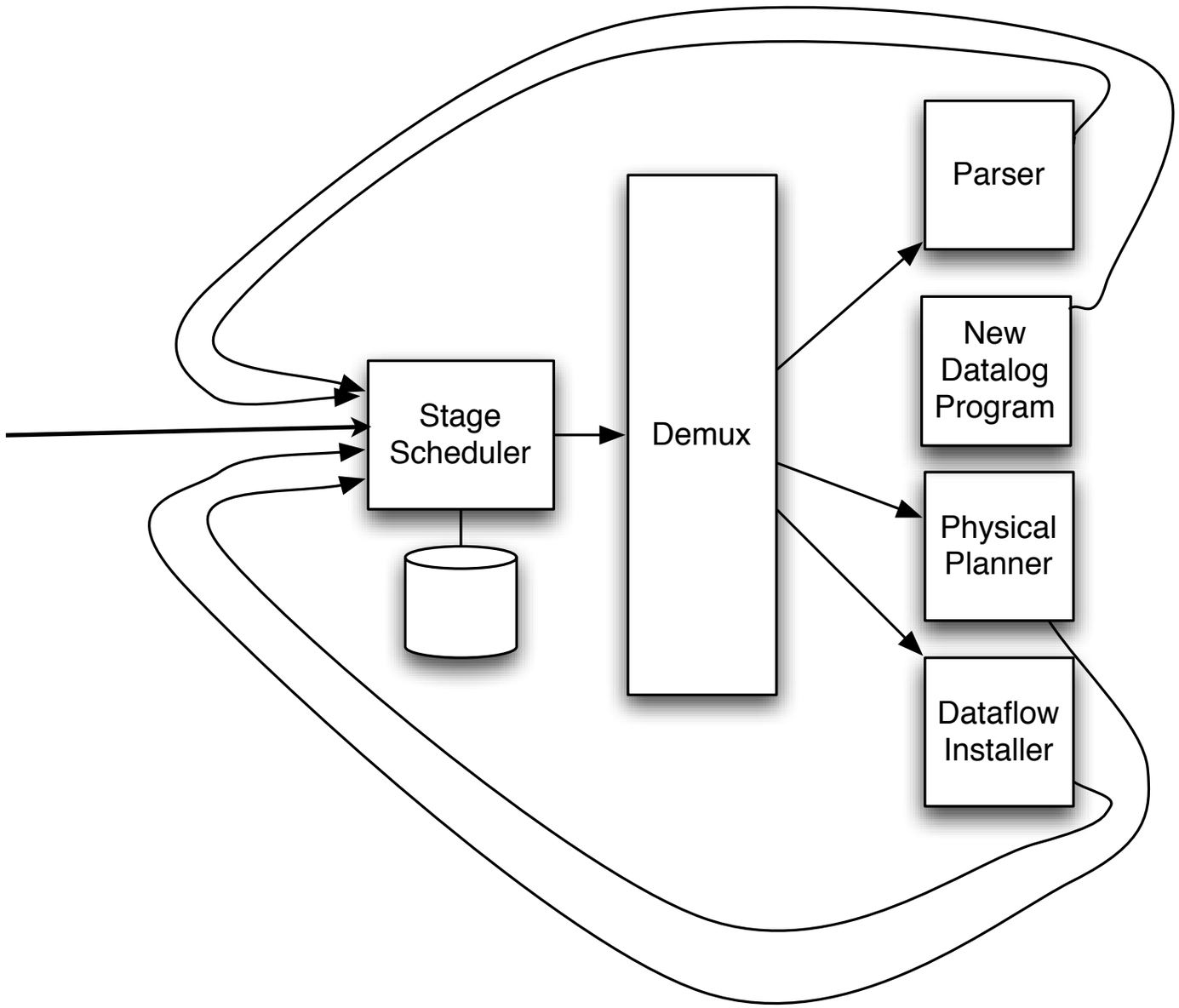
EVITA RACED:
A P2 METACOMPILER



WHY METACOMPILATION

- datalog a good fit to datalog optimizations
  - dynamic programming = recursive table-building
  - magic sets: traversal of “rule/goal graph”
  - statistics-gathering = query processing & inference
- extensibility required
  - application domains still evolving
- but can it be done elegantly?

THE EVITA RACED STAGE CYCLE



INITIAL RESULTS

- system r in dozens of lines
- magic sets rewriting in dozens of lines
- sensornet rendezvous placement in a handful of lines
  - cross-compiled to DSN
- used to support security extensions to overlog

TODAY

- WHY WHAT?
- SAY WHAT
- WHAT: HOW
- WHAT IS NEXT?
- WHAT’S IT TO YOU

TOOLS FOR RESEARCHERS

- overlay construction
- sensornet programming
- network/datacenter monitoring
- coordination protocols
- secure networking
- distributed ML (coming)

RESEARCH OPPORTUNITIES

- language design: declarative/imperative
  - attractiveness, analysis, interoperability
- multicore, traditional parallelism
- protocol optimization/synthesis
- distributed ML, control, robotics
  - engineering ensembles
- ... people keep identifying more!

QUERIES

http://www.declarativity.net