

**THE CLOUD
GOES BOOM**

**DATA-CENTRIC
PROGRAMMING FOR
DATACENTERS**

**JOSEPH M HELLERSTEIN
UC BERKELEY**

JOINT WORK

- Peter ALVARO
Tyson CONDIE
Neil CONWAY
Bill MARCZAK



- Khaled ELMELEEGY
Rusty SEARS



TODAY

- data-centric cloud programming
- datalog and overlog
- a look at BOOM
- a whiff of Bloom
- directions



THE FUTURE'S SO CLOUDY

<http://www.flickr.com/photos/kky/704056791/>

- ✿ a new software dev/deploy *platform*
 - ✿ shared, dynamic, evolving
 - ✿ spanning sets of 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/Connection_Machine



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



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



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



<http://www.flickr.com/photos/kky/704056791/>

THEY LAUGHED WHEN I SAT DOWN AT THE KEYBOARD

Squeals of derision rang through the room. "You, program a computer?" someone asked incredulously. "Now I've heard everything!"

"Enjoy your laugh, beetface," I thought. "You won't be chuckling for long." Little did they know I had MICROSOFT BASIC II, the powerful programming language that uses simple English commands.

I slipped the potent little cartridge into my ATARI Home Computer and closed the door with a confident slap. In a very short time, my friends were astounded at my programming prowess. Information, sounds, colors — even player-missile graphics — leapt

across the screen. True, at one point I did have a little bug in a program, but MICROSOFT BASIC II's debugging features helped me correct it easily. I finished my *tour de force* by typing in a program written in another computer's MICROSOFT BASIC dialect.

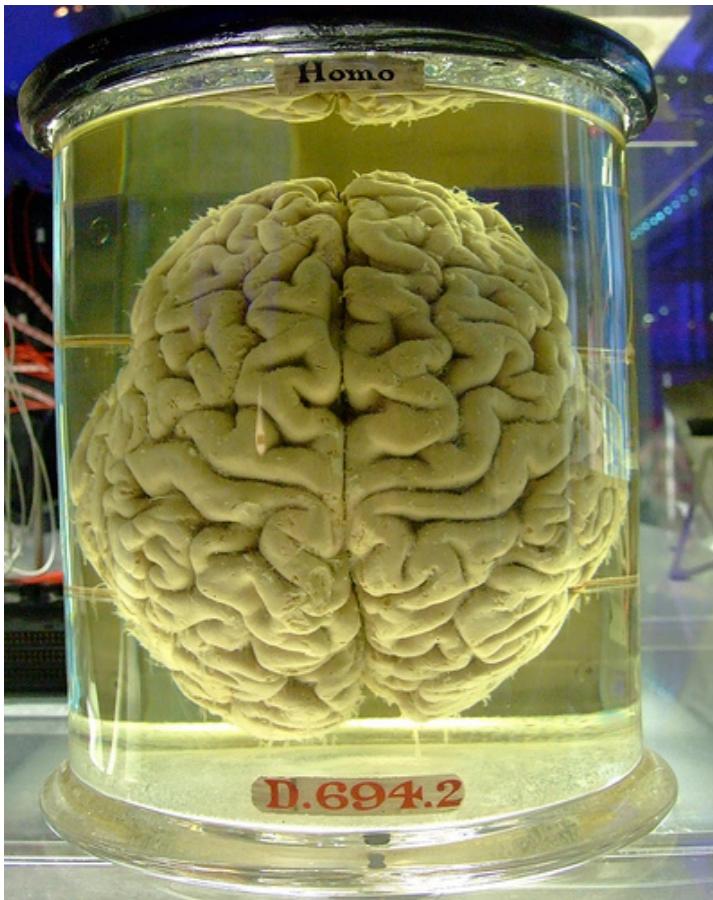
Oohs and ahs filled the air. "Top drawer," snapped the Colonel. "What a man," Mimi cooed. MICROSOFT BASIC II and I had won the day.



DEVELOPERS!



CLOUD DEVELOPMENT



<http://www.flickr.com/photos/gaetanlee/421949167/>

- ➊ the ultimate challenge?
 - ➊ parallel
 - ➋ distributed
 - ➌ elastic
 - ➍ 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
- ✿ and deriving/updating/communicating that state!



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



WINNING STRATEGY

pshanmugam@mac.com

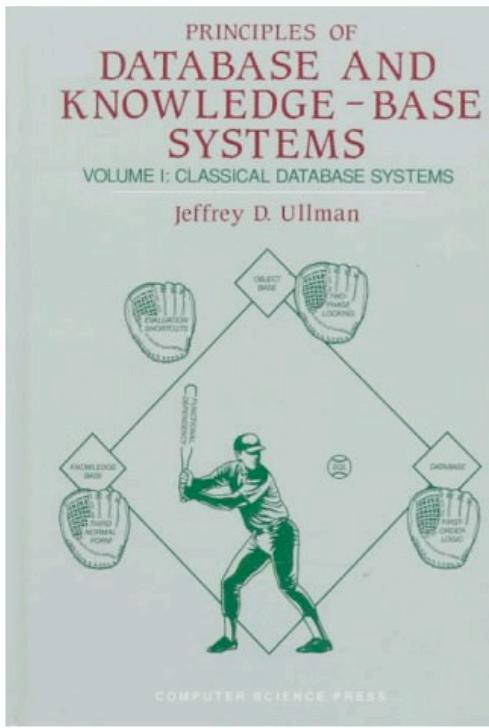
<http://www.flickr.com/photos/pshan427/2331162310/>

✿ *reify state as data*

- ✿ system state is 1st-class data.
- ✿ model. react. evolve.

✿ **data-centric programming**

- ✿ declarative specs for event handling, state safety and transitions
- ✿ reduces hard problems to easy ones
 - ✿ e.g. concurrent programming => data parallelism
 - ✿ e.g. synchronize only for counting



DATA-CENTRIC LANGUAGES

- ✿ decades of theory
 - ✿ logic programming, dataflow
- ✿ but: recent groundswell of applied research
 - ✿ networking, distributed computing, statistical machine learning, **multiplayer games**, 3-tier services, robotics, natural language processing, compiler analysis, security...
 - ✿ see <http://declarativity.net/related>
 - ✿ and CCC Blog:
<http://www.cccblog.org/2008/10/20/the-data-centric-gambit/>

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

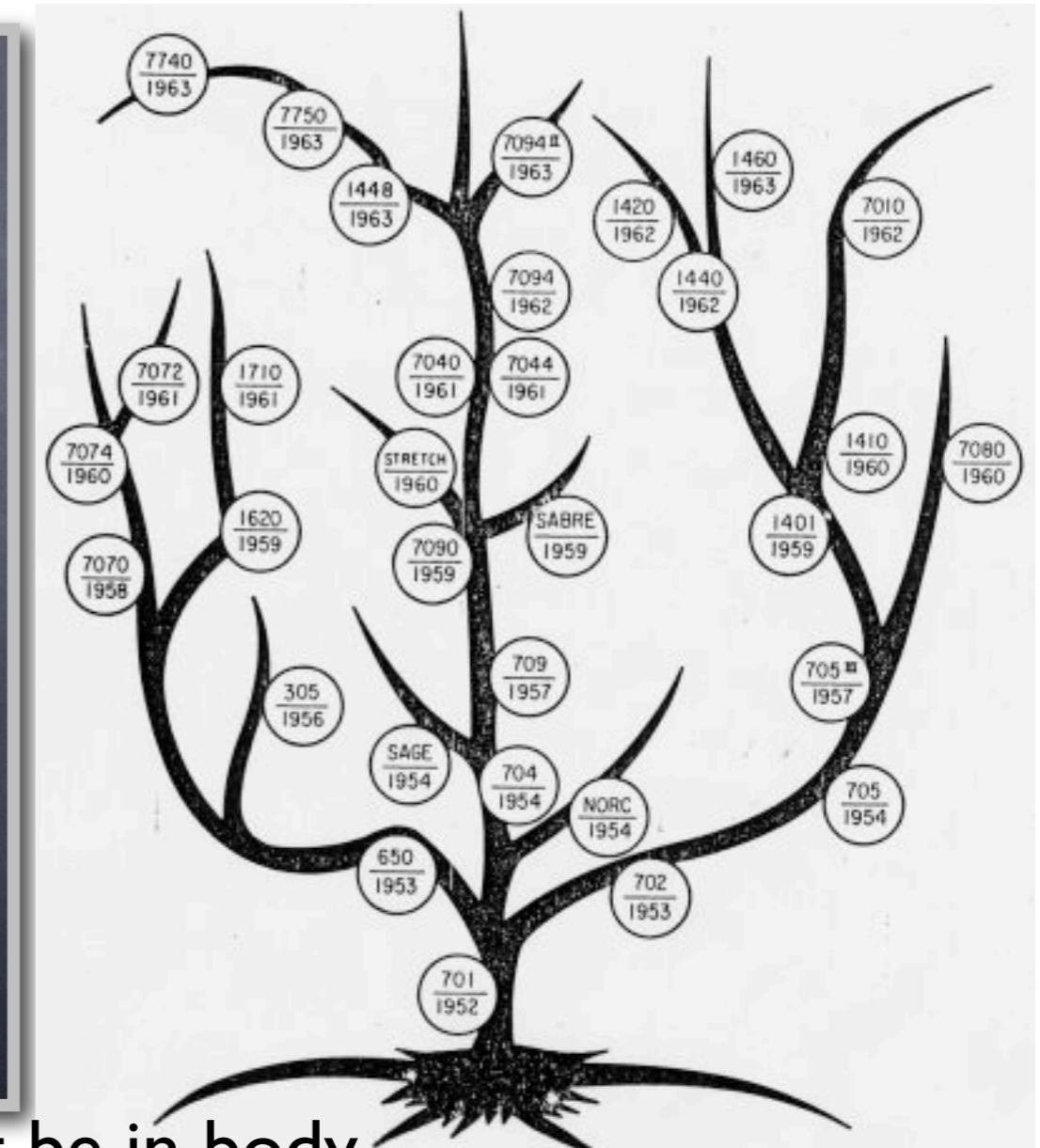
- data-centric cloud programming
- **datalog and overlog**
- a look at BOOM
- a whiff of Bloom
- directions

DATA BASICS

- Data (stored).
- Logic: what we can deduce from the data
 - $P :- q.$
 - SQL “views” (stored/named queries)
- This is all of computing
 - Really!
 - But until recently, it helped to be European.

DUSTY OLD DATALOG

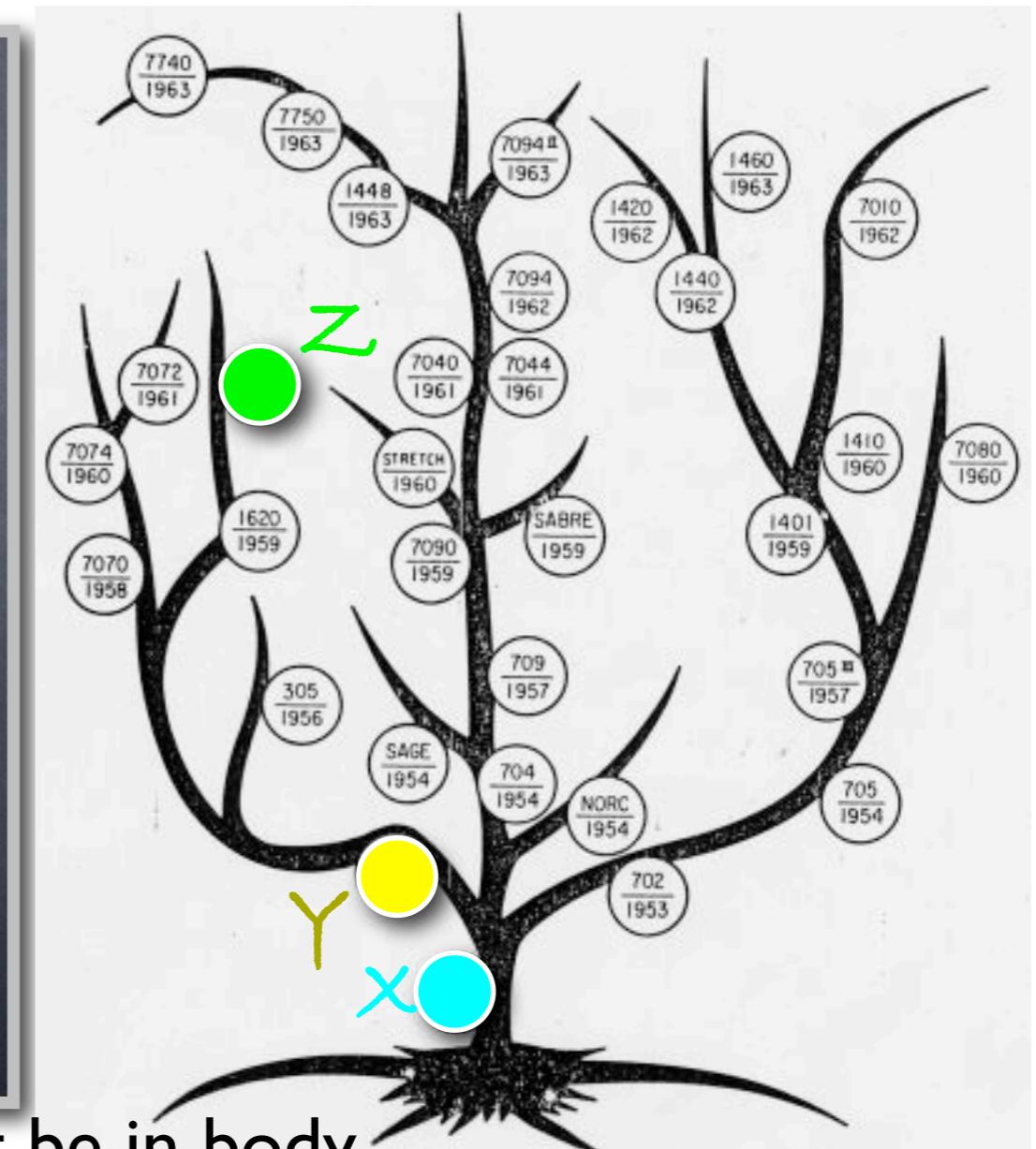
- $\text{parent}(X, Y).$
- $\text{anc}(X, Y) :- \text{parent}(X, Y).$
- $\text{anc}(X, Z) :- \text{parent}(X, Y),$
 $\text{anc}(Y, Z).$
- $\text{anc}(X, s)?$



Notes: *unification*, vars in caps, head vars must be in body.
Set semantics (no dups).

DUSTY OLD DATALOG

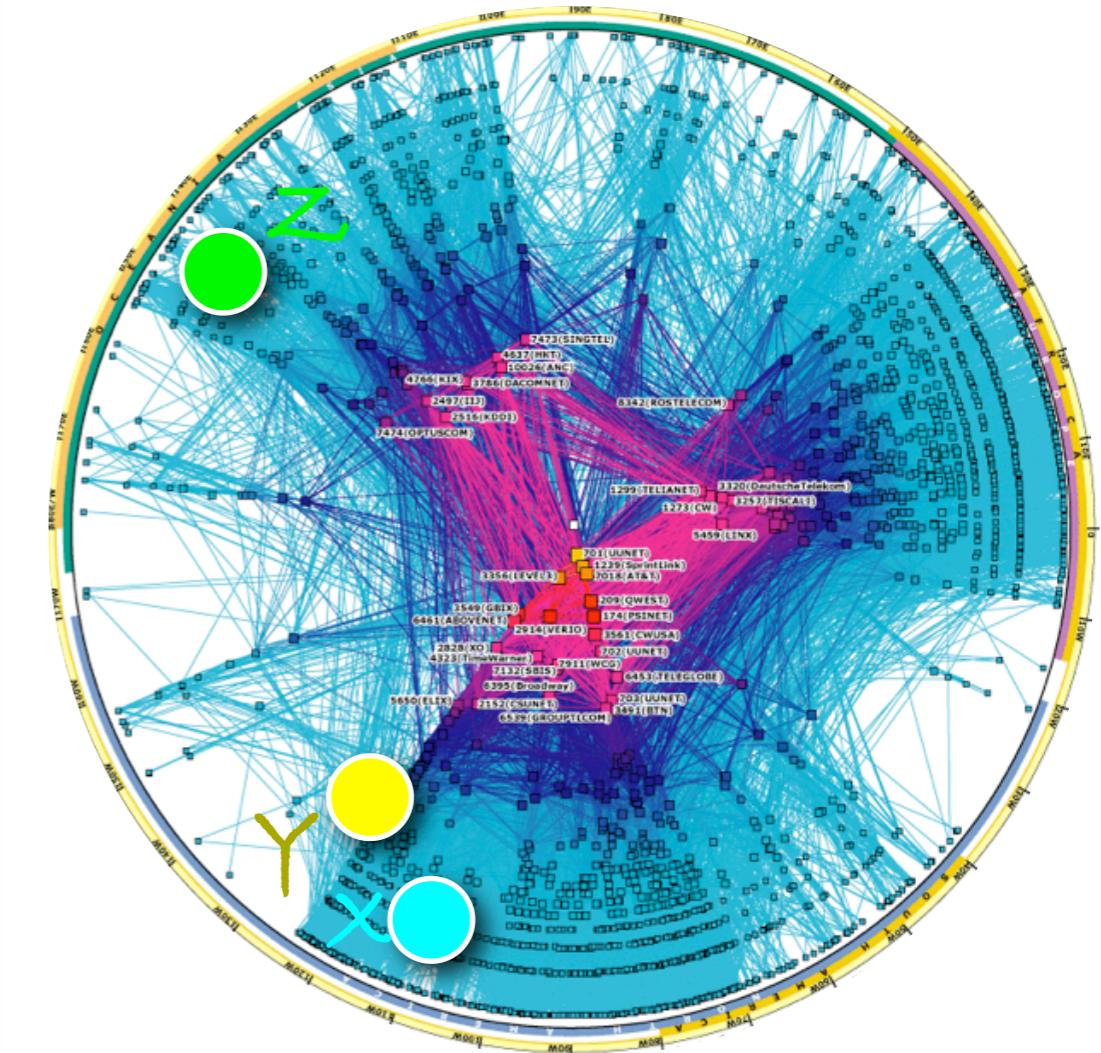
- $\text{parent}(X, Y).$
- $\text{anc}(X, Y) :- \text{parent}(X, Y).$
- $\text{anc}(X, Z) :- \text{parent}(X, Y), \text{anc}(Y, Z).$
- $\text{anc}(X, s)?$



Notes: *unification*, vars in caps, head vars must be in body.
Set semantics (no dups).

THE INTERNET CHANGES EVERYTHING?

- $\text{link}(X, Y).$
- $\text{path}(X, Y) :- \text{link}(X, Y).$
- $\text{path}(X, Z) :- \text{link}(X, Y),$
 $\text{path}(Y, Z).$
- $\text{path}(X, s)?$



Notes: *unification*, vars in caps, head vars must be in body.
Set semantics (no dups).

DATALOG SEMANTICS

- $\text{link}(X, Y).$
- $\text{path}(X, Y) :- \text{link}(X, Y).$
- $\text{path}(X, Z) :- \text{link}(X, Y),$
 $\quad \text{path}(Y, Z).$
- $\text{path}(X, s)?$

- *minimal model*
- i.e. smallest derived DB consistent with stored DB
- Lemma: datalog programs have a *unique* minimal model
- “least model”
- Lemma: natural recursive join strategy computes this model
- “semi-naive” evaluation

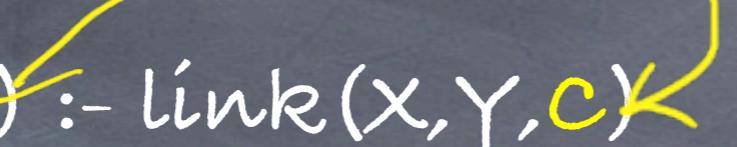
FORMING PATHS

- $\text{link}(X, Y, C)$
- $\text{path}(X, Y, Y, C) :- \text{link}(X, Y, C)$
- $\text{path}(X, Z, Y, C + D)$
 $:- \text{link}(X, Y, C), \text{path}(Y, Z, N, D)$

FORMING PATHS

- $\text{link}(X, Y, C)$ ← COST
- $\text{path}(X, Y, Y, C) :- \text{link}(X, Y, C)$
- $\text{path}(X, Z, Y, C + D)$
 $:- \text{link}(X, Y, C), \text{path}(Y, Z, N, D)$

FORMING PATHS

- $\text{link}(X, Y, C)$  COST
- $\text{path}(X, Y, Y, C) :- \text{link}(X, Y, C)$ 
- $\text{path}(X, Z, Y, C + D)$
 $:- \text{link}(X, Y, C), \text{path}(Y, Z, N, D)$

FORMING PATHS

- $\text{link}(X, Y, C)$
 - $\text{path}(X, Y, Y, C) :- \text{link}(X, Y, C)$
 - $\text{path}(X, Z, Y, C + D) :- \text{link}(X, Y, C), \text{path}(Y, Z, N, D)$
- NEXT HOP*

FORMING PATHS

- $\text{link}(X, Y, C)$
- $\text{path}(X, Y, Y, C) :- \text{link}(X, Y, C)$
- $\text{path}(X, Z, Y, C+D)$
 $:- \text{link}(X, Y, C), \text{path}(Y, Z, N, D)$

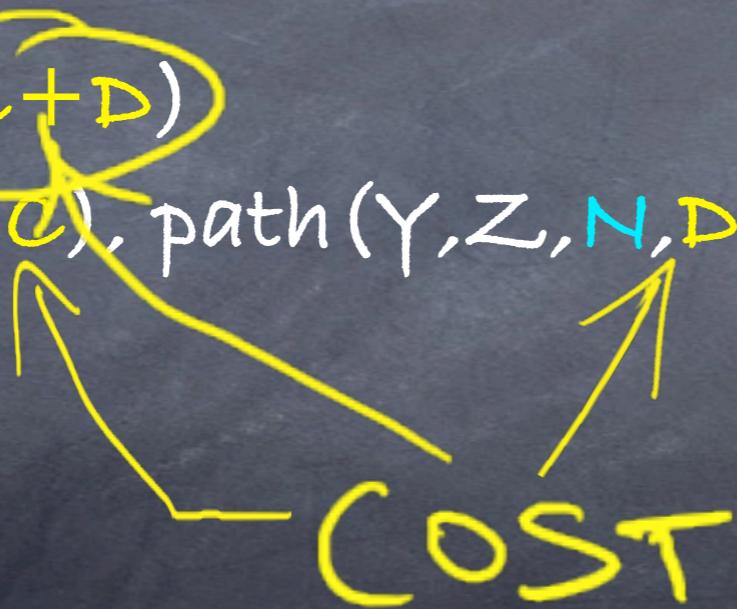
FORMING PATHS

- $\text{link}(X, Y, C)$
- $\text{path}(X, Y, Y, C) :- \text{link}(X, Y, C)$
- $\text{path}(X, Z, Y, C+D)$
 $:- \text{link}(X, Y, C), \text{path}(Y, Z, N, D)$



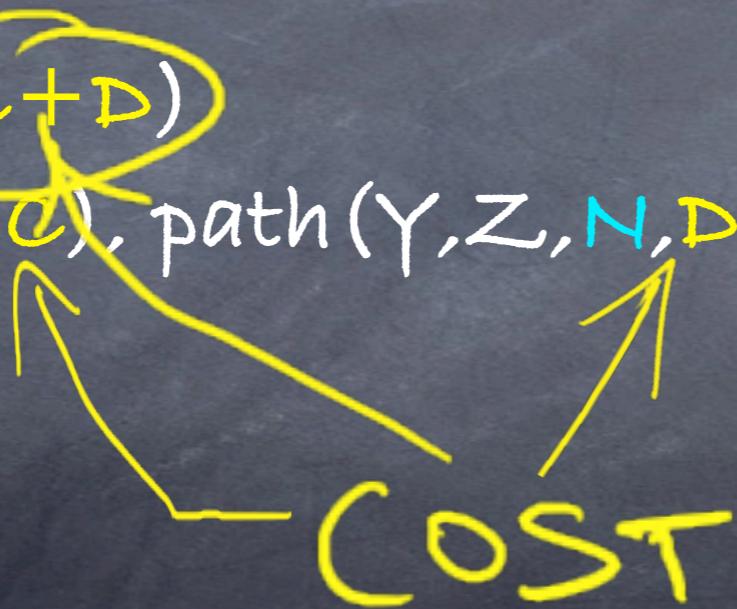
FORMING PATHS

- $\text{link}(X, Y, C)$
- $\text{path}(X, Y, Y, C) :- \text{link}(X, Y, C)$
- $\text{path}(X, Z, Y, C+D)$
 $:- \text{link}(X, Y, C), \text{path}(Y, Z, N, D)$



FORMING PATHS

- $\text{link}(X, Y, C)$
- $\text{path}(X, Y, Y, C) :- \text{link}(X, Y, C)$
- $\text{path}(X, Z, Y, C+D)$
 $:- \text{link}(X, Y, C), \text{path}(Y, Z, N, D)$



Note: we just extended Datalog with functions, which are infinite relations.
E.g. $\text{sum}(C, D, E)$. Need to be careful that programs are still “safe” (finite model). 18

FORMING PATHS

- $\text{link}(X, Y, C)$
- $\text{path}(X, Y, Y, C) :- \text{link}(X, Y, C)$
- $\text{path}(X, Z, Y, C+D)$
 $:- \text{link}(X, Y, C), \text{path}(Y, Z, N, D)$

Note: we just extended Datalog with functions, which are infinite relations.
E.g. $\text{sum}(C, D, E)$. Need to be careful that programs are still “safe” (finite model). 18

FORMING PATHS

- $\text{link}(X, Y, C)$
 - $\text{path}(X, Y, Y, C) :- \text{link}(X, Y, C)$
 - $\text{path}(X, Z, Y, C+D) :- \text{link}(X, Y, C), \text{path}(Y, Z, N, D)$
- NEXT HOP

Note: we just extended Datalog with functions, which are infinite relations.
E.g. $\text{sum}(C, D, E)$. Need to be careful that programs are still “safe” (finite model). 18

BEST PATHS



BEST PATHS

• link(X,Y)



BEST PATHS

- $\text{link}(X, Y)$
- $\text{path}(X, Y, Y, C) :- \text{link}(X, Y, C)$

BEST PATHS

- $\text{link}(X, Y)$
- $\text{path}(X, Y, Y, C) :- \text{link}(X, Y, C)$
- $\text{path}(X, Z, Y, C + D) :- \text{link}(X, Y, C), \text{path}(Y, Z, N, D)$

BEST PATHS

- $\text{link}(X, Y)$
- $\text{path}(X, Y, Y, C) :- \text{link}(X, Y, C)$
- $\text{path}(X, Z, Y, C + D) :- \text{link}(X, Y, C), \text{path}(Y, Z, N, D)$
- $\text{mincost}(X, Z, \min(C)) :- \text{path}(X, Z, Y, C)$

BEST PATHS

- $\text{link}(X, Y)$
- $\text{path}(X, Y, Y, C) :- \text{link}(X, Y, C)$
- $\text{path}(X, Z, Y, C + D) :- \text{link}(X, Y, C), \text{path}(Y, Z, N, D)$
- $\text{mincost}(X, Z, \text{min} < C >) :- \text{path}(X, Z, Y, C)$
- $\text{bestpath}(X, Z, Y, C) :- \text{path}(X, Z, Y, C), \text{mincost}(X, Z, C)$

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)?`

BEST PATHS

- $\text{link}(X, Y)$
- $\text{path}(X, Y, Y, C) :- \text{link}(X, Y, C)$
- $\text{path}(X, Z, Y, C + D) :- \text{link}(X, Y, C), \text{path}(Y, Z, N, D)$
- $\text{mincost}(X, Z, \min(C)) :- \text{path}(X, Z, Y, C)$
- $\text{bestpath}(X, Z, Y, C) :- \text{path}(X, Z, Y, C), \text{mincost}(X, Z, C)$
- $\text{bestpath}(\text{src}, D, Y, C)?$

Note: we just extended Datalog with *aggregation*. You can't compute an aggregate until you fully compute its inputs (*stratification*).

SO FAR...

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

TOWARD DISTRIBUTION: DATA PARTITIONING

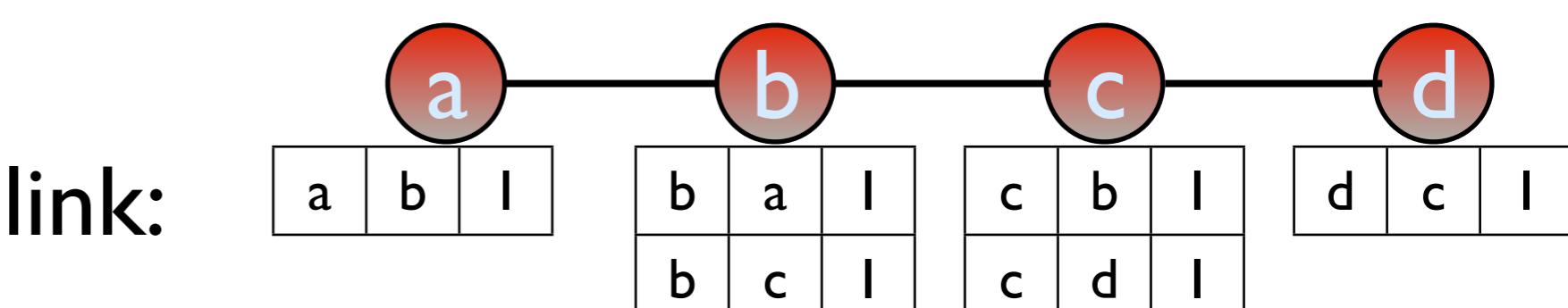
- logically global tables
- horizontally partitioned
- an address field per table
 - *location specifier: @*
 - data placement based on loc.spec.

LOCATION 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)`

LOCATION 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)`



LOCATION SPECS INDUCE COMMUNICATION

- $\text{link}(@X, Y, C)$
- $\text{path}(@X, Y, Y, C) :- \text{link}(@X, Y, C)$
- $\text{path}(@X, Z, Y, C + D) :- \text{link}(@X, Y, C), \text{path}(@Y, Z, N, D)$

path:

a	b	b	
b	c	c	

b	a	a	
b	c	c	

c	b	b	
c	d	d	

d	c	c	
d	c	c	

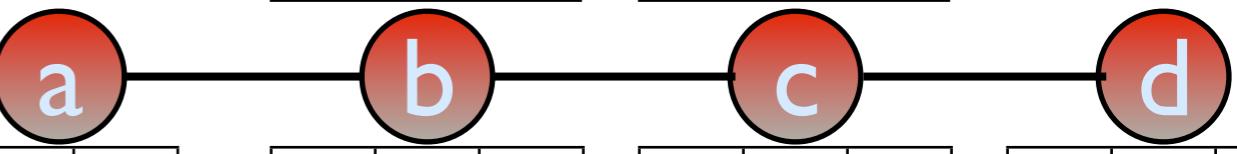
link:

a	b	
b	c	

b	a	
b	c	

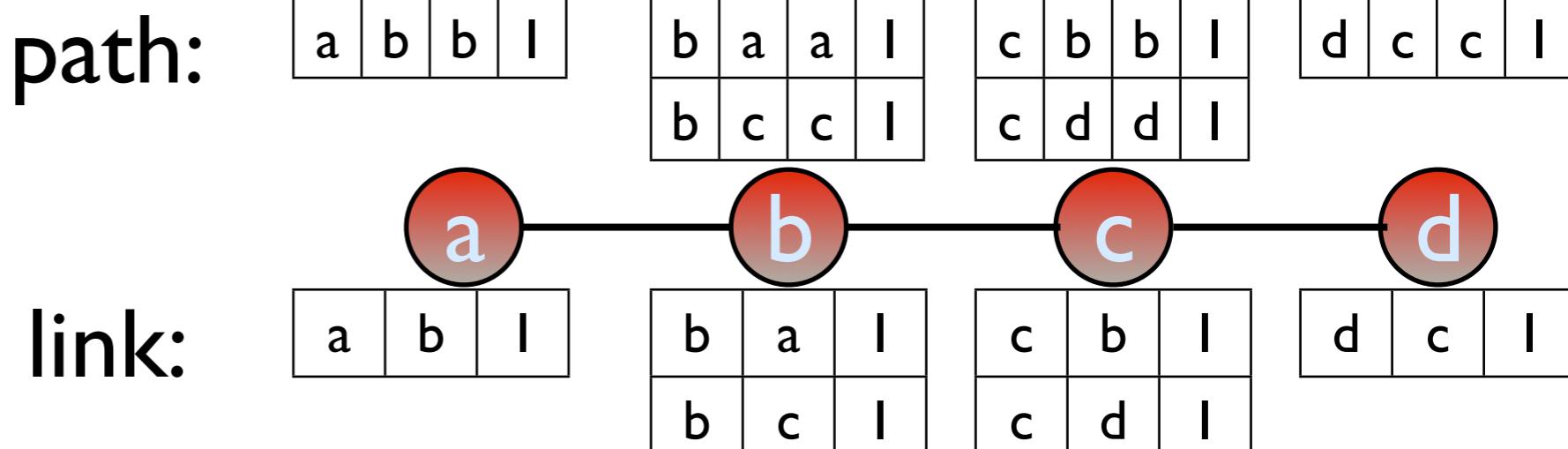
c	b	
c	d	

d	c	
d	c	



LOCATION 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)`



LOCATION SPECS INDUCE COMMUNICATION

- $\text{link}(@X, Y, C)$
- $\text{path}(@X, Y, Y, C) :- \text{link}(@X, Y, C)$
- $\text{path}(@X, Z, Y, C+D) :- \text{link}(@X, Y, C), \text{path}(@Y, Z, N, D)$

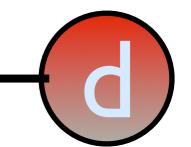
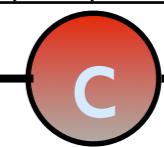
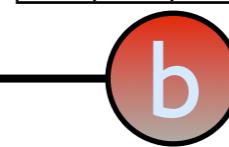
path:

a	b	b	
b	c	c	

b	a	a	
b	c	c	

c	b	b	
c	d	d	

d	c	c	
d	c	c	



link:

a	b	
b	c	

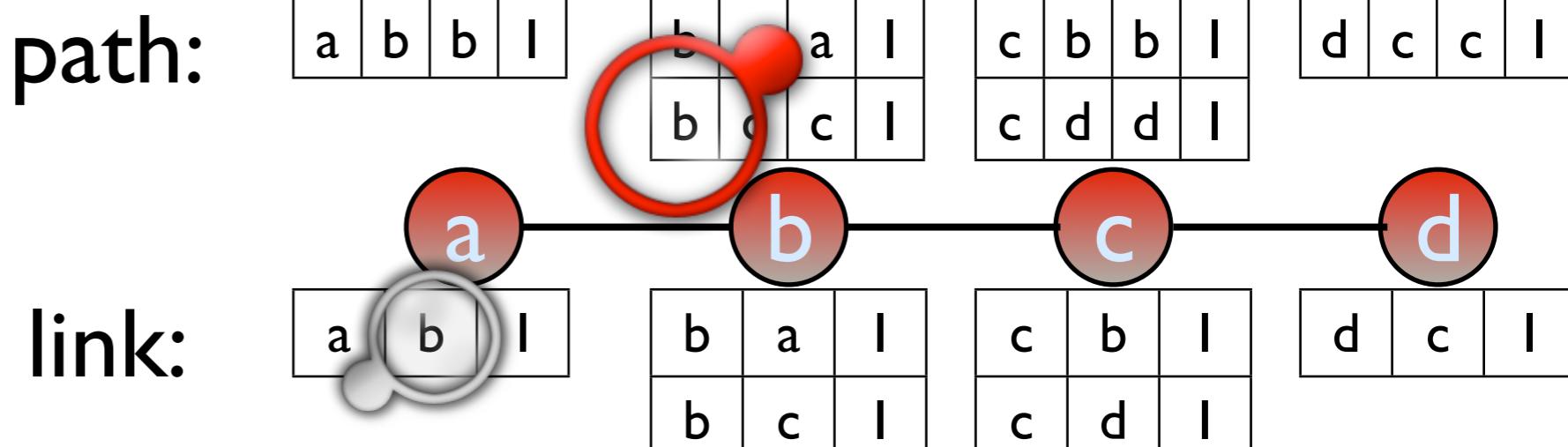
b	a	
b	c	

c	b	
c	d	

d	c	
d	c	

LOCATION SPECS INDUCE COMMUNICATION

- $\text{link}(@X, Y, C)$
- $\text{path}(@X, Y, Y, C) :- \text{link}(@X, Y, C)$
- $\text{path}(@X, Z, Y, C+D) :- \text{link}(@X, Y, C), \text{path}(@Y, Z, N, D)$



LOCATION SPECS INDUCE COMMUNICATION

- $\text{link}(@X, Y, C)$
- $\text{path}(@X, Y, Y, C) :- \text{link}(@X, Y, C)$
- $\text{path}(@X, Z, Y, C+D) :- \text{link}(@X, Y, C), \text{path}(@Y, Z, N, D)$

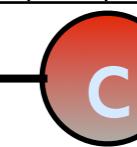
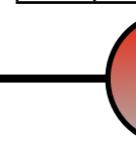
path:

a	b	b	
b	c	c	

b	a	a	
b	c	c	

c	b	b	
c	d	d	

d	c	c	
d	c	c	



link:

a	b	
b	c	

b	a	
b	c	

c	b	
c	d	

d	c	
d	c	

LOCATION SPECS INDUCE COMMUNICATION

- $\text{link}(@X, Y, C)$
- $\text{path}(@X, Y, Y, C) :- \text{link}(@X, Y, C)$
- $\text{path}(@X, Z, Y, C+D) :- \text{link}(@X, Y, C), \text{path}(@Y, Z, N, D)$

path:

a	b	b	
b	c	c	

b	a	a	
b	c	c	

c	b	b	
c	d	d	

d	c	c	
d	c	c	

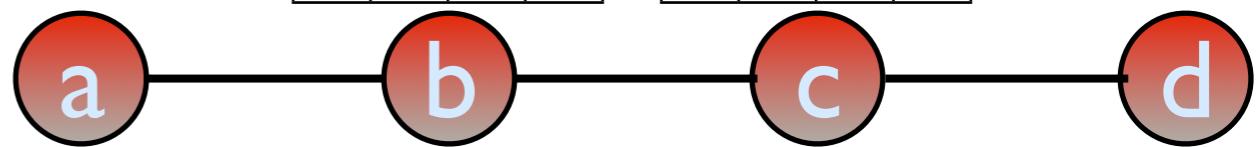
link:

a	b	
b	c	

b	a	
b	c	

c	b	
c	d	

d	c	
d	c	

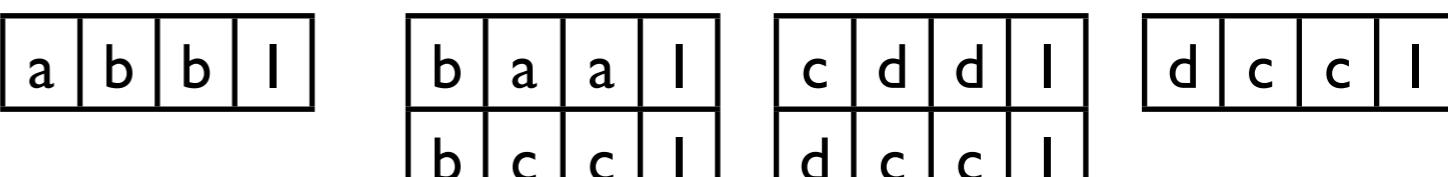


LOCATION SPECS INDUCE COMMUNICATION

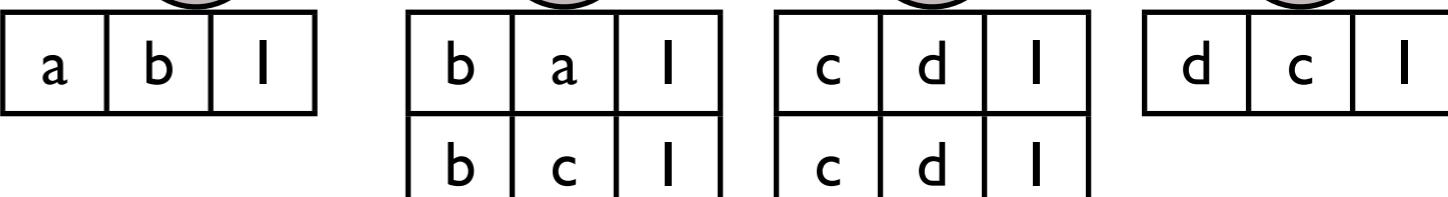
- $\text{link}(@X, Y)$ Localization Rewrite
- $\text{path}(@X, Y, Y, C) :- \text{link}(@X, Y, C)$
- $\text{link_d}(X, @Y, C) :- \text{link}(@X, Y, C)$
- $\text{path}(@X, Z, Y, C+D) :- \text{link_d}(X, @Y, C), \text{path}(@Y, Z, N, D)$

link_d:

path:



link:



LOCATION SPECS INDUCE COMMUNICATION

- $\text{link}(@X, Y)$ Localization Rewrite
- $\text{path}(@X, Y, Y, C) :- \text{link}(@X, Y, C)$
- $\text{link_d}(X, @Y, C) :- \text{link}(X, Y, C)$
- $\text{path}(@X, Z, Y, C+D) :- \text{link_d}(X, @Y, C), \text{path}(@Y, Z, N, D)$

link_d:

path:

a b b l	b a a l	c d d l	d c c l
b c c l	d c c l		

link:

a b l	b a l	c d l	d c l
b c l	d c l	c d l	d c l



LOCATION SPECS INDUCE COMMUNICATION

- $\text{link}(@X, Y)$
- $\text{path}(@X, Y, Y, C) :- \text{link}(@X, Y, C)$
- $\text{link_d}(X, @Y, C) :- \text{link}(X, Y, C)$
- $\text{path}(@X, Z, Y, C+D) :- \text{link_d}(X, @Y, C), \text{path}(@Y, Z, N, D)$

Localization Rewrite

link_d:

a	b	
---	---	--

path:

a	b	b	
b	a	a	

b	a	a	
b	c	c	

c	d	d	
d	c	c	

d	c	c	
---	---	---	--

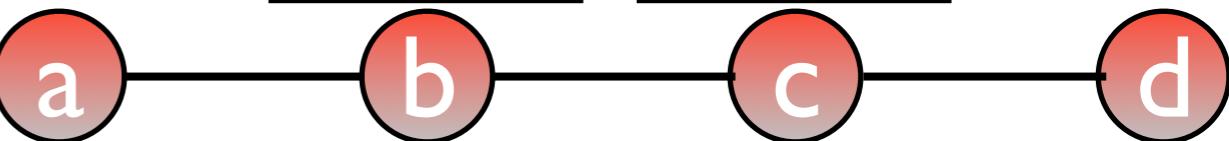
link:

a	b	
b	c	

b	a	
b	c	

c	d	
c	d	

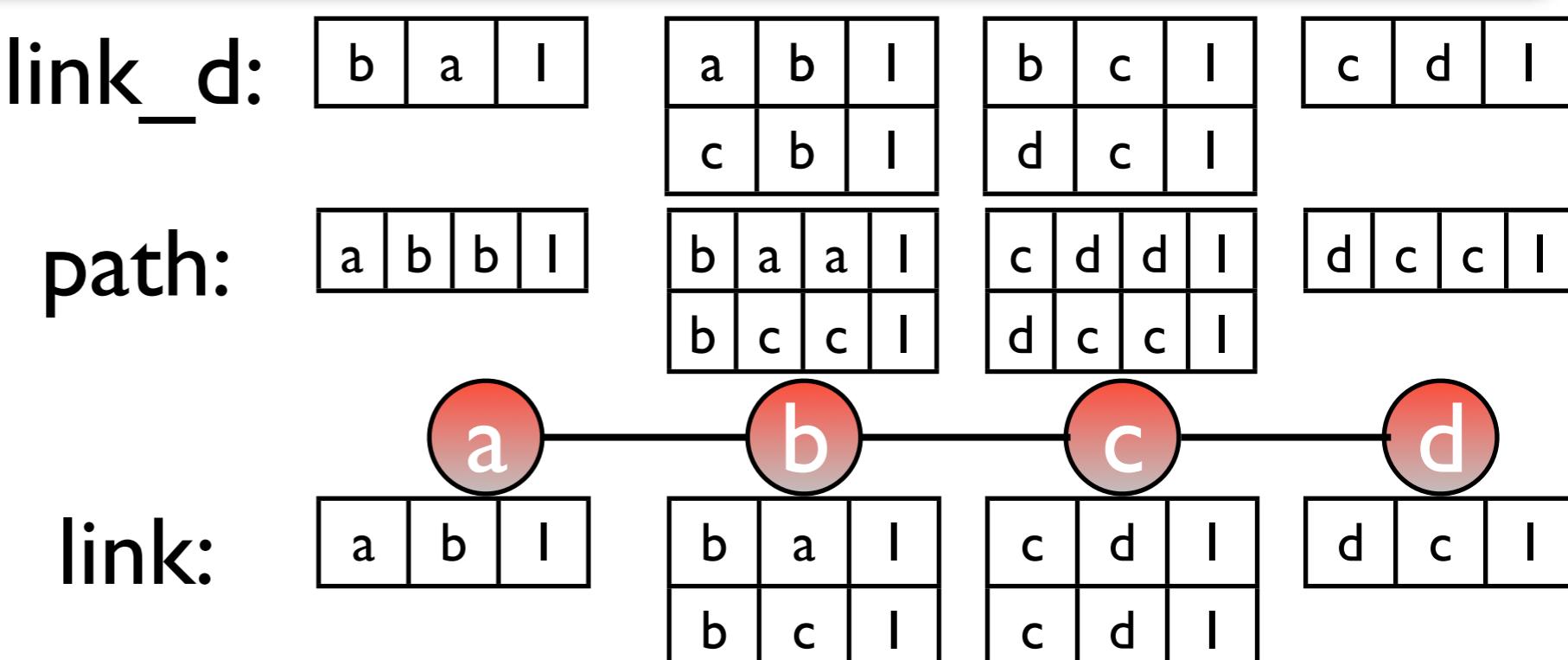
d	c	
---	---	--



LOCATION SPECS INDUCE COMMUNICATION

- $\text{link}(@X, Y)$
- $\text{path}(@X, Y, Y, C) :- \text{link}(@X, Y, C)$
- $\text{link_d}(X, @Y, C) :- \text{link}(X, Y, C)$
- $\text{path}(@X, Z, Y, C+D) :- \text{link_d}(X, @Y, C), \text{path}(@Y, Z, N, D)$

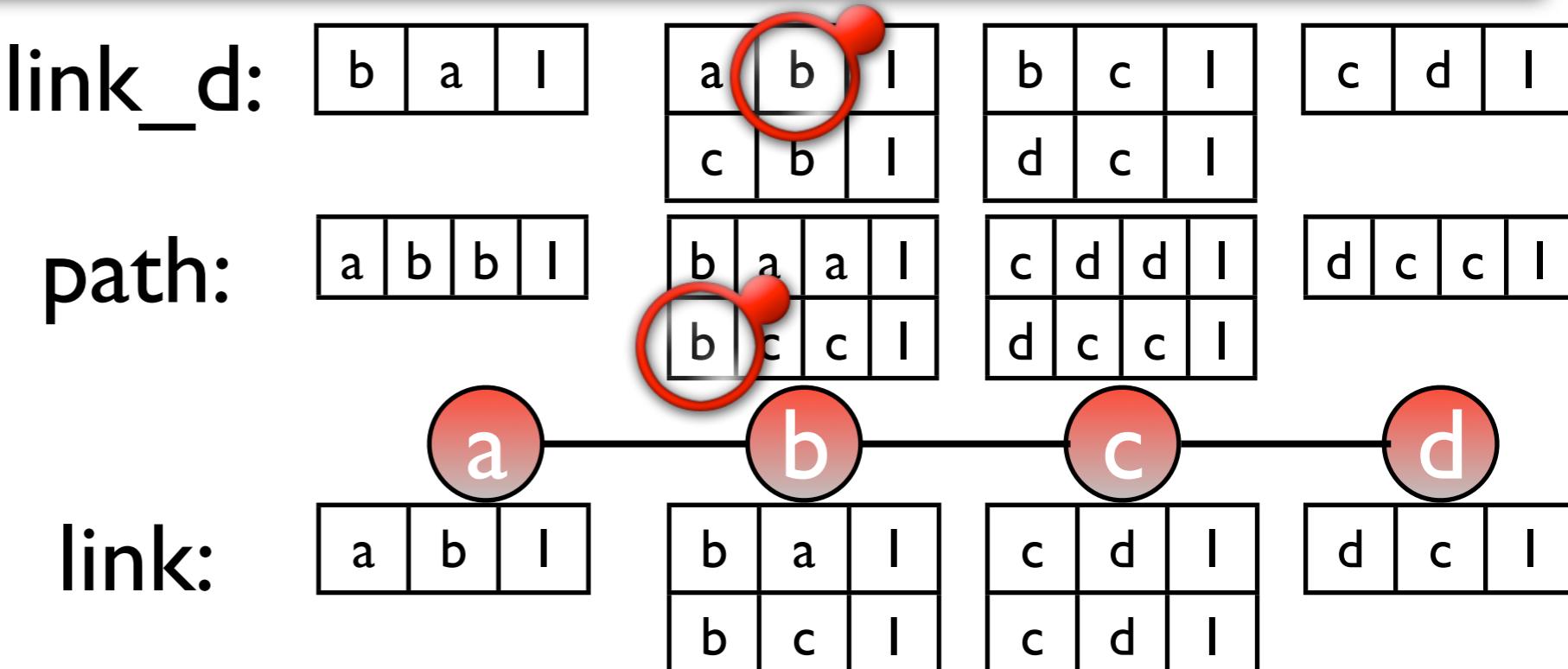
Localization Rewrite



LOCATION SPECS INDUCE COMMUNICATION

- $\text{link}(@X, Y)$
- $\text{path}(@X, Y, Y, C) :- \text{link}(@X, Y, C)$
- $\text{link_d}(X, @Y, C) :- \text{link}(X, Y, C)$
- $\text{path}(@X, Z, Y, C+D) :- \text{link_d}(X, @Y, C), \text{path}(@Y, Z, N, D)$

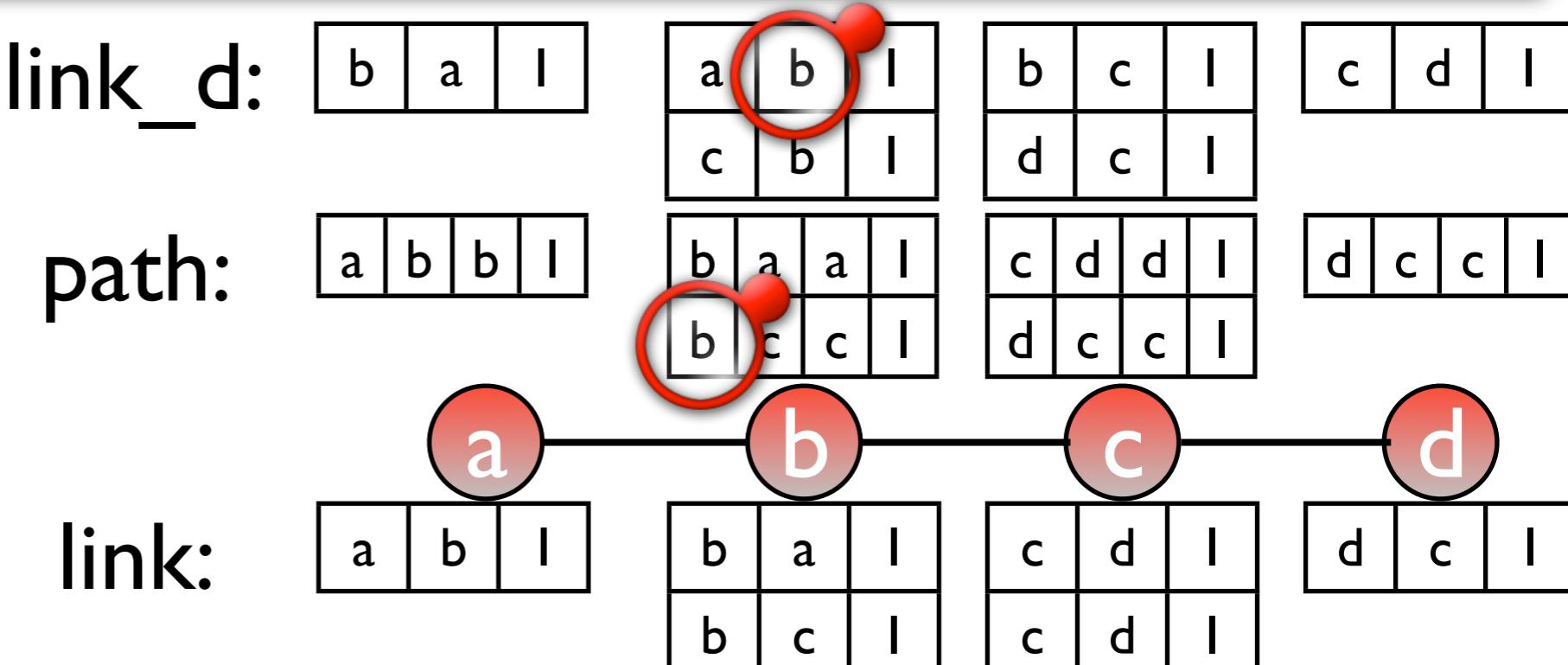
Localization Rewrite



LOCATION SPECS INDUCE COMMUNICATION

- $\text{link}(@X, Y)$
- $\text{path}(@X, Y, Y, C) :- \text{link}(@X, Y, C)$
- $\text{link_d}(X, @Y, C) :- \text{link}(X, Y, C)$
- $\text{path}(@X, Z, Y, C+D) :- \text{link_d}(X, @Y, C), \text{path}(@Y, Z, N, D)$

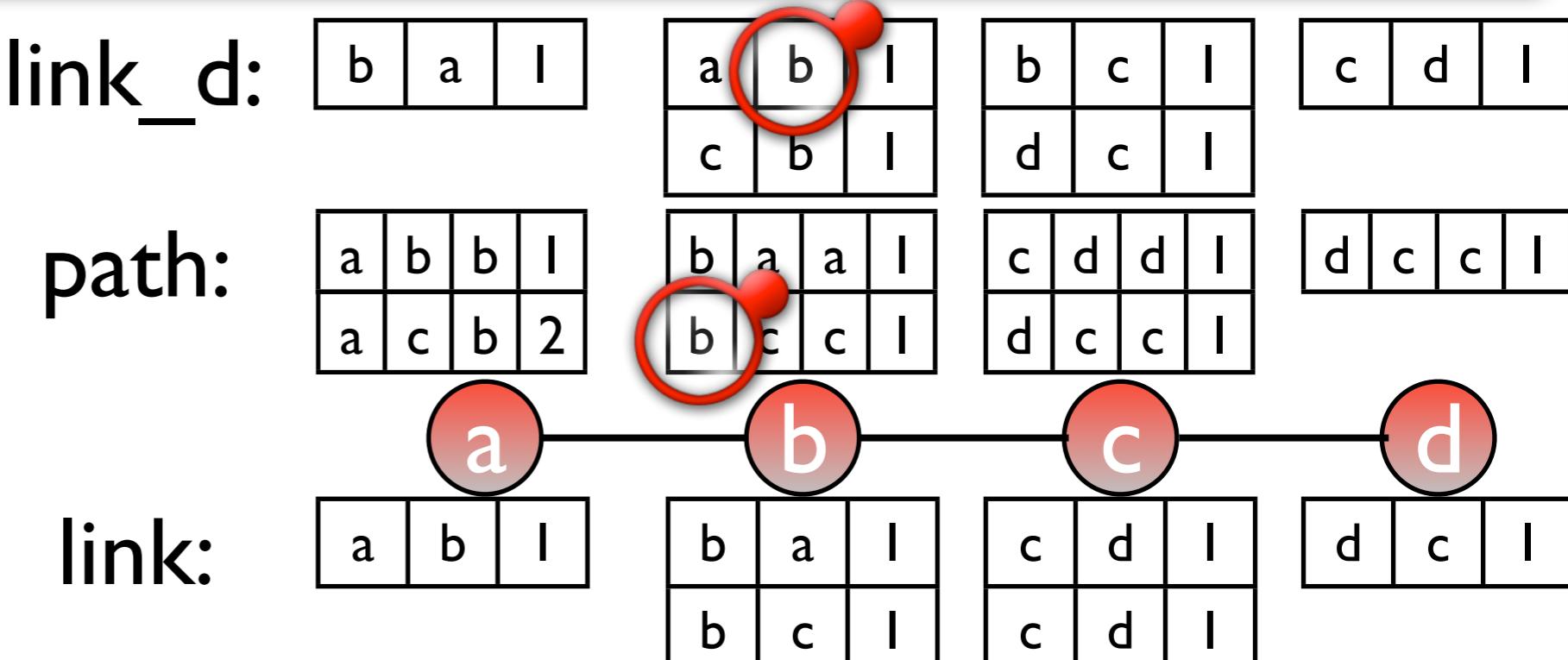
Localization Rewrite



LOCATION SPECS INDUCE COMMUNICATION

- $\text{link}(@X, Y)$
- $\text{path}(@X, Y, Y, C) :- \text{link}(@X, Y, C)$
- $\text{link_d}(X, @Y, C) :- \text{link}(X, Y, C)$
- $\text{path}(@X, Z, Y, C+D) :- \text{link_d}(X, @Y, C), \text{path}(Y, Z, N, D)$

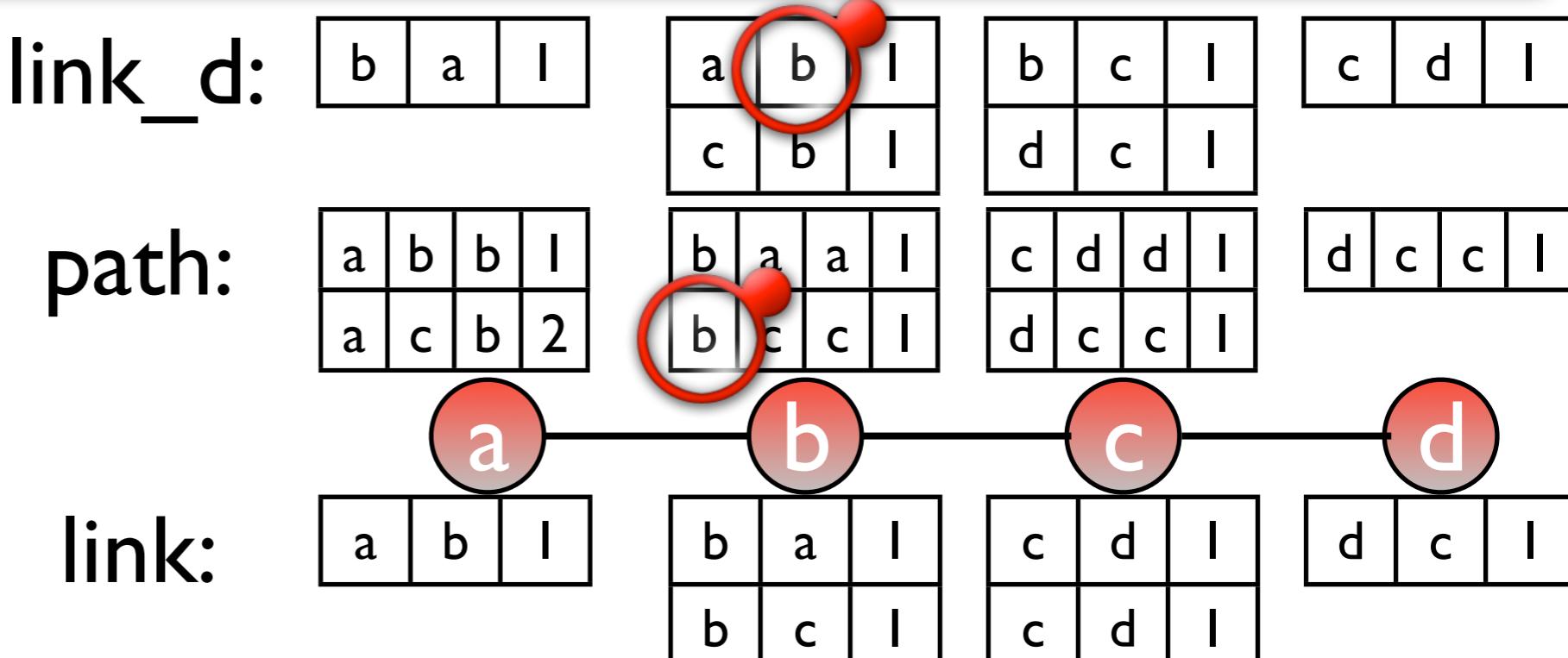
Localization Rewrite

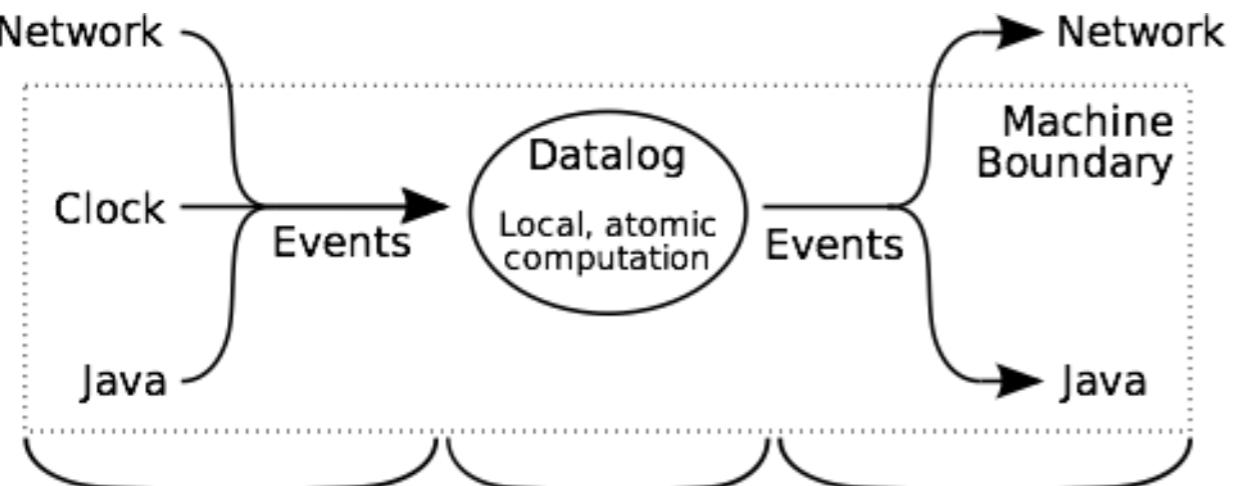


LOCATION SPECS INDUCE COMMUNICATION

- $\text{link}(@X, Y)$
- $\text{path}(@X, Y, Y, C) :- \text{link}(@X, Y, C)$
- $\text{link_d}(X, @Y, C) :- \text{link}(X, Y, C)$
- $\text{path}(@X, Z, Y, C+D) :- \text{link_d}(X, @Y, C), \text{path}(@Y, Z, N, D)$

Localization Rewrite





OVERLOG IS...

- ✿ Datalog w/aggregation & function symbols
- ✿ + Horizontally partitioned tables (data, not messages!)
- ✿ “Event” tables for clock/net/host (data again!)
- ✿ + iterated (single-machine) fixpoints
- ✿ “state update” happens atomically between fixpoints
- ✿ formal temporal logic treatment in *Dedalus* (foundation of *Bloom*)

DSN-TRICKLE

Levis, et al., Sensys 2004

Chu, et al., Sensys 2007

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.

```

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-TRICKLE

Levis, et al., Sensys 2004

Chu, et al., Sensys 2007

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.

```

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



<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



<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

```

/* 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, 15, 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).

11 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].
12 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).
13 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 > Z.
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,I,BI,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 (N,B), BI != NI.

```

<http://www.flickr.com/photos/3862144@N02/2765974909/>

```

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,"-",P) :- pingEvent@NI(NI,E),
   pendingPing@NI(NI,PI,E), pred@NI(NI,P,PI).
```

chord distributed hash table

Internet overlay for content-based routing

high-function implementation

all the research bells and whistles

48 rules, 13 table definitions

P2P HORD

```

/* 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, 15, 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 > Z.
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,I,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 (N,B), BI != NI.

```

<http://www.flickr.com/photos/3862144@N02/2765974909/>

```

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,"-",P) :- pingEvent@NI(NI,E),
    pendingPing@NI(NI,PI,E), pred@NI(NI,P,PI).
```



chord distributed hash table

Internet overlay for content-based routing



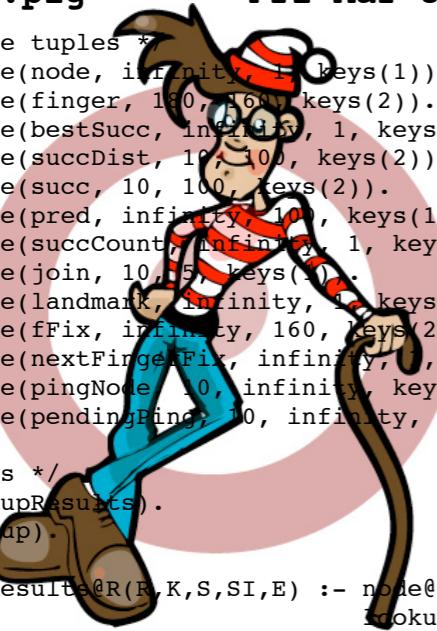
high-function implementation

all the research bells and whistles

48 rules, 13 table definitions

100x LESS CODE THAN CHORD

P2P-HORD



TODAY

- data-centric cloud programming
- datalog and overlog
- a look at BOOM
- a whiff of Bloom
- directions



THE CLOUD GOES BOOM!

- ✿ Berkeley Orders Of Magnitude
 - ✿ OOM bigger systems
 - ✿ OOM less code
- ✿ we did it for network protocols, time to generalize
 - ✿ and make attractive to developers
 - ✿ *Bloom* is the language for BOOM



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 (Bloom)
 - ✿ for 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 (prototype Hadoop's future)
 - ✿ availability rev: hot-standby masters
 - ✿ scalability rev: scale out master state
 - ✿ monitoring rev: invariant checking, logging
- ✿ 9 months, 4 grad student developers
 - ✿ most work in a 3-month span



CURRENT CODE BASE

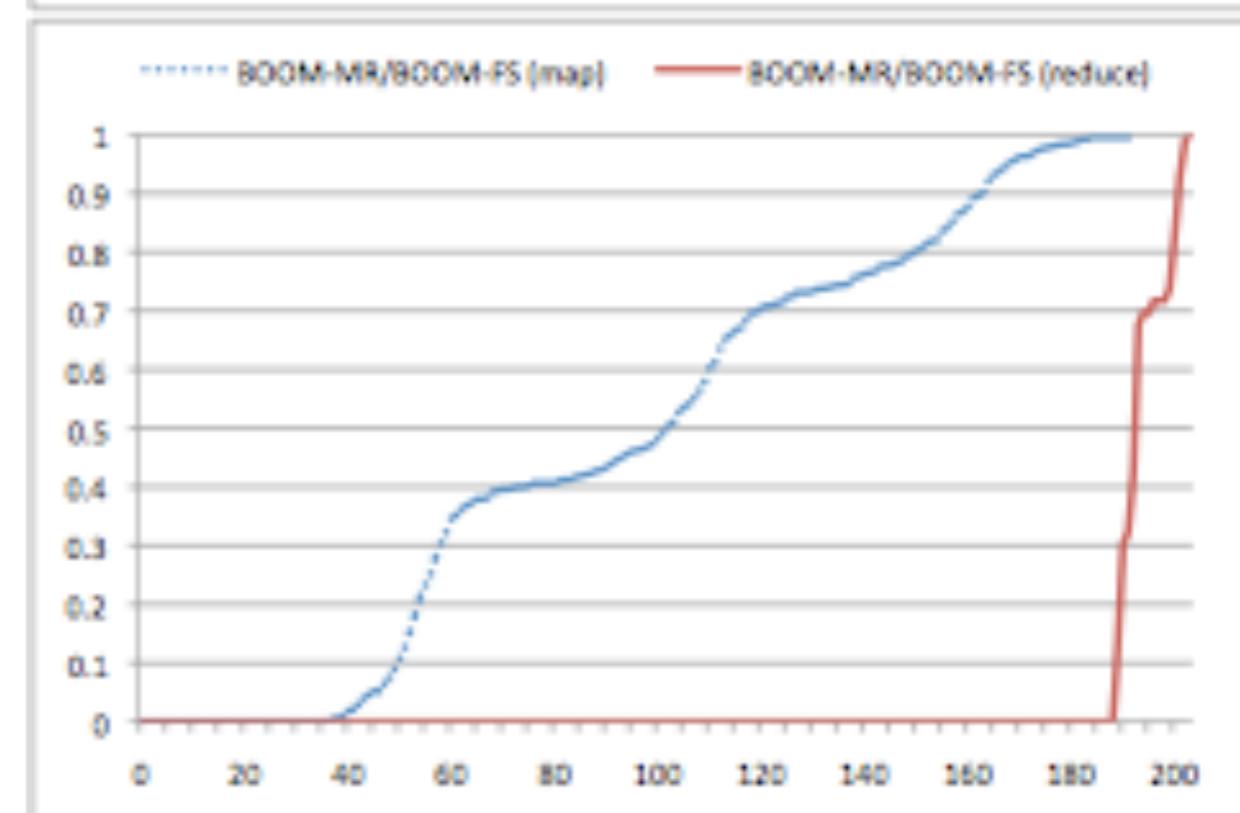
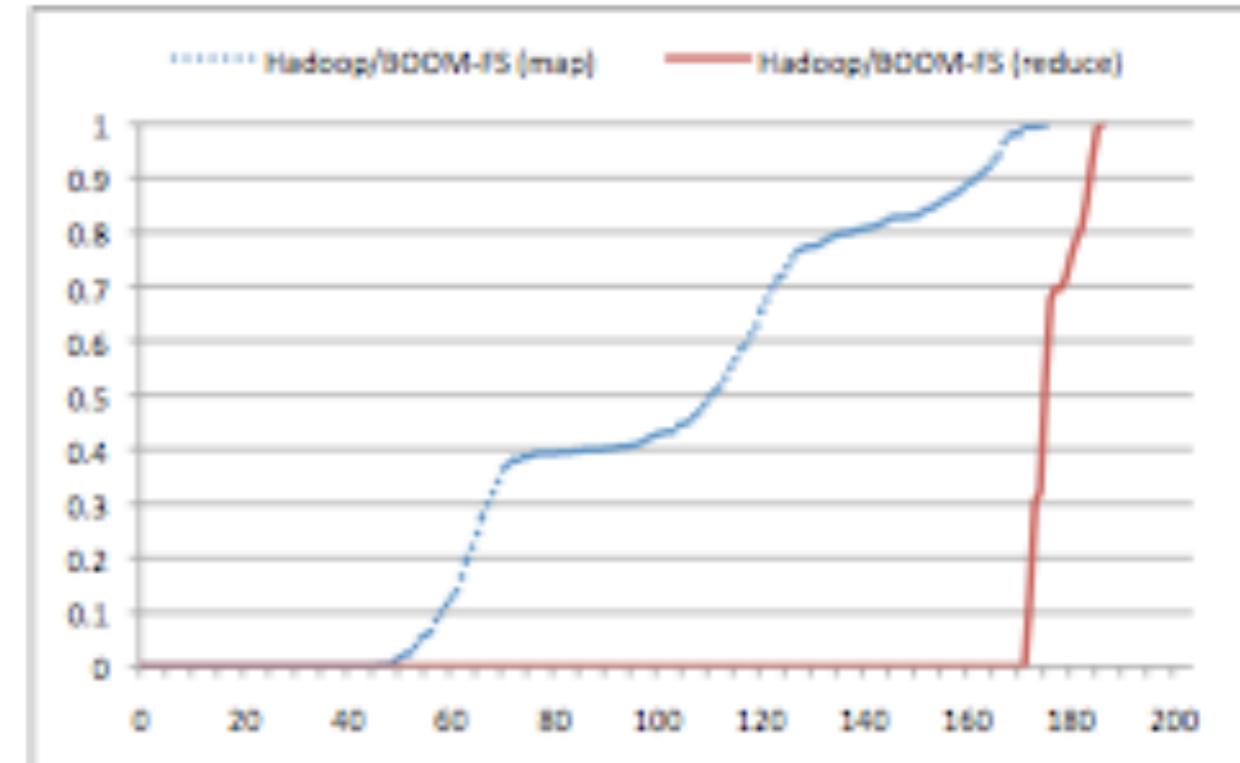
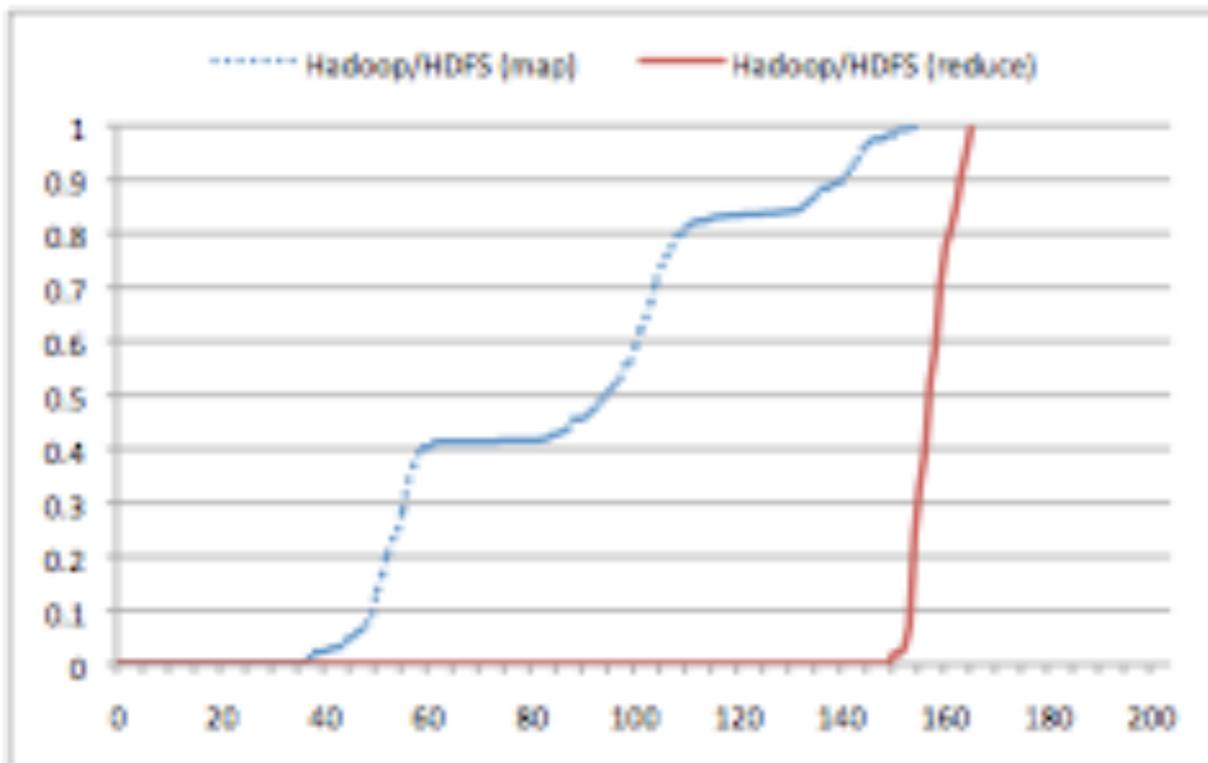
Filesystem

	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-MR	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);
```



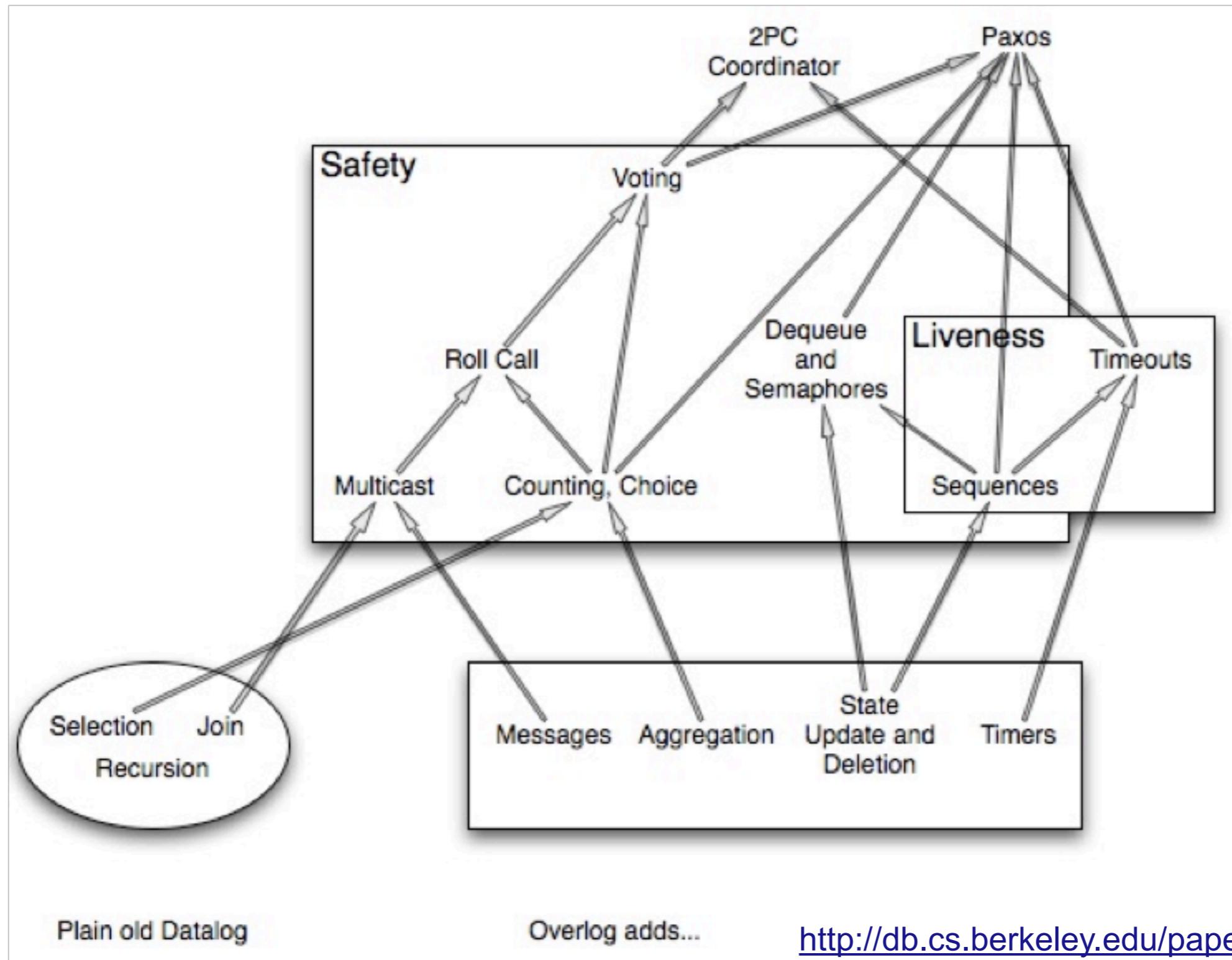
BASIC PAXOS

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

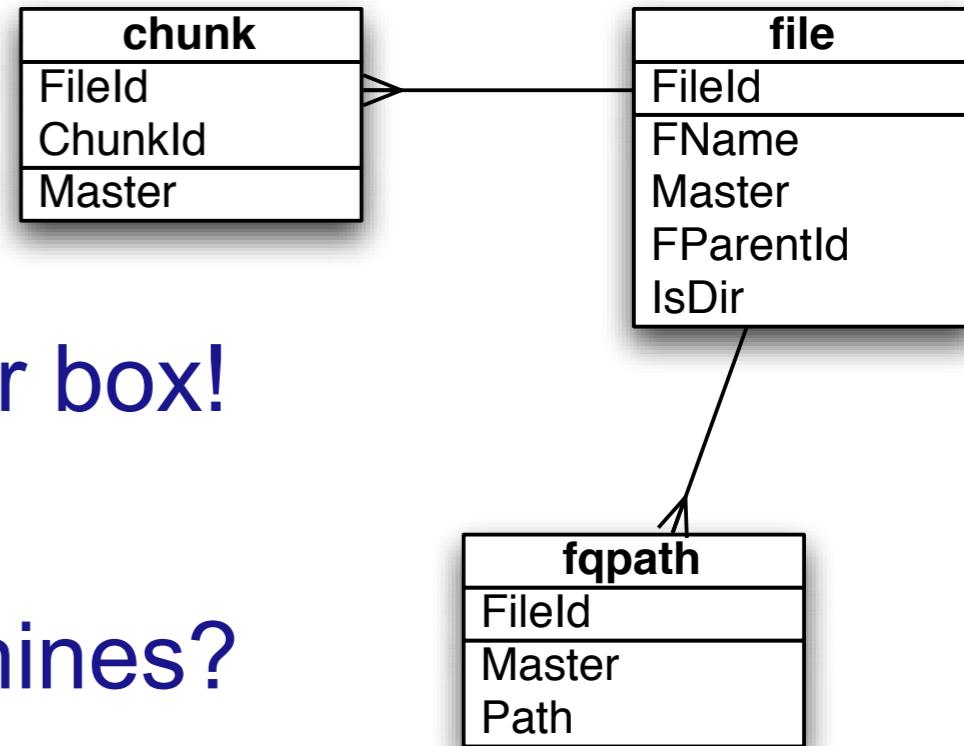
“I Do Declare...”, Alvaro, et al. NetDB 09





SCALABILITY REV

- ✿ master scaling woes? buy a bigger box!
- ✿ a real problem at Yahoo
- ✿ “scale out” master to multiple machines?
- ✿ massive rewrite in HDFS. trivial in BOOM-FS!
- ✿ hash-partition metadata tables as you would in a DB
- ✿ lookups by unicast or broadcast
- ✿ task completed in *one day*
- ✿ by Rusty Sears, the “OS guy” on the team





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

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



LESSONS 2

✿ overlog limitations

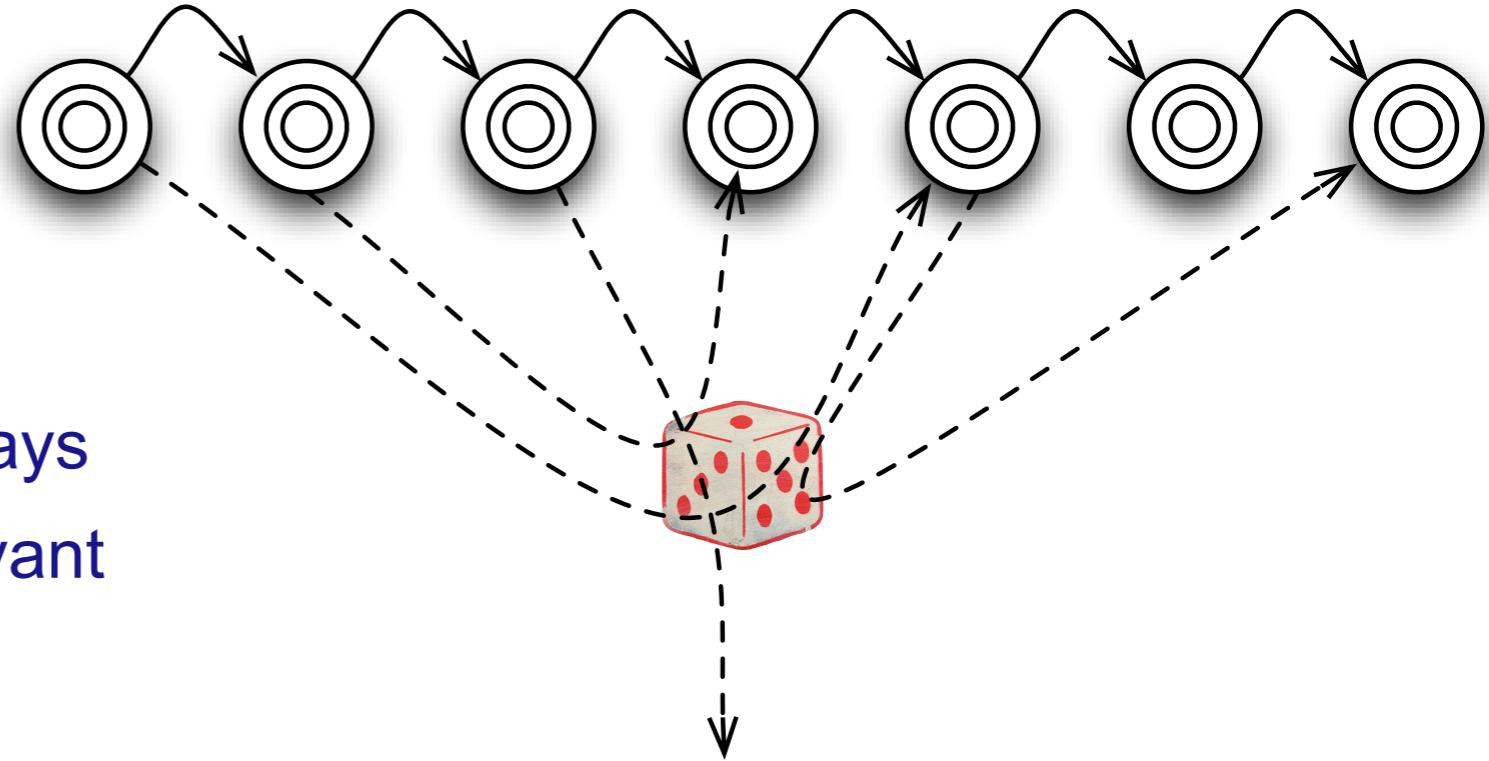
- ✿ datalog syntax: hard to write, *really* hard to read
- ✿ partitioned tables are a *lie*, so we don't use them
 - ✿ except as a layer above Paxos/2PC etc.
- ✿ state update is “illogical”
 - ✿ as noted in recent papers on operational semantics of P2

TODAY

- data-centric cloud programming
- datalog and overlog
- a look at BOOM
- a whiff of Bloom
- directions

TIME AND SPACE

- ✿ there is no space. only time.
 - ✿ now.
 - ✿ next.
 - ✿ later.
 - ✿ machine boundaries induce unpredictable delays
 - ✿ otherwise space is irrelevant
- ✿ time is a fiction
- ✿ *Dedalus*: a temporal logic capturing state update, atomicity/visibility, and delays





BLOOM: CORE LANGUAGE

Batch

- what to deQ when. defines a “trace”

Logic

- “now”: derivations, assertions, invariants

Operations

- “next”: local state modification, side effects

Orthography

- i.e., acronym enforcement

Messages

- “later”: network xmission, asynchronous calls

A NOTE TO READERS OF THE SLIDES

- the following slide is a ruby-ish “mockup” of what Bloom might look like.
- Bloom itself was not specified at the time of this talk
- Hence “v. -1”



SHORTEST PATHS: BLOOM v. -1

BATCH:

each path or every 1 second;

LOGIC:

```
table link [String from, String to] [integer cost];  
  
define path [String from, String to] [String nexthop, integer cost] {  
    link.each |l| :  
        yield { [l.from, l.to] => [l.to, l.cost] };  
  
    (path.to->link.from).each |p,l| :  
        yield { [p.from, l.to] => [p.nexthop, p.cost + l.cost] };  
}  
  
define shortest_paths [String from, String to] [integer cost] {  
    least = path.reduce([from,to] => [min(cost)]);  
    (path.[from,to]->least[from,to]).each |p,l| :  
        yield { [p.from, p.to] => [p.nexthop, l.cost] }  
}
```

OPS:

MSGS:

```
path.each |p| { send(p.from, p) if p.from != localhost }
```

TODAY

- data-centric cloud programming
- datalog and overlog
- a look at BOOM
- a whiff of Bloom
- directions



BOOM AGENDA

- ✿ continue pushing Hadoop community
 - ✿ e.g. HOP for streams and online agg
- ✿ from analytics to interactive apps
 - ✿ C4: a low-latency (explosive) runtime
 - ✿ towards a more complete Cloudstack
 - ✿ multifaceted/ambitious look at storage consistency
 - ✿ cloud operator/service management
 - ✿ monitoring/prediction/control (w/Guestrin@CMU)
 - ✿ secure analytics, nets (w/DawnSong, Mitchell@Stanford, Feamster@GTU)



BLOOM AGENDA

✿ Syntax & Semantics

- ✿ nail down *Dedalus*
- ✿ integral syntax for time
 - ✿ now/next/later
- ✿ logic made approachable
- ✿ list comprehensions

✿ Static analysis

- ✿ parallelism/concurrency
- ✿ redistribution
- ✿ concurrency

✿ Debugging

- ✿ static checks
- ✿ message provenance
- ✿ distributed checkpt

✿ Complexity?

- ✿ resources are free
- ✿ coordination is expensive
- ✿ “coordination surfaces”
- ✿ randomization & approximation

QUERIES?



<http://www.declarativity.net>

 **remaining slides are backup**

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
- ✿ distributed ML inference CMU/Berkeley 08

OTHERS

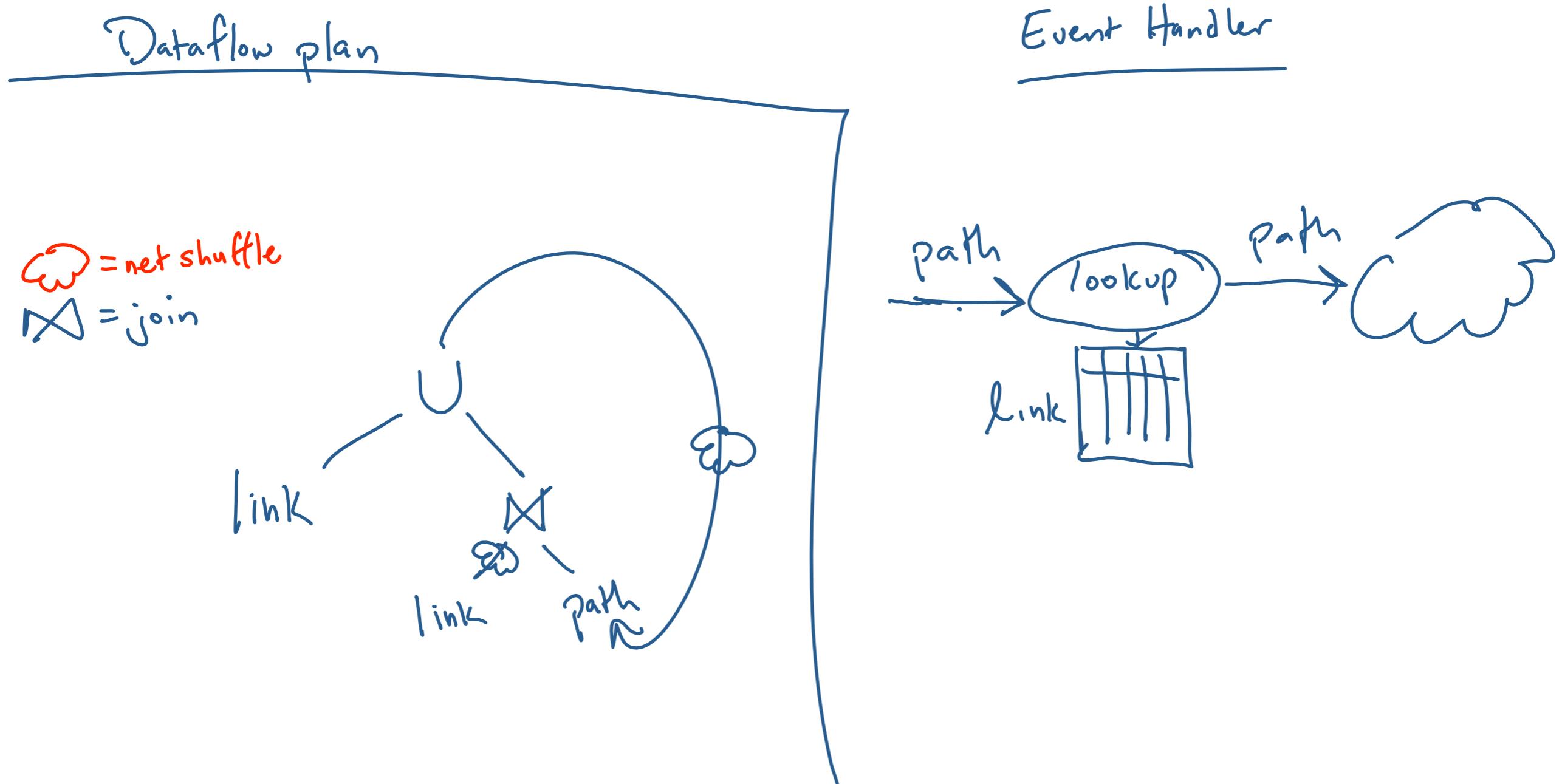
- ✿ video games ([sgl](#)) Cornell
- ✿ 3-tier apps ([hilda](#), [xquery](#)) Cornell, ETH, Oracle
- ✿ compiler analysis ([bddbddb](#)) Stanford
- ✿ nlp ([dyna](#)) Johns Hopkins
- ✿ modular robotics ([meld](#)) CMU
- ✿ trust management ([lbtrust](#)) Penn/LogicBlox
- ✿ security protocols ([pcl](#)) Stanford
- ✿ ... see <http://declarativity.net/related>

“BOTTOM-UP” EXECUTION

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

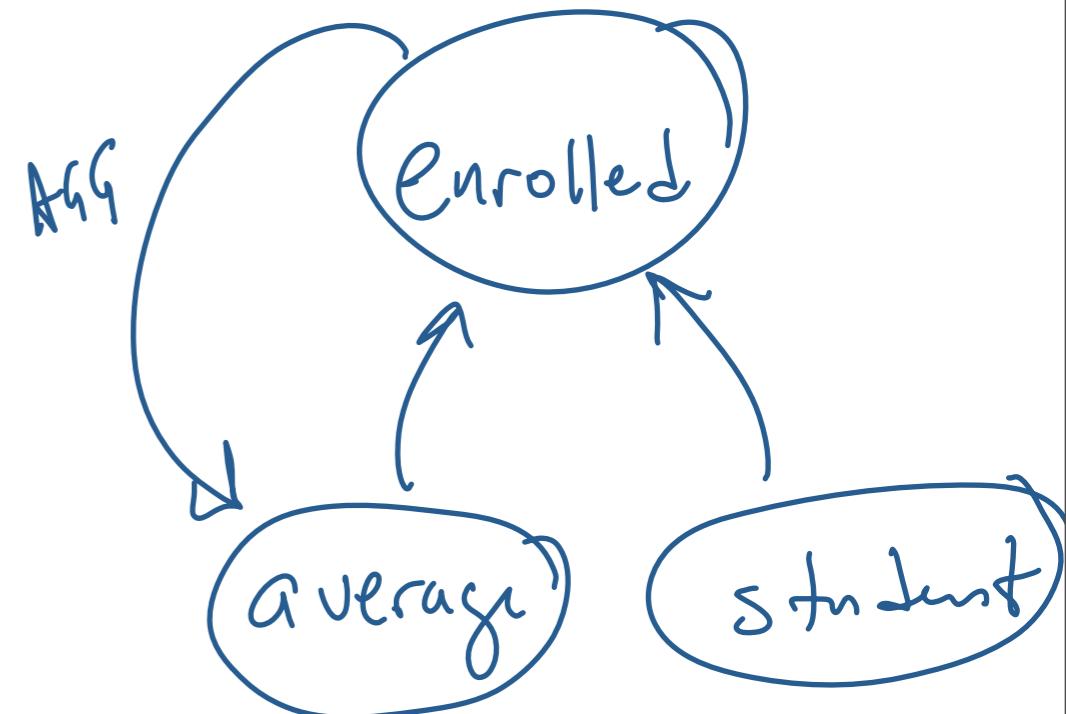
- ✿ Akin to RDBMS with recursion
 - ✿ join/project body predicates to derive new head facts.
 - ✿ repeat until *fixpoint*
- ✿ Optimization: avoid rederiving known facts
 - ✿ *semi-naive* evaluation

BOTTOM-UP EXECUTION

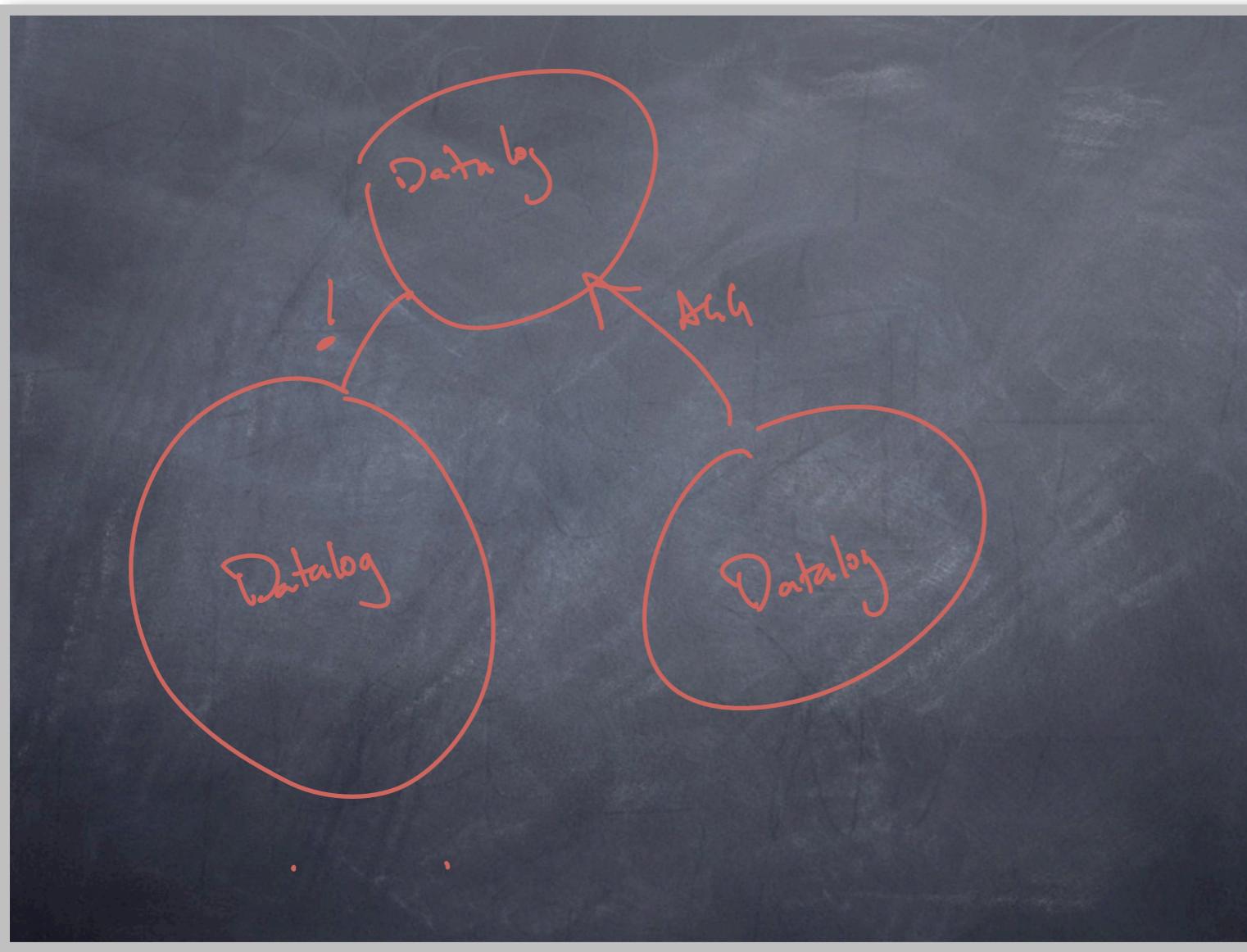


PROBLEMS IN LAKE WOBEGON (AGGS)

- ④ enrolled(N,A) :- student(N,A),
average(B), A > B.
- ④ average(avg<A>) :-
enrolled(N,A).
- ④ student(carlos, 30).
- ④ student(joey, 20).
- ④ enrolled(carlos, 30)?



STRATIFICATION



- no recursion through negation/aggregation
- lemma: evaluating strata in order of the dependency graph produces a (natural) minimal model!
- *local stratification:* similar lemma if no facts can ever recurse through negation/aggregation

SOME SIMPLE OVERLOG

SOME SIMPLE OVERLOG



Async 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).
```

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:

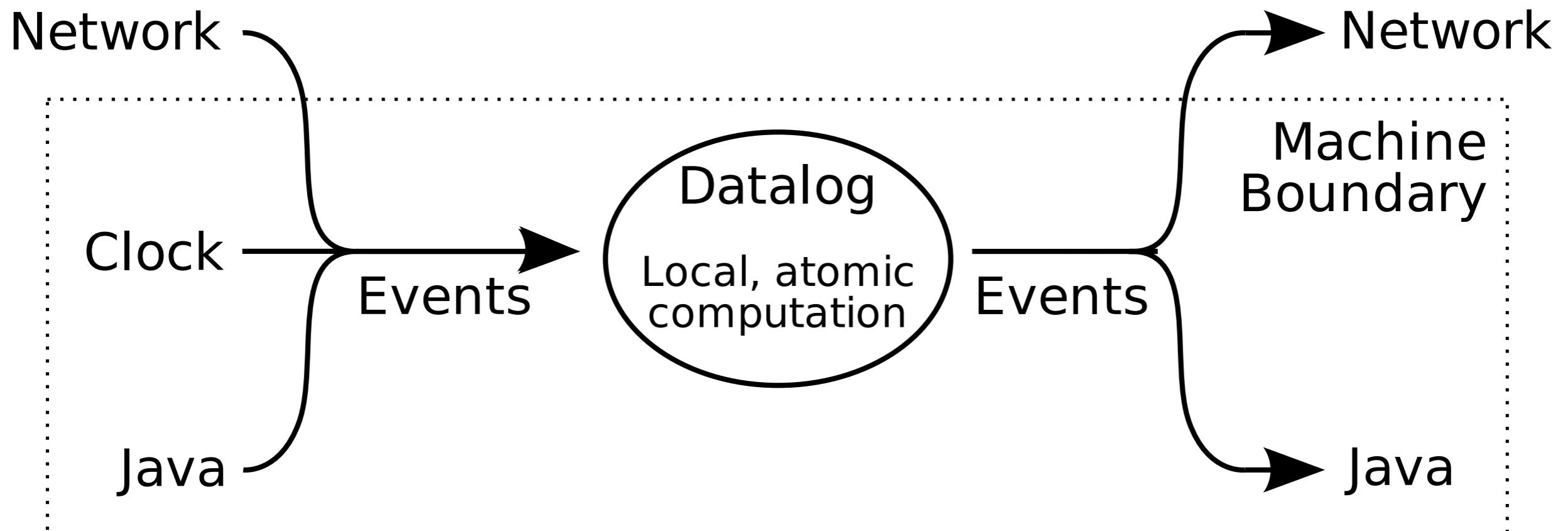
```
msg(@Dest, Payload) :- xmission(@Src, Payload),  
                      group(@Src, Dest).
```



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).
```

OVERLOG EXECUTION



KEY CONCEPTS IN DEDALUS

```
• link@4(1,2).  
• link@next(F,T) :- link(F, T).  
• path(F,T) :- link(F,N),  
              path(N,T).  
• msg@later(T,F,M)  
  :- link(F,T),  
    M = "howdy, neighbor".
```

- facts @constant.
- head predicates have timespecs
 - N, N+1, N+r()
- body predicates implicitly @N.

STATE UPDATE IN DEDALUS

- persistence:

- $r@next(X) :- r(X)$ and $\neg del_r(X)$.

- deletion:

- $del_r(X) :- msg(X)$.

- key update:

- $del_s(K,W) :- s(K,W), new(K,V)$.

- $s@next(K,V) :- new(K,V)$.

- “deferred” delete and update

- there's a gotcha here we're still ironing out...



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



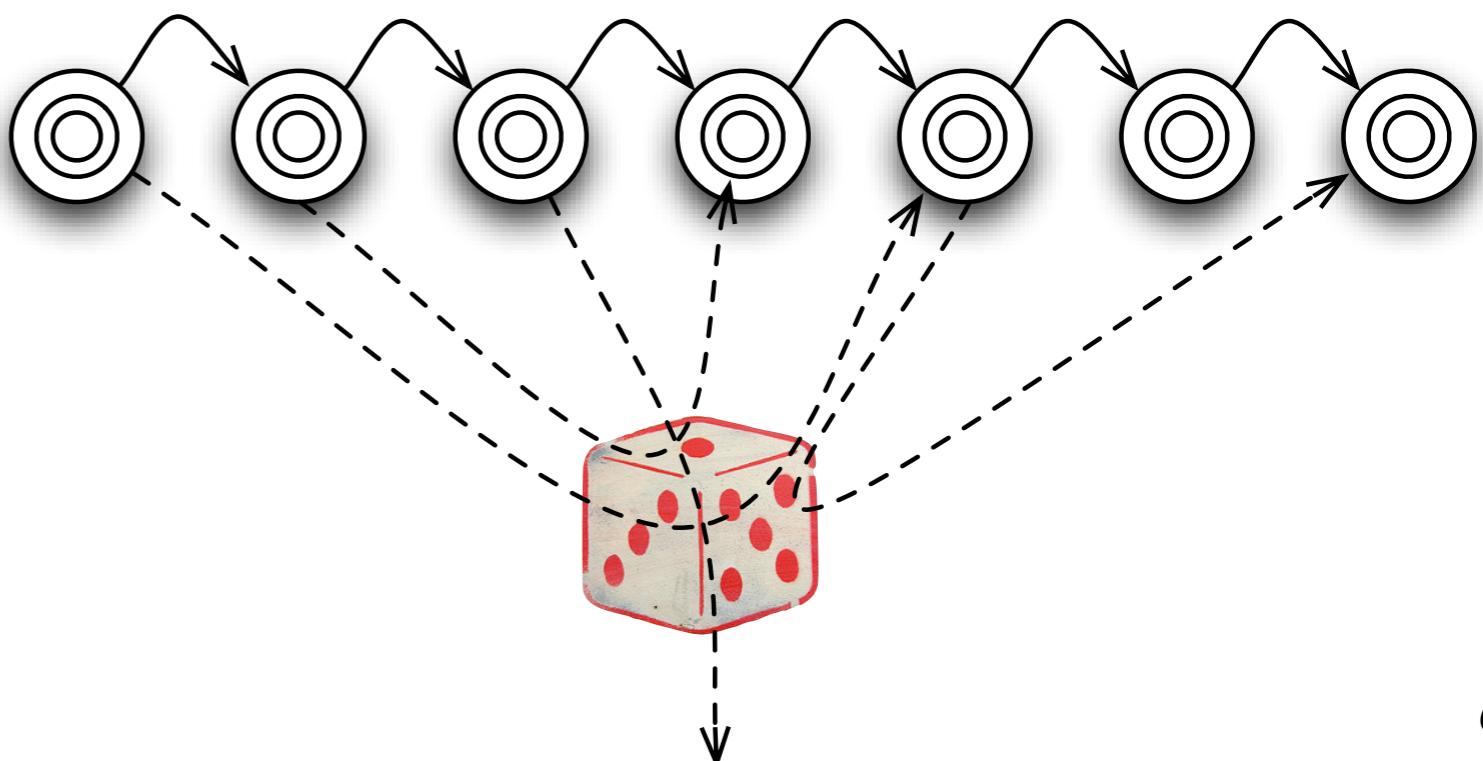
PARALLELISM?

- ✿ aggregation = stratification = “wait”
- ✿ natural analogy to counting semaphores
- ✿ this is the *only* reason for parallel barriers
- ✿ delay iff data dependencies depend on parallelism
 - ✿ or even cheat: approximate aggregates, speculation.



WORLD OUTSIDE THE LOGS

- ✿ the “trace” of a system
 - ✿ mapping between external sequence (msg queue) and system time
- ✿ “entanglement” of 2 systems
 - ✿ relationship between msgs in their traces





TIME IS STRATIFICATION

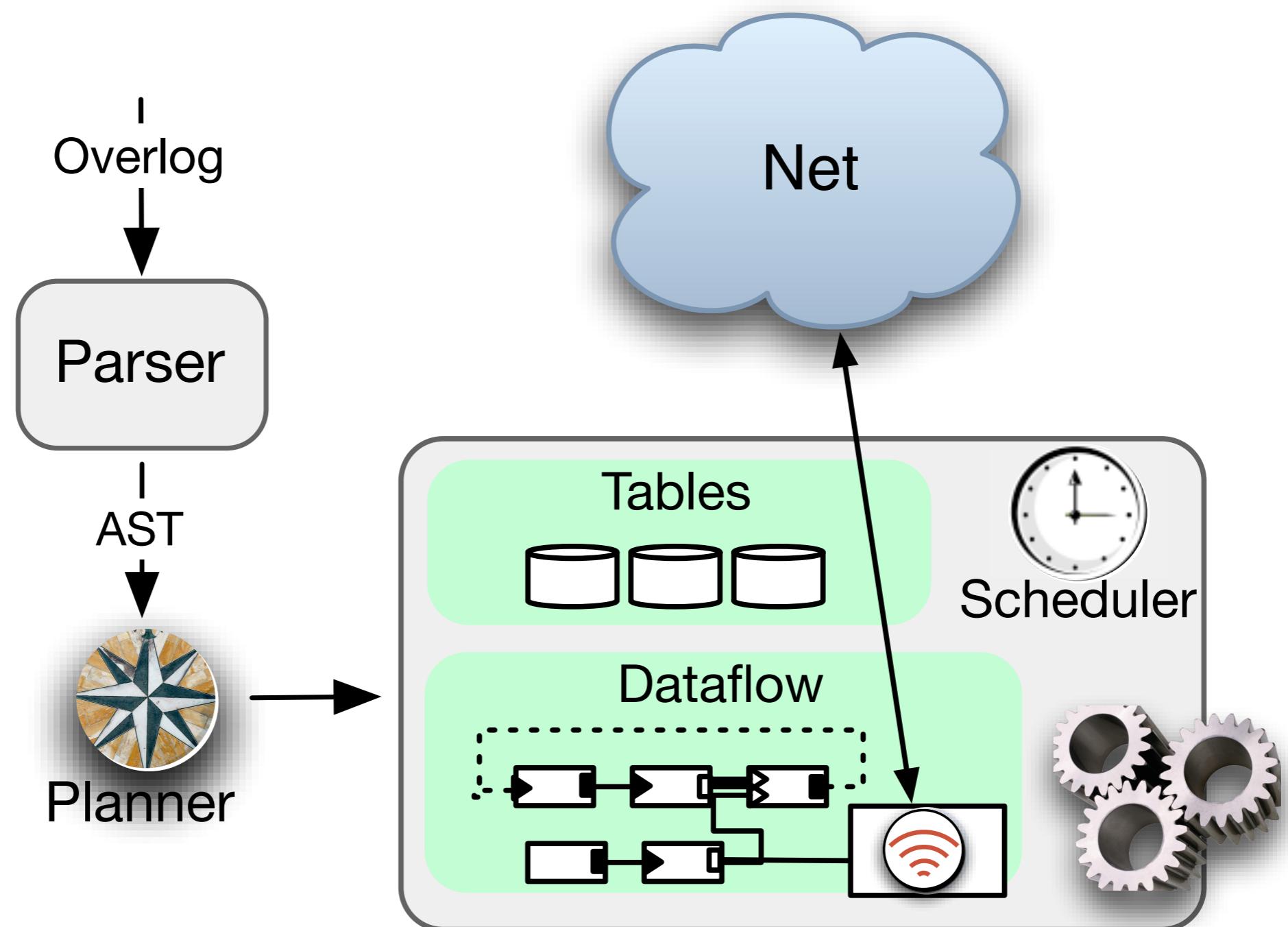
- chains of inference on independent data can be “rescheduled”
- prove two “traces” equivalent.



LAMPORT CLOCKS?

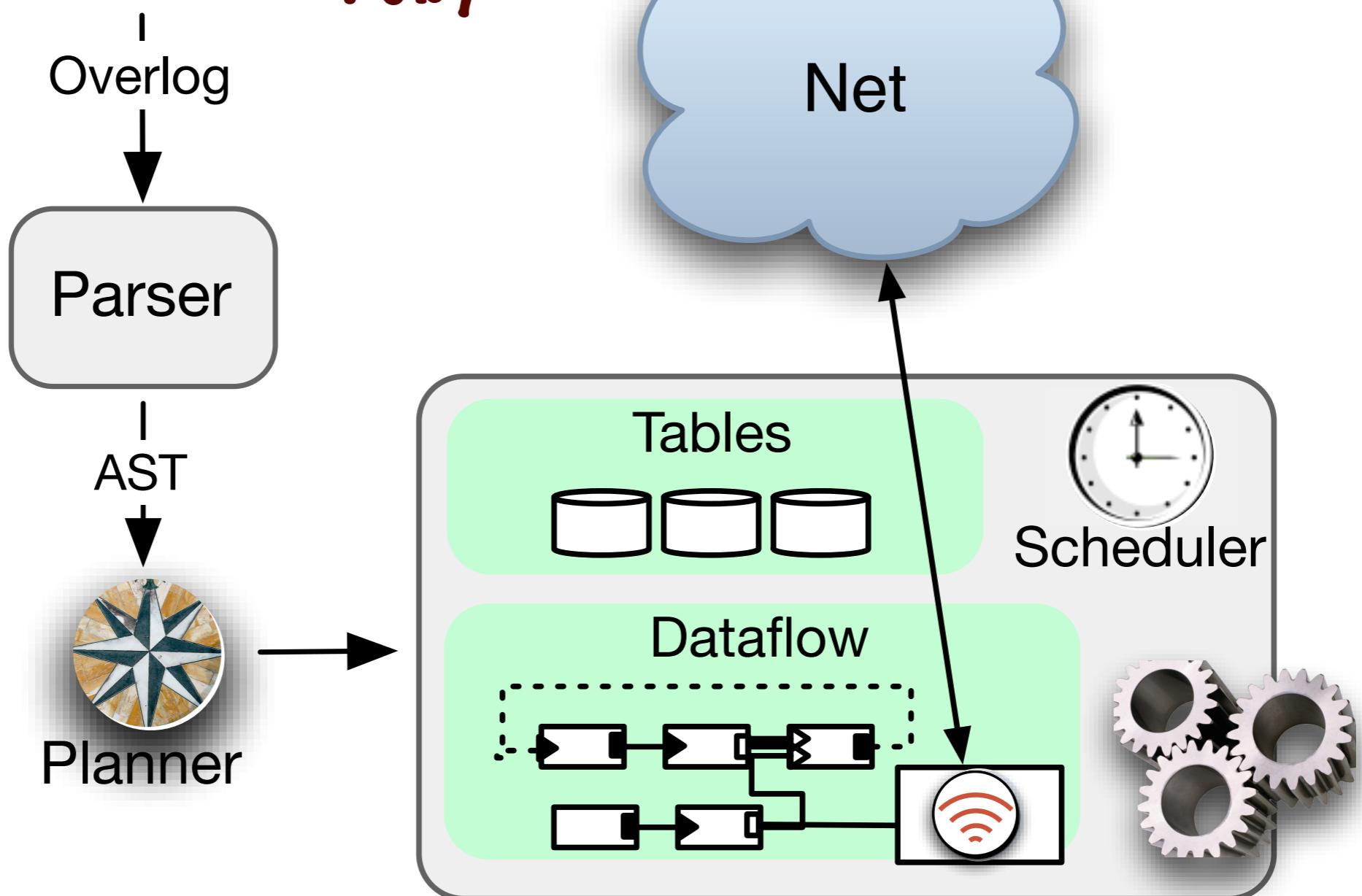
- ✿ “causal” ordering
- ✿ “happens before”
- ✿ our “cause” is data dependency.
- ✿ what else “happens”?!
 - ✿ captured faithfully (statically and dynamically) via logic.

P2 @ 10,000 FEET



~~P2~~ @ 10,000 FEET

java,
ruby

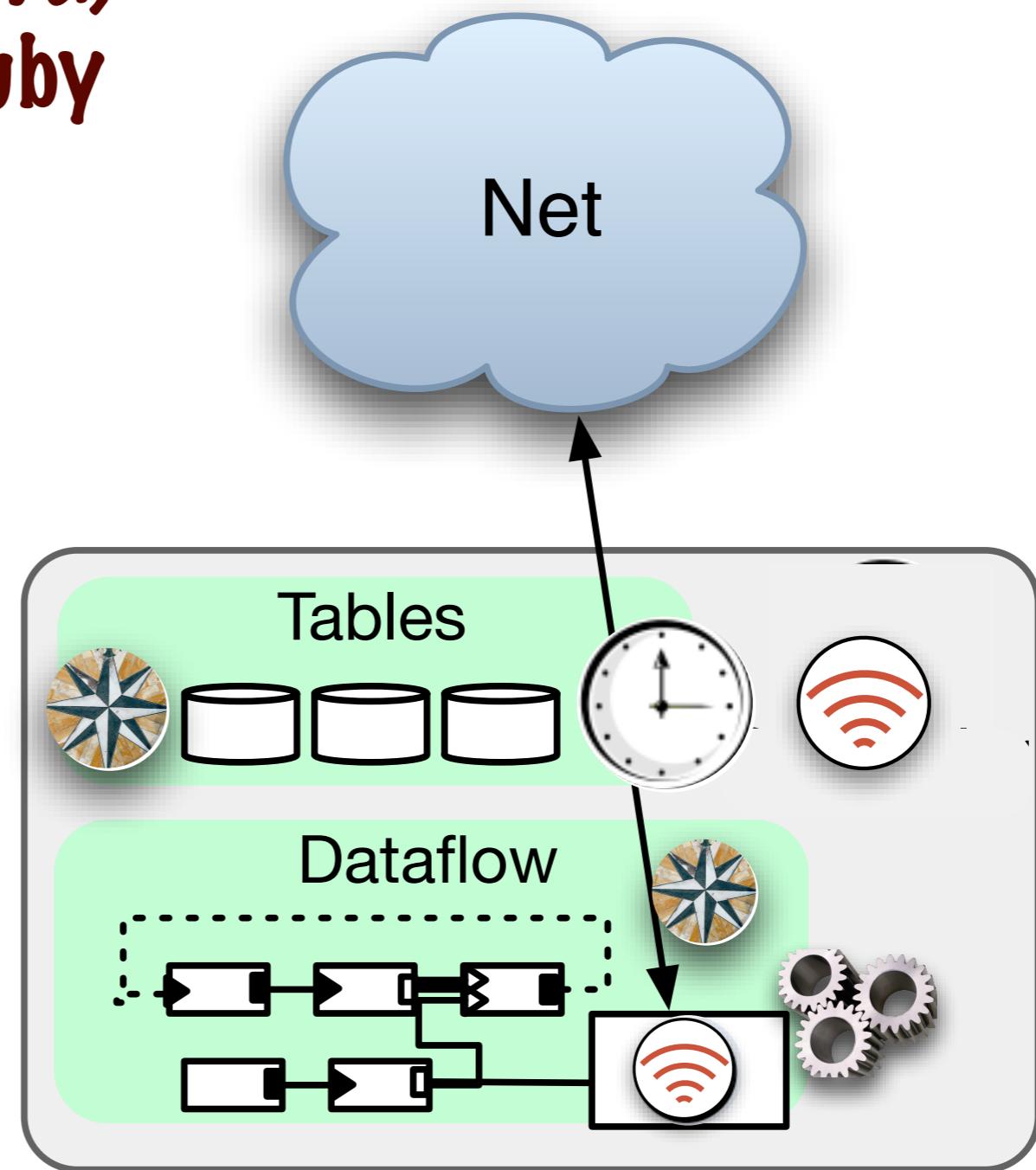


~~P2~~ @ 10,000 FEET

java,
ruby

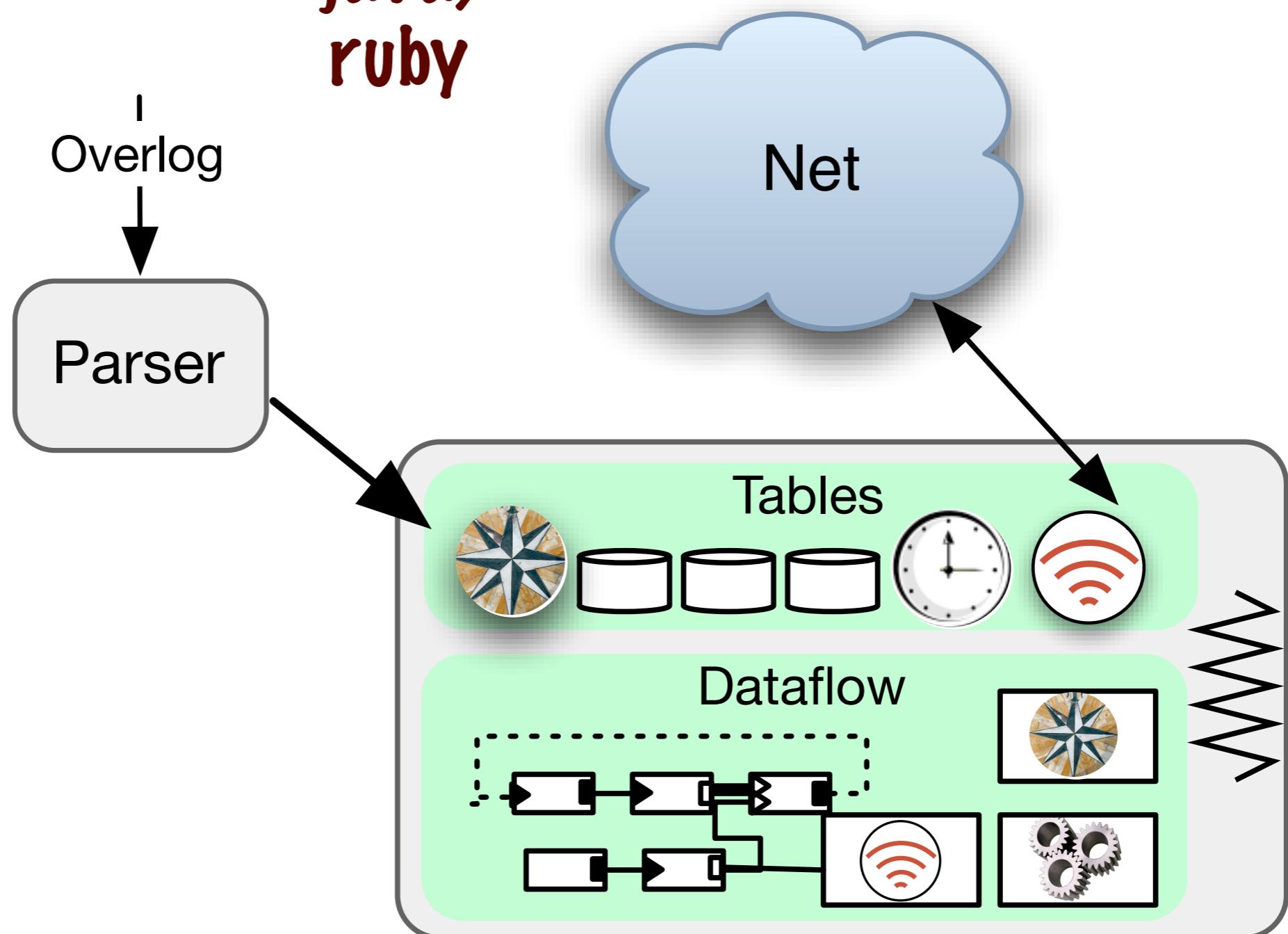
I
Overlog

Parser



~~P2~~ @ 10,000 FEET

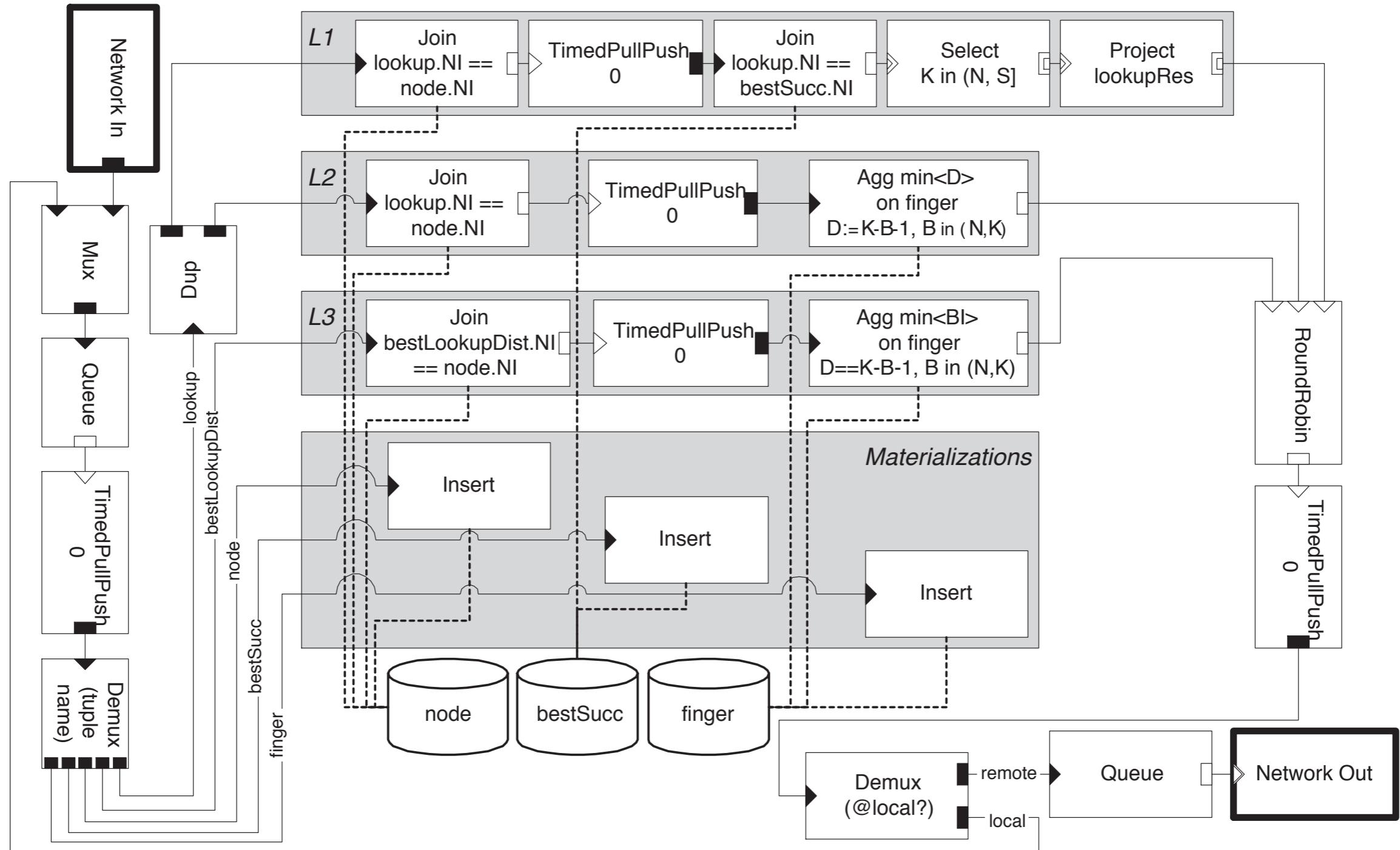
java,
ruby



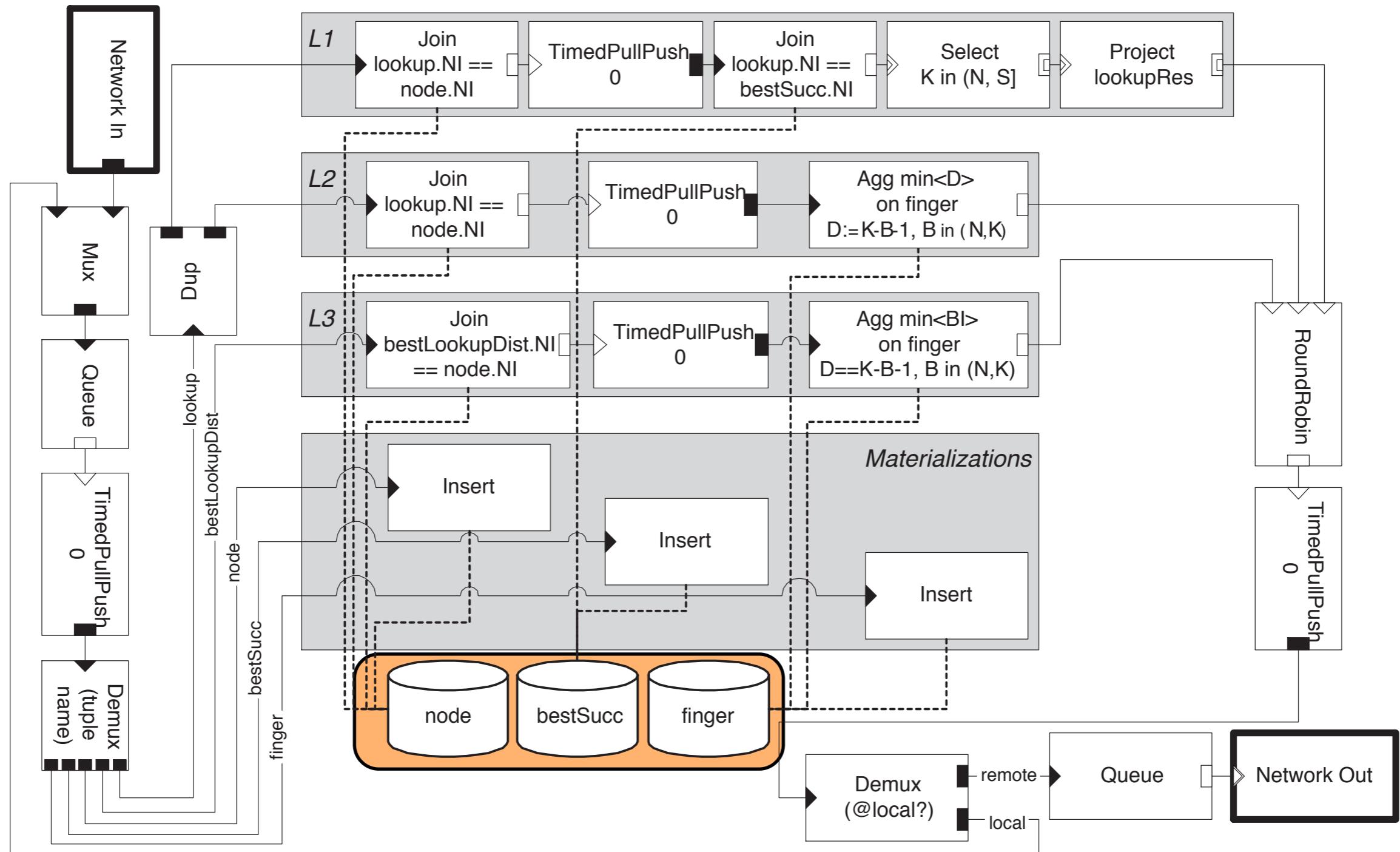
DATAFLOW EXAMPLE IN P2

- ✿ 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).`

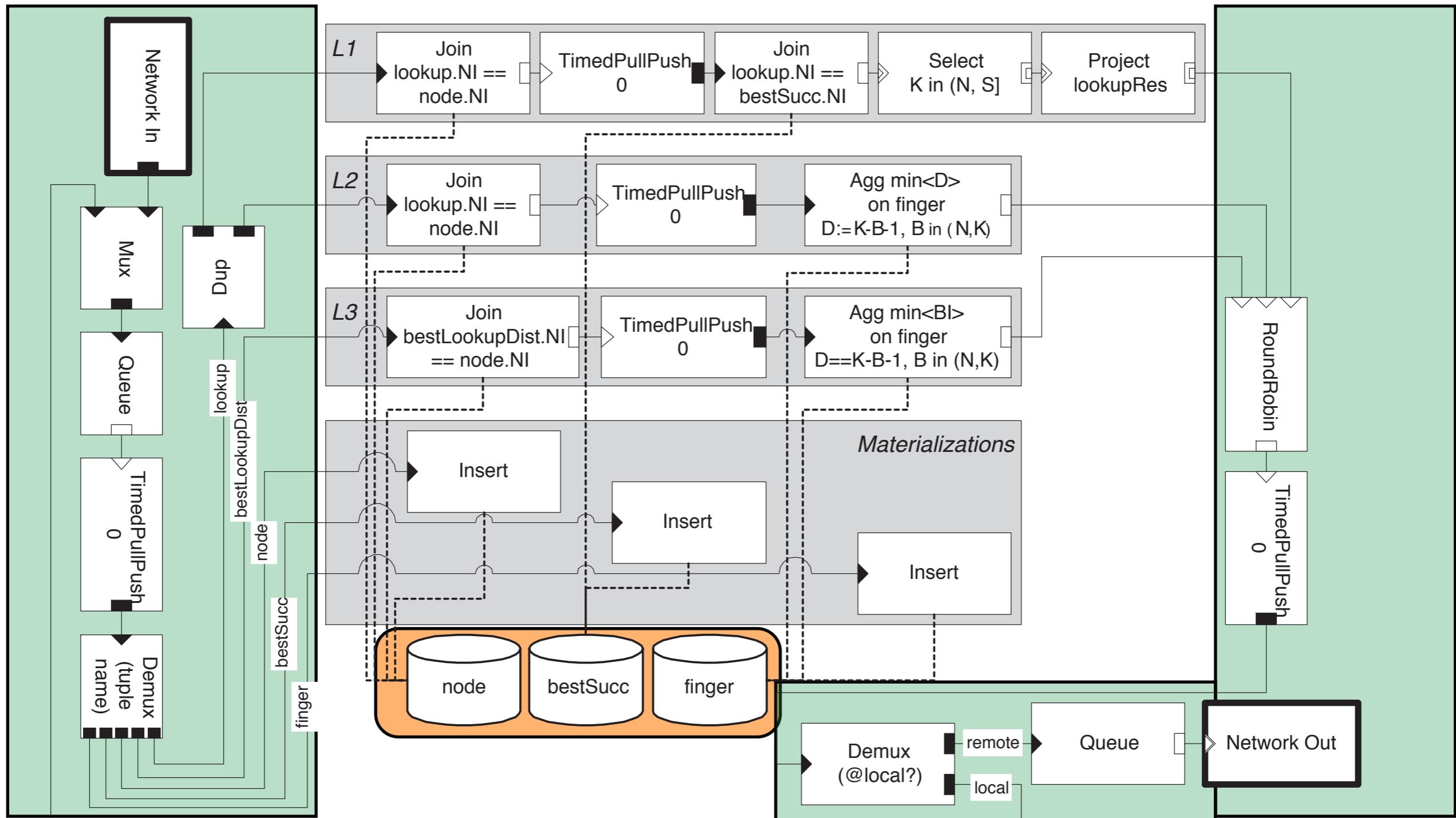
DATAFLOW EXAMPLE IN P2



DATAFLOW EXAMPLE IN P2



DATAFLOW EXAMPLE IN P2

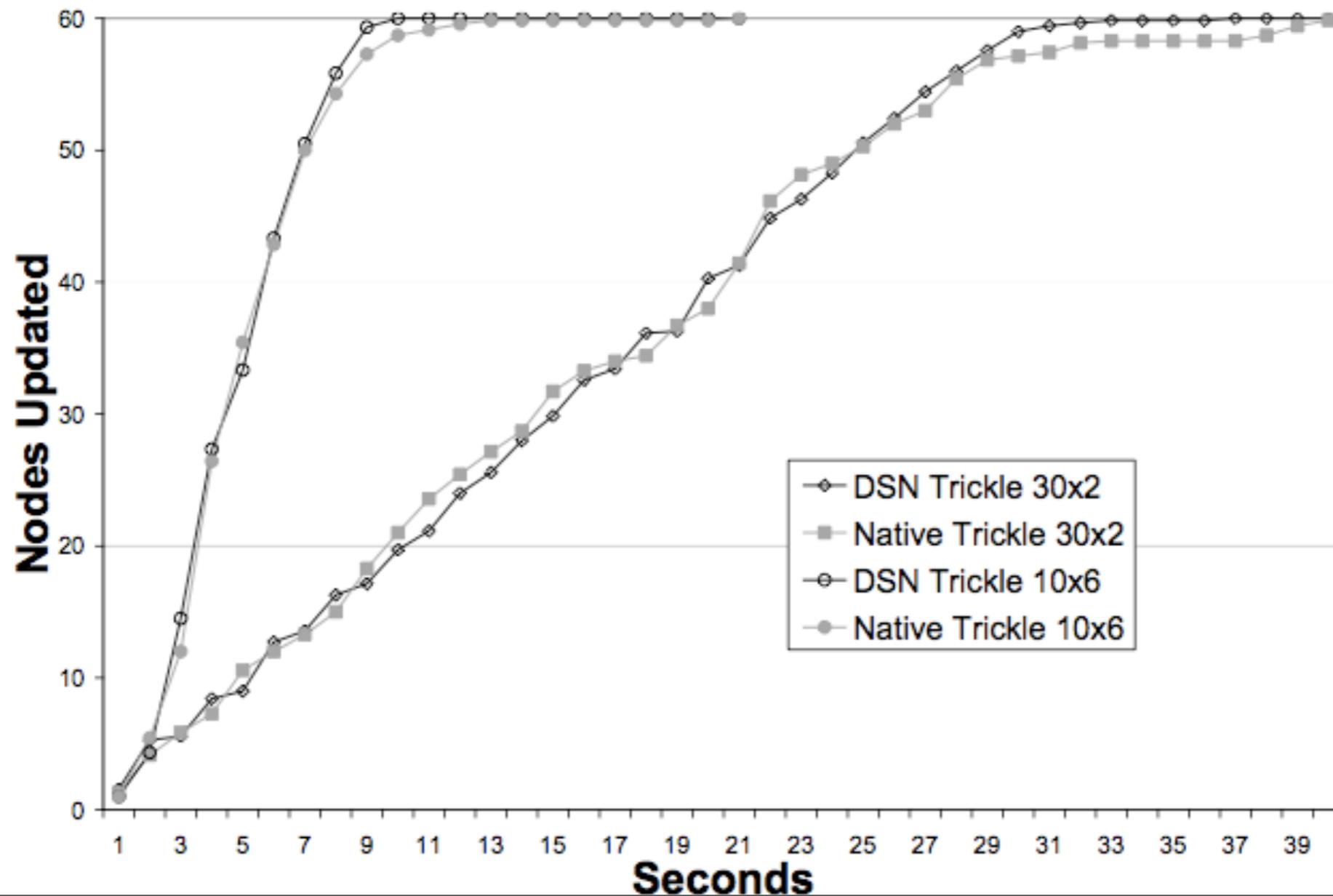


NOTES

- flow runs at multiple nodes
- data partitioned by locspec
- this is SPMD parallel dataflow
- a la database engines, MapReduce
 - locspecs can be hash functions via content routing
 - unlike MapReduce, finer-grained operators that pipeline

DSN vs NATIVE TRICKLE

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

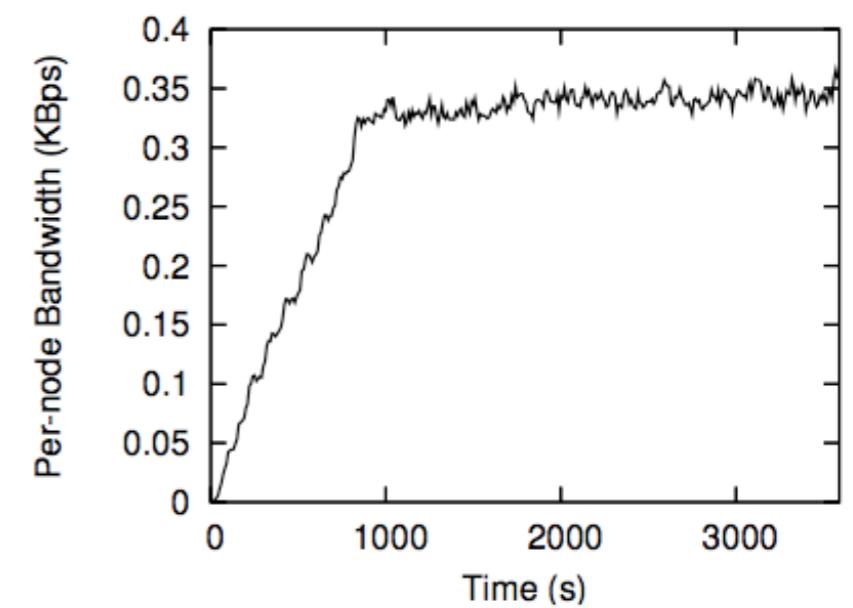
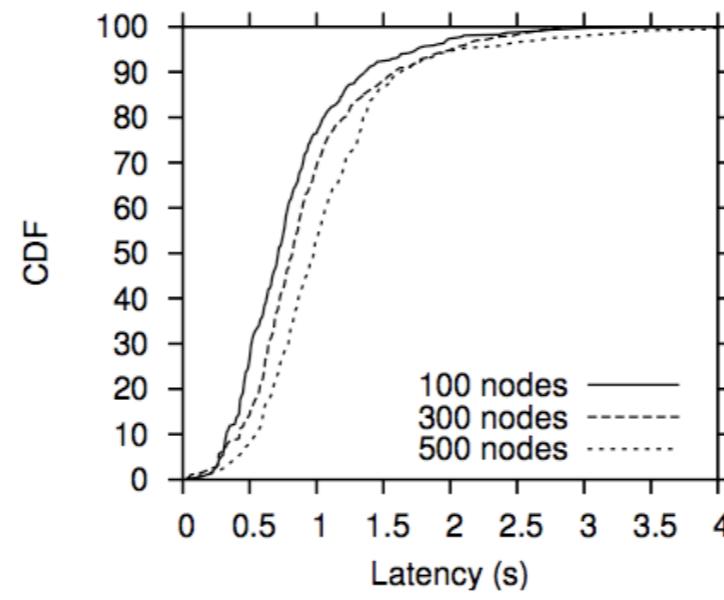
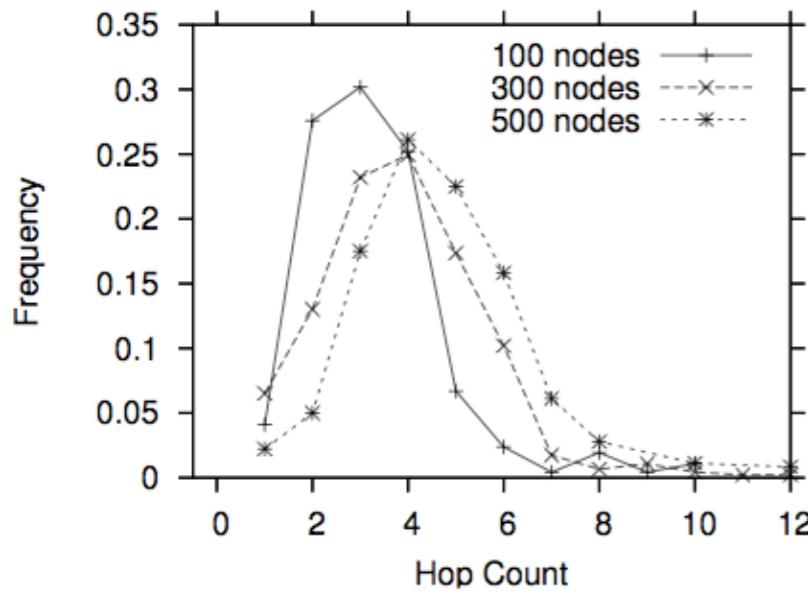


DSN vs NATIVE TRICKLE

	Native	DSN
LOC	560 (NesC)	13 rules, 25 lines
Code Sz	12.3KB	24.4KB
Data Sz	0.4KB	4.1KB

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

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

SEMANTICS

- ★ Dedalus is really Datalog

- ★ with negation/aggs, a *successor* relation for time, and a non-deterministic function (for *later*)
- ★ time an attribute of each table
- ★ rewrite rule bodies to include “now predicates”.

- ★ Dedalus semantics: minimal model

- ★ with “don’t-care” semantics on non-deterministic values
- ★ some details to work out here

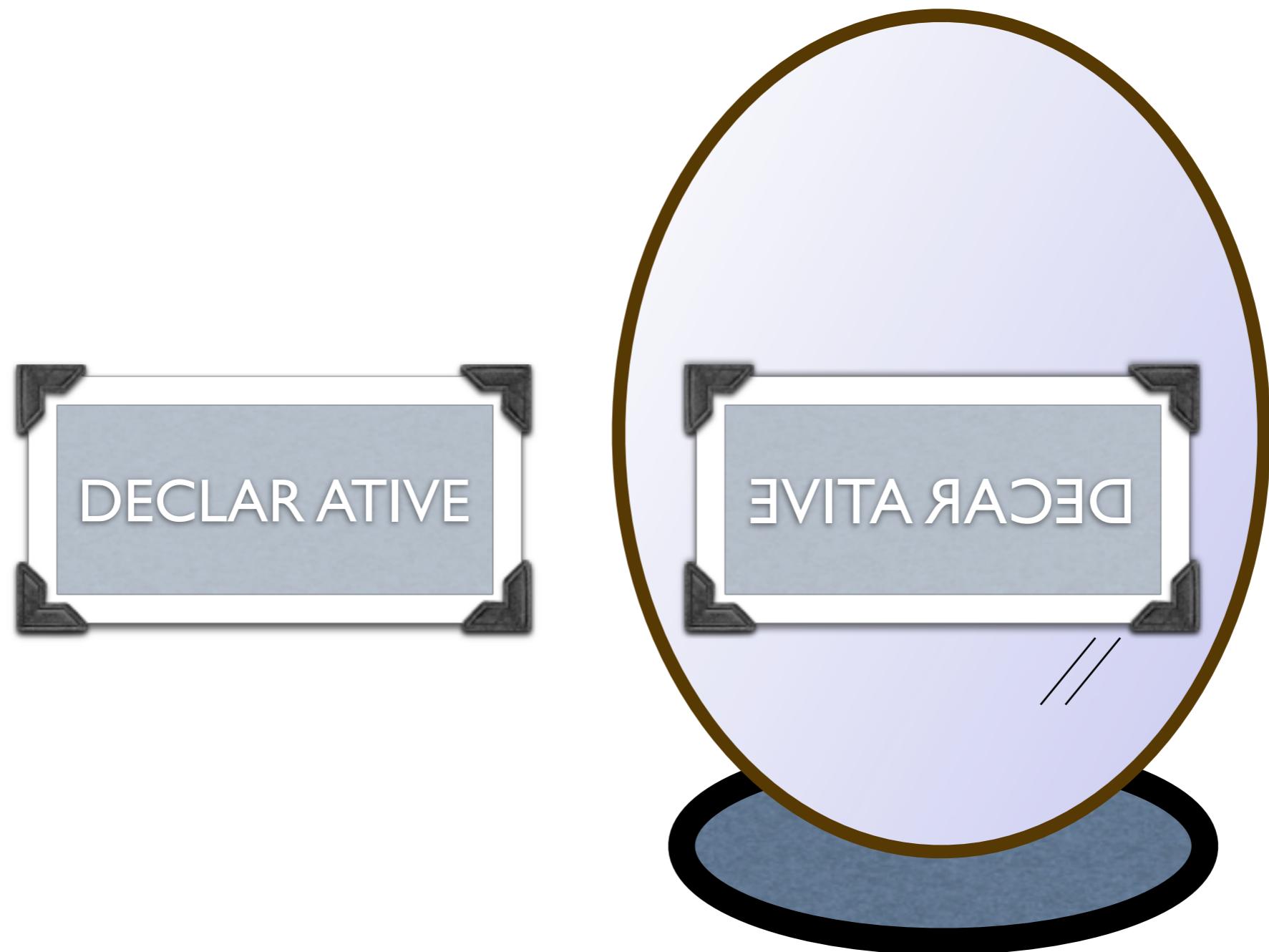
DEDALUS EXECUTION

- ✿ given a fixed input DB, can just run semi-naive eval.
- ✿ assertion: “now predicate” locally stratifies on (monotonically increasing) time
- ✿ challenge: “implement” minimal model of a Dedalus program via “traditional” persistence
 - ✿ i.e. store, don’t re-derive.

EVITA RACED: OVERLOG METACOMPILER



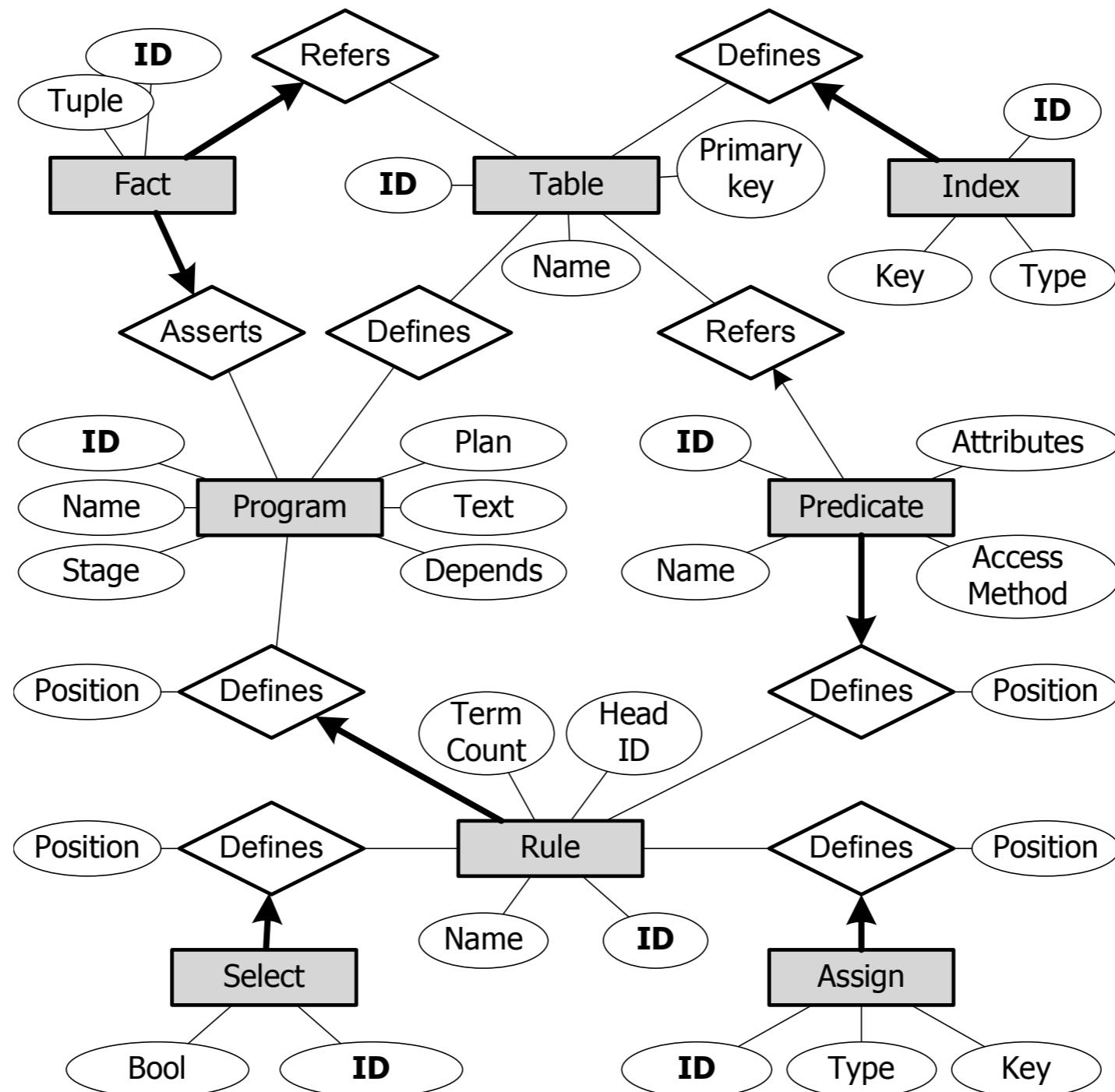
EVITA RACED: OVERLOG METACOMPILER



EVITA RACED: OVERLOG METACOMPILER

- ✿ represent:
 - ✿ overlog as data
 - ✿ optimizations as overlog
 - ✿ optimizer stage schedule as a lattice -- i.e. data
- ✿ needs just a little bootstrapping
 - ✿ optimization as “hand-wired” dataflow

OVERLOG AS DATA



OPTIMIZER AS OVERLOG

- ✿ System R's Dynamic Programming
 - ✿ 38 rules
- ✿ Magic Sets Rewriting
 - ✿ 68 rules
 - ✿ close translation to Ullman's course notes
- ✿ VLDB Feedback story
 - ✿ replaced System R with Cascades Branch-and-Bound search
 - ✿ 33 rules, 24 hours
 - ✿ paper accepted

SOME LESSONS

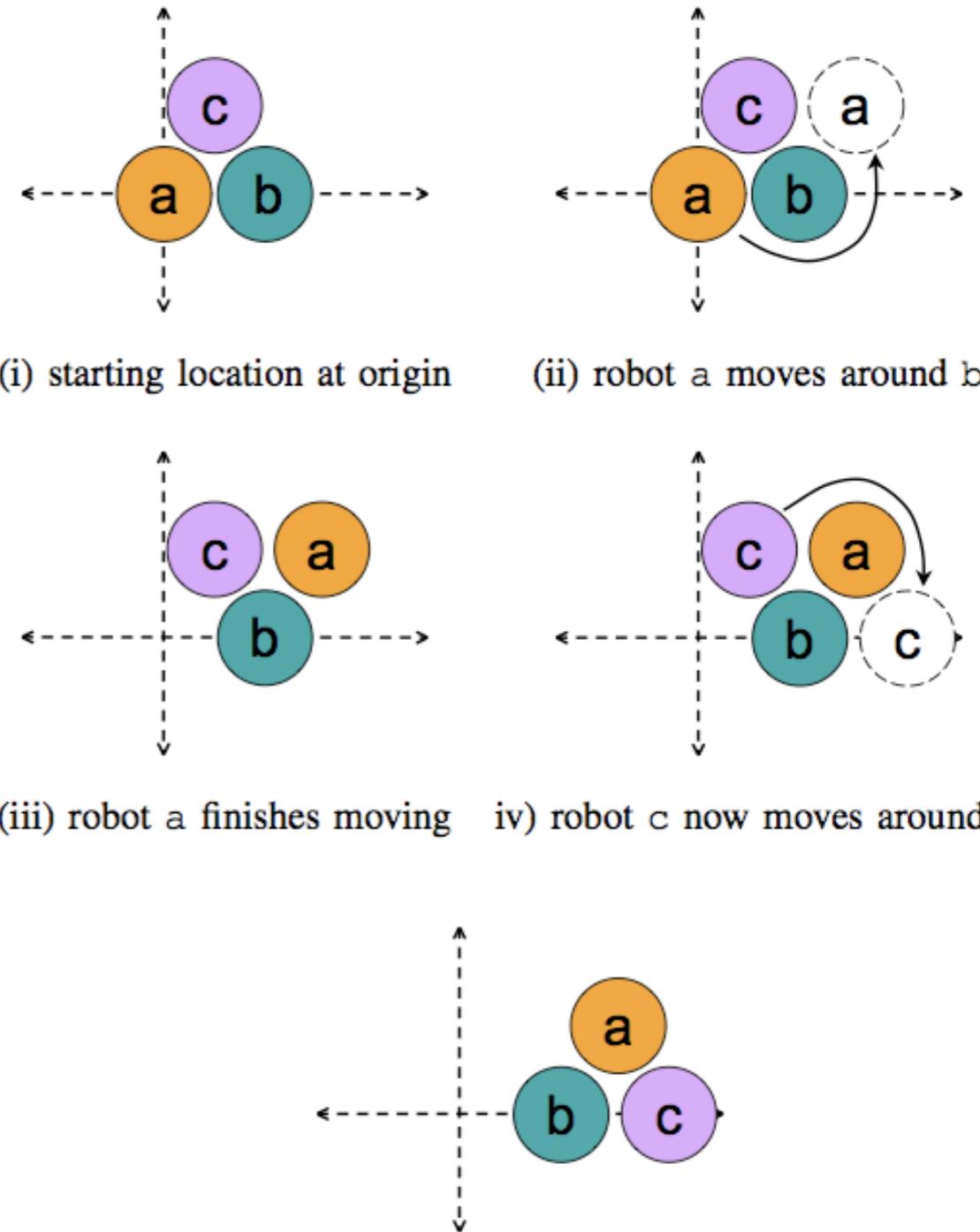
- ✿ dynamic programming & search
 - ✿ another nice fit for declarative programming
- ✿ extensible optimizer really required
 - ✿ e.g. protocol optimization not like a DBMS
 - ✿ graph algorithms vs. search-space enumeration

MOVING CATOMS IN MELD

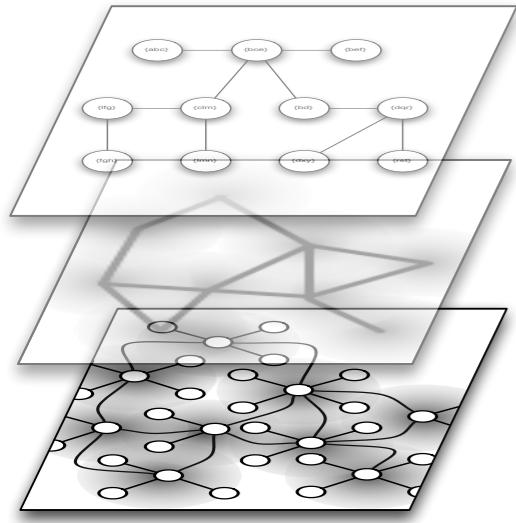
RULE1: $\text{Dist}(S, D) :- \text{At}(S, P),$
 $P_d = \text{destination}(),$
 $D = |P - P_d|,$
 $D > \text{robot radius}.$

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

RULE3: $\text{MoveAround}(S, T, U) :- \text{Farther}(S, T),$
 $\text{Farther}(S, U),$
 $U \neq T.$



[Ashley-Rollman, et al. IROS '07]

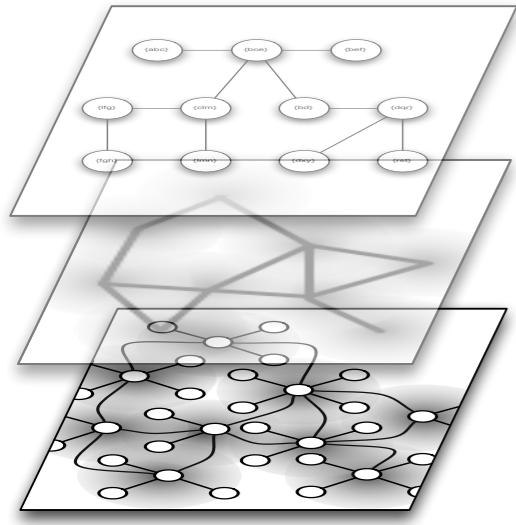


DISTRIBUTED INFERENCE

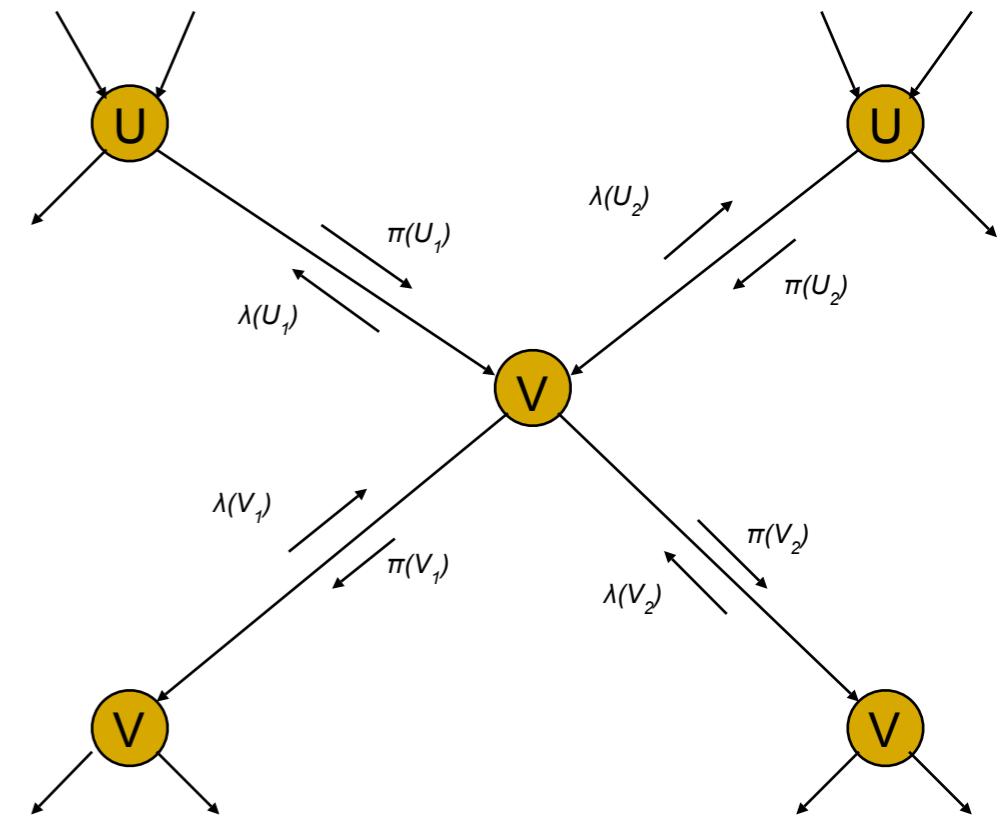
- ✿ challenge: real-time distributed info
 - ✿ despite uncertainty and acquisition cost

- ✿ applications
 - ✿ internet security, building control, disaster response, robotics
 - ✿ really ANY distributed query.

INFERENCE (CENTRALIZED)

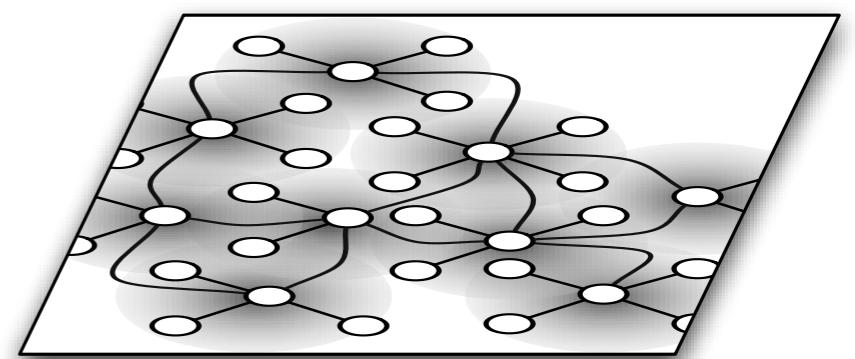


- ✿ given:
 - ✿ a graphical model
 - ✿ node: random variable
 - ✿ edge: correlation
 - ✿ evidence (data)
- ✿ find probabilities for RVs
- ✿ tactic: belief propagation
 - ✿ a “message passing” algorithm



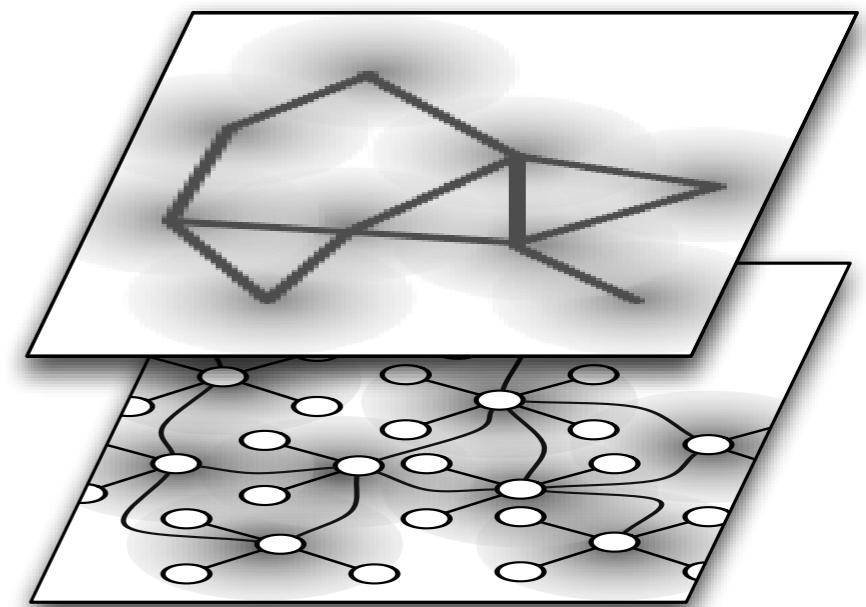
DISTRIBUTED INFERENCE

- ✿ graphs upon graphs
- ✿ each can be easy to build
- ✿ opportunity for rich cross-layer optimization



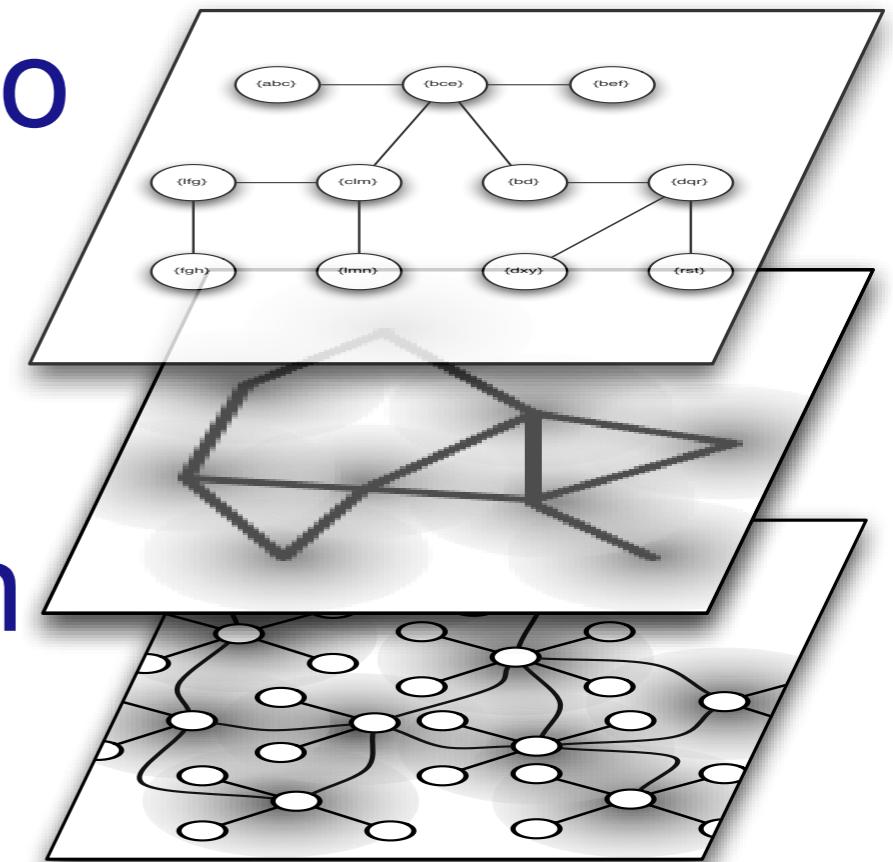
DISTRIBUTED INFERENCE

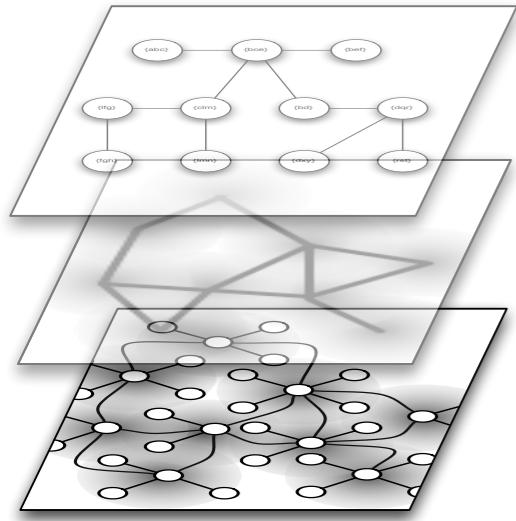
- graphs upon graphs
- each can be easy to build
- opportunity for rich cross-layer optimization



DISTRIBUTED INFERENCE

- graphs upon graphs
- each can be easy to build
- opportunity for rich cross-layer optimization





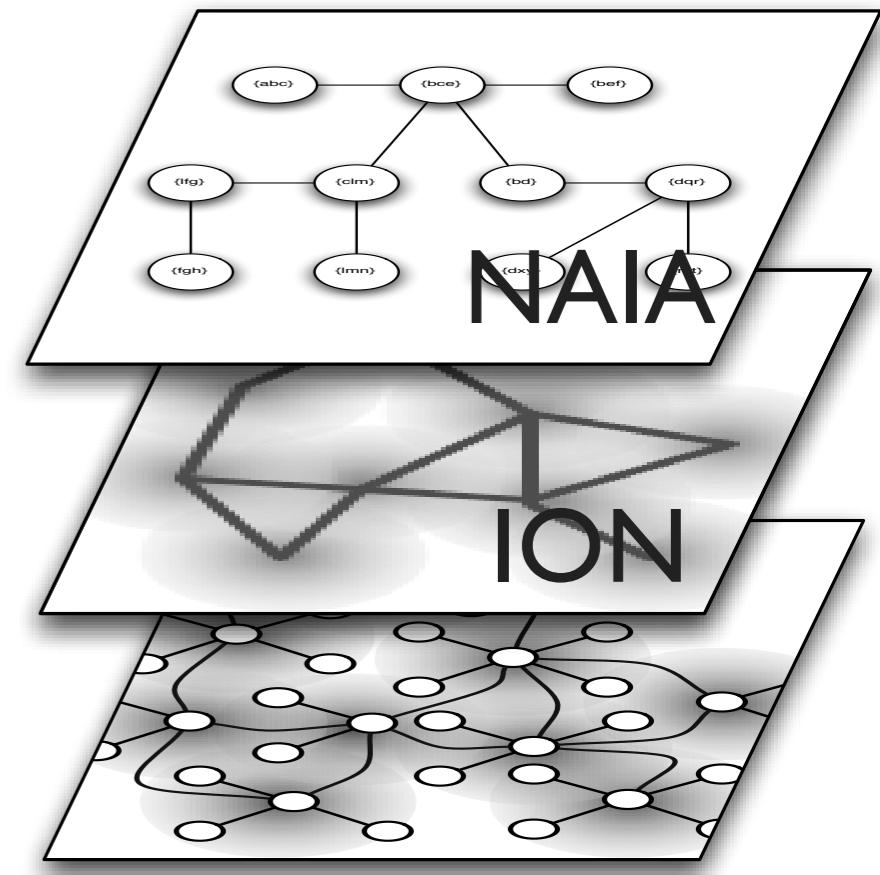
DECLARATIVE DISTRIBUTED INFERENCE

- ✿ even fancy belief propagation is not bad
- ✿ robust distributed junction tree 39 rules
 - ✿ 5x smaller than Paskin's Lisp
 - ✿ + identified a race condition
- ✿ also variants of Loopy Belief Propagation

[Funiak, Atul, Chen, Guestrin, Hellerstein, 2008]

RESEARCH ISSUES

- optimization at each layer.
- custom Inference Overlay Networks (IONs)
- network-aware approximate inference algorithms (NAIAs)
- optimization across layers?
- co-design to balance NW cost and approximation quality



RESEARCH ISSUES

- ✿ optimization at each layer.
- ✿ custom Inference Overlay Networks (IONs)
- ✿ network-aware approximate inference algorithms (NAIAs)
- ✿ optimization across layers?
- ✿ co-design to balance NW cost and approximation quality

