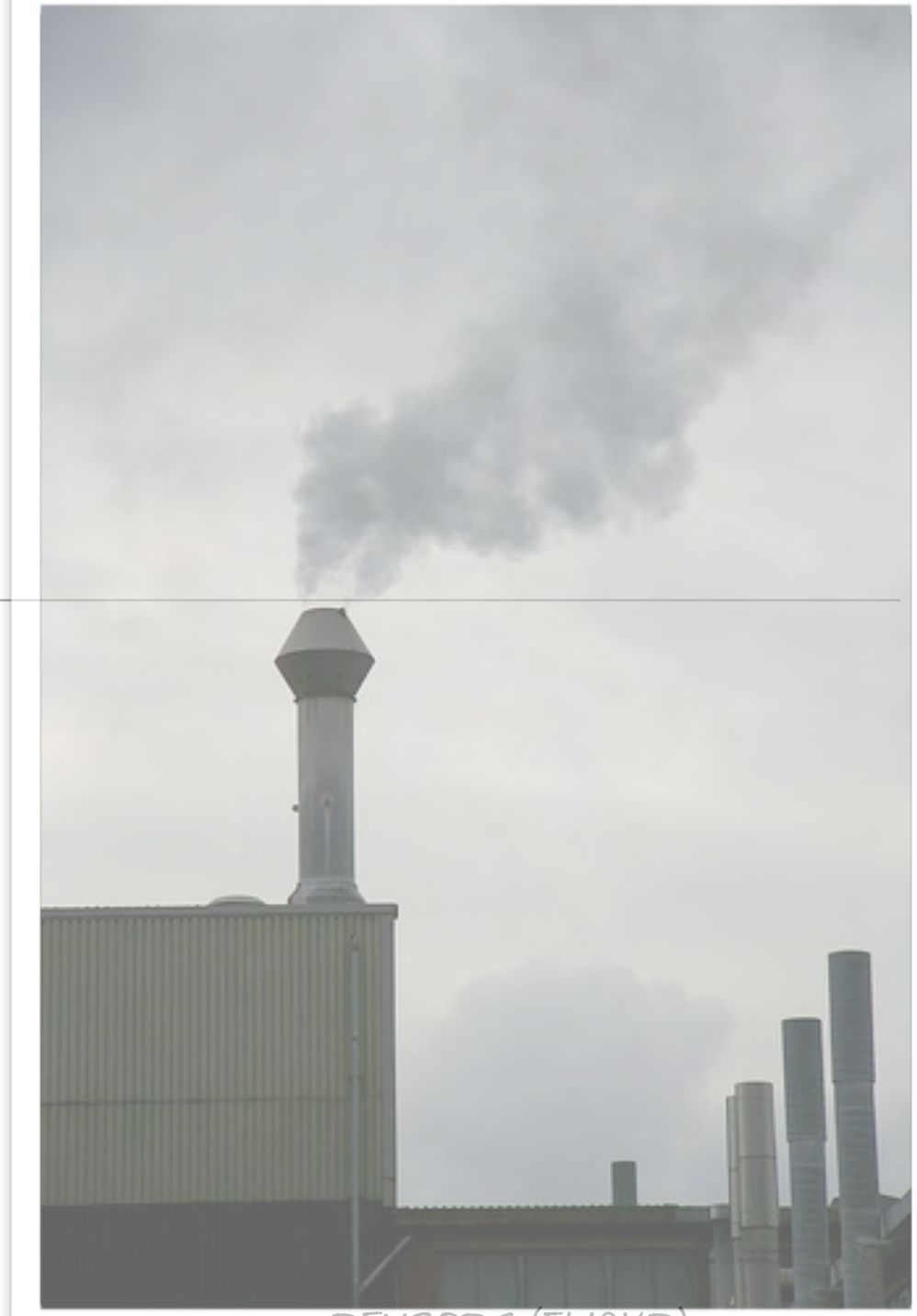
industrial revolution: data ... and software?

joe hellerstein
uc berkeley

CCC Council Meeting
03.10.2008

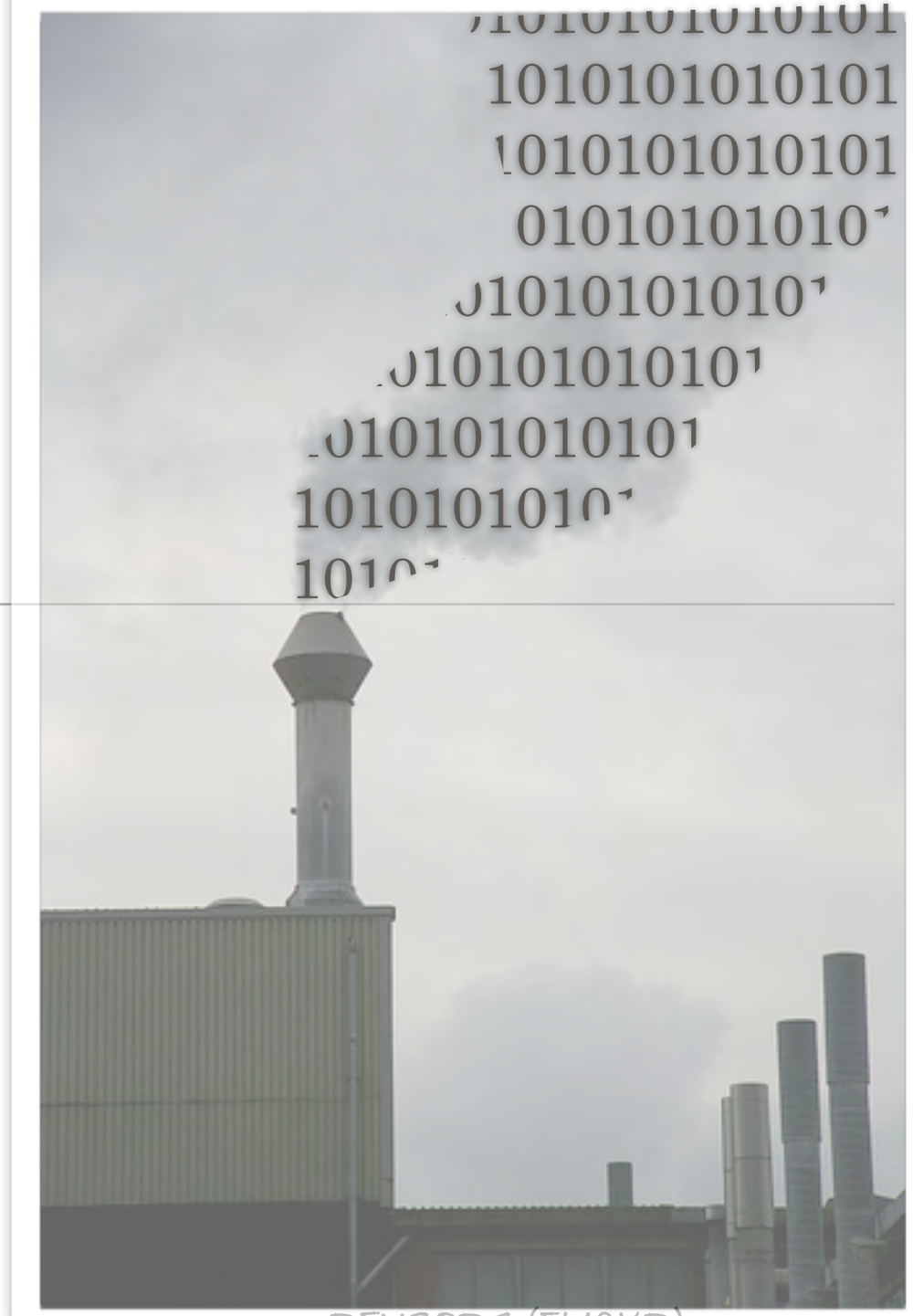


REVSORG (FLICKR)

industrial revolution: data ... and software?

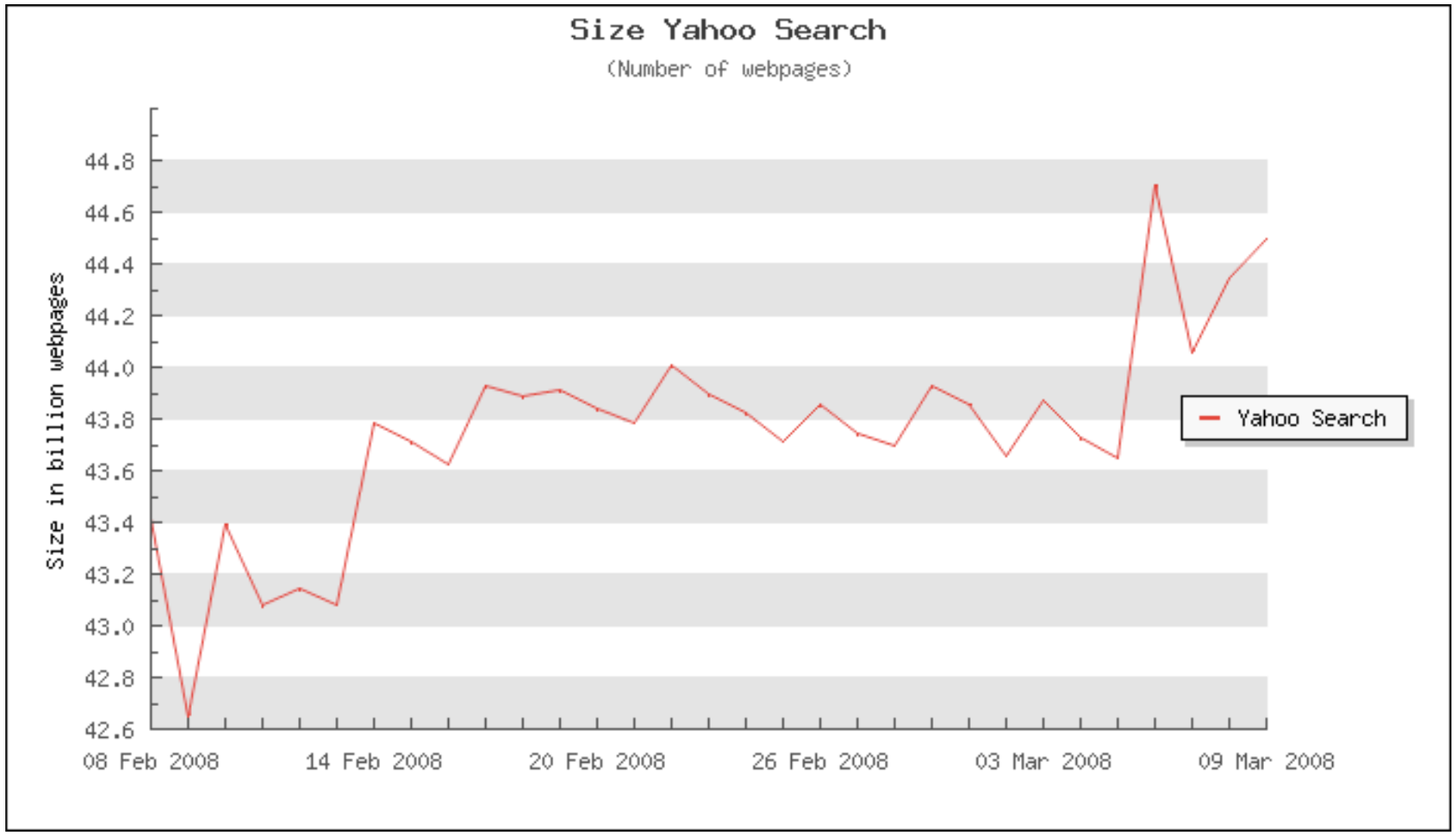
joe hellerstein
uc berkeley

CCC Council Meeting
03.10.2008



REVSORG (FLICKR)

the web is big



HTTP://WWW.WORLDWIDEWEBSIZE.COM

the web is big

HTTP://FLICKR.COM/PHOTOS/MRICON/1836673/

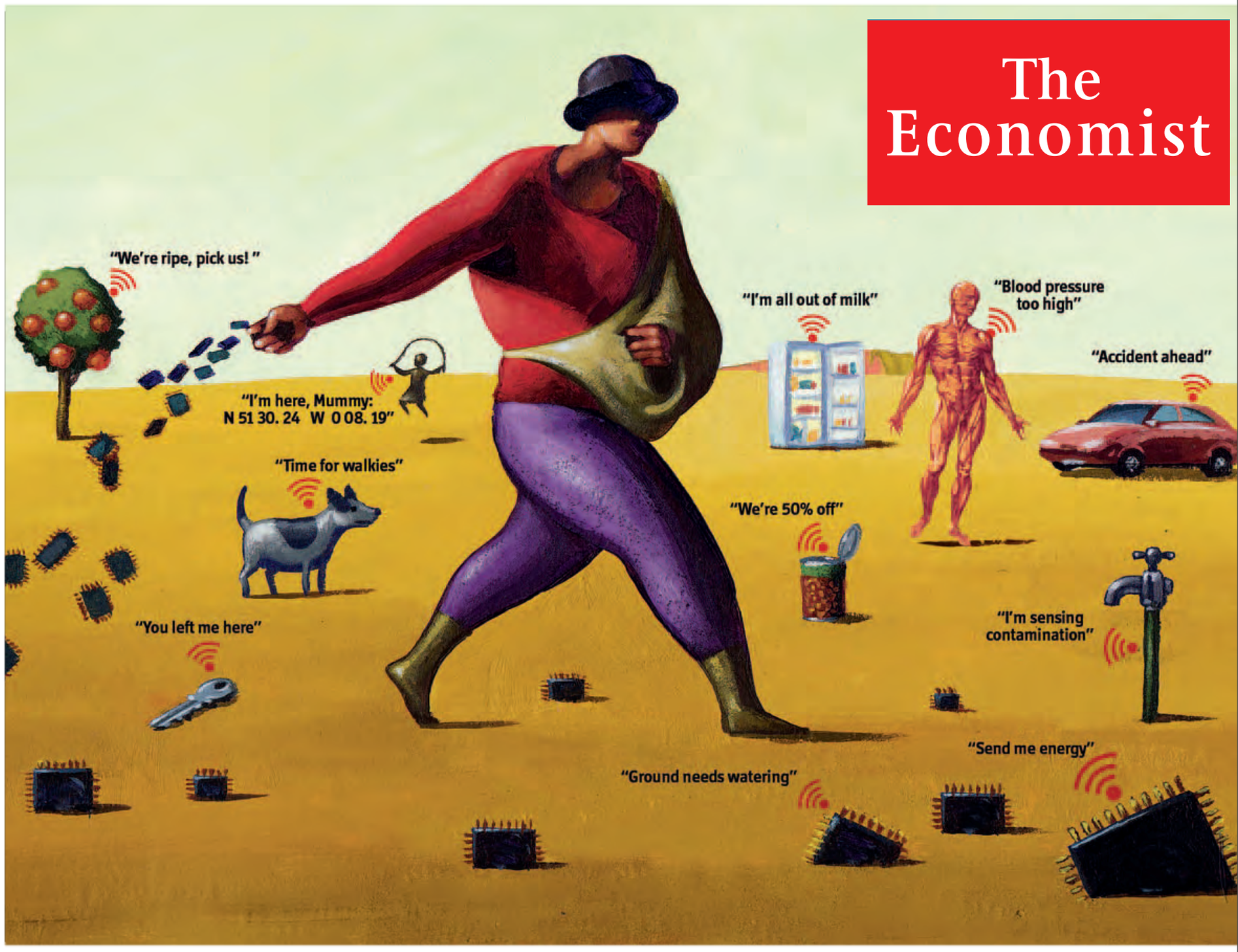
the web is big

HTTP://FLICKR.COM/PHOTOS/MRICON/1836673/

- so was hand-spun cotton

in progress: industrial revolution in data

- production
- transportation
- uniformity



data ... networking ... *uncertainty*

- data != understanding
  uniformity != certainty
    - e.g. sensornets
    - e.g. internet monitoring

- distributed learning/inference:
  graphs upon graphs
    - overlay/ad-hoc networks
    - graphical models

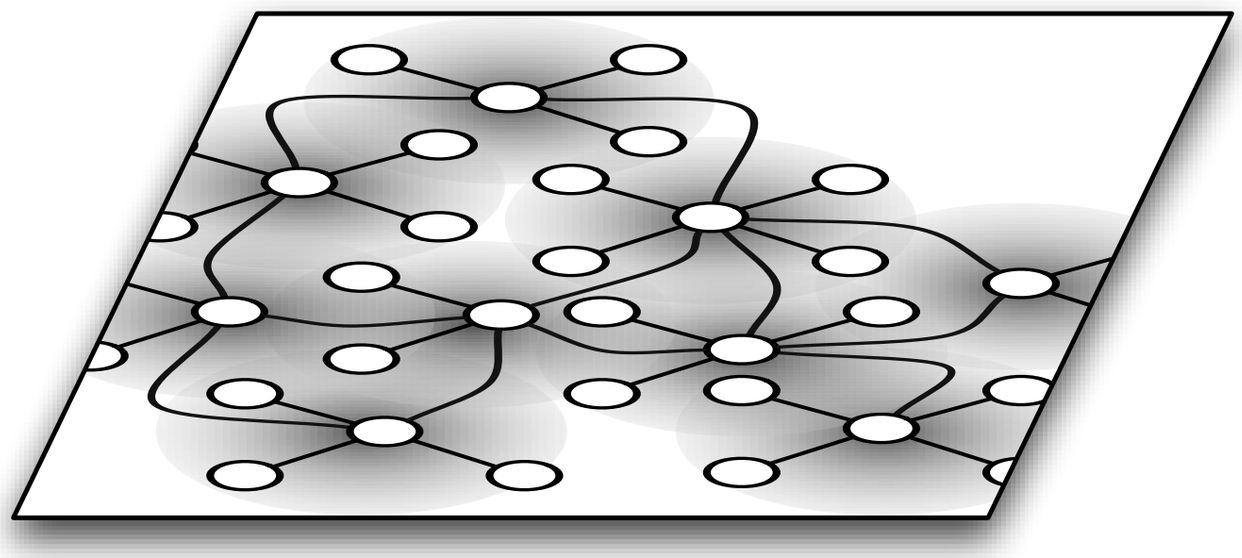
- programming each is hard
    - yet strangely similar

data ... networking ... *uncertainty*

- data != understanding
  uniformity != certainty
  - e.g. sensornets
  - e.g. internet monitoring

- distributed learning/inference:
  graphs upon graphs
  - overlay/ad-hoc networks
  - graphical models

- programming each is hard
  - yet strangely similar

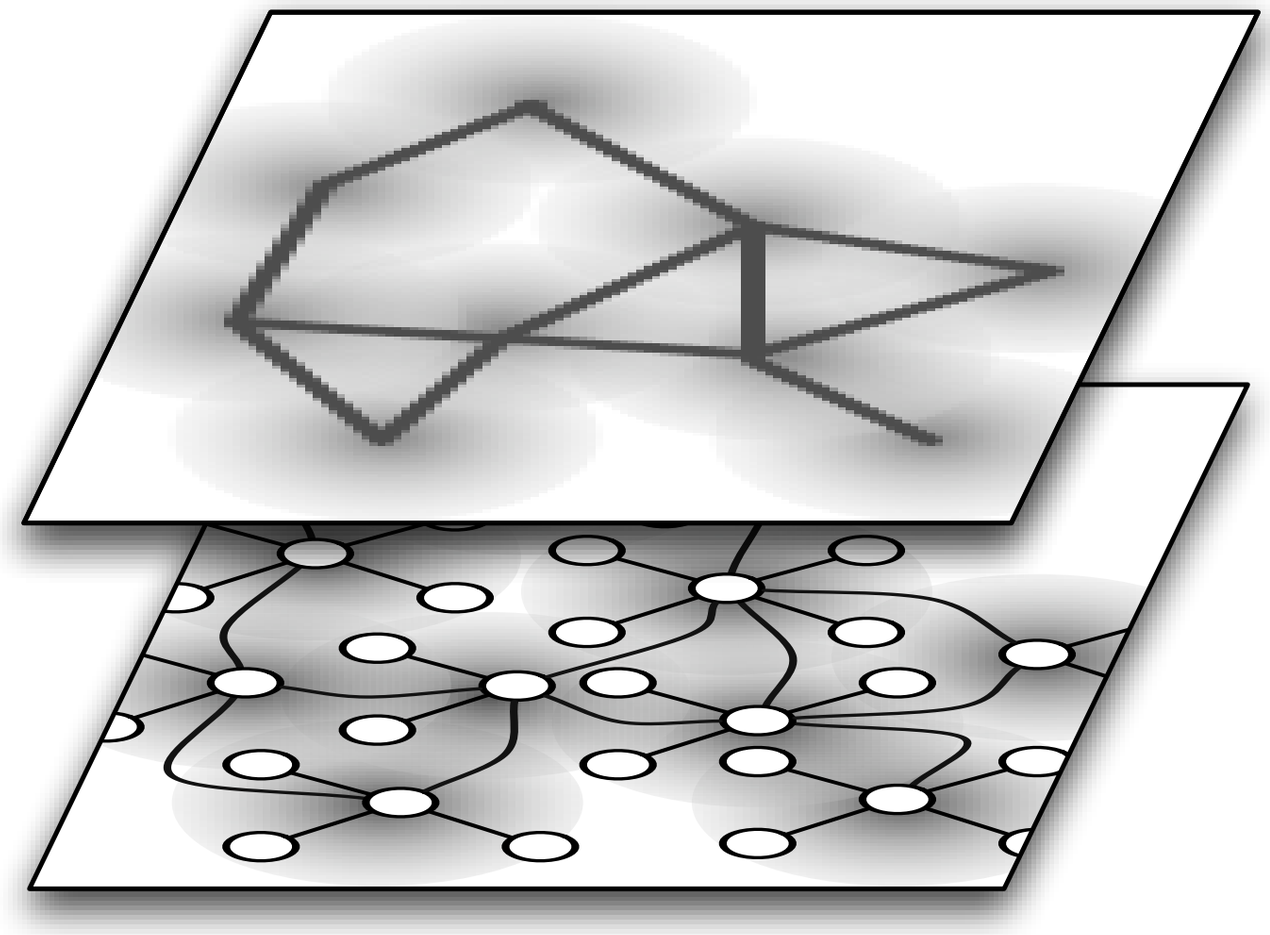


data ... networking ... *uncertainty*

- data != understanding
  uniformity != certainty
    - e.g. sensornets
    - e.g. internet monitoring

- distributed learning/inference:
  graphs upon graphs
    - overlay/ad-hoc networks
    - graphical models

- programming each is hard
    - yet strangely similar

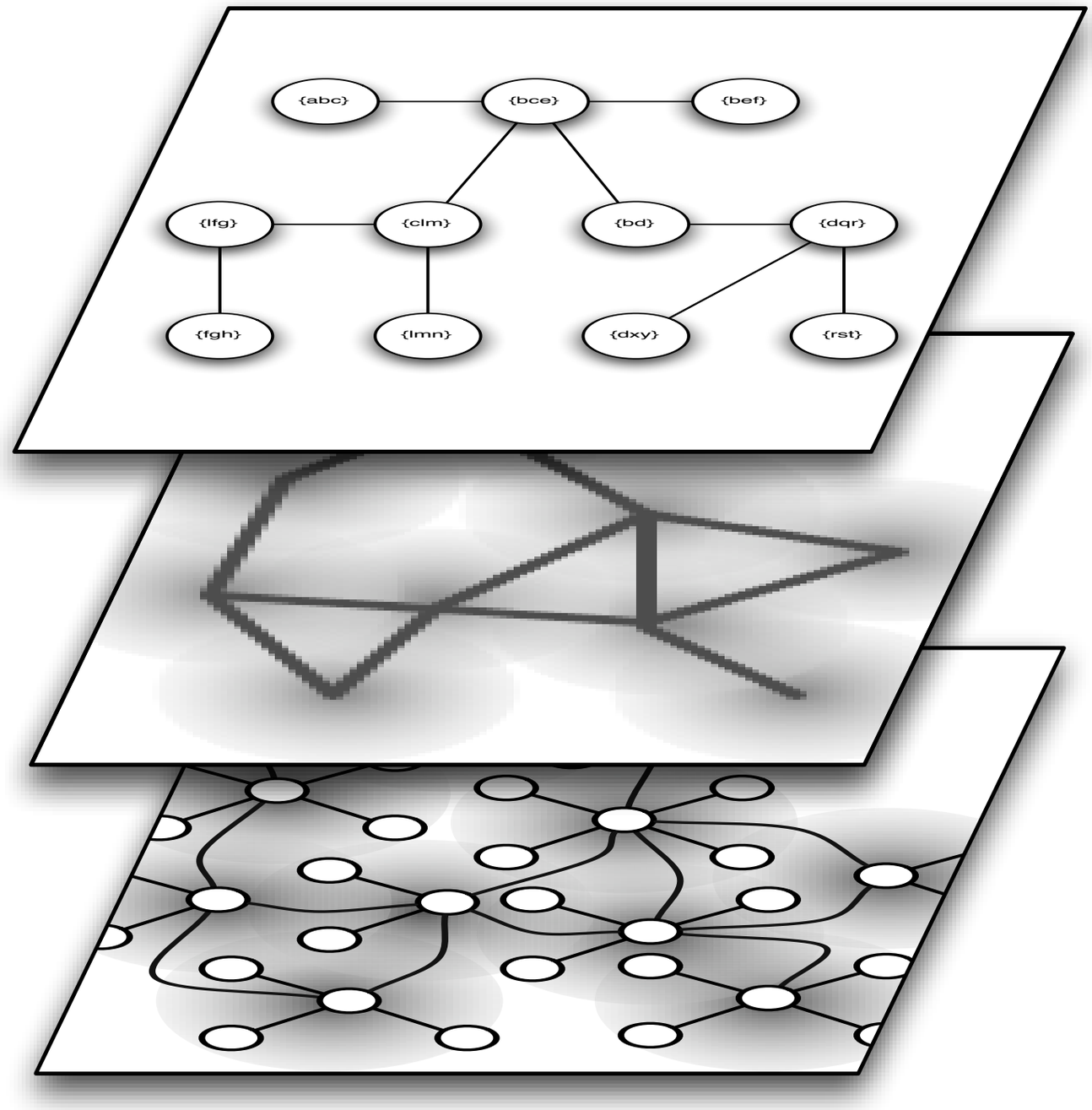


data ... networking ... *uncertainty*

- data != understanding
  uniformity != certainty
  - e.g. sensornets
  - e.g. internet monitoring

- distributed learning/inference:
  graphs upon graphs
  - overlay/ad-hoc networks
  - graphical models

- programming each is hard
  - yet strangely similar



software: last bastion of manual labor?

industrial revolutions in software production?

industrial revolutions in software production?

- 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)

industrial revolutions in software production?

- 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)

- example: declarative networking
  - routing = distributed transitive closure on link tables
  - rendezvous = join of data streams
  - examples: 100× less code, line-for-line from pseudocode

industrial revolutions in software production?

- 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)

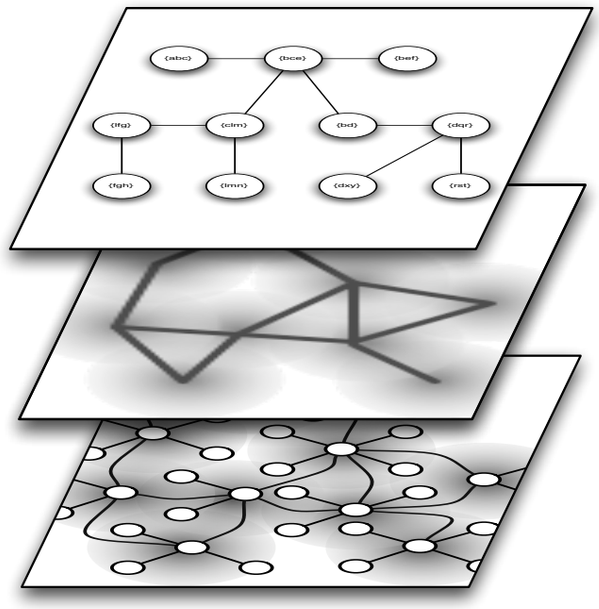
- example: declarative networking
  - routing = distributed transitive closure on link tables
  - rendezvous = join of data streams
  - examples: 100× less code, line-for-line from pseudocode

- looks like we* can do this for machine learning
  - inference in graphical models = transitive closure again
  - dynamic programming = forward-chained deduction

industrial revolutions in software production?

- 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)

- example: declarative networking
  - routing = distributed transitive closure on link tables
  - rendezvous = join of data streams
  - examples: 100× less code, line-for-line from pseudocode



- looks like we* can do this for machine learning
  - inference in graphical models = transitive closure again
  - dynamic programming = forward-chained deduction

- hypothesis: declarative graphs upon graphs, optimization crosses layers

*w/carlos guestrin, stano funiak, ashima atul, kuang chen

related trends/directions

- parallelism
  - embarrassing vs. impossible
  - scientific computing redux? mapreduce? domain-specific languages?
  - declarative maps to dataflow ... auto-parallelization for constrained programs
- data tarpits
  - coming true in sensornets. manycore? disks?
- programming with probability
  - data-centric approaches? IBAL, BLOG, probabilistic DBs.

backup slides follow

P2-CHORD

- chord *distributed hash table*
  - Internet overlay for content-based routing
- high-function implementation
  - multiple successors
  - stabilization
  - optimized finger maintenance
  - failure detection
- 48 rules

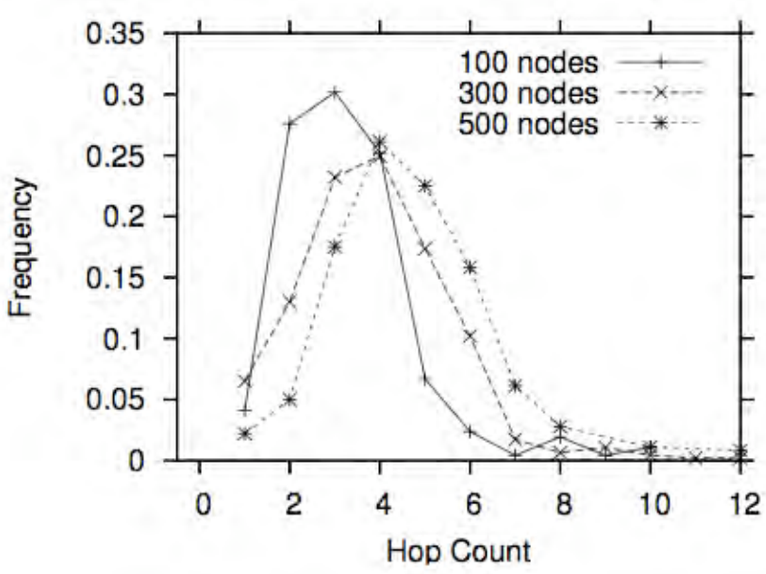
```
/* The base tuples */
materialize(node, infinity, 1, keys(1)).
materialize(finger, 180, 160, keys(2)).
materialize(bestSucc, infinity, 1, keys(1)).
materialize(succDist, 10, 100, keys(2)).
materialize(succ, 10, 100, keys(2)).
materialize(pred, infinity, 100, keys(1)).
materialize(succCount, infinity, 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).
```

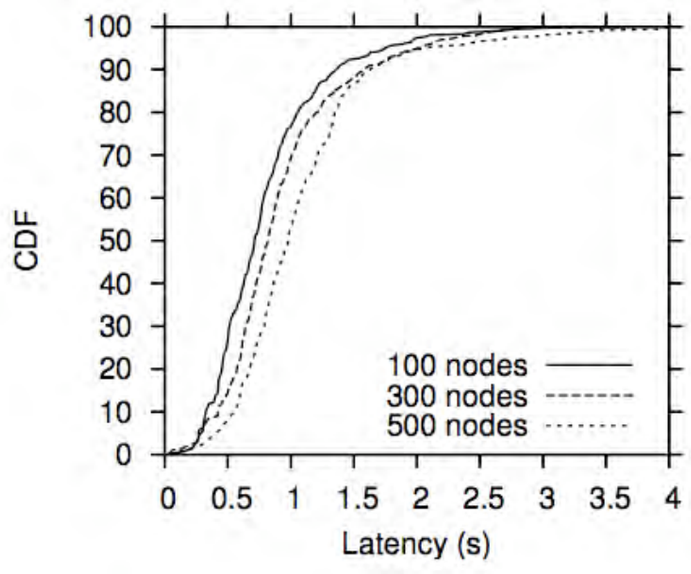
```
/* The base tuples */
materialize(node, infinity, 1, keys(1)).
materialize(finger, 180, 160, keys(2)).
materialize(bestSucc, infinity, 1, keys(1)).
materialize(succDist, 10, 100, keys(2)).
materialize(succ, 10, 100, keys(2)).
materialize(pred, infinity, 100, keys(1)).
materialize(succCount, infinity, 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).
```

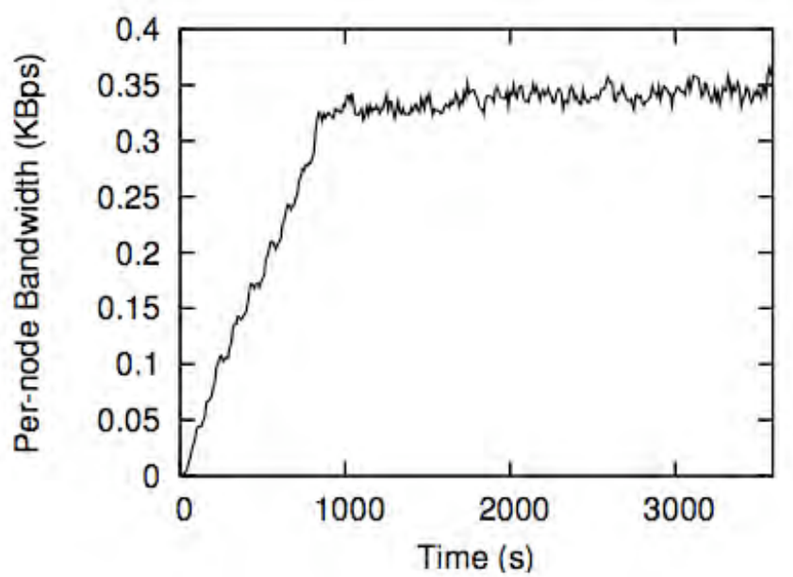
100x LESS CODE THAN MIT CHORD

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

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-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.

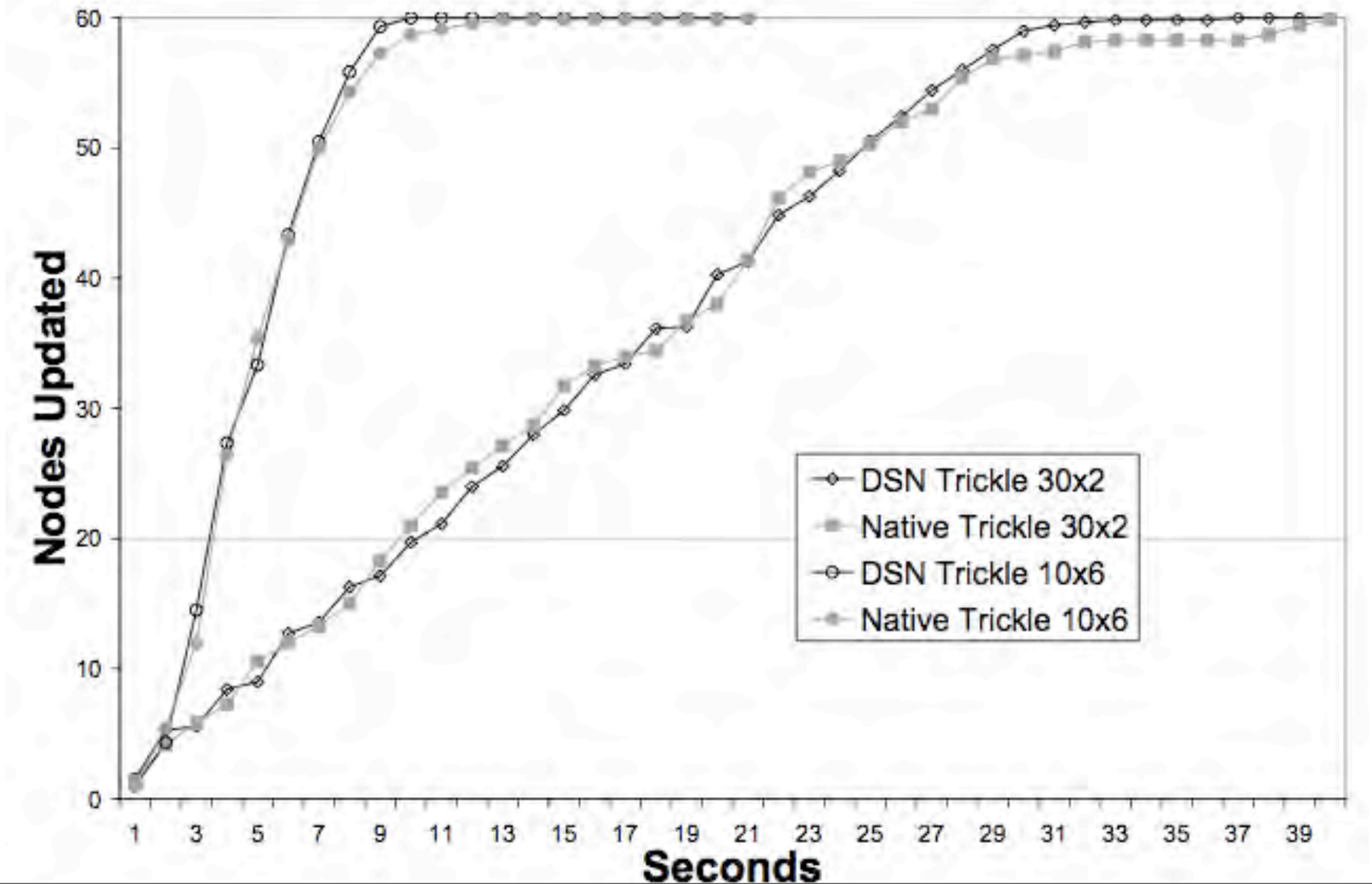
Chu, et al., Sensys 2007

```
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



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