WHAT IS THE FUTURE OF THE INTERNET

JOSEPH M HELLERSTEIN
U C BERKELEY

WHAT IS THE FUTURE OF THE INTERNET.

JOSEPH M HELLERSTEIN
U C BERKELEY

JOINT WORK

- Tyson CONDIE Boon Thau LOO Ion STOICA Berkeley
- Minos GAROFALAKIS David GAY Petros MANIATIS Timothy ROSCOE intel
- Raghu RAMAKRISHNAN Wisconsin
- Atul SINGH Rice
- Peter DRUSCHEL Max Planck

NETWORKS IN FLUX

NETWORKS IN FLUX

- the internet is about to change

NETWORKS IN FLUX

- the internet is about to change
  - revolutionevolution

NETWORKS IN FLUX

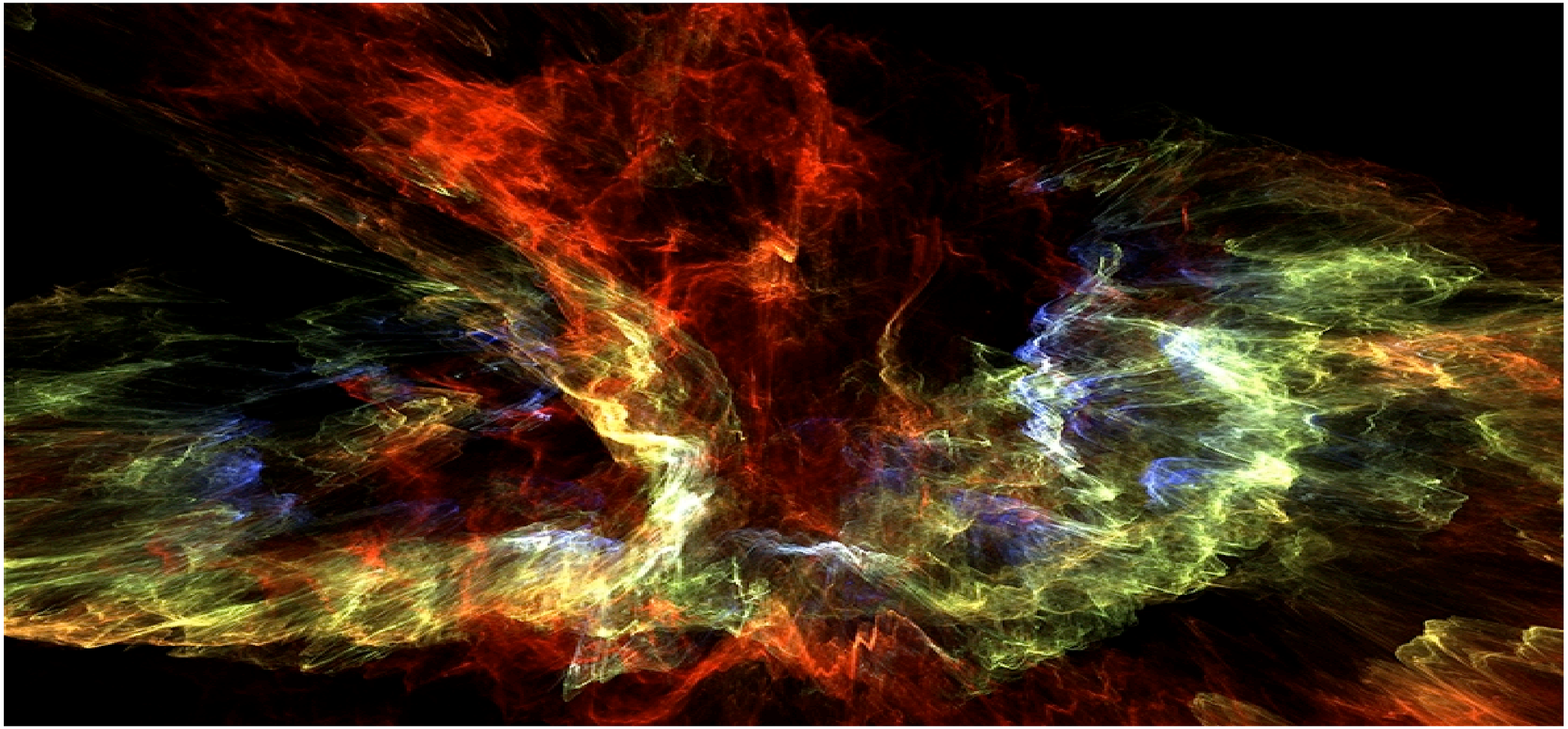
- the internet is about to change
  - revolutionevolution
- sensornets shift the paradigm

NETWORKS IN FLUX

- the internet is about to change
  - revolutionevolution
- sensornets shift the paradigm
  - end: the postal analogy

NETWORKS IN FLUX

- the internet is about to change
  - revolutionevolution
- sensornets shift the paradigm
  - end: the postal analogy
- begin: NETDB



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 IS THE FUTURE: DECLARATIVE NETWORKING

- WHAT: beyond the network *how*
  - topology by specification
  - routing by constraints
  - addressing by content

WHAT IS THE FUTURE: DECLARATIVE NETWORKING

- WHAT: beyond the network *how*
  - topology by specification
  - routing by constraints
  - addressing by content
- and who when where why
  - consensus, snapshots, forensics
  - innate search, query, inference

THE EVOLUTION OF WHAT

- netDB roots
  - query (in) the network
- the deep dive
  - networking VIA queries

- **WATCH THE SYNTHESIS**

TODAY

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

TODAY

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

WHY WHAT?

WHY WHAT?

- ease, insight:
  - rapid prototyping
  - customizability
  - uplevel ideas and their synergies

WHY WHAT?

- ease, insight:
  - rapid prototyping
  - customizability
  - uplevel ideas and their synergies
- towards safety
  - static checks
  - (synthesized) runtime checks

HELLERSTEIN'S INEQUALITY

$$\frac{dapp}{dt} << \frac{denv}{dt}$$

HELLERSTEIN'S INEQUALITY

$$\frac{dapp}{dt} << \frac{denv}{dt}$$

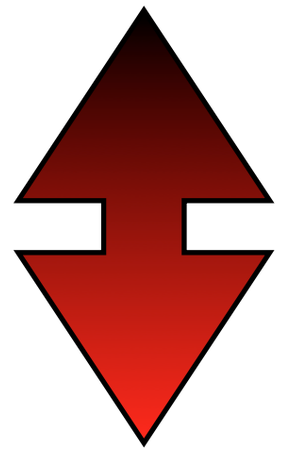
DATA INDEPENDENCE

THAT'S WHAT.

DECLARATIVE NETWORK ARCHITECTURES

- logical blueprint

WHAT



- structure assembly

HOW

  - and mutation!

WHAT NOW

- textbook routing protocols
  - internet-style and wireless (3-8 lines, UCB/Wisconsin)
- routing by content
  - chord dht (47 lines, UCB/Intel)
- debugging
  - distributed watchpoints
  - chandy-lamport snapshots (20 lines, Intel/Rice/MPI)
- paxos distributed consensus protocol
  - (44 lines, Harvard)

WHAT NOW

- textbook routing protocols
  - internet-style and wireless (3-8 lines, UCB/Wisconsin)
- routing by content
  - chord dht (47 lines, UCB/Intel)
- debugging
  - distributed watchpoints
  - chandy-lamport snapshots (20 lines, Intel/Rice/MPI)
- paxos distributed consensus protocol
  - (44 lines, Harvard)

WHAT'S NEXT

- distributed machine learning
  - distributed junction trees (17 lines so far!)
- workable protocol implementations
  - ospf
  - bgp?
- synthesis of multiple layers
  - public health for the Internet

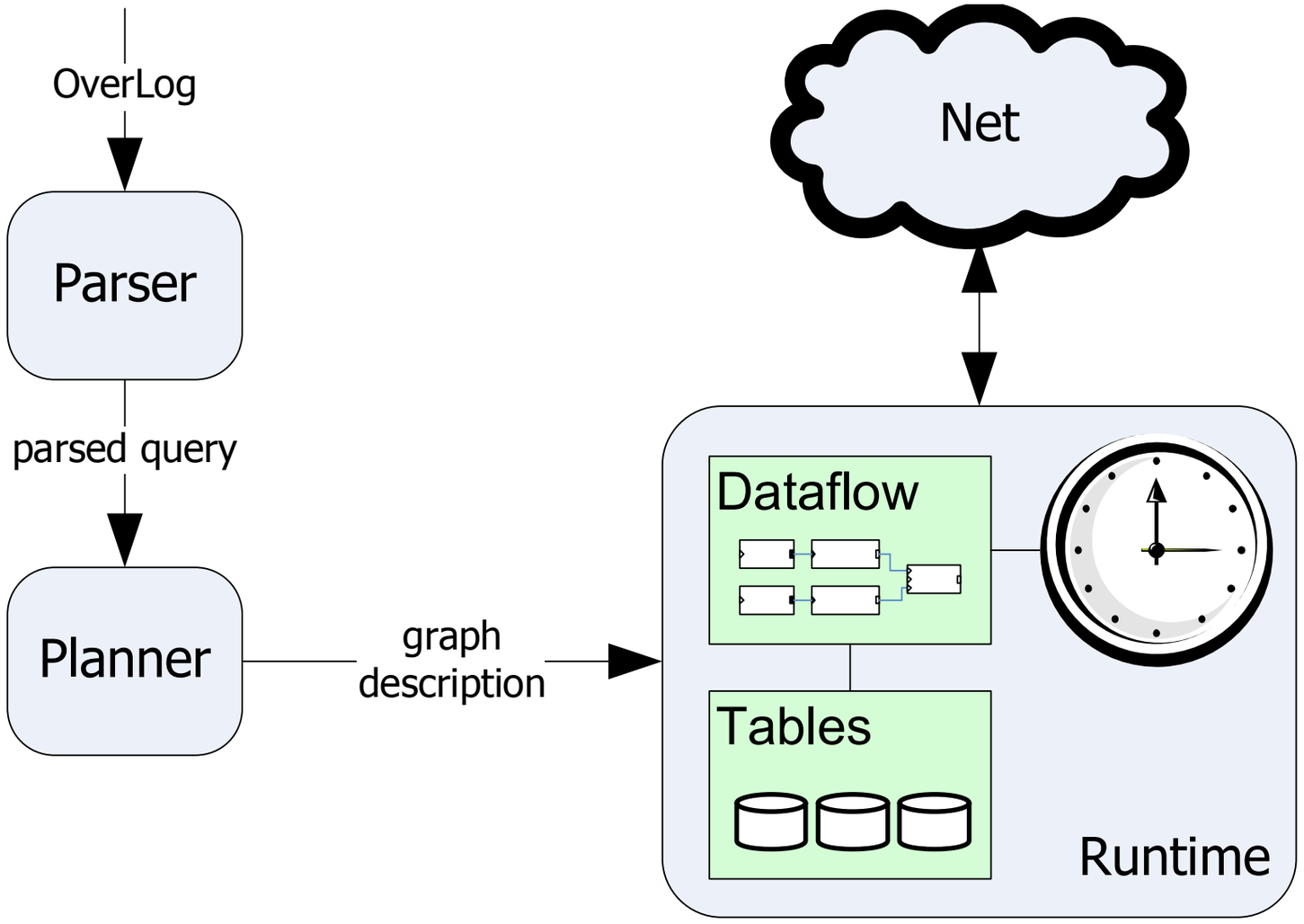
TODAY

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

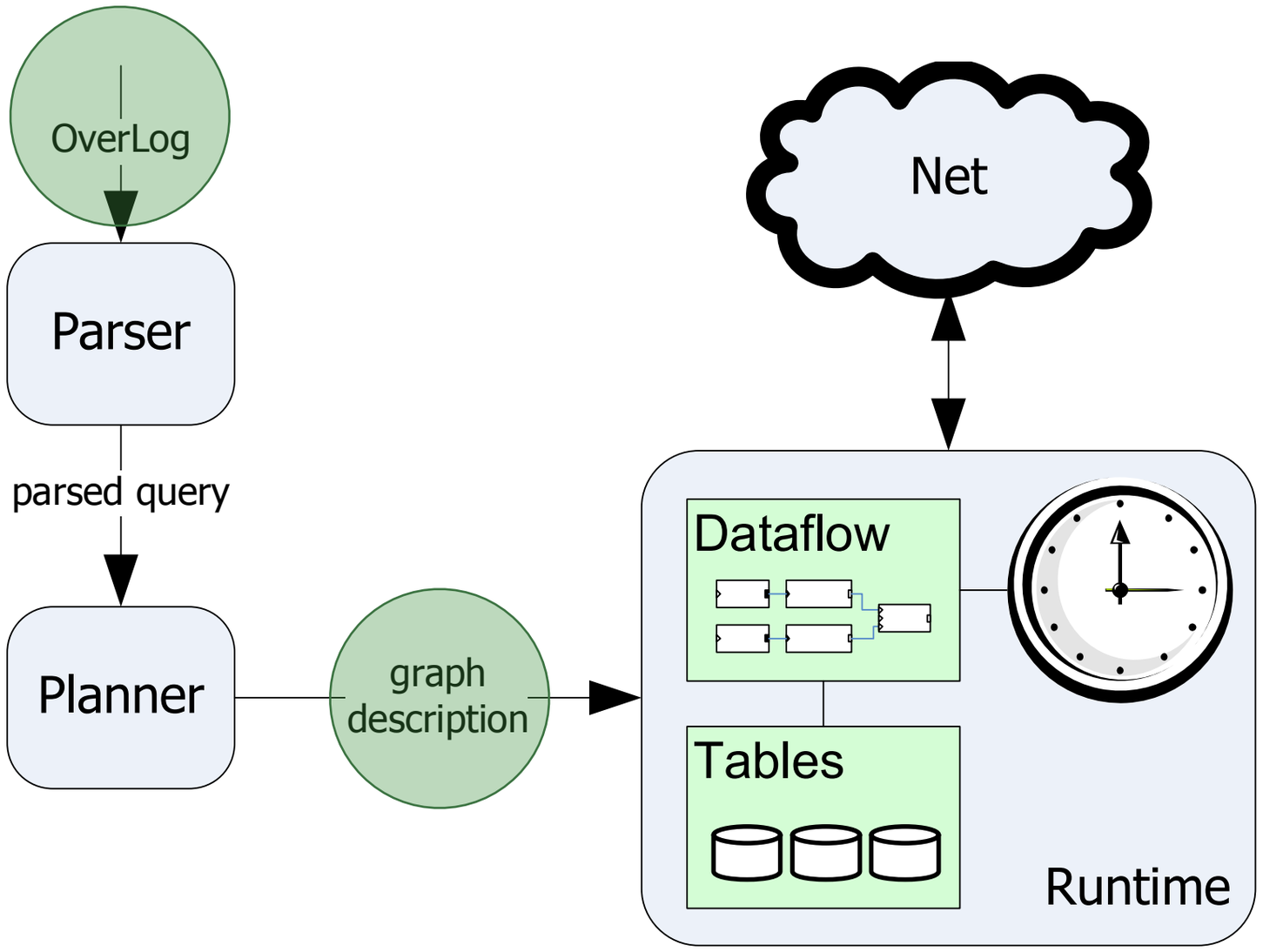
TODAY

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

P2 @ 10,000 FEET

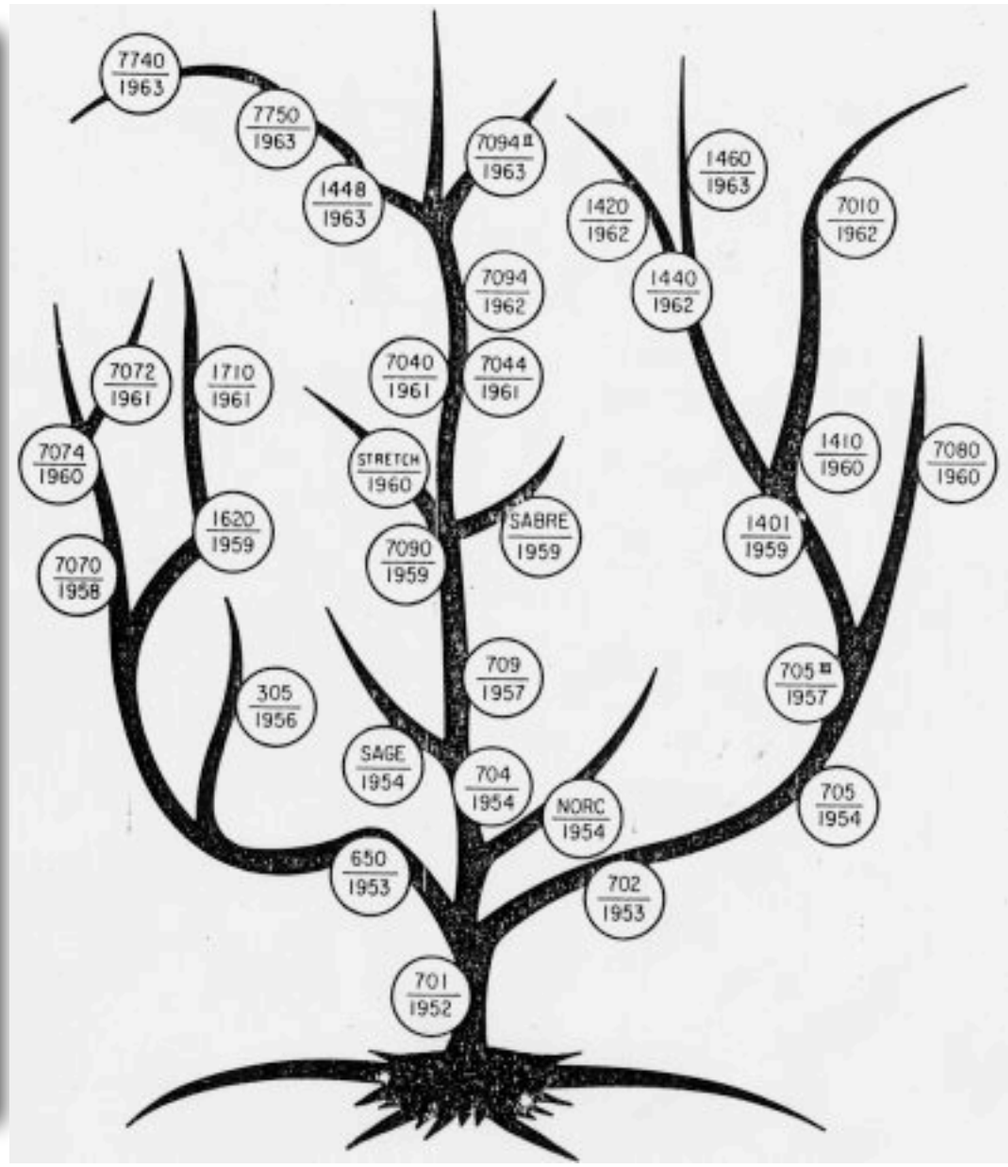


P2 @ 10,000 FEET



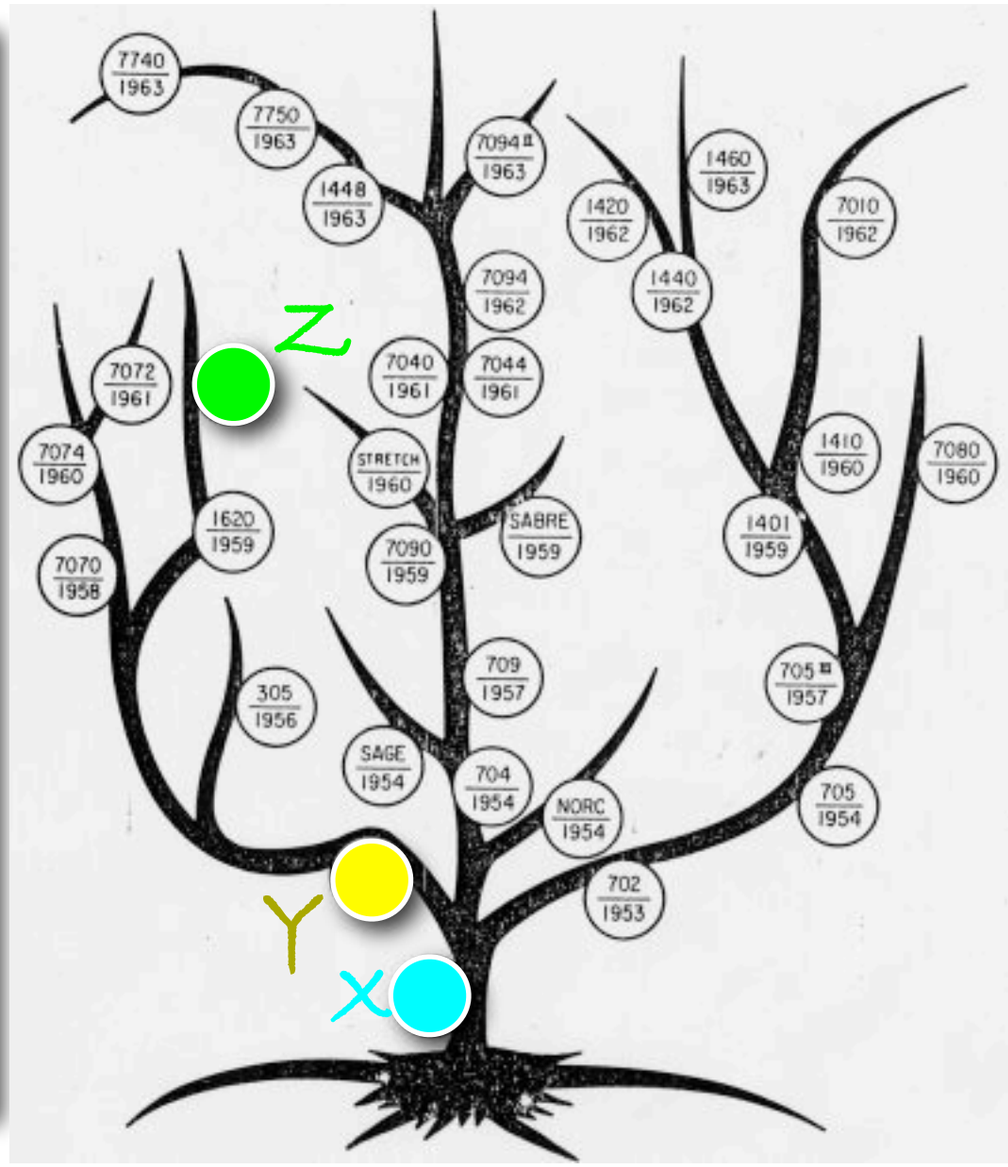
DUSTY OLD DATALOG

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



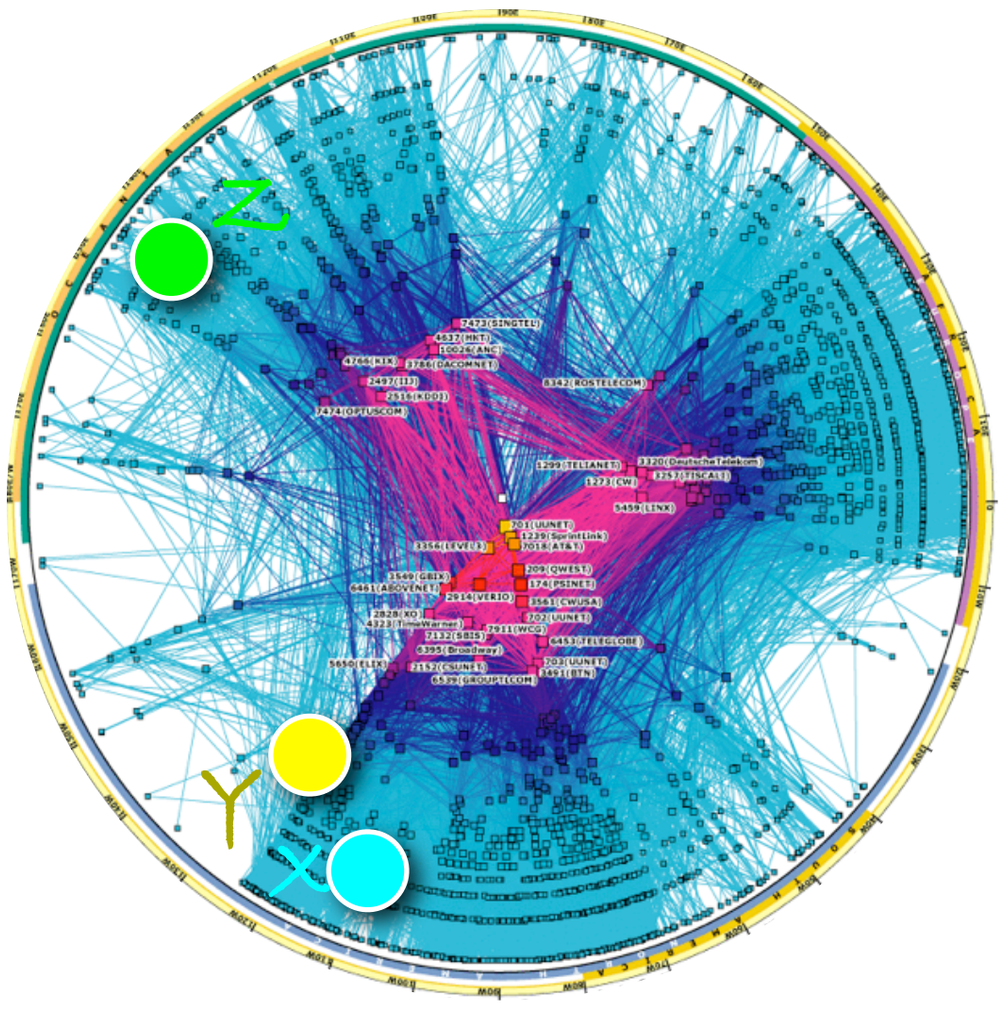
DUSTY OLD DATALOG

- 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) ← COST
- 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)

COST

- path(X,Y,Y,C) :- link(X,Y,C)

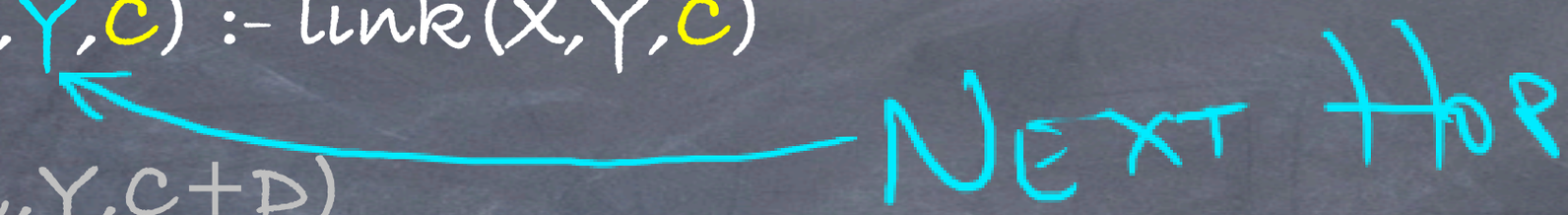
- path(X,Z,Y,C+D)
  :- link(X,Y,C), path(Y,Z,N,D)

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)

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)

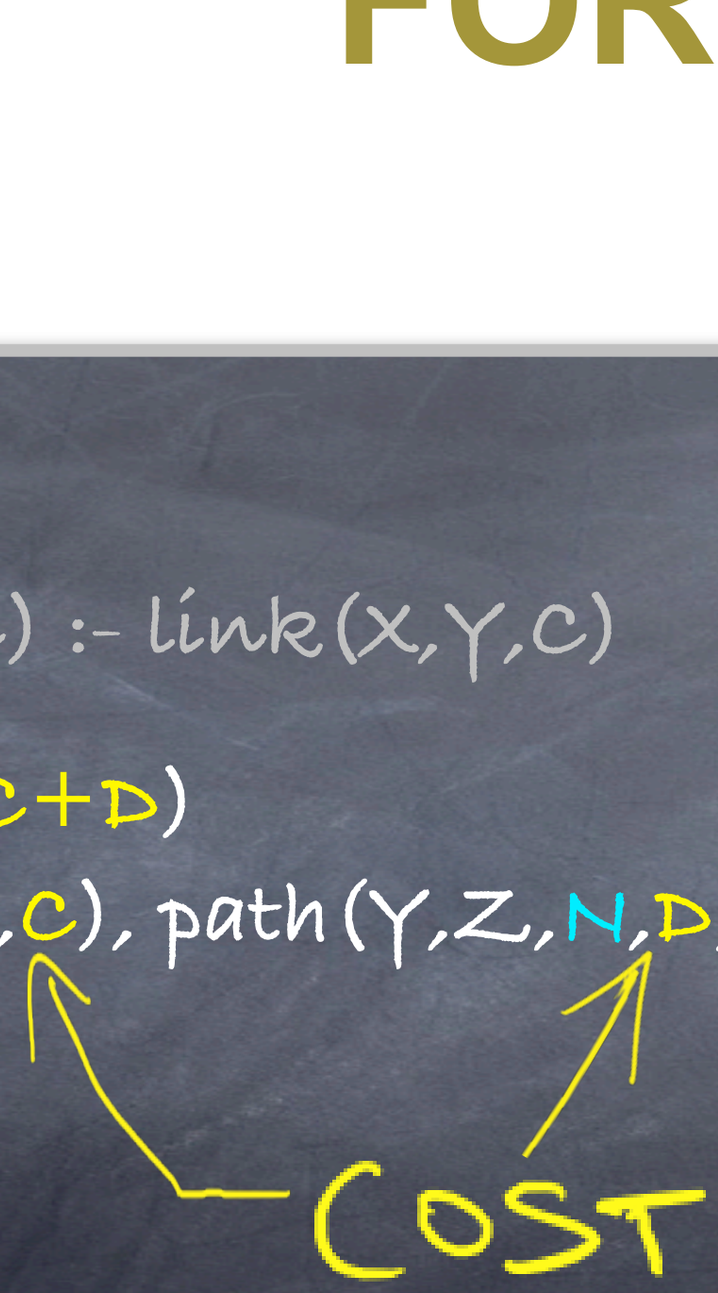


FORMING PATHS

- 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)

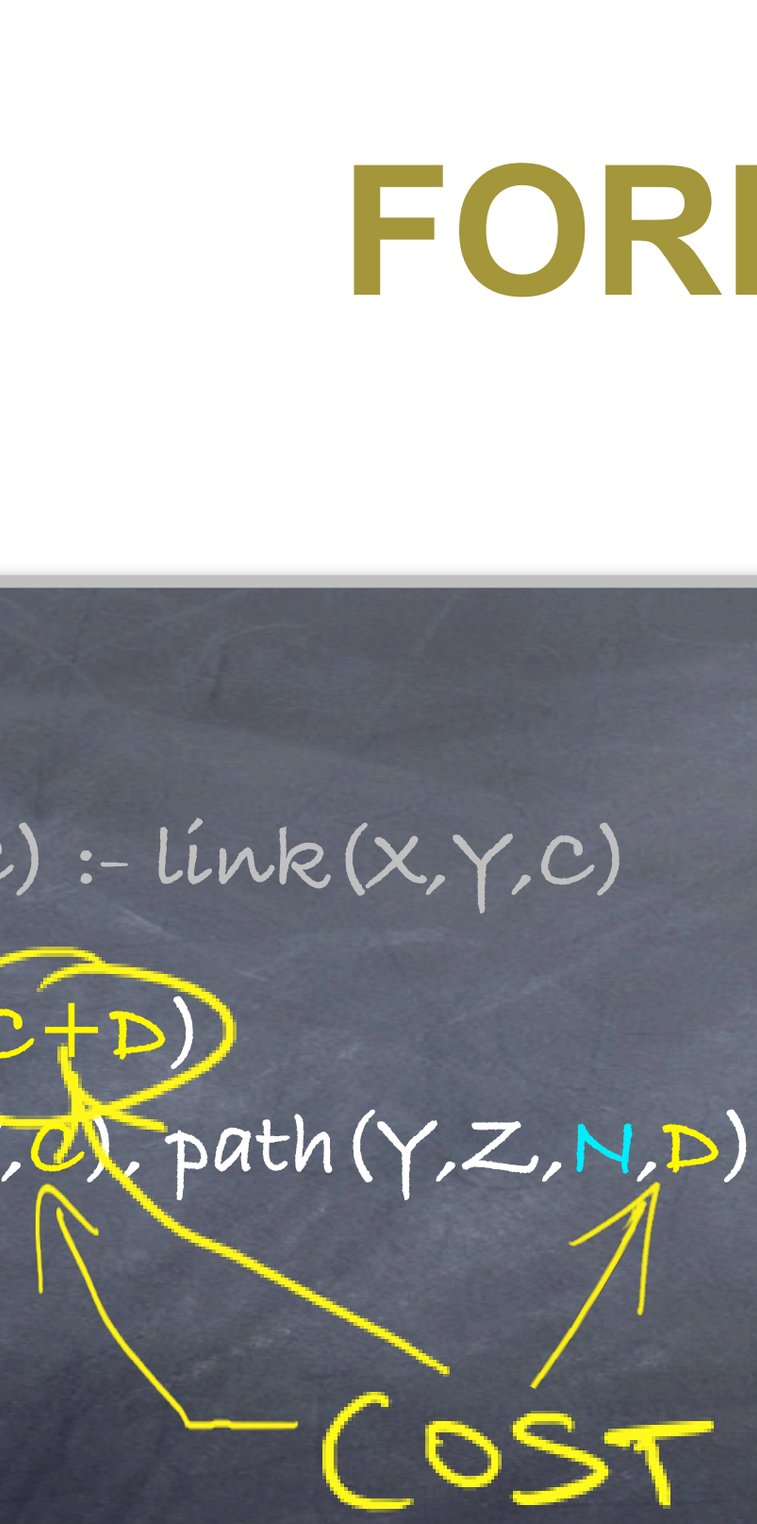
FORMING PATHS

- 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)



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)

NEXT HOP

BEST PATHS

BEST PATHS

- link(X,Y)

BEST PATHS

- link(X,Y)
- path(X,Y,Y,C) :- link(X,Y,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)

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)

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)

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
  - could run in a *routing service*
- but can this lead to protocols?

TOWARD DISTRIBUTION: DATA PARTITIONING

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

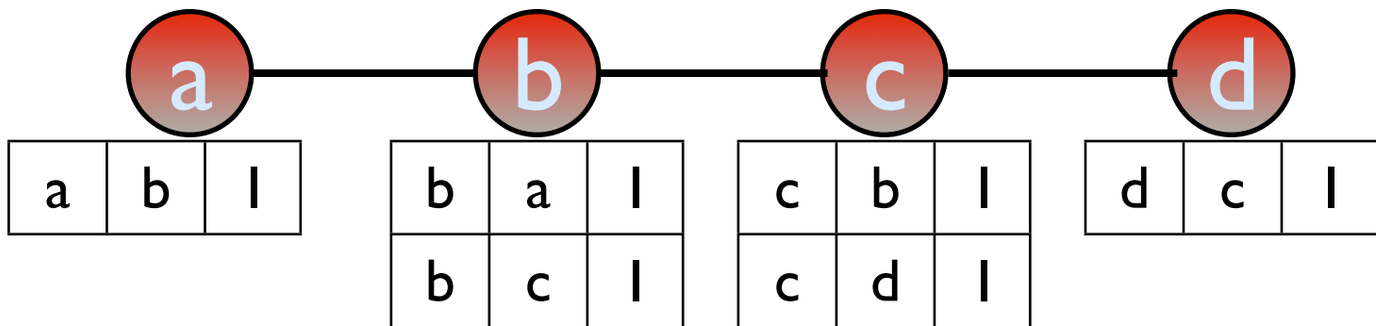
PARTITION SPECS INDUCE COMMUNICATION

- 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

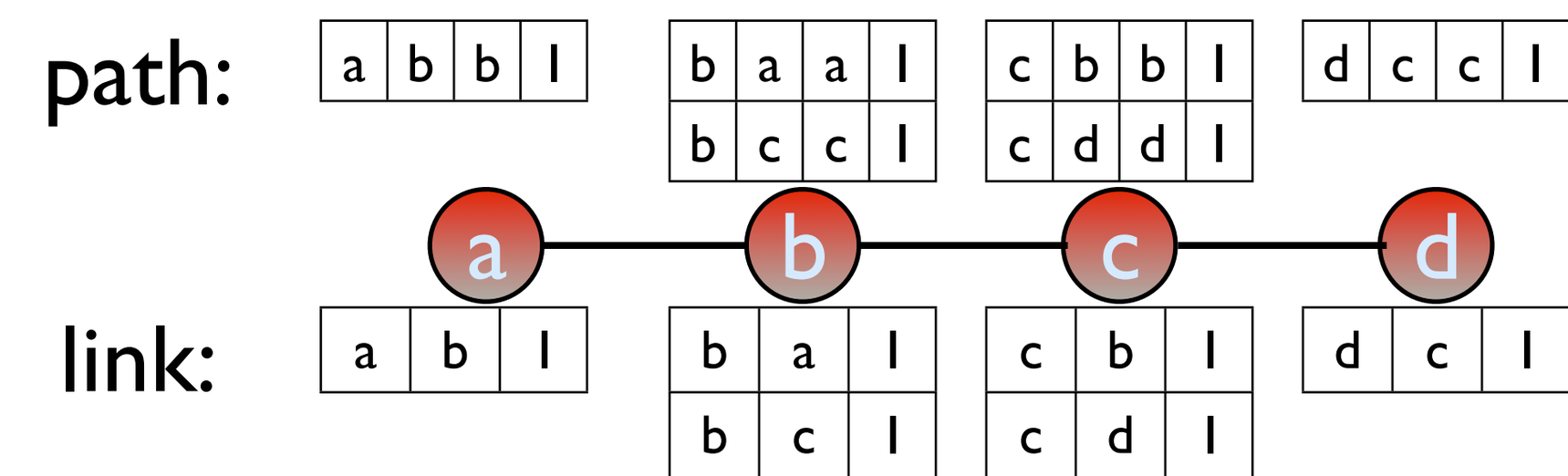
- link(@X,Y,C)
- path(@X,Y,Y,C) :- link(@X,Y,C)
- path(@X,Z,Y,C+D) :- link(@X,Y,C), path(@Y,Z,N,D)

link:



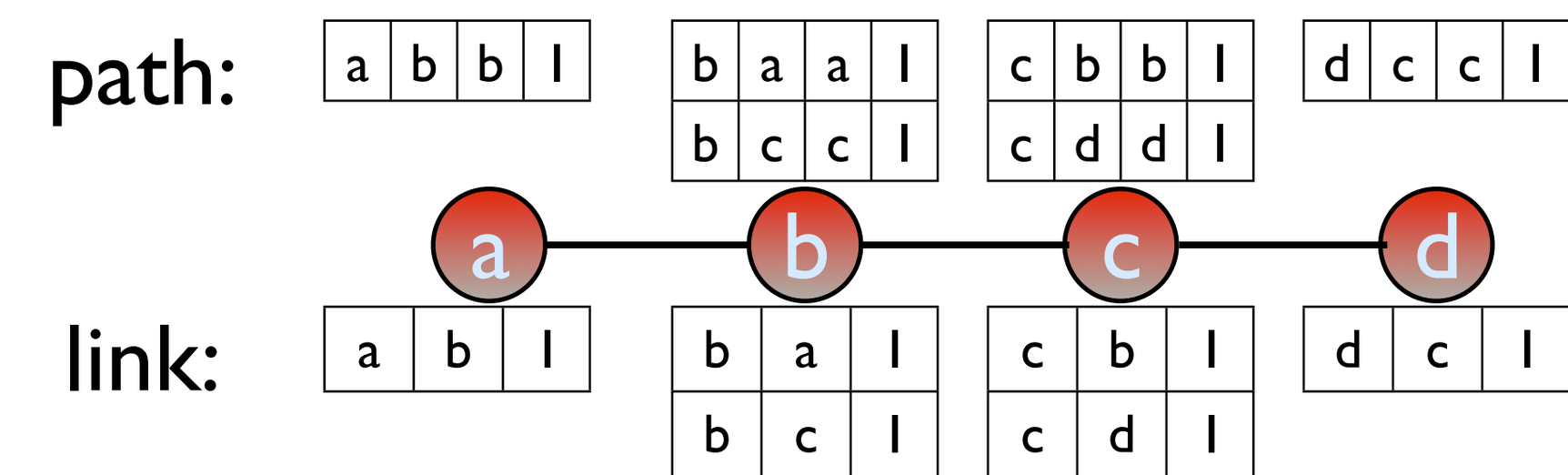
PARTITION SPECS INDUCE COMMUNICATION

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



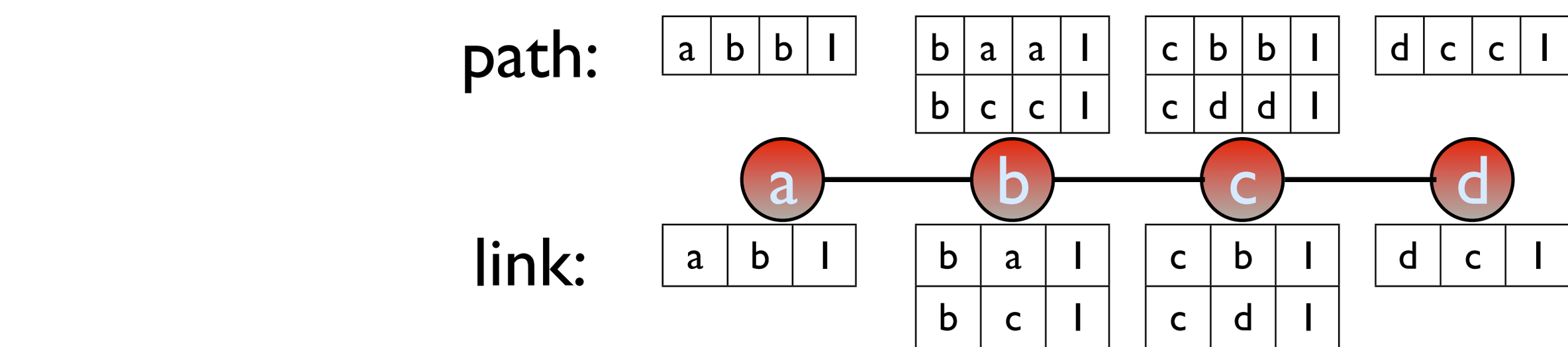
PARTITION SPECS INDUCE COMMUNICATION

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



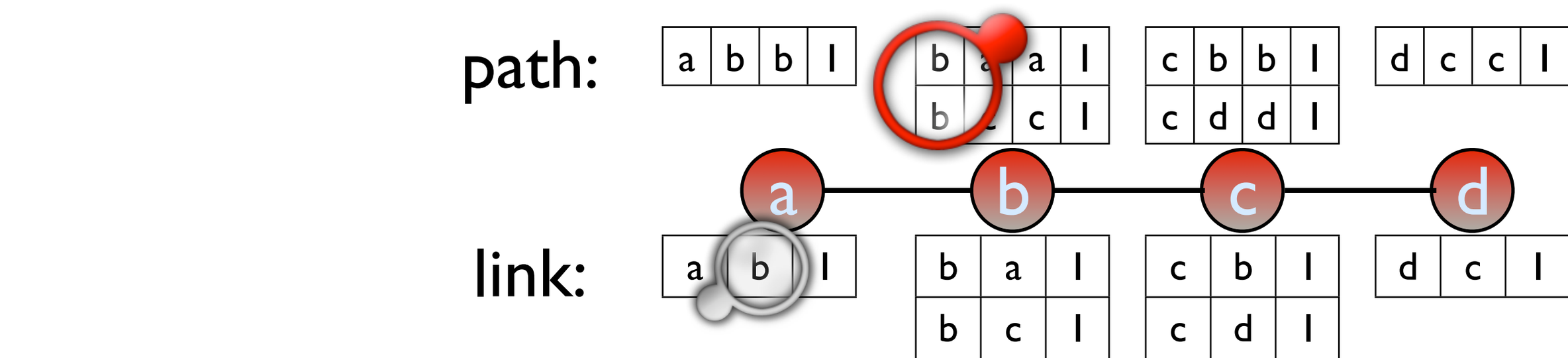
PARTITION SPECS INDUCE COMMUNICATION

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



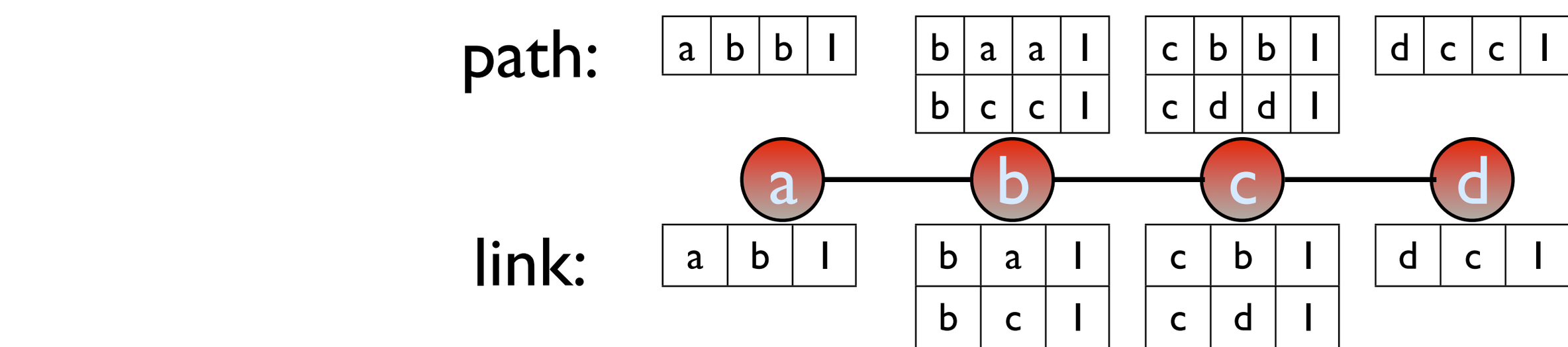
PARTITION SPECS INDUCE COMMUNICATION

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



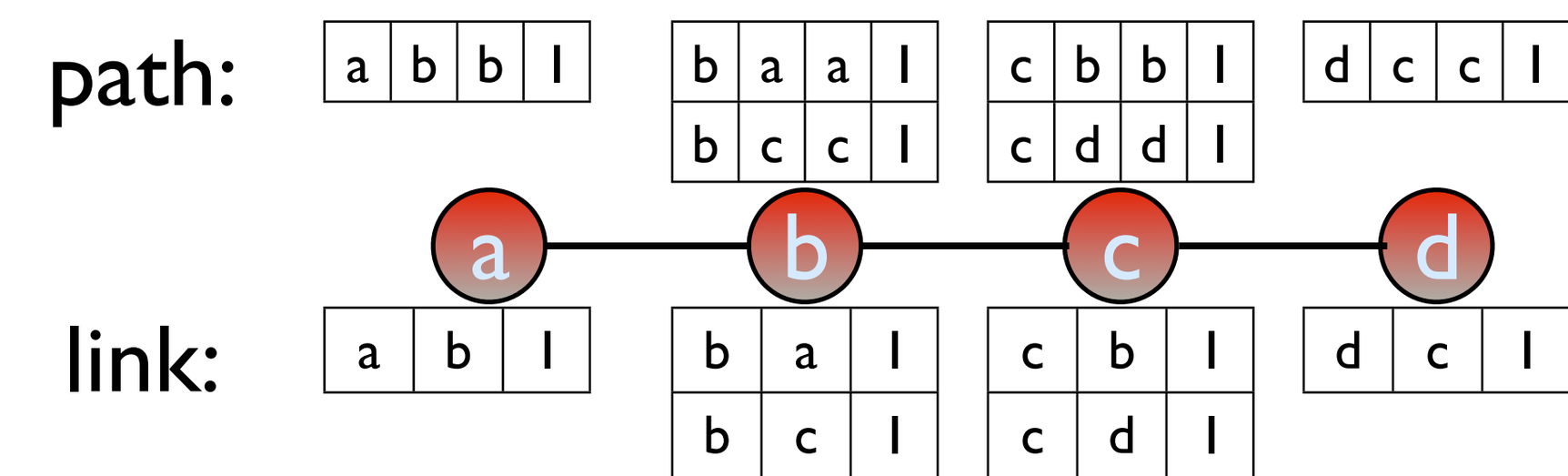
PARTITION SPECS INDUCE COMMUNICATION

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



PARTITION SPECS INDUCE COMMUNICATION

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



PARTITION SPECS INDUCE COMMUNICATION

- link(@X,Y)

Localization Rewrite

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

link_d:

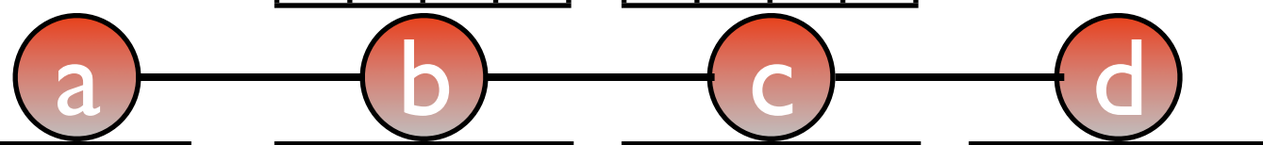
path:

a	b	b	l

b	a	a	l
b	c	c	l

c	d	d	l
d	c	c	l

d	c	c	l



link:

a	b	l

b	a	l
b	c	l

c	d	l
c	d	l

d	c	l

PARTITION SPECS INDUCE COMMUNICATION

- 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)

Localization Rewrite

link_d:

path:

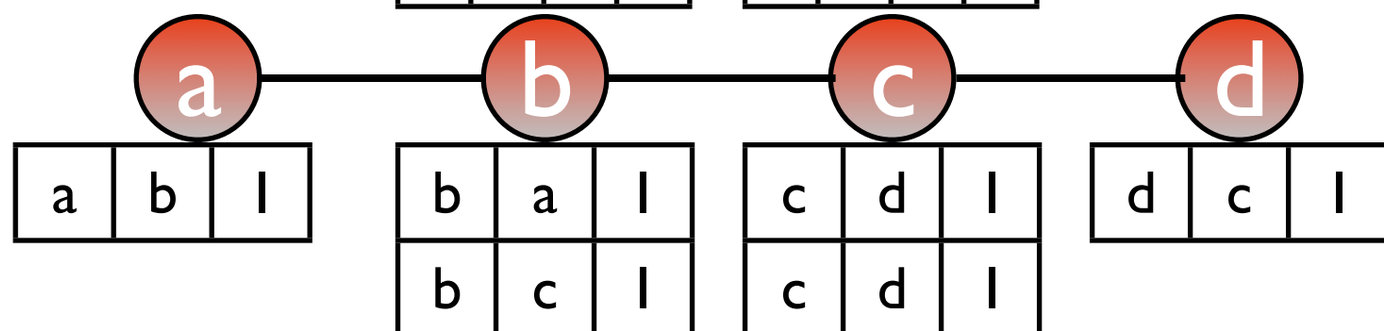
a	b	b	l

b	a	a	l
b	c	c	l

c	d	d	l
d	c	c	l

d	c	c	l

link:



PARTITION SPECS INDUCE COMMUNICATION

- link(@X,Y)

Localization Rewrite

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

link_d:

a	b	l

path:

a	b	b	l

b	a	a	l
b	c	c	l

c	d	d	l
d	c	c	l

d	c	c	l

link:

a b c d

a	b	l

b	a	l
b	c	l

c	d	l
c	d	l

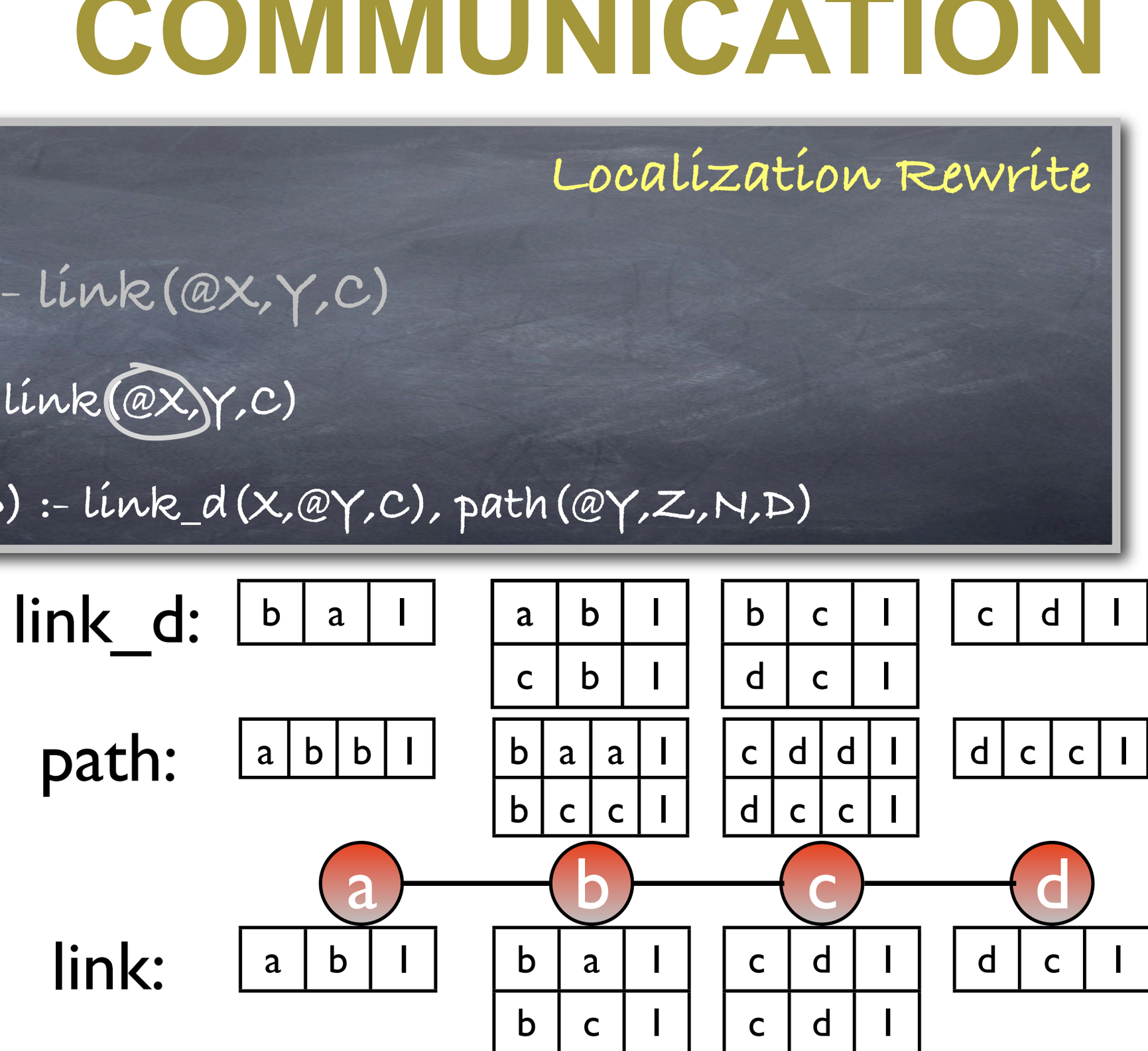
d	c	l

PARTITION SPECS INDUCE COMMUNICATION

- link(@X,Y)

Localization Rewrite

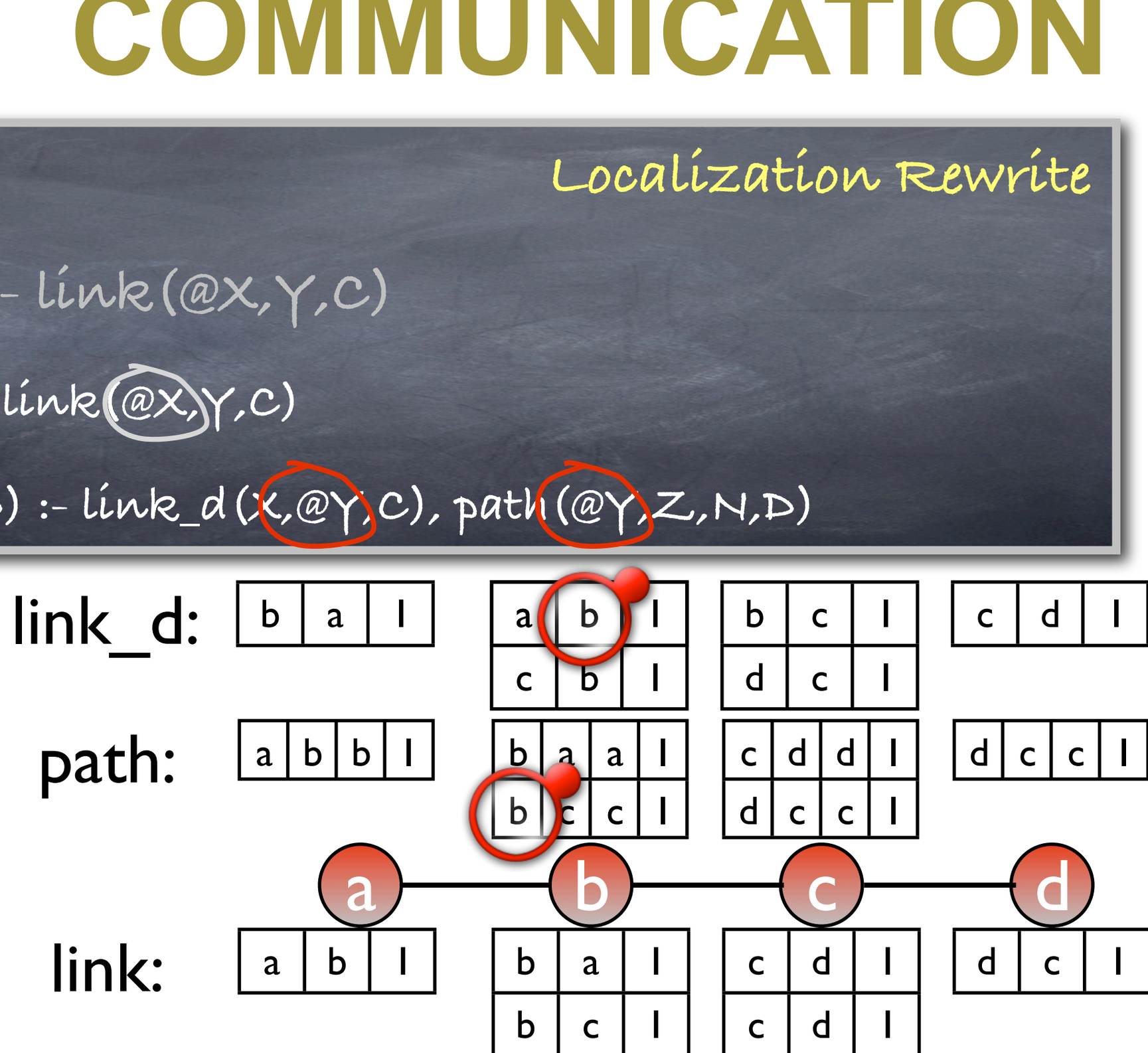
- path(@X,Y,Y,C) :- link(@X,Y,C)
- link_d(X,@Y,C) :- link(@X,Y,C)
- path(@X,Z,Y,C+D) :- link_d(X,@Y,C), path(@Y,Z,N,D)



PARTITION SPECS INDUCE COMMUNICATION

Localization Rewrite

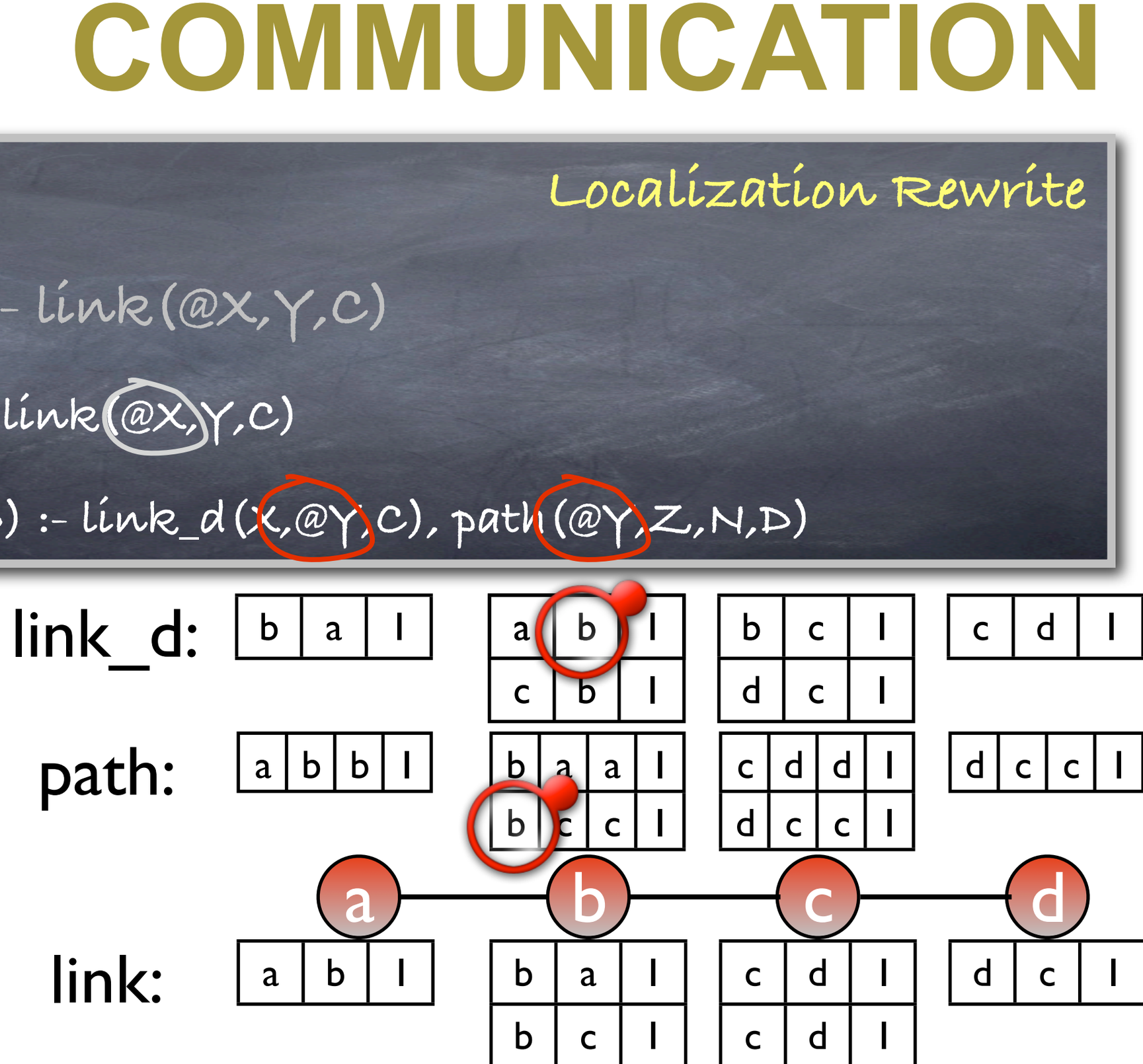
- link(@X,Y)
- path(@X,Y,Y,C) :- link(@X,Y,C)
- link_d(X,@Y,C) :- link(@X,Y,C)
- path(@X,Z,Y,C+D) :- link_d(X,@Y,C), path(@Y,Z,N,D)



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)



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)

link_d:

b	a	1

a	b	1
c	b	1

b	c	1
d	c	1

c	d	1

path:

a	b	b	1
a	c	b	2

b	a	a	1
b	c	c	1

c	d	d	1
d	c	c	1

d	c	c	1

link:

a	b	1

b	a	1
b	c	1

c	d	1
c	d	1

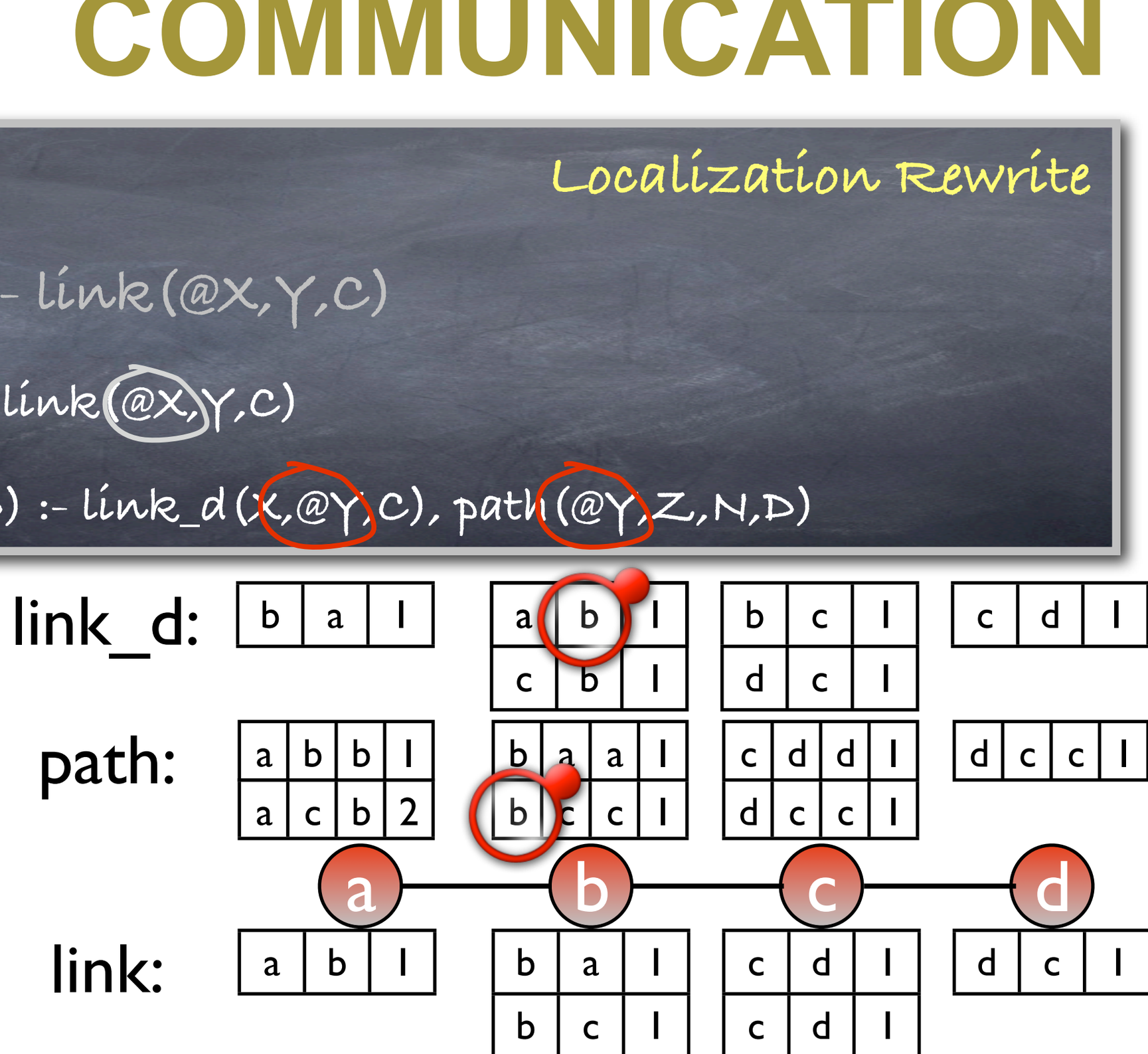
d	c	1

PARTITION SPECS INDUCE COMMUNICATION

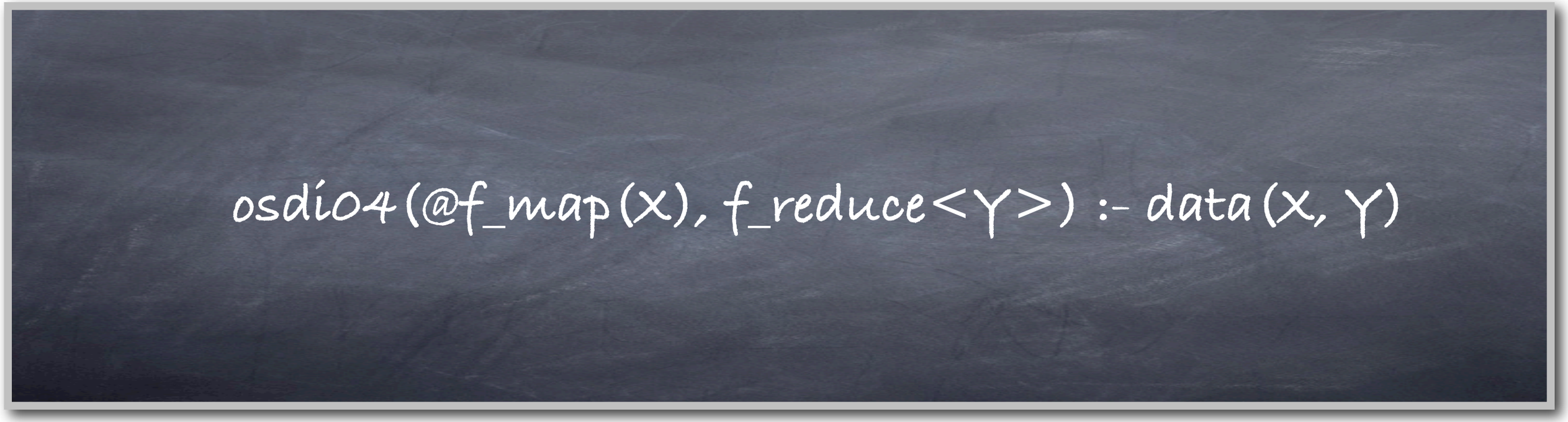
Localization Rewrite

- 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



OH BY THE WAY



THE POWER OF MINOR VARIATIONS

- Best-Path Routing: MAX/MIN, MAX/EXPECT, etc.
- Distance Vector
- Dynamic Source Routing
- Policy: Peering, VPN subsetting
- QoS-based Routing
- Link-state
- Multicast Overlays (Single-Source & CBT)

THE POWER OF MINOR VARIATIONS

- Best-Path Routing: MAX/MIN, MAX/EXPECT, etc.
- Distance Vector
- Dynamic Source Routing
- Policy: Peering, VPN subsetting
- QoS-based Routing
- Link-state
- Multicast Overlays (Single-Source & CBT)

REMEMBER, HUMANS AND CHIMPS SHARE 95% OF THEIR DNA

TODAY

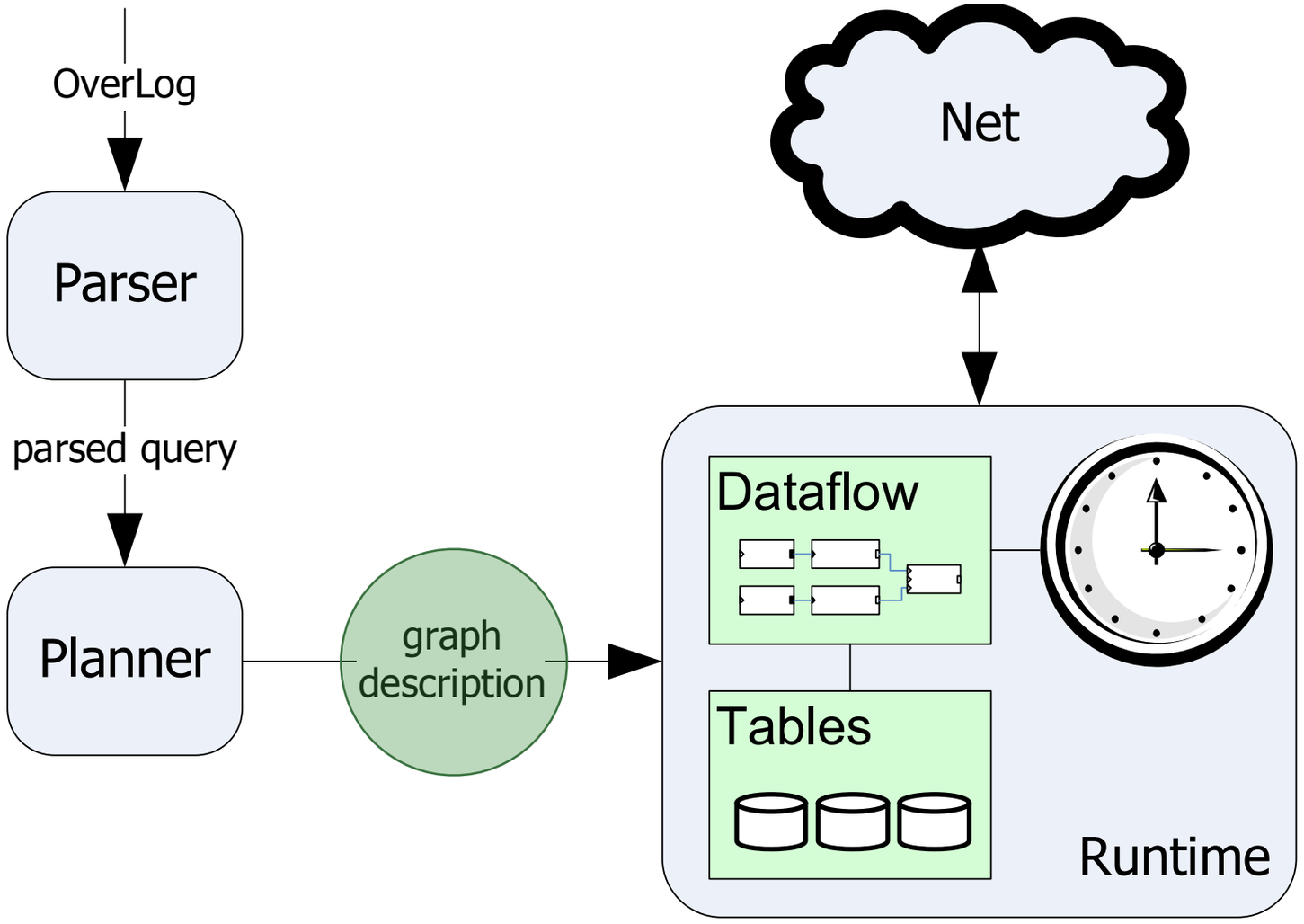
- WHY WHAT?
- SAY WHAT

- WHAT: HOW
- WHAT MORE
- WHAT’S IT TO YOU

TODAY

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

P2 @ 10,000 FEET



DATAFLOW BACKGROUND

- "boxes and arrows"
  - framework for data-intensive processing
- multiple contexts
  - UNIX pipes
  - DBMS query plans (relational algebra)
  - scientific workflows
  - click router toolkit

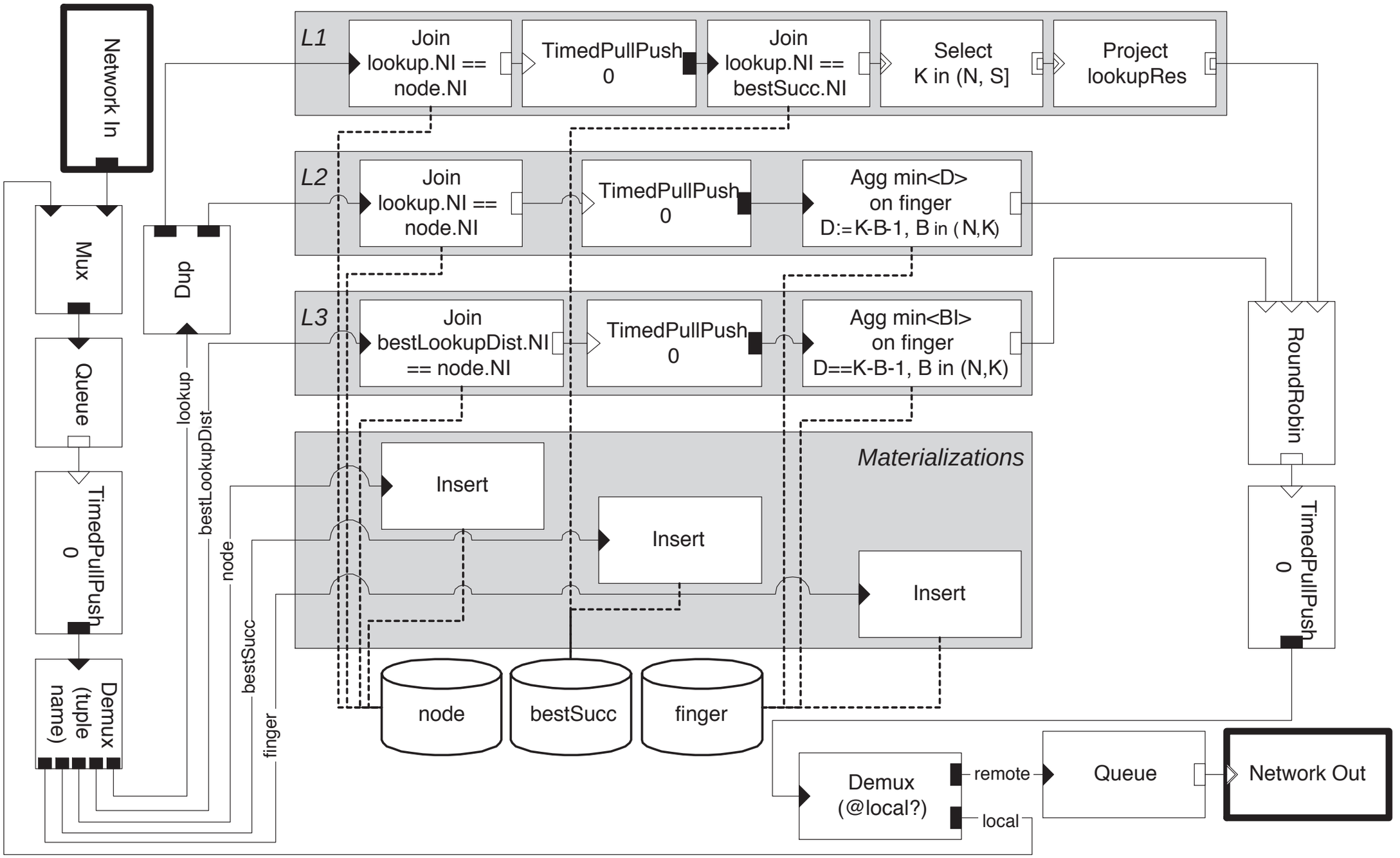
DATAFLOW EXAMPLE IN P2

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

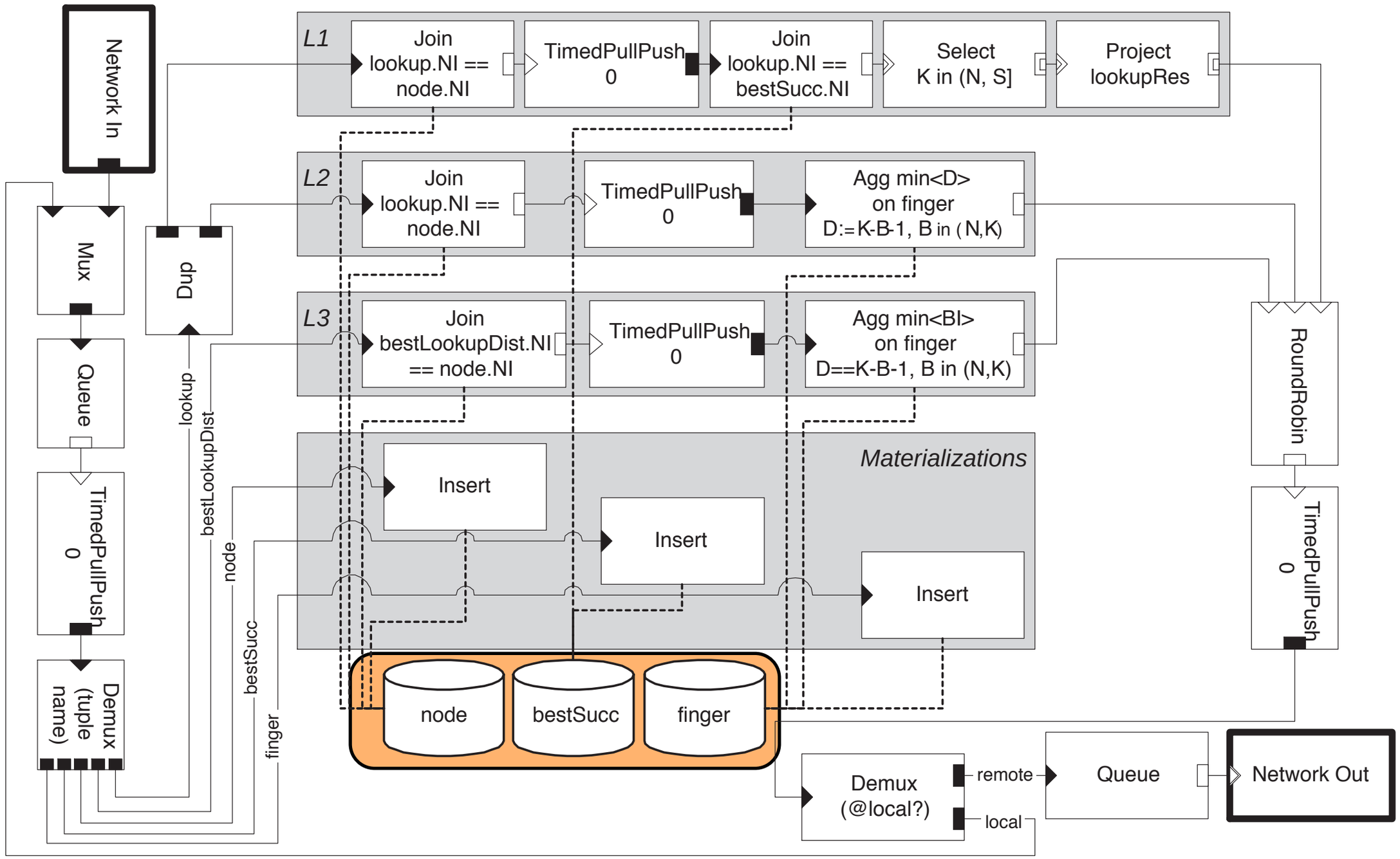
- L2 bestLookupDist(@NI,K,R,E,min<D>) :- node(@NI,N),
  lookup(@NI,K,R,E),
  finger(@NI,I,B,BI),
  D:=K-B-1, B in (N,K)

- 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).

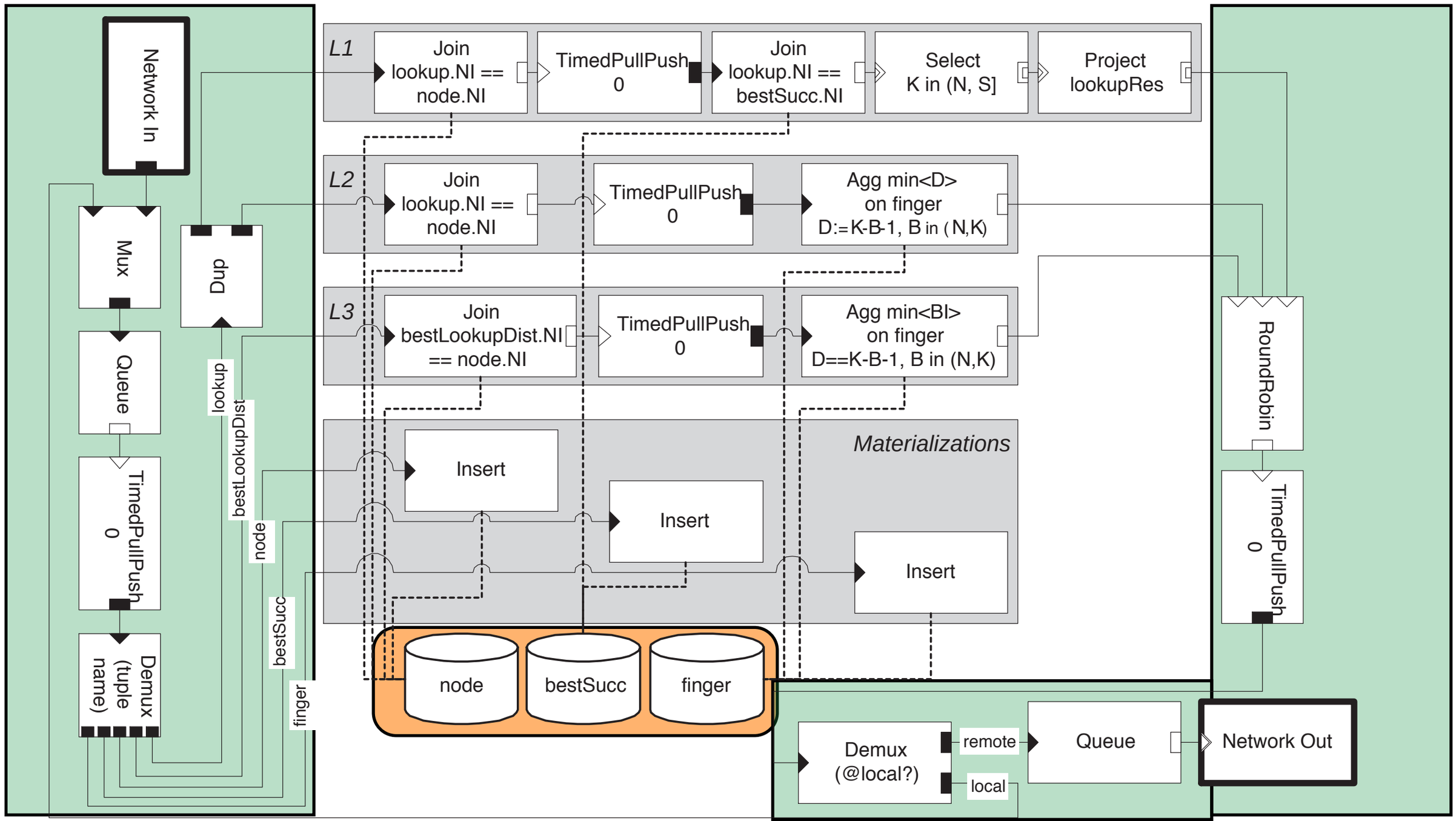
DATAFLOW EXAMPLE IN P2



DATAFLOW EXAMPLE IN P2



DATAFLOW EXAMPLE IN P2



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);
```

DATAFLOW DESIGN

- “just” producer/consumer?
  - a subtle design space
- indirection in *space* and *time*

DATAFLOW DESIGN

- “just” producer/consumer?
- a subtle design space
- indirection in *space* and *time*

DATAFLOW AND CONTROL FLOW

A NOTE ON DESIRED BEHAVIOR

- packets go deep on arrival
  - *queue before goo*
- send window reaches deep
  - *data later*
- app handling of exceptions

TODAY

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

TODAY

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

OVERLAY NETWORKS

- distributed apps on the network
  - the game: track...
    - subset of participating nodes
    - names for participating nodes
    - methods for routing to nodes via other nodes
- e.g. VPNs, DHT, P2P

DECLARATIVE OVERLAYS

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

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

P2-CHORD

- chord routing, with:
  - multiple successors
  - stabilization
  - optimized finger maintenance
  - failure detection
- 47 rules

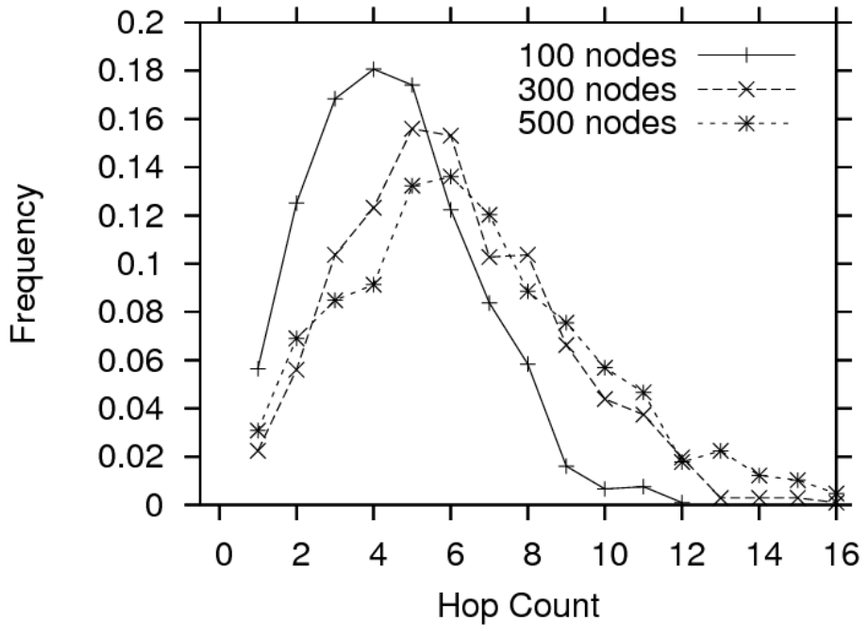
```
/* The base tuples */
materialize(node, infinity, 1, keys(1)).
materialize(finger, 180, 160, keys(2)).
materialize(bestSucc, infinity, 1, keys(1)).
materialize(succDist, 10, 100, keys(2)).
materialize(succ, 10, 100, keys(2)).
materialize(pred, infinity, 100, keys(1)).
materialize(succCount, infinity,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

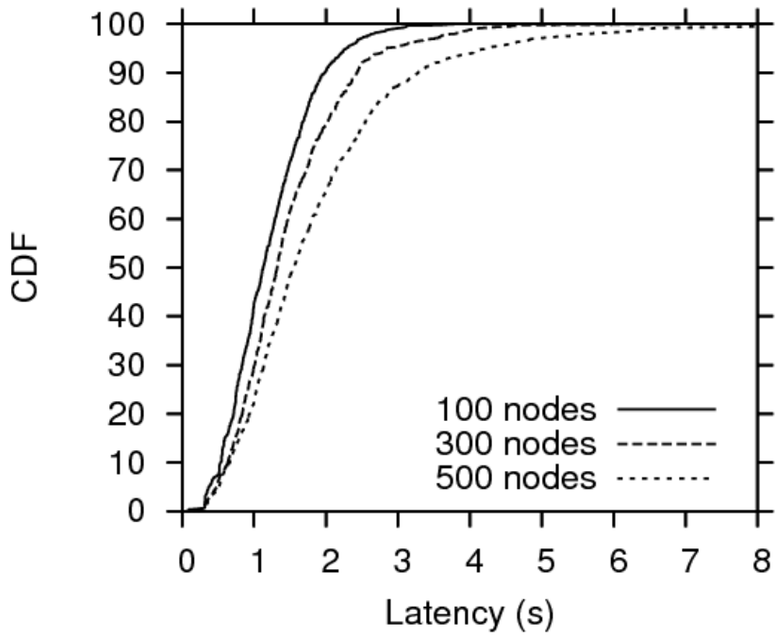
```
/* The base tuples */
materialize(node, infinity, 1, keys(1)).
materialize(finger, 180, 160, keys(2)).
materialize(bestSucc, infinity, 1, keys(1)).
materialize(succDist, 10, 100, keys(2)).
materialize(succ, 10, 100, keys(2)).
materialize(pred, infinity, 100, keys(1)).
materialize(succCount, infinity,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

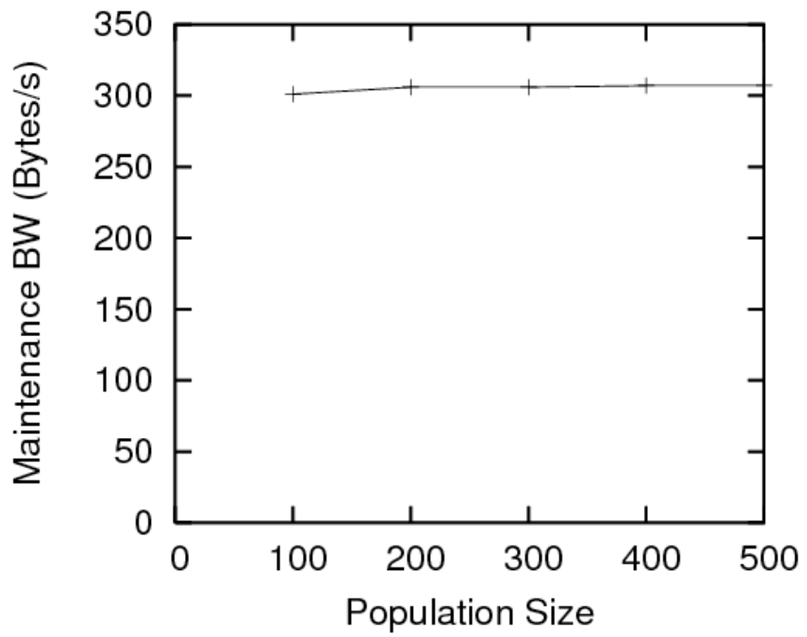
100x LESS CODE THAN MIT CHORD

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@64mins:
    97% consistency
  - P2-Chord@16mins:
    84% consistency
  - P2-Chord@8min:
    42% consistency
- C++ Chord:
  - MIT-Chord@47mins:
    99.9% consistency

TODAY

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

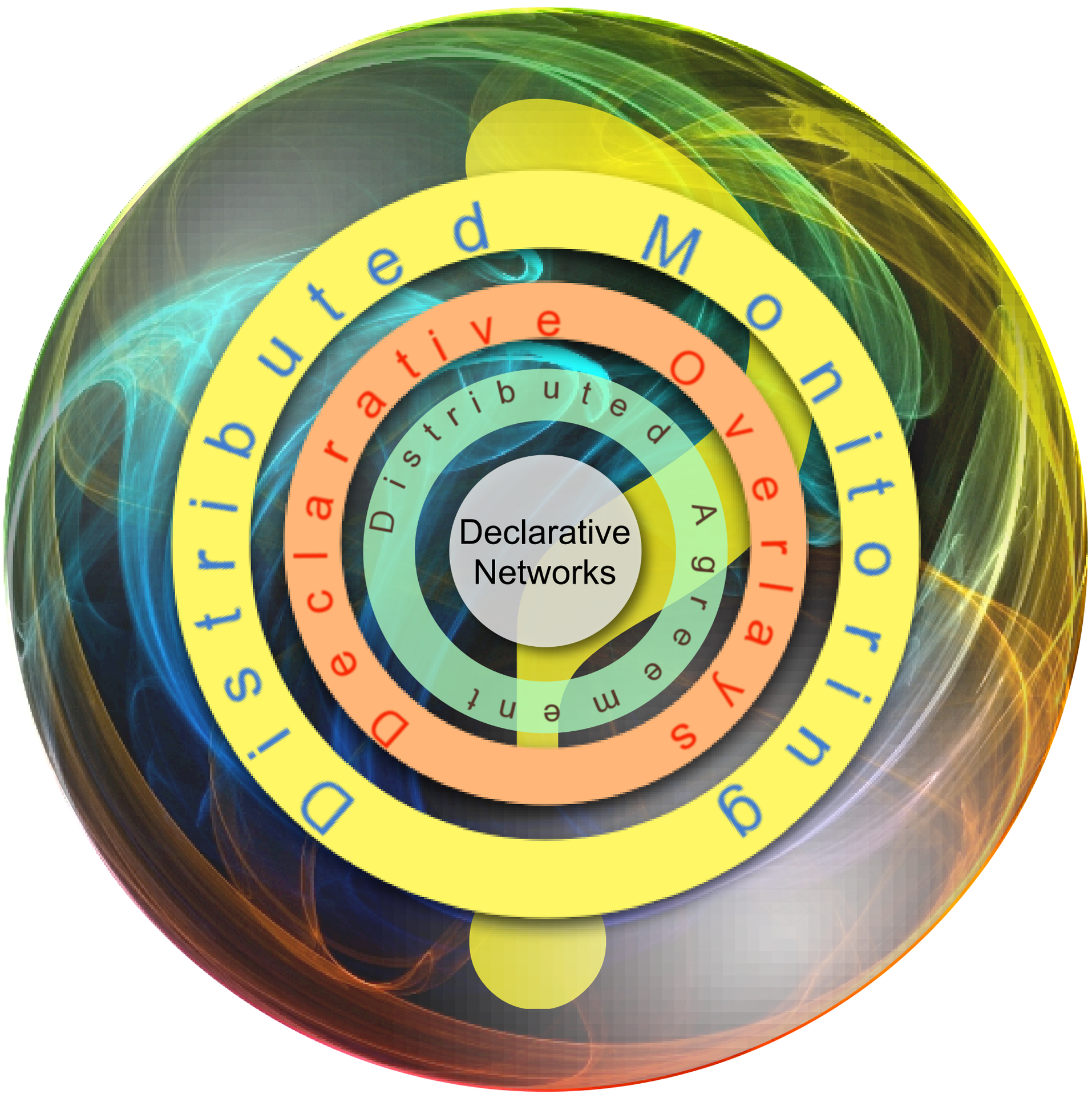
TODAY

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


Declarative
Networks


Distributed Agreement
Declarative Networks


Declarative Overlays
Distributed Management
Declarative Networks


Distributed Monitoring
Declarative Overlays
Distributed Agreement
Declarative Networks


Distributed Inference
Distributed Monitoring
Declarative Overlays
Distributed Debugging
Secure Accountable Declarative Networks

NETDB:

SAY

WHAT.

QUERIES

HTTP://P2.CS.BERKELEY.EDU