Lecture 4-extra: Datalog over Streams

David Maier
in collaboration with
Badrish Chandramouli,
Jonathan Goldstein

Use of Datalog in Declarative Networking

Datalog has been proposed as a language for expressing computations over networks

- Connectivity
- Routing
- Forensics

http://portal.acm.org/citation.cfm?id=1592785

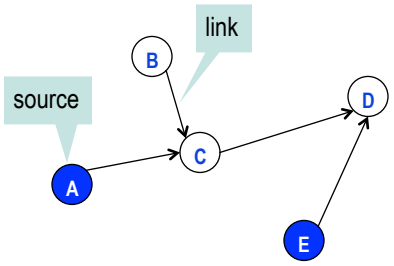
One issue in such settings is the frequently changing DB.

Distributed evaluation is another big issue

Recursive Network Query

```
reach(X) :- source(X).
reach(X) :- link(Y, X), reach(Y).
```

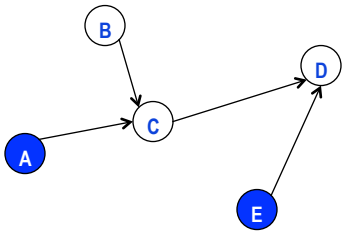
```
source(A)
source(E)

link(A, C)
link(B, C)
link(C, D)
link(E, D)
```



Bottom-up Execution

Apply rules right to left (a *phase*) until no new results

```
reach(X) :- source(X).
reach(X) :- link(Y, X), reach(Y).
```

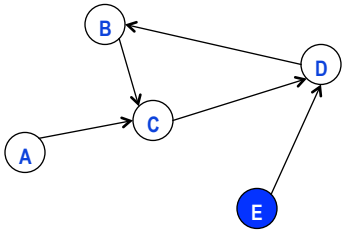



```
source(A)        Phase 1:
source(E)        reach(A)
                 reach(E)
link(A, C)       Phase 2:
link(B, C)       reach(A)
link(C, D)       reach(E)
link(E, D)       reach(C)
                 reach(D)
```

Changing Data

Changes to extensional DB

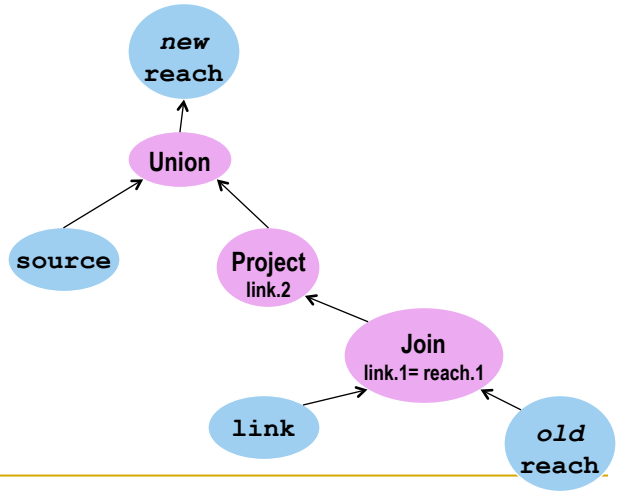
```
reach(X) :- source(X).
reach(X) :- link(Y, X), reach(Y).
```




```
source(E)

link(A, C)
link(B, C)
link(C, D)
link(E, D)
link(D, B)
```

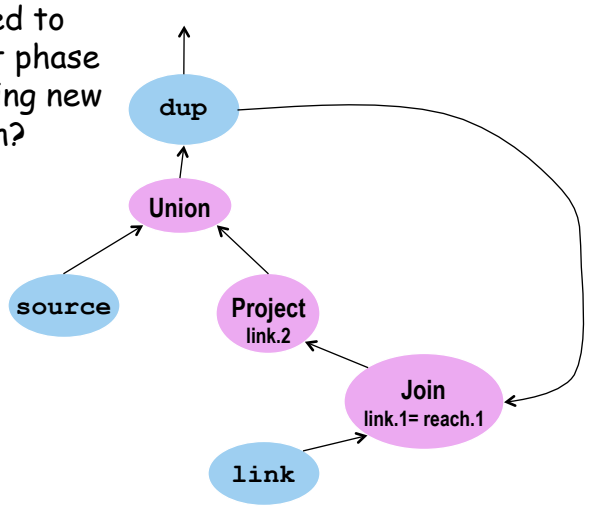
```
Phase 1:
reach(E)
Phase 2:
reach(E)
reach(D)
Phase 3:
reach(E)
reach(D)
reach(B)
Phase 4:
reach(E)
reach(D)
reach(B)
reach(C)
```


Can Represent Phase as Query



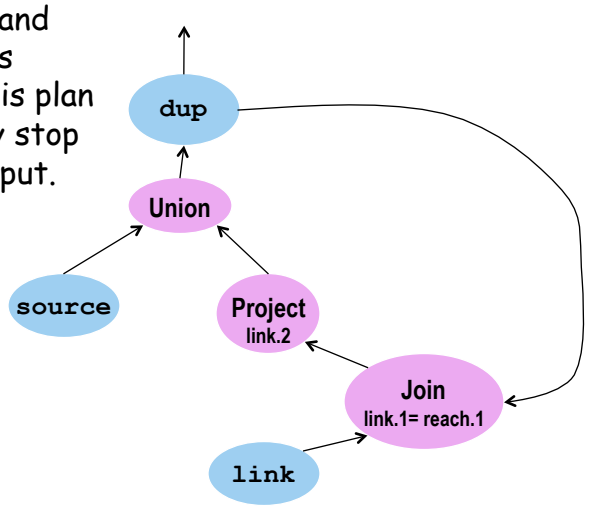
Free-Running Evaluation

Do we really need to finish current phase to start pushing new reach through?



Convergence

If inputs finite and Union removes duplicates, this plan will eventually stop producing output.



Cyclic Plan with Changing Data

If base data changes over time, we can just do a timestamped version of the data.

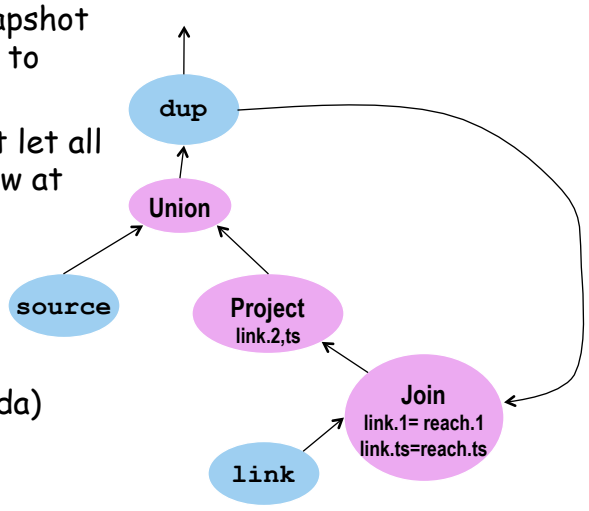
```
source(A,1)          link(E, D, 1)
                     link(E, D, 2)
source(E,1)          link(E, D, 3)
source(E,2)
source(E,3)          link(D, B, 3)
```


Modify Join to Compare Times

Could do one snapshot at a time, run to completion.

But why not just let all the tuples flow at once?

This works! (kinda)



Issue

Duplicate elimination is hard to support if inputs can be undone or evaluation is distributed

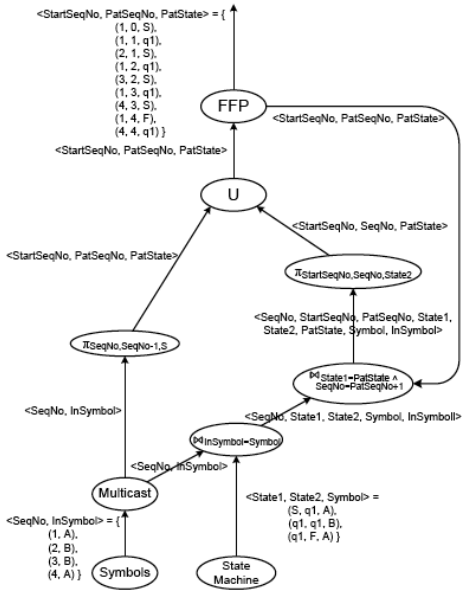
- Need to keep track of visited nodes to keep from going around cycles forever
  - We used a bit-vector of visited nodes
  - Berkeley people kept a path

Not an issue if derivation is bounded by the number of tuples on one of the inputs

- E.g., pattern matching with a finite automaton

Pattern matching with a finite automata is strongly convergent.

Cool bit: State machine can be a streaming input!



Issue 2

- Re-deriving the same info at different timesteps if the facts don't change.

Use time-interval semantics

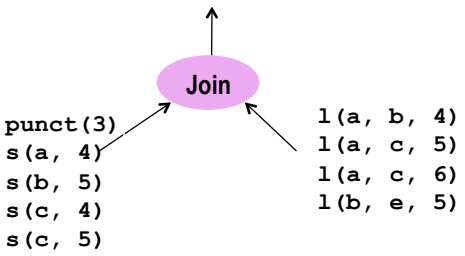
```
reach(E,[1,3]), link(E,D,[1,3]) →
  reach(D,[1,3])
reach(A,[1,1]), link(A,B,[1,3]) →
  reach(B,[1,1])
```


Punctuation: Signaling Progress

Don't want to require streams be ordered on time

Insert punctuations to mark lower bound on advancement of time

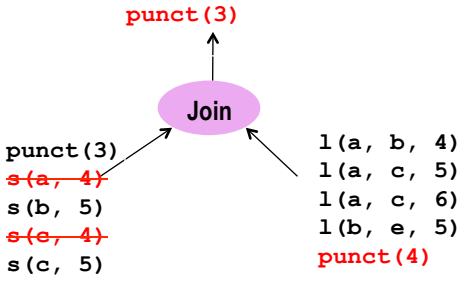
`punct(t)`: **have received all events up to time t**



Purge and Propagate

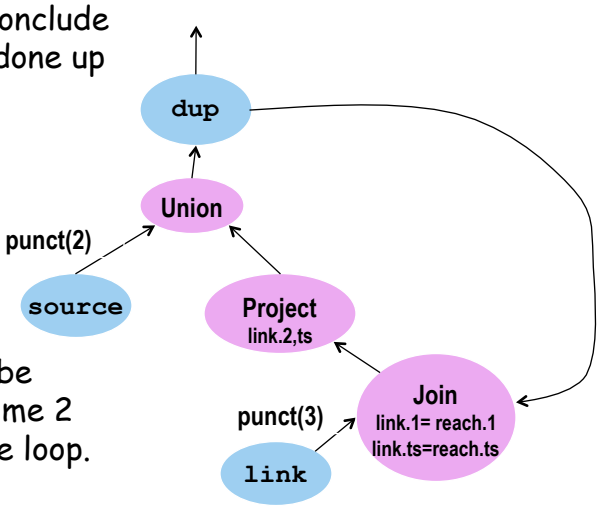
Use punctuation to purge state

Propagate progress to downstream operators



Punctuation

Would like to conclude that we are done up to time 2.



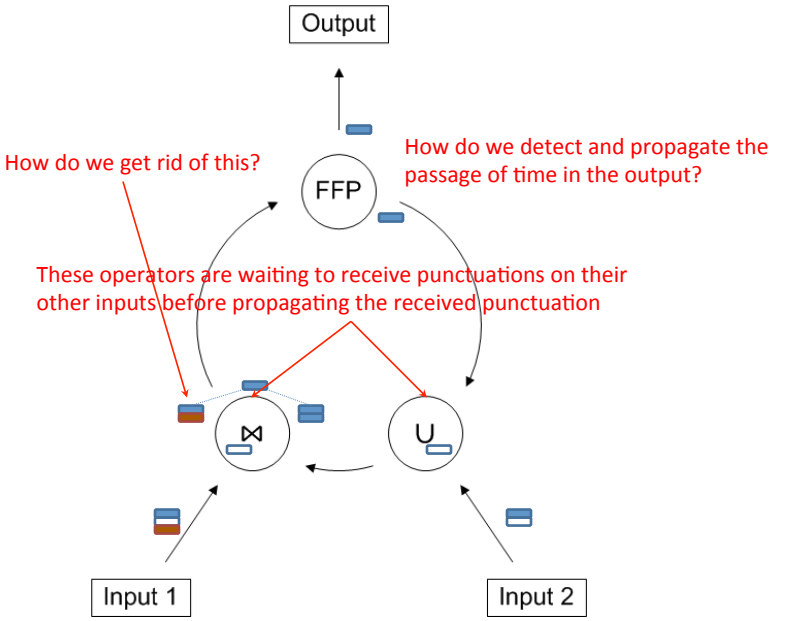
But might still be phases for time 2 running in the loop.

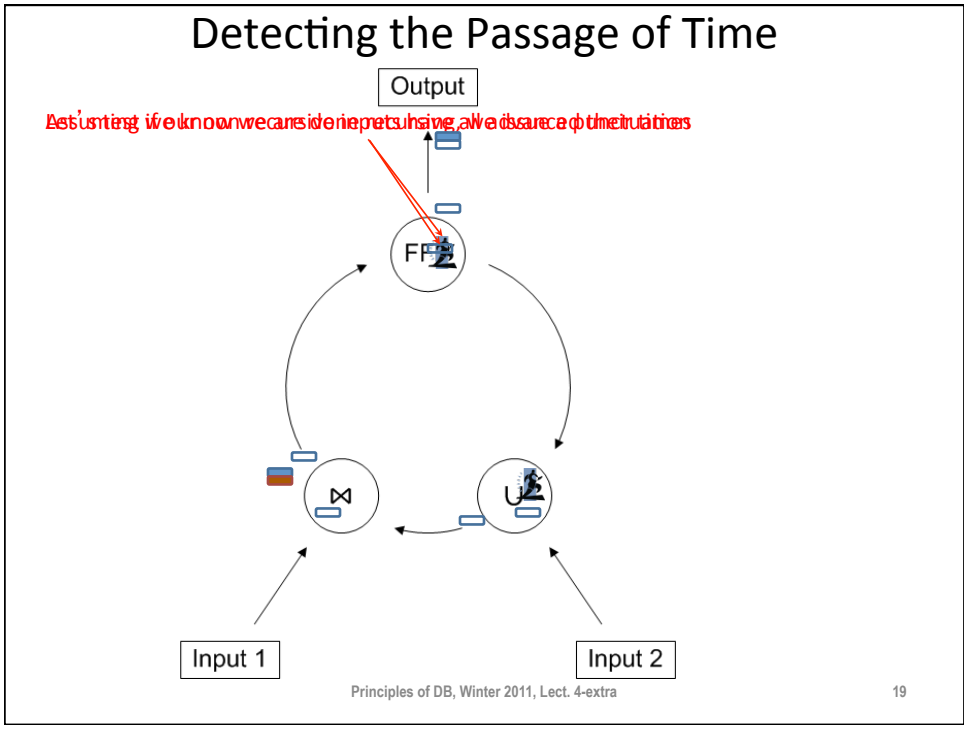
Solution: Flying Fixed Point (FFP)

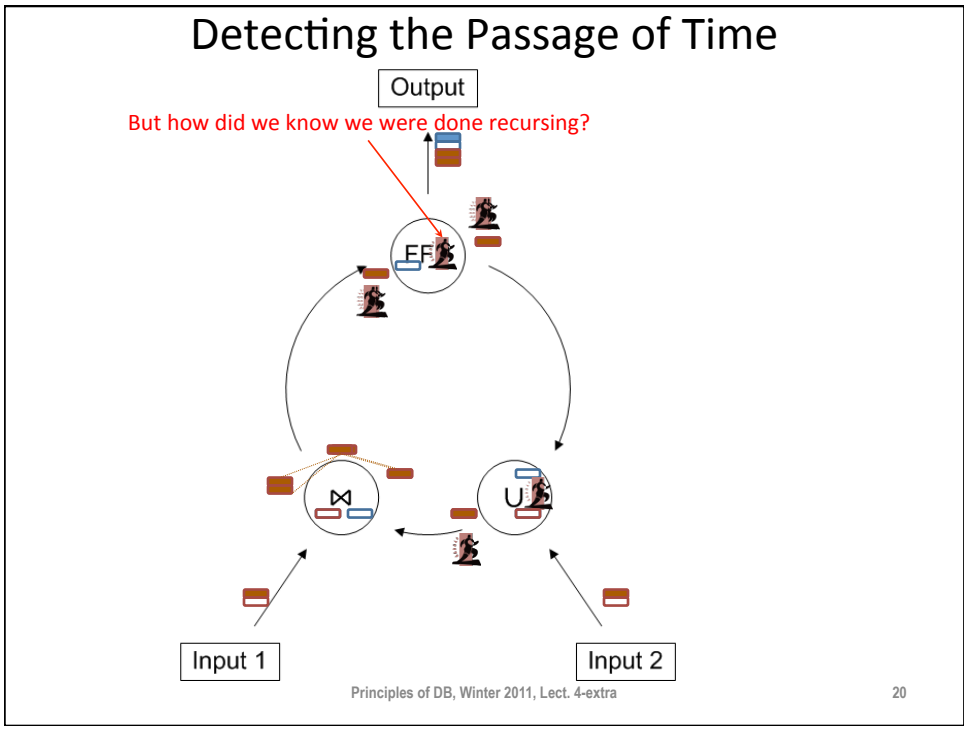
- Replaces Dup
- It does "guess & check" on progress

  Uses *speculative* punctuations (SPs): "I think we might have finished everything up until time 2"

Detecting the Passage of Time




Detecting the Passage of Time
Output
Input 1
Input 2
Principles of DB, Winter 2011, Lect. 4-extra
19

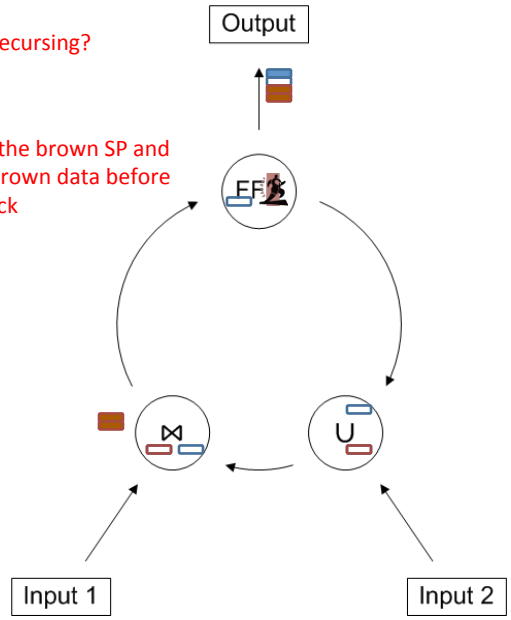

Detecting the Passage of Time
Output
But how did we know we were done recursing?
Input 1
Input 2
Principles of DB, Winter 2011, Lect. 4-extra
20

Detecting the Passage of Time

Are we done recursing?

Yes:

The FFP sent the brown SP and received no brown data before the SP got back



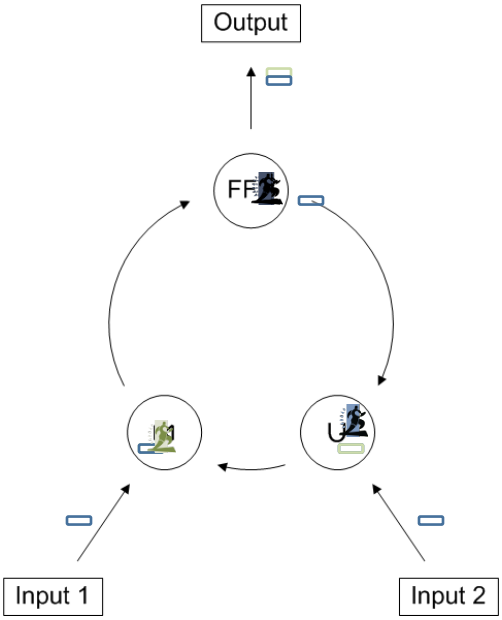
Detecting the passage of time

- How did we know to send the brown SP?
  - Proposal #1 : High watermark
    - FFP maintains a high watermark for the latest data received.
    - After the current SP is converted to a punctuation, a new SP with the latest high watermark is started.
  - What if we receive punctuations from the input but no data?
    - High watermark will never produce punctuations in the output

Proposal #2: Probing

- Always have an active SP. Start it off with the maximum punctuation time
- Allow the SP to demote its time based on received guarantees in the input
- If the SP comes back with the last propagated time, send it out with the maximum time.

Detecting the Passage of Time



Probing

- Passage of time in the input always propagates into passage of time in the output

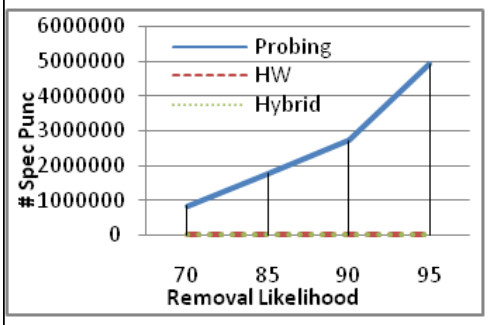
BUT

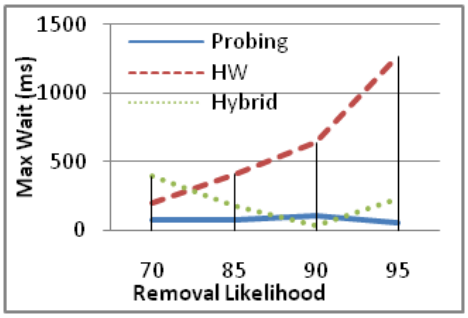
- Expensive during periods of low activity
- Guarantees wasteful 100% CPU utilization all the time

Proposal #3: Hybrid

- Like probing, allow the demotion of SPs to earlier times. Start them at max time.
- Don't have an SP at all times:
  - External progress (EP(t)) events are sent through the plan to FFP when a punctuation at time t is received
  - Start an SP only if an EP(t) is received with t > latest output punctuation
  - When a punctuation is produced, don't automatically start a new SP

Effect of Lulls in the Data





Lulls vs. # SPs

Lulls vs. responsiveness