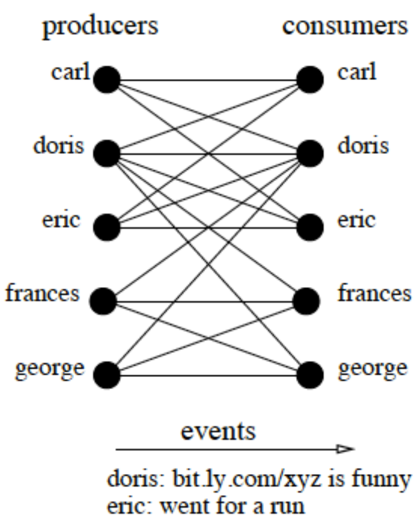
Feed Following

CompSci 590.03
Instructor: Ashwin Machanavajjhala

Feed Following Architecture

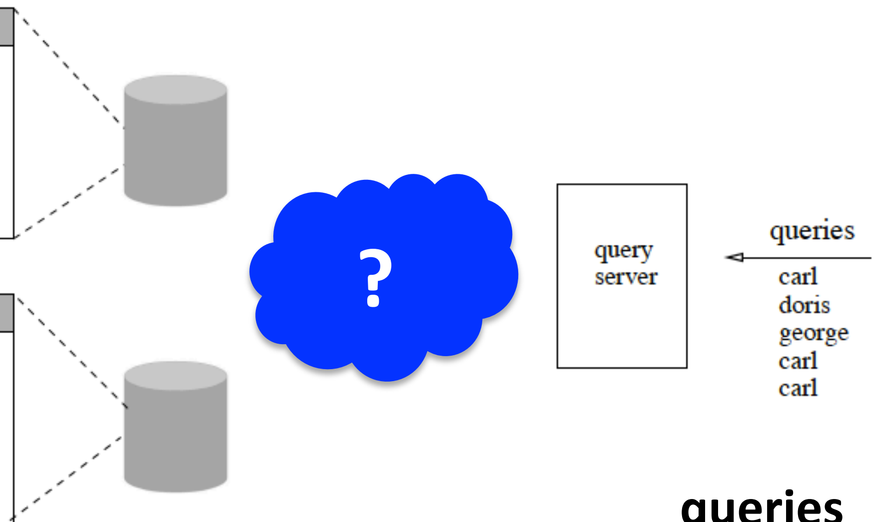


Connections

consumer	followed producers
carl	carl, doris, eric
doris	carl, doris, eric, frances, george
eric	carl, doris, eric
frances	doris, frances, george
george	doris, frances, george

Events

producer	events
carl	..., ..., ...
doris	bit.ly.com/xyz is funny, ..., ...
eric	went for a run, ..., ...
frances	..., ..., ...
george	..., ..., ...



queries

connections, events

Duke UNIVERSITY

Feed Queries

- Each user may ask queries related to the events generated by the producers they follow
  - Recent events are more important than older ones
  - Collect events from all or subset of producers
  - Filter events based on category

- K most recent events (based on criterion q) generated by producers that the consumers follow

- Queries may be posed by users, or posed on behalf of them by websites
  - When reading a new article, Google/Yahoo retrieves the latest k tweets that the user is following related to this article.

Constraints

- Latency: Most queries must be answered very quickly.
- Freshness: Ideally a user would like the answer to their query to reflect the current state of events generated by the producer.
  - But event processing is not instantaneous
- Relaxed Freshness: e.g., Answers may miss events that were generated in the last few seconds.

Constraints

- Time ordered: If e1 was generated before e2, them e1 precedes e2 in the output.
- Gapless: Suppose e1, e2 and e3 were all generated by the same producer, and they all satisfy the query. If e1 and e3 are output, then e2 should also be output.
- No duplicates

Formalizing Feed Following

- **Feed Query:** K most recent events (based on criterion q) generated by producers that the consumers follow
  - E.g., latest K events.
  - E.g., latest K events related to sports.

- **Performance Constraints (SLAs):**
  - Latency: p_L% of queries must be answered in less than t_L time.
  - Freshness: p_F % of the queries must return a feed that was up-to-date in the last t_F time units.

- **Minimize Cost(s):**
  - Possible bottlenecks: CPU, communication, memory footprint

Push vs Pull

- Pull: on receiving a customer query, pull events from each producer that satisfy the query, and construct the query answer.

- Push: Continuously keep track of the consumer feed (answer). When a producer generates a new event, push it to the consumers who follow the producer and update their feeds.

- **Which is better?**

Push vs Pull

- Bob follows Alice
- If Alice creates an event once a day, but Bob queries for events every 5 minutes
  - Push > Pull
- If Alice generated events every second, but Bob queries once a day
  - Pull > Push

Cost model

- H: cost of pushing an event to a consumer's feed
- **Push model:**
  Pay a cost of H for every event that is generated in the system.

Cost model

- Suppose the query is "K most recent events"
- Lj: cost of pulling from a producer j

- **Pull model:**
  Cost depends on the rate at which events are produced and queries are generated. Cost of pulling an event from producer p_j for customer c_i:

$$L_j \phi_{c_i} / \sum_{p_j \in F_i} \phi_{p_j}$$

Query rate for consumer ci

Producers followed by ci

Producer rate

MinCost

- Policy that minimizes cost for handling events generated by producer pj for consumer ci:

$$\begin{aligned} If\ \phi_{c_i} / \sum_{p_j \in F_i} \phi_{p_j} &\quad >= \quad H/L_j,\ push\ for\ all\ events\ by\ p_j \\ If\ \phi_{c_i} / \sum_{p_j \in F_i} \phi_{p_j} &\quad < \quad H/L_j,\ pull\ for\ all\ events\ by\ p_j \end{aligned}$$

- Decision is made on a per-edge basis

Latency Constrained Problem

- Pull strategy may reduce cost, but increases query latency.
- If pL% of the queries are required to have low latency, then one may need to change some of the edges from Pull to Push.
- Equivalent to a Knapsack problem.

Summary

Push vs Pull

- If a consumer queries the system more often than its producer create updates, then **use Push**
- If a producer creates updates more often than queries from a consumer, then **use Pull**

OPEN RESEARCH PROBLEMS

Open Questions

- View Selection:
  - which views to materialize
- View Scheduling:
  - when to build views, when to incrementally maintain and when to expire views
- View Placement:
  - Optimally place views in a distributed setting
- Access control and fine grained queries
- Handling Changes in the Connections graphs

Duke UNIVERSITY

Materialized views for Feed Following

- Push can be thought of as:
  Maintain a view for every consumer which contains the answer to the consumer query.
  On every new event, push ensures these views are up-to-date

- Pull can be thought of as:
  Maintain a view for each producer (e.g., containing their latest k events).
  When a new query comes, pull answers the consumer query using the views.

View Selection

Which type of views should be materialized?

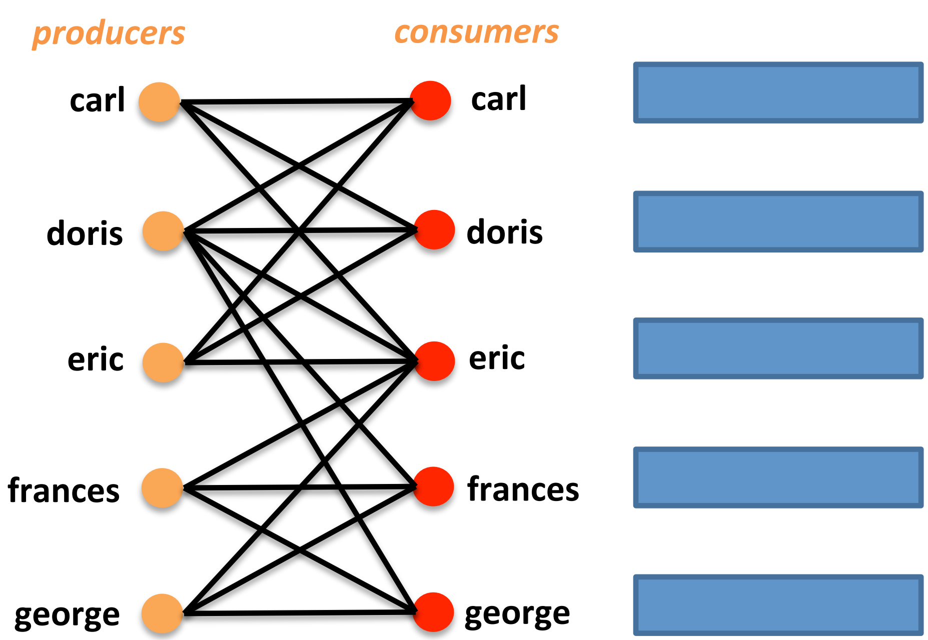
Optimization Criterion:

- *Update*: Cost of maintaining views when a new event enters the system
- *Query*: Cost of generating a user feed from views
- *Memory footprint*: Total size of all views

View Selection

Query: Return latest k events produced by friends.

Design 1: One view per consumer *(with latest k events from friends)*



Duke UNIVERSITY

View Selection

Query: Return latest k events produced by friends.

Design 1: One view per consumer *(with latest k events from friends)*

Update: O(degree(producer))

Query: O(1)

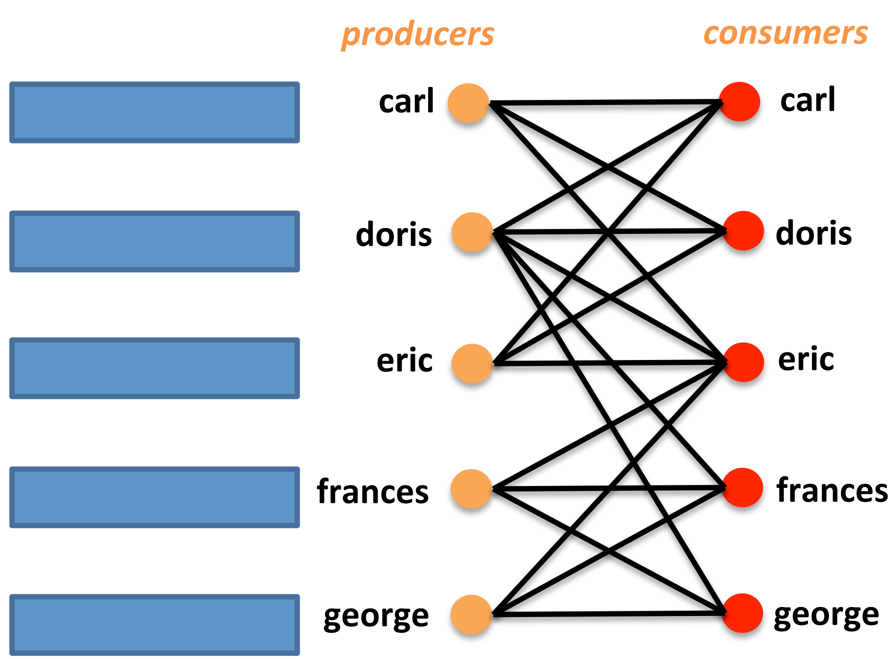
Memory footprint: O(# consumers)

Duke UNIVERSITY

View Selection

Query: Return latest k events produced by friends.

Design 2: One view per producer*(with latest k events from producer)*



View Selection

Query: Return latest k events produced by friends.

Design 2: One view per producer *(with latest k events from producer)*

Update cost: O(1)

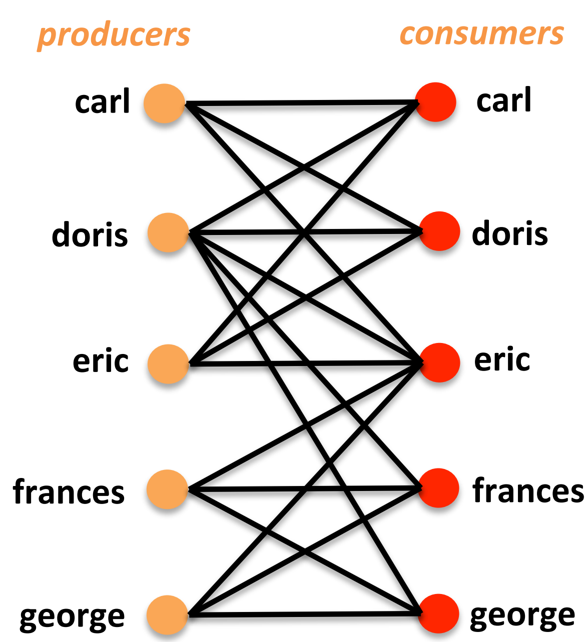
Query cost: O(degree(consumer))

Memory footprint: O(# producers)

View Selection

Query: Return latest k events produced by friends.

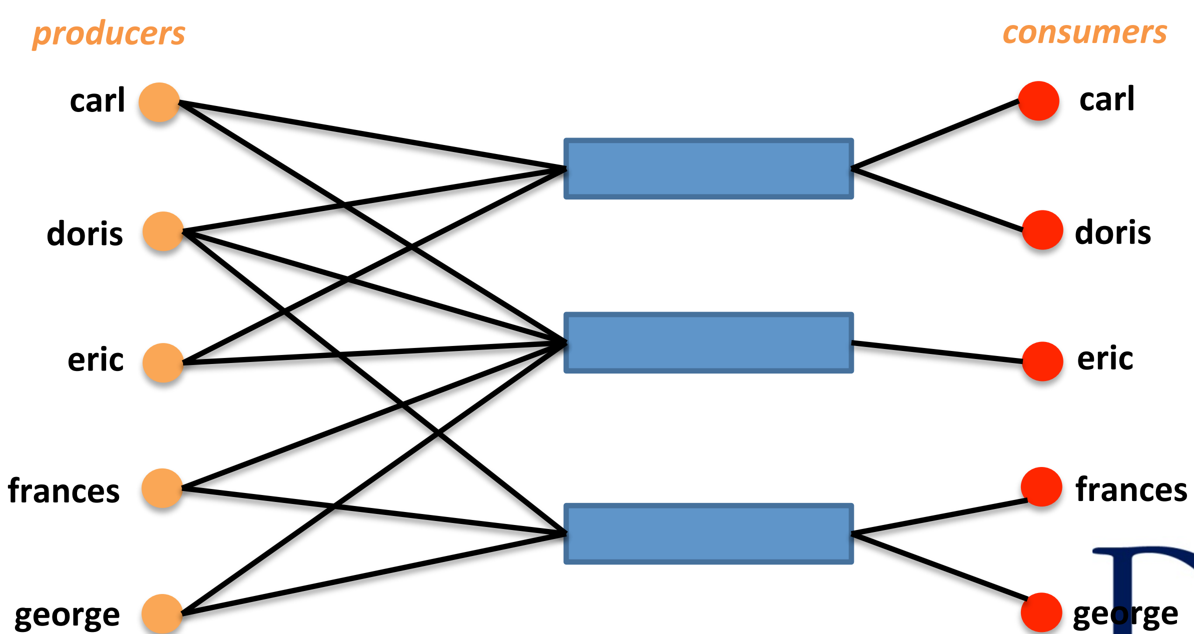
Design 3: One view per set of producers S *(with latest k events from producer in S)*



View Selection

Query: Return latest k events produced by friends.

Design 3: One view per set of producers S *(with latest k events from producer in S)*

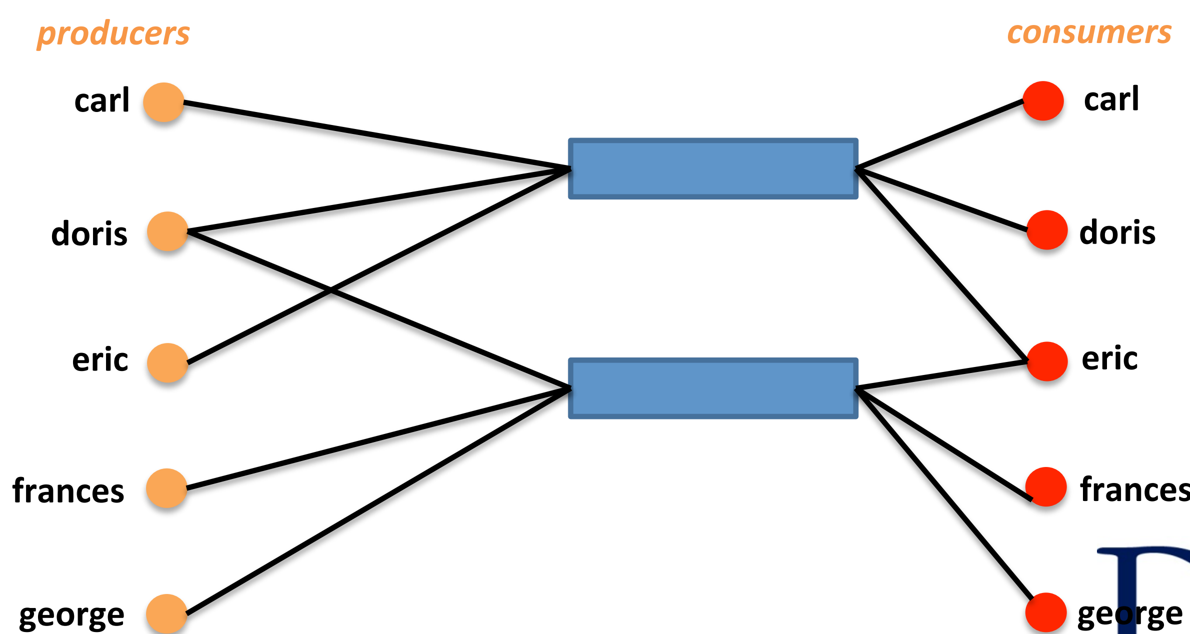


Duke UNIVERSITY

View Selection

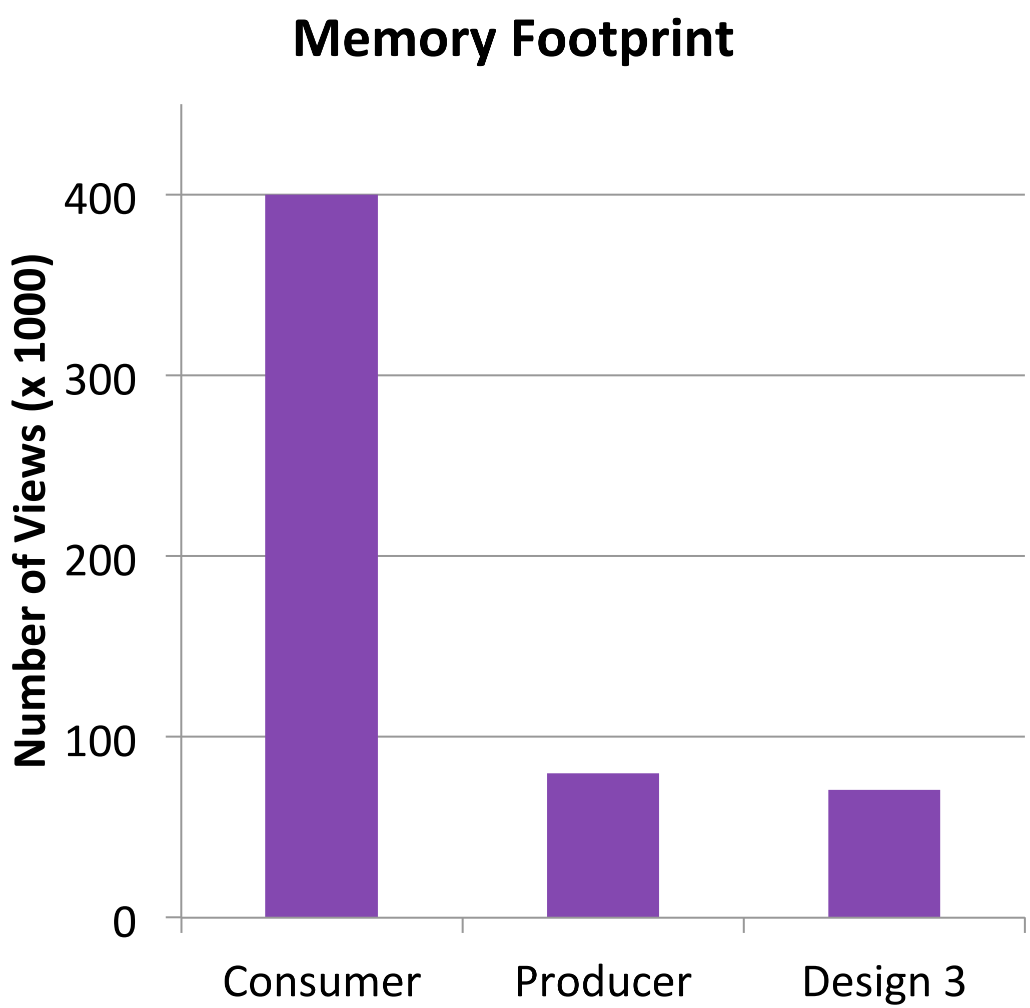
Query: Return latest k events produced by friends.

Design 3: One view per set of producers S *(with latest k events from producer in S)*



Duke UNIVERSITY

View Selection



- Subset of Twitter social graph
- 400,000 consumers
- 79,842 producers
- Design 3: 70,926 views
  - **5.6x improvement** over consumer views
  - **12% improvement** over producer views

View Scheduling

We do not need all views at all times. When do we evict them/let them grow stale and when do we rebuild/refresh them?

- May be able to predict when users will pose queries.
- In certain cases, there is a fixed schedule for queries
  - Regression tasks on a codebase are always run at the same time everyday.

Signature Scheduling

Feasibility of view scheduling:
users typically have a diurnal access pattern

- Based on access logs generate access signature

Logged Accesses by Eric

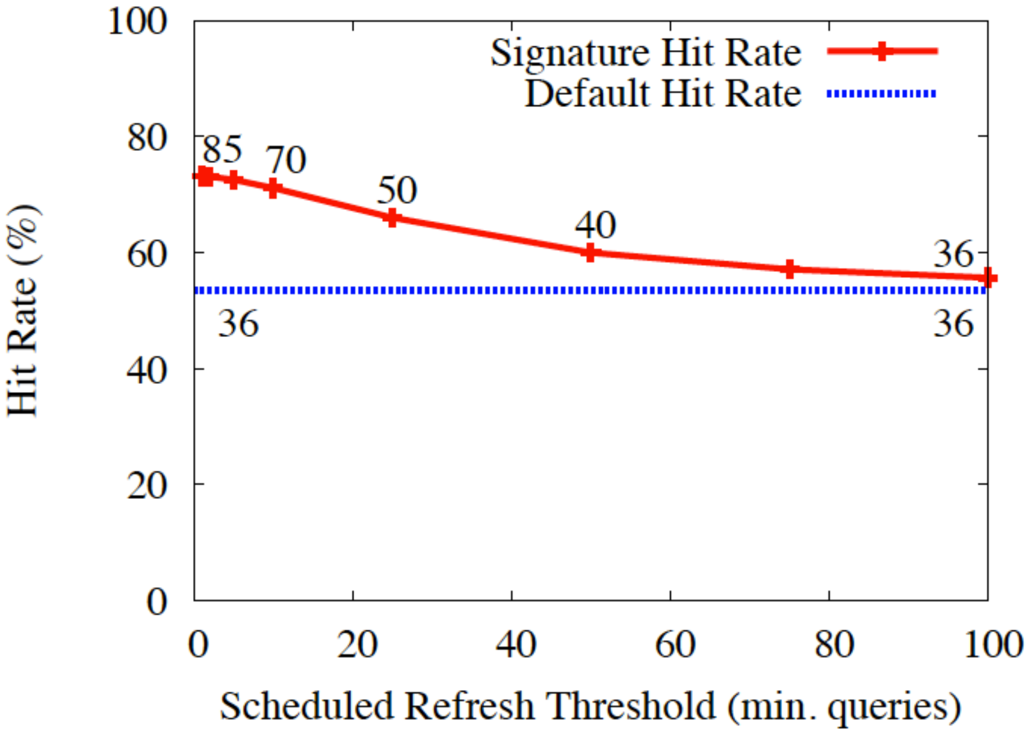
Monday, 4:30 PM
Monday, 6:10 PM
Thursday, 7:45 PM
Friday, 1:15 PM
Friday, 6:40 PM
Friday, 10:20 PM

→ Signature: 0000000000000100101110010

Signature Scheduling

Hit Rate: percentage of queries answered with fresh results

Schedule Refresh Threshold: number of queries a consumer must make in training to get signature refreshes



Duke UNIVERSITY

View Placement

How to optimally place views in a distributed setting?

Optimization criteria:

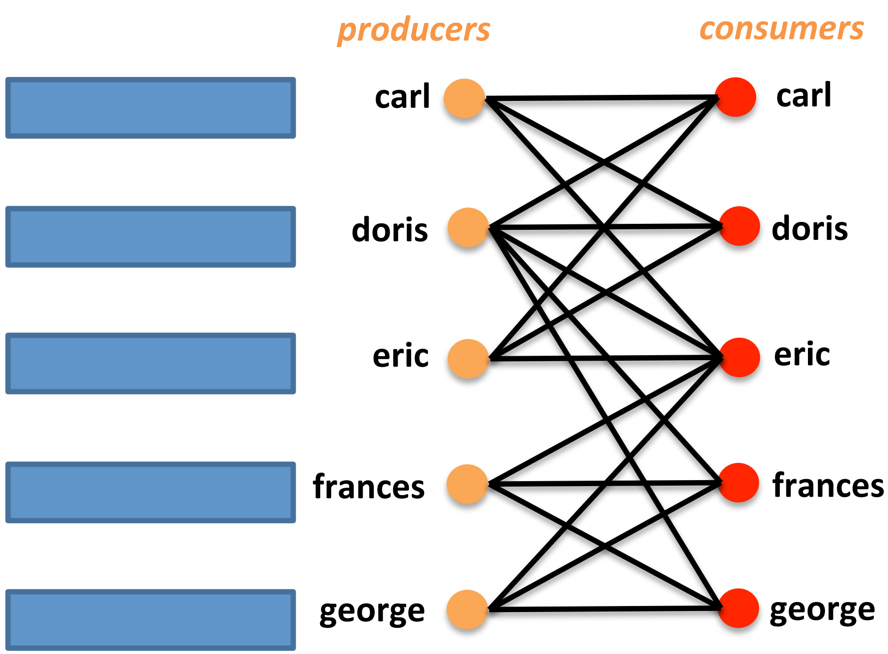
Update: Number of machines to be accessed to update views on a new event

Query: Number of machines to be accessed to answer a user query

Size of each machine

View Placement

Suppose consumer views must be distributed on 2 machines, at most 3 views per machine

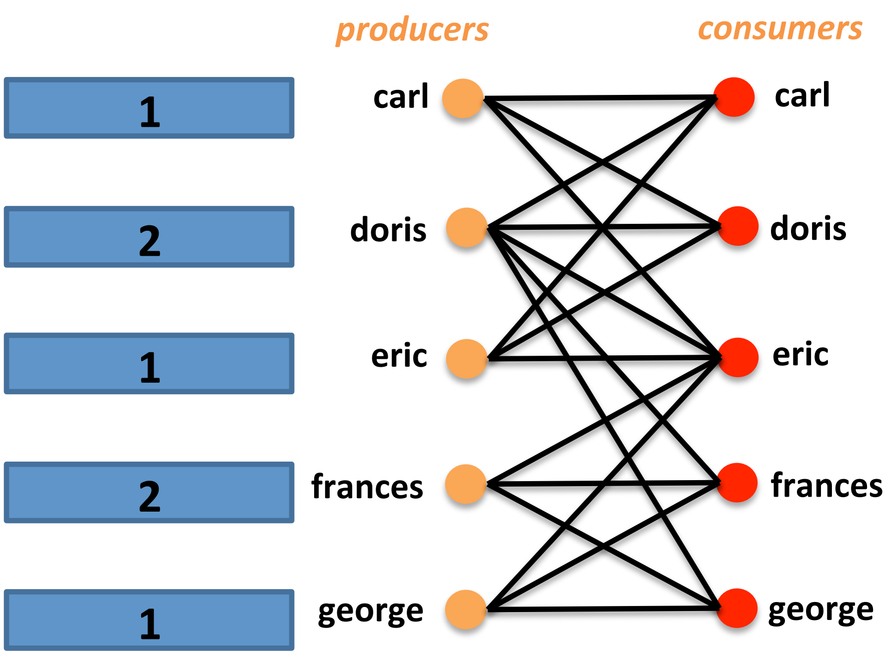


Duke UNIVERSITY

View Placement

Random placement:

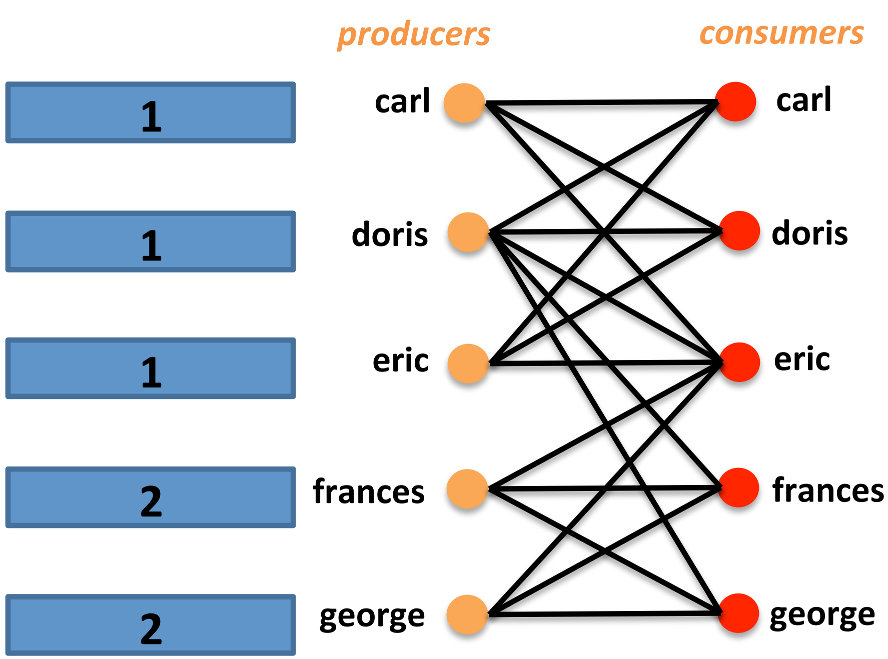
Every query must access 2 machines



View Placement

Intelligent placement:

Carl and Doris only need to access one machine.



Duke UNIVERSITY

Open Questions

- View Selection
- View Scheduling
- View Placement

- Access control and fine grained queries

- Handling Changes in the Connections graph

- Answering more complex aggregate queries over recent events

Duke UNIVERSITY