TelegraphCQ: Continuous Dataflow Processing for an Uncertain World

Michael Franklin

UC Berkeley

March 2003

Joint work w/the Berkeley DB Group.

Telegraph: Context

- **Networked data streams are central to current and future applications.**

  Data is a *commodity* — it is *useful* only when it is moved to where it is needed.

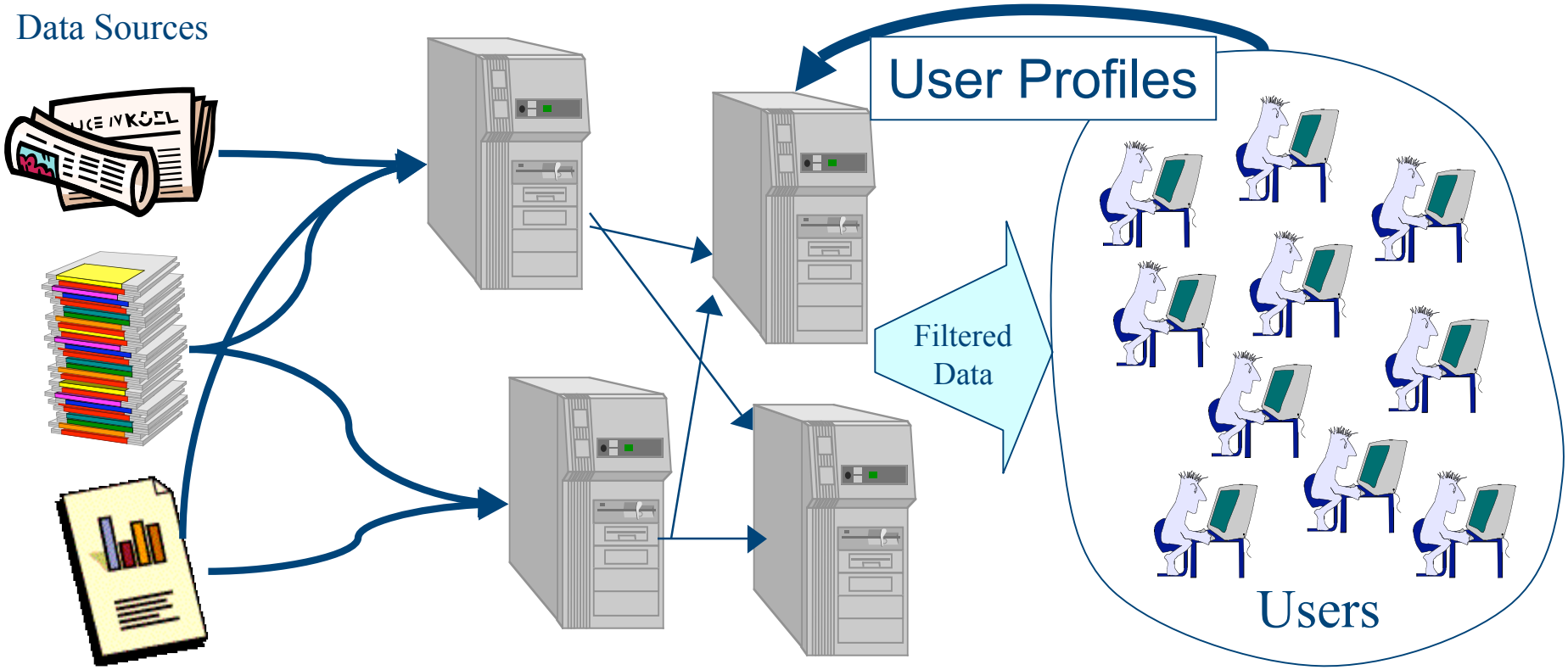
- **Existing data management & query processing infrastructure is not up to the task:**
  - Adaptability
  - Continuous and Incremental Processing
  - Work Sharing for Large Scale
  - Resource scalability: from "smart dust" up to clusters to grids.

Example App1: “Real-Time Business”

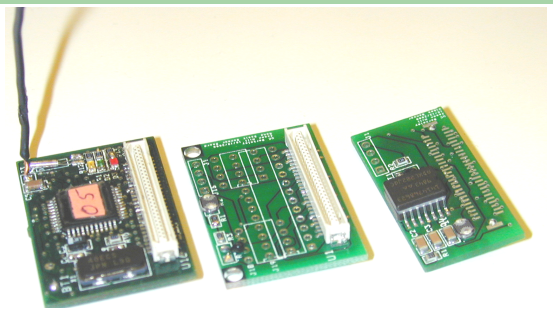
- **Event-driven processing**
- **B2B and Enterprise apps**
  - Supply-Chain, CRM
  - Trade Reconciliation, Order Processing etc.
- **(Quasi) real-time flow of *events* and *data***
- **Must manage these flows to drive business processes.**
- **Mine flows to create and adjust business rules.**
- **Can also “tap into” flows for on-line analysis.**

Example App 2: Information Dissemination

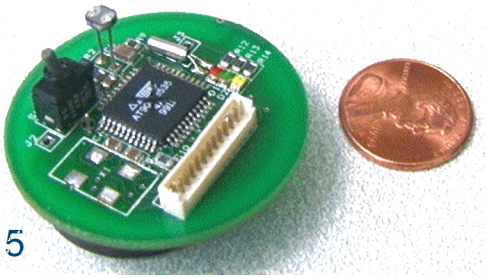
- *Doc creation* or *crawler* initiates flow of data towards users.
- Users + system initiate flow of *profiles* back towards data.



Example App 3: Sensor Networks



- **Tiny devices measure the environment.**
  - Berkeley “motes”, Smart Dust, Smart Tags, …
- **Form dynamic *ad hoc* networks, aggregate and communicate streams of values.**
- **Major research thrust at Berkeley.**
  - Apps: Transportation, Seismic, Energy,…



Common Features

- **Centrality of *dataflow***
  - Architecture is focused on data movement.
  - Moving streams of data through operators in a network.

    ➡ Requires *intelligent, low-overhead, shared* routing and processing
- ***Volatility* of the environment**
  - Dynamic resources & topology, partial failures
  - Long-running (never-ending?) tasks
  - Potential for user interaction during the flow
  - Large Scale: users, data, resources, …

    ➡ Requires *adaptivity*

Adaptive Query Processing

static plans

current DBMS

- **Existing systems are adaptive but at a slow rate.**
  - Collect Stats
  - Compile and Optimize Query
  - Eventually collect stats again or change schema
  - Re-compile and optimize if necessary.

[Survey: Hellerstein, Franklin, et al., DE Bulletin 2000]

Adaptive Query Processing

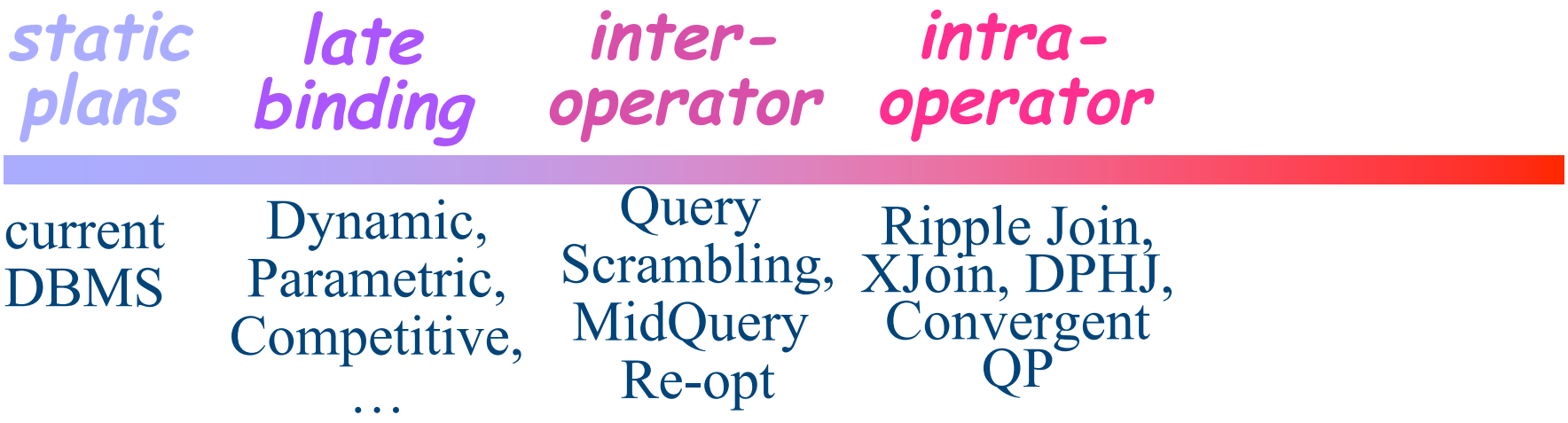
- **Goal: allow adjustments for runtime conditions.**
- **Basic idea: leave "options" in compiled plan.**
- **At runtime, bind options based on observed state:**
  - available memory, load, cache contents, etc.
- **Once bound, plan is followed for entire query.**

[HP88,GW89,IN+92,GC94,AC+96, AZ96,LP97]

Adaptive Query Processing

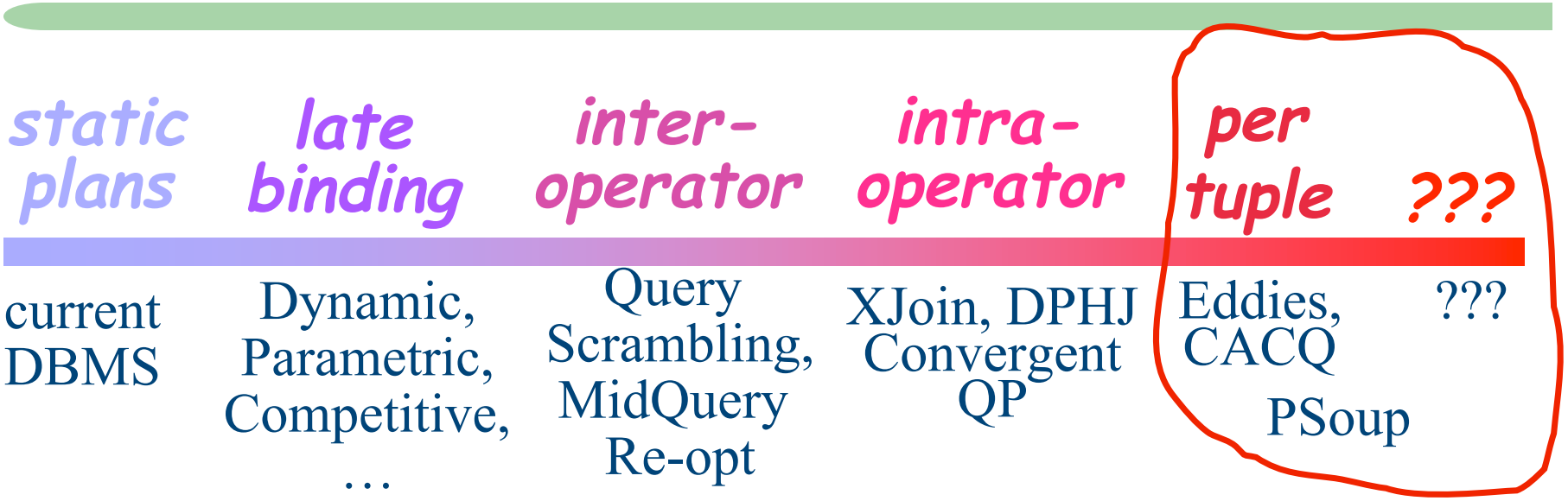
- **Start with a compiled plan.**
- **Observe performance between blocking (or blocked) operators.**
- **Re-optimize remainder of plan if divergence from expected data delivery rate** [AF+96,UFA98] **or data statistics** [KD98].

Adaptive Query Processing



- **Join operators themselves can be made adaptive:**
  - to user needs (Ripple Joins [HH99])
  - to memory limitations (DPHJ [IF+99])
  - to memory limiations and delays (XJoin [UF00])
- **Plan Re-optimization can also be done in mid-operation** (Convergent QP [IHW02])

Adaptive Query Processing



- **This is the region that we are exploring in the *Telegraph* project at Berkeley.**

Outline

- Motivation and context
- Telegraph: basic technology
- The new TelegraphCQ system
- Stream semantics and language Issues
- Conclusions

Telegraph Overview

- **An *adaptive* system for large-scale *shared* dataflow processing.**
  - *Sharing and adaptivity go hand-in-hand*

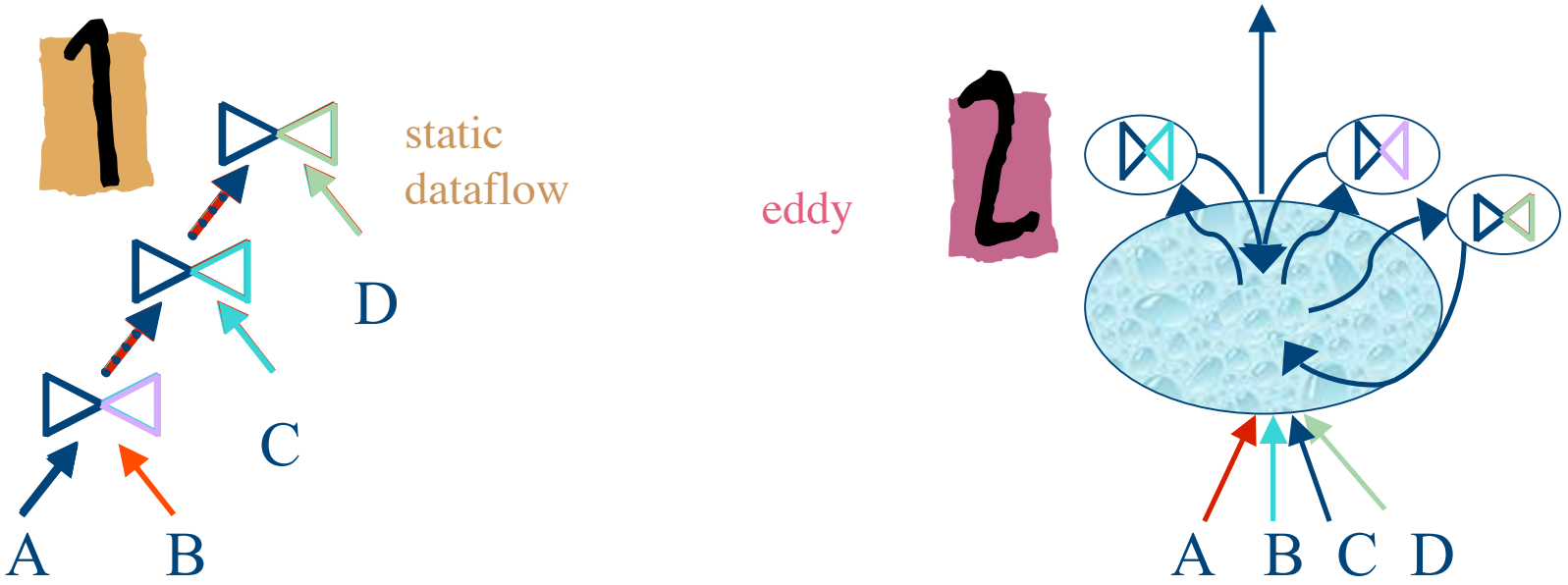
- **Based on an extensible set of operators:**
  1) **Ingress** (data access) operators
     - Screen Scraper, Napster/Gnutella readers,
     - File readers, Sensor Proxies
  2) Non-Blocking **Data processing** operators
     - Selections (filters), XJoins, …
  3) **Adaptive Routing** Operators
     - Eddies, STeMs, FLuX, etc.
- **Operators connected through "Fjords" [MF02]**
  - queue-based framework unifying push&pull.

The Telegraph Project

- **We've explored sharing and adaptivity in …**
  - Eddies: Continuously adaptive queries [SIGMOD 00]
  - Fjords: Inter-module communication [ICDE 02]
  - CACQ: Sharing, Tuple-lineage [SIGMOD 02]
  - PSoup: Query=Data duality [VLDB 02]
  - STeMs: Half-a-symmetric-join, tuple store [ICDE 03]
  - FLuX: Fault tolerance, load balancing [ICDE 03]
- **.. and built a first generation prototype** [SIGMODRec01]
  - Built from scratch in Java
- **Currently finishing up 2nd generation** [CIDR 03]
  - In "C", based on open-source PostgreSQL

Routing Operators: Eddies

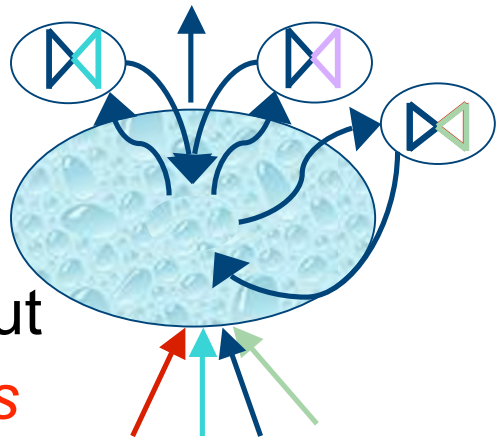
[Avnur & Hellerstein, SIGMOD 00]



- **How to order and reorder operators over time?**
  - Traditionally, use performance, economic/admin feedback
  - won't work for never-ending queries over volatile streams
- **Instead, use adaptive record routing.**

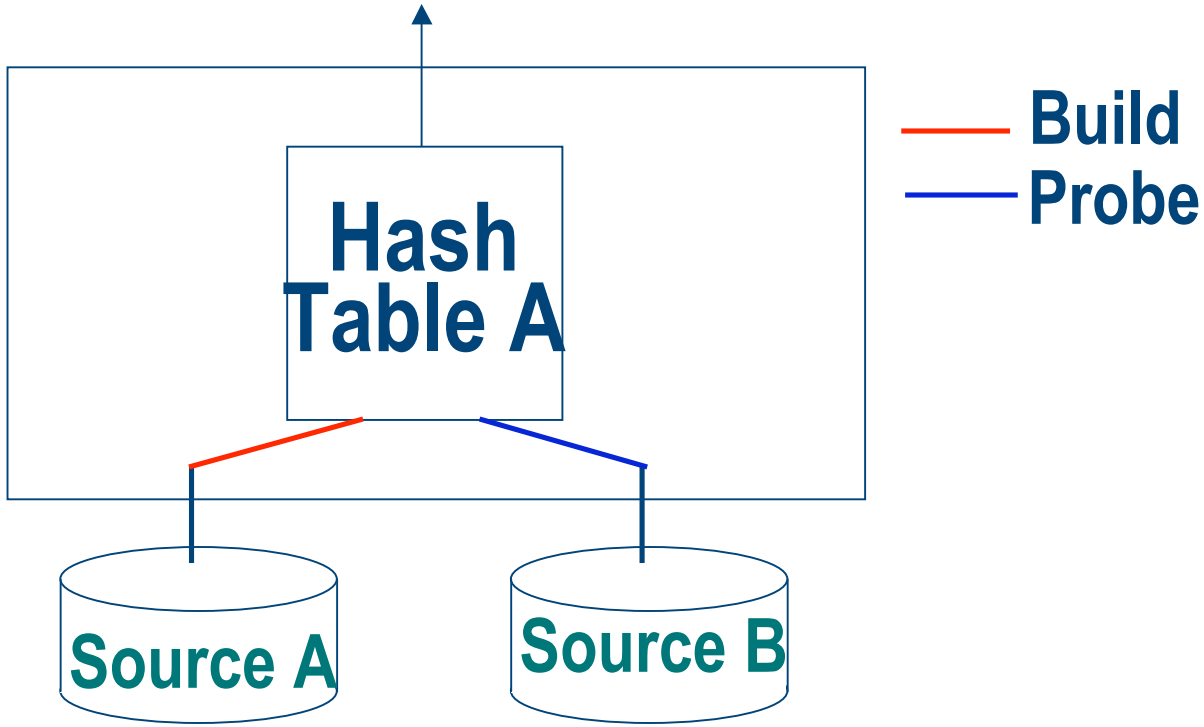
  Reoptimization = change in routing policy

Eddy – Per Tuple Adaptivity



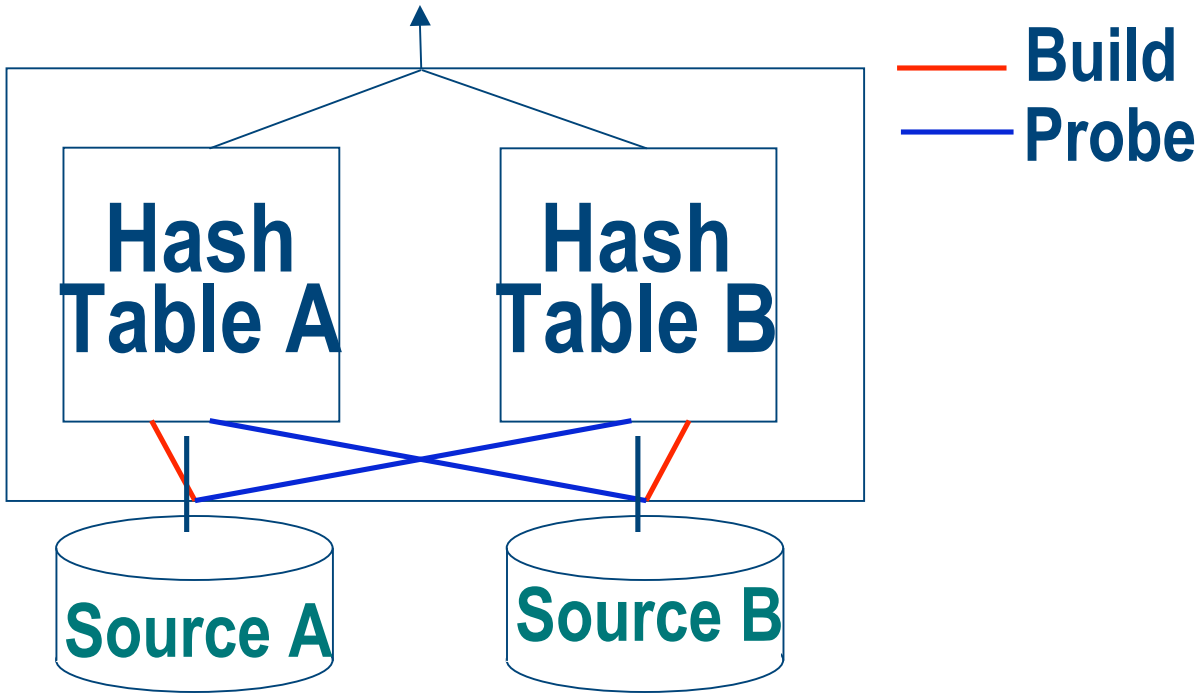
- **Adjusts flow adaptively**
  - Tuples routed through ops in diff. orders
  - Must visit each operator once before output
    - *State is maintained on a per-tuple basis*
- **Two complementary routing mechanisms**
  - Back pressure: each operator has a queue, don't route to ops with full queue – avoids expensive operators.
  - Lottery Scheduling: Give preference to operators that are more selective.
- **Initial Results showed eddy could "learn" good plans and adjust to changes in stream contents over time.**
- **Currently in the process of exploring the inherent tradeoffs of such fine-grained adaptivity.**

Non-Blocking Operators – Join



Traditional Hash Joins block when one input stalls.

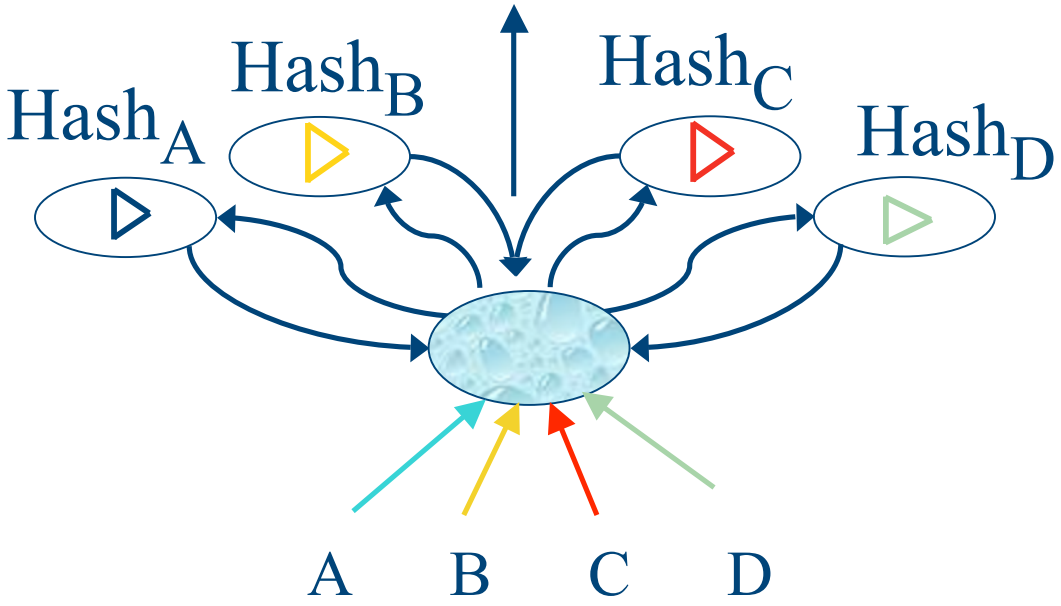
Non-Blocking Operators – Join



- Symm Hash Join [WA91] blocks only if both stall.
- Processes tuples as they arrive from sources.
- Produces all answer tuples and no spurious duplicates.

SteMs:“State Modules”

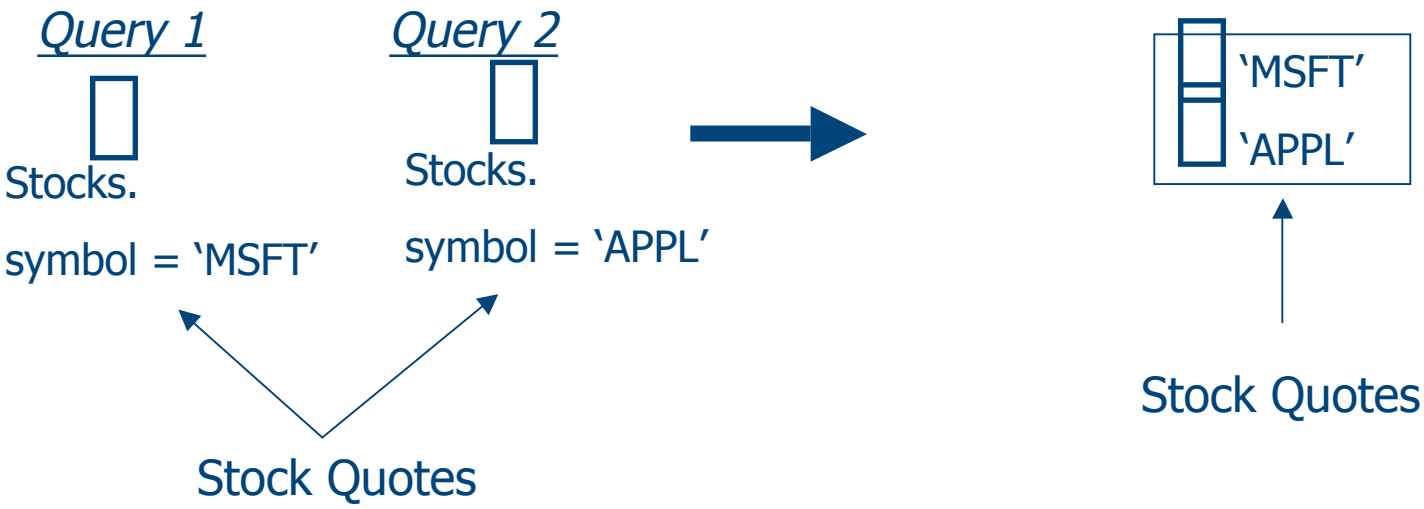
[Raman & Hellerstein ICDE 03]



- A generalization of the symmetric hash join (n-way)
- SteMs maintain intermediate state for multiple joins.
- Use Eddy to route tuples through the necessary modules.
  - Be sure to enforce “build then probe” processing.
- Note, can also maintain results of individual joins in SteMs.
- Lots of other interesting/useful properties

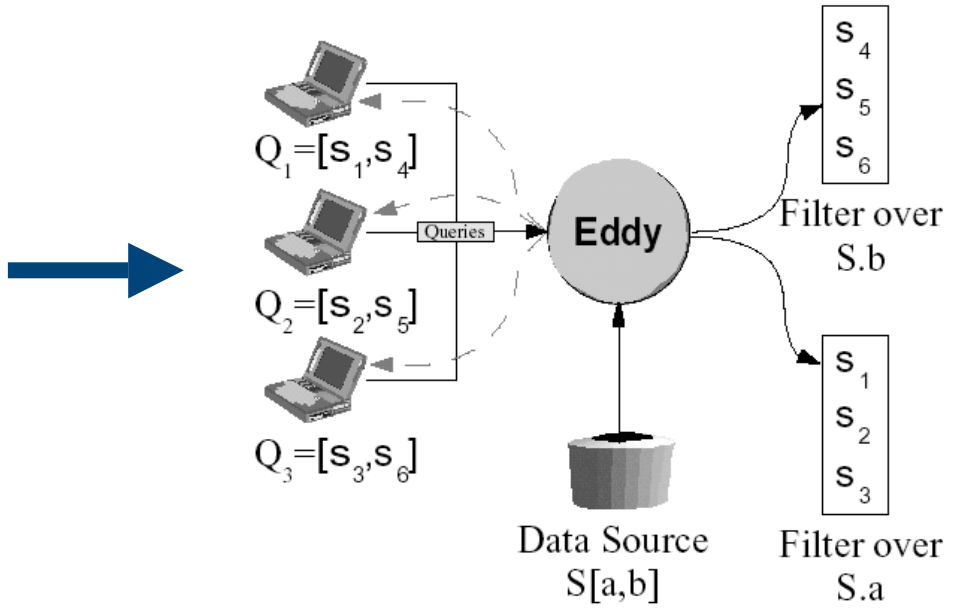
Shared Processing: CACQ
[Madden et al. SIGMOD 02]

- **In some cases there will be hundreds to thousands of queries over the same sources.**
- **Continuously Adaptive Continuous Queries**
  - combine operators from many queries to improve efficiency (i.e. share work).



Combining Queries in CACQ

Q1 = Select * From S where
S.a = s1 and S.b = s4;
Q2 = Select * From S where
S.a = s2 and S.b = s5;
Q3 = Select * From S where
S.a = s3 and S.b = s6;



- Eddy now is routing tuples for multiple queries simultaneously
  - Need additional *per tuple* state (a.k.a. "tuple lineage")
- Can also use SteMs to store query specifications.
  - Need a good predicate index if many queries.
- CACQ leverages Eddies for adaptive CQ processing
  - Results show advantages over static approaches.

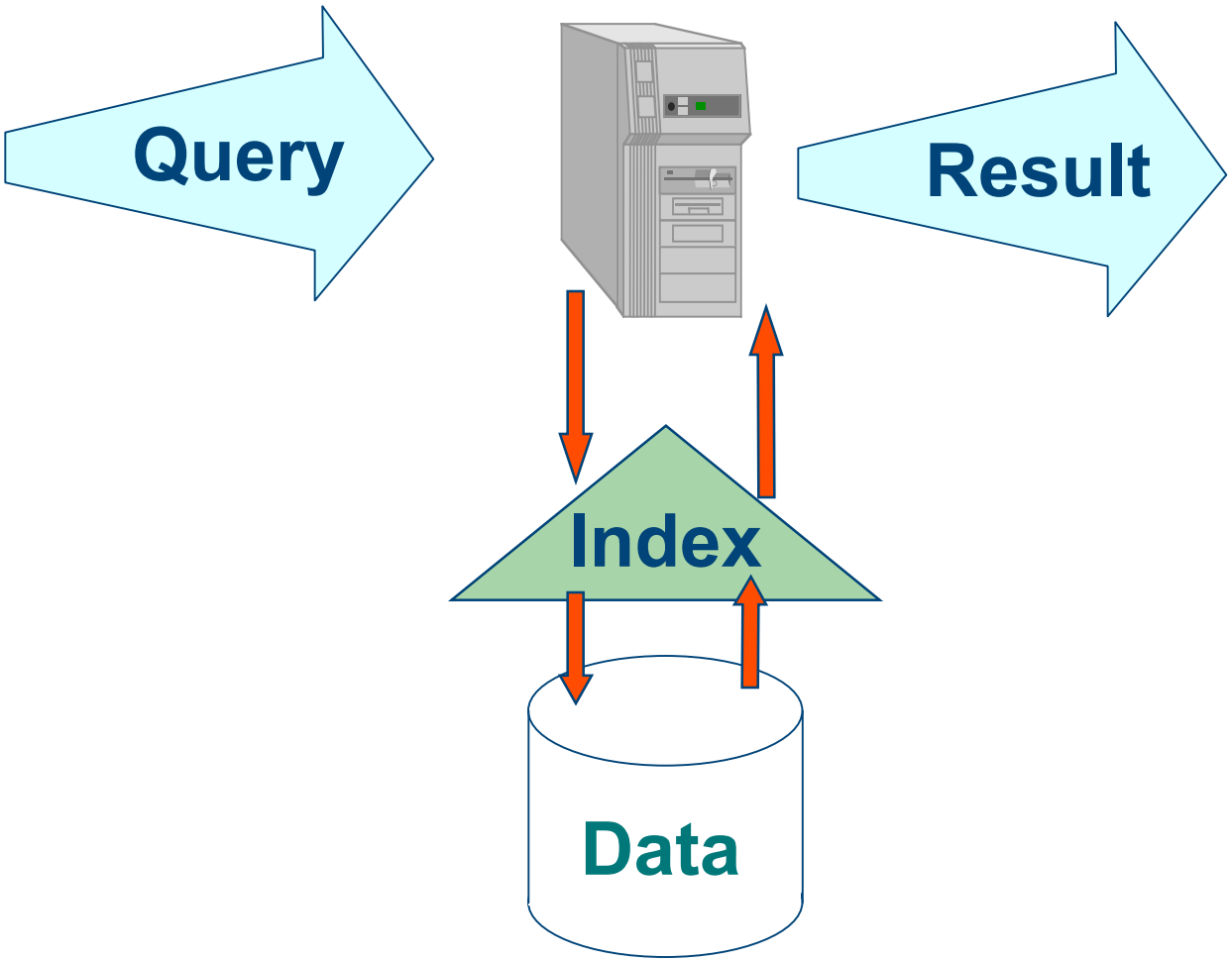
PSoup [Chandrasekaran & Franklin VLDB 02]

CQ systems tend to be "filters"

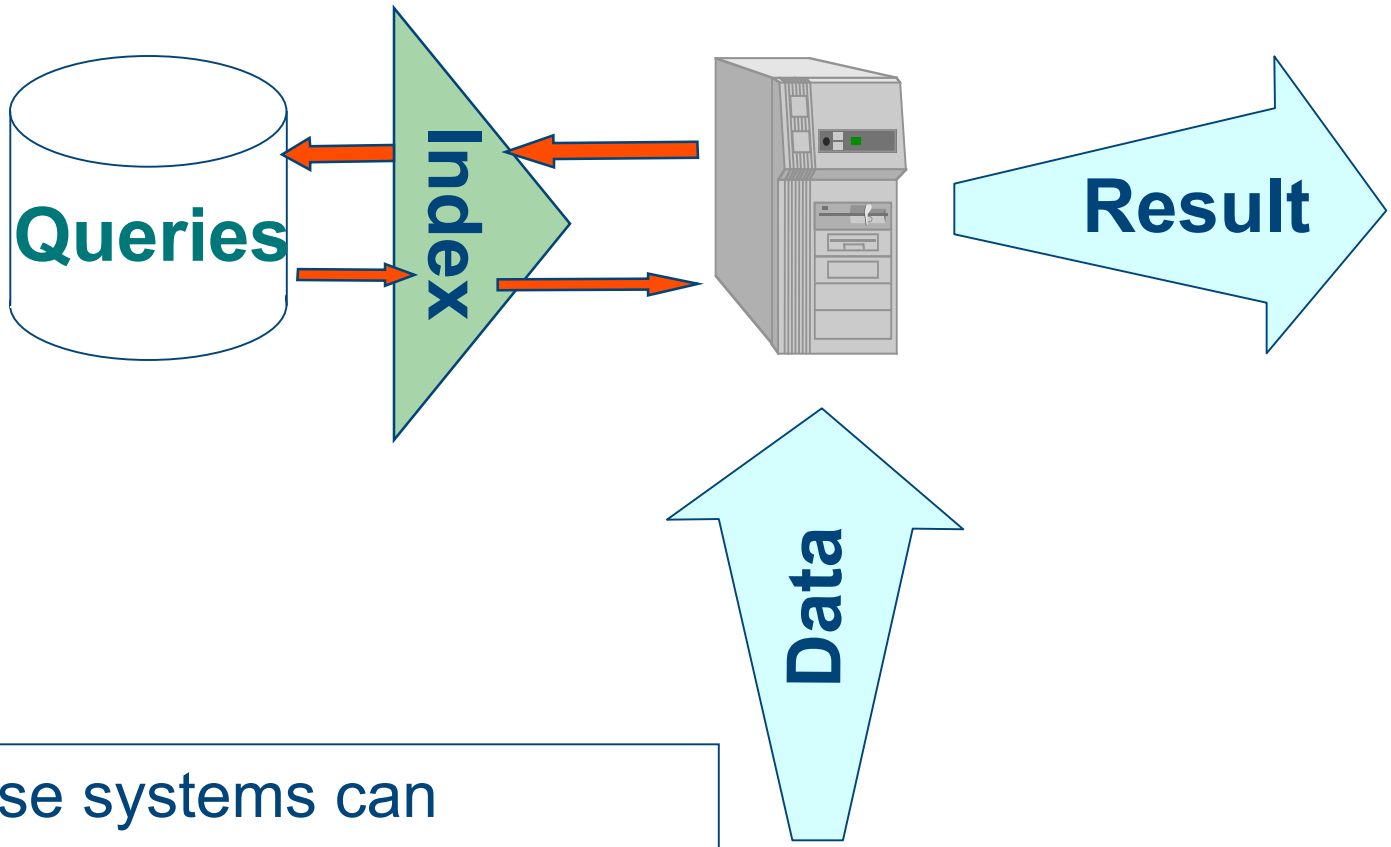
- **No historical data.**
  - Would like "new" queries to access "old" data.
- **Answers continuously returned as new data streams into the system.**
  - Such continuous answers are often:
    - Infeasible – intermittent connectivity and "Data Recharging" profiles.
    - Inefficient – often user does not want to be continuously interrupted.

- **Logical (?) conclusion:**
  - ***Treat queries and data as duals.***

A Traditional Database System

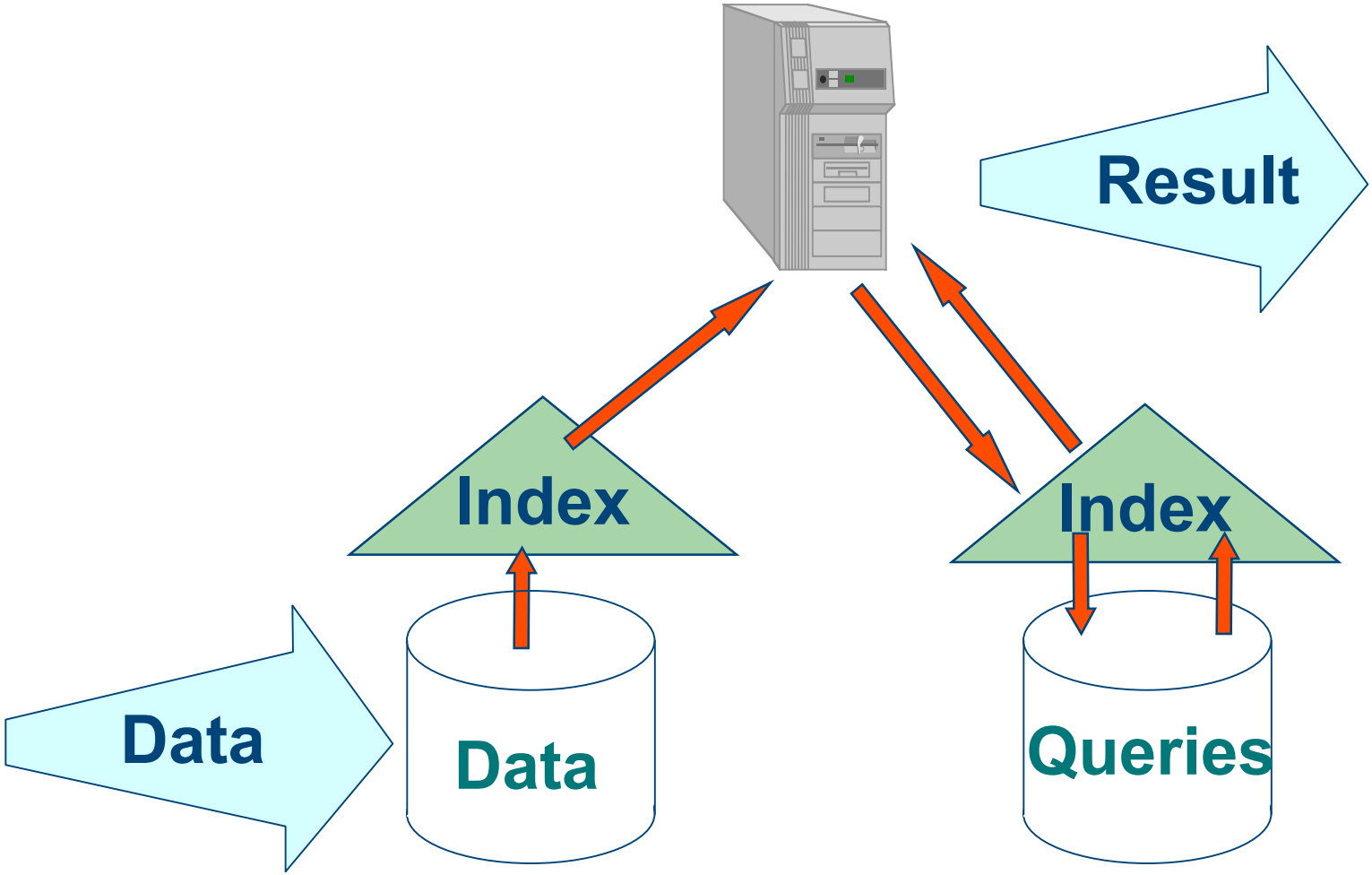


Publish-Subscribe/CQ/Filtering Systems

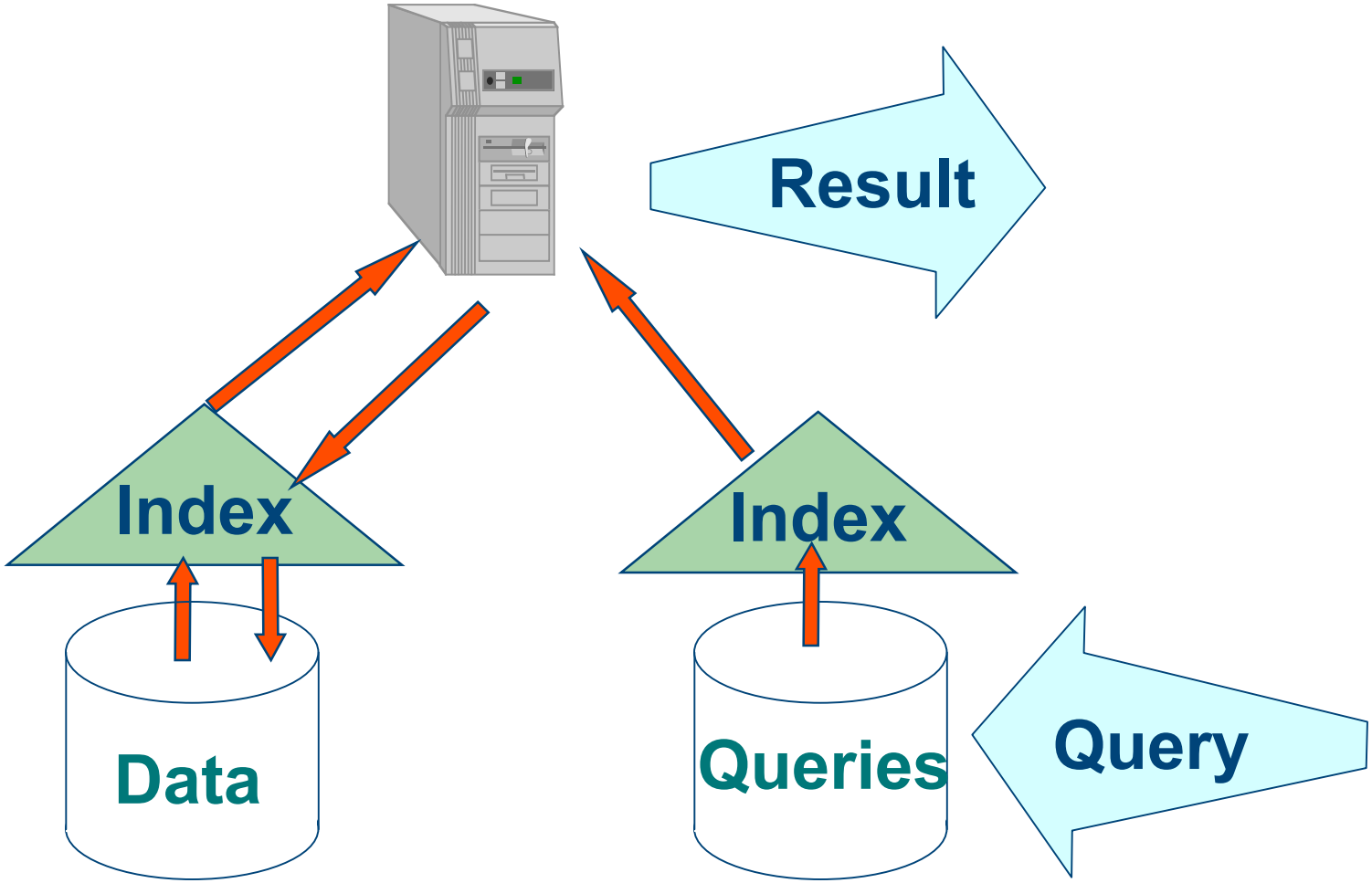


These systems can
be looked at as database
systems turned upsidedown.

PSoup: Query & Data Duality



PSoup: Query & Data Duality



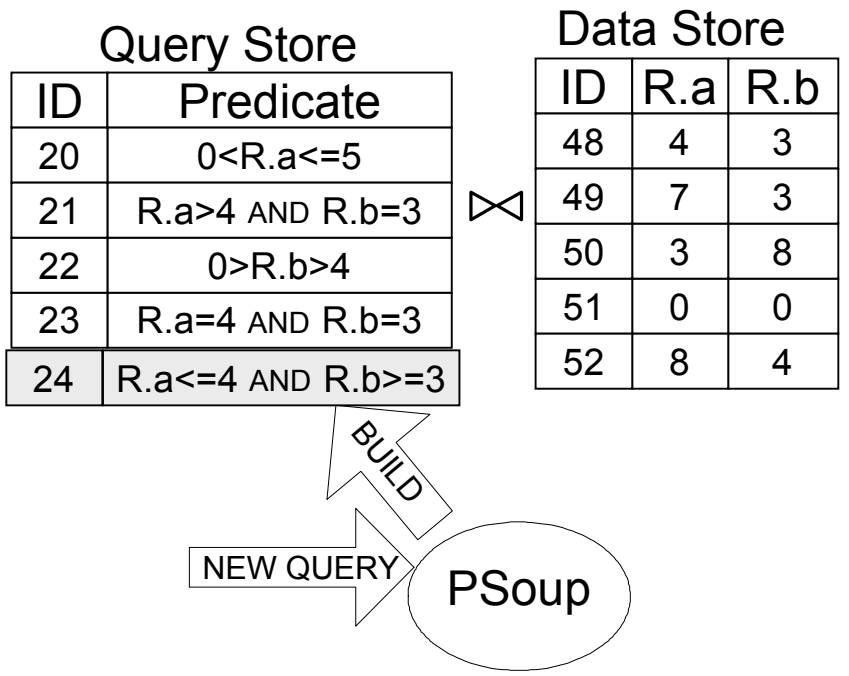
PSoup (cont.)

- **Query processing is treated as an *n-way symmetric join* between data tuples and query specifications.**

- **PSoup model:**

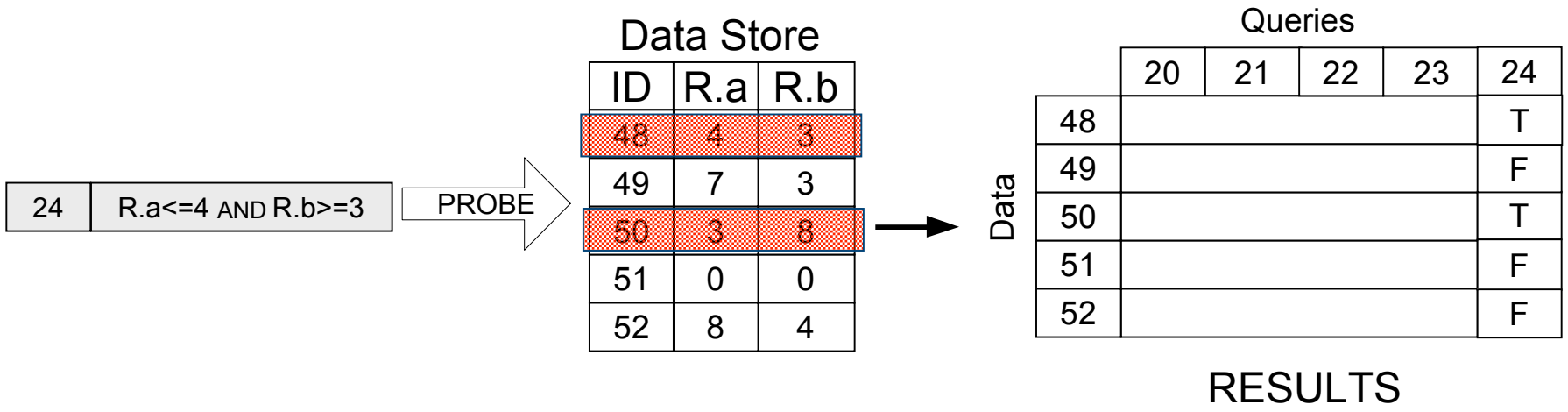
  repeated invocation of standing queries over windows of streaming data.

PSoup: New Select Query Arrival

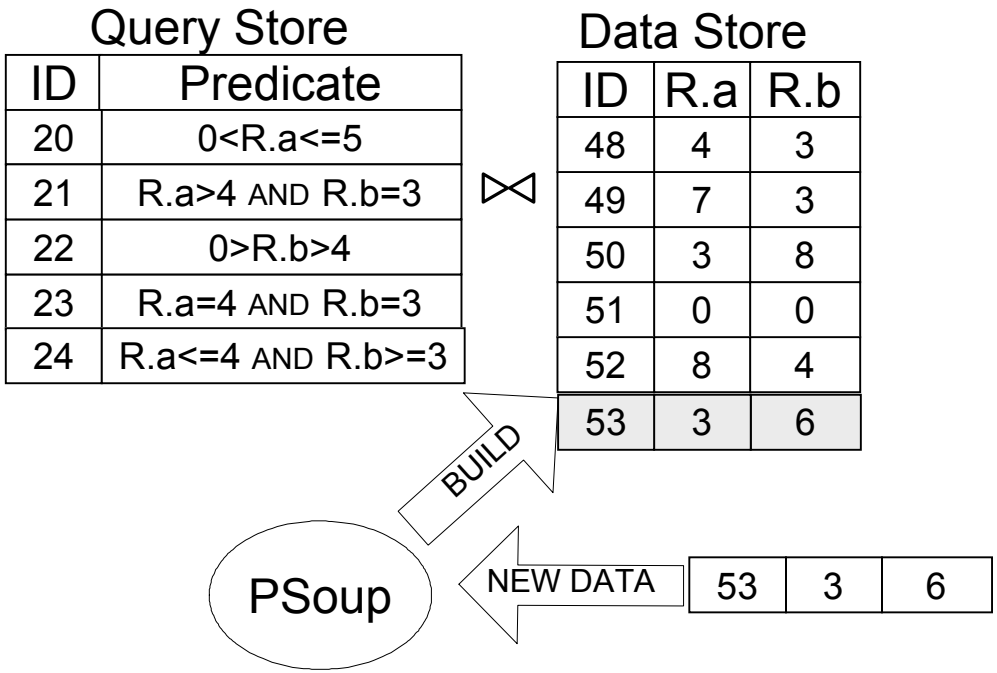



```
SELECT *
FROM R
WHERE R.a <= 4
  AND R.b >=3
BEGIN (NOW – 600)
END (NOW)
```

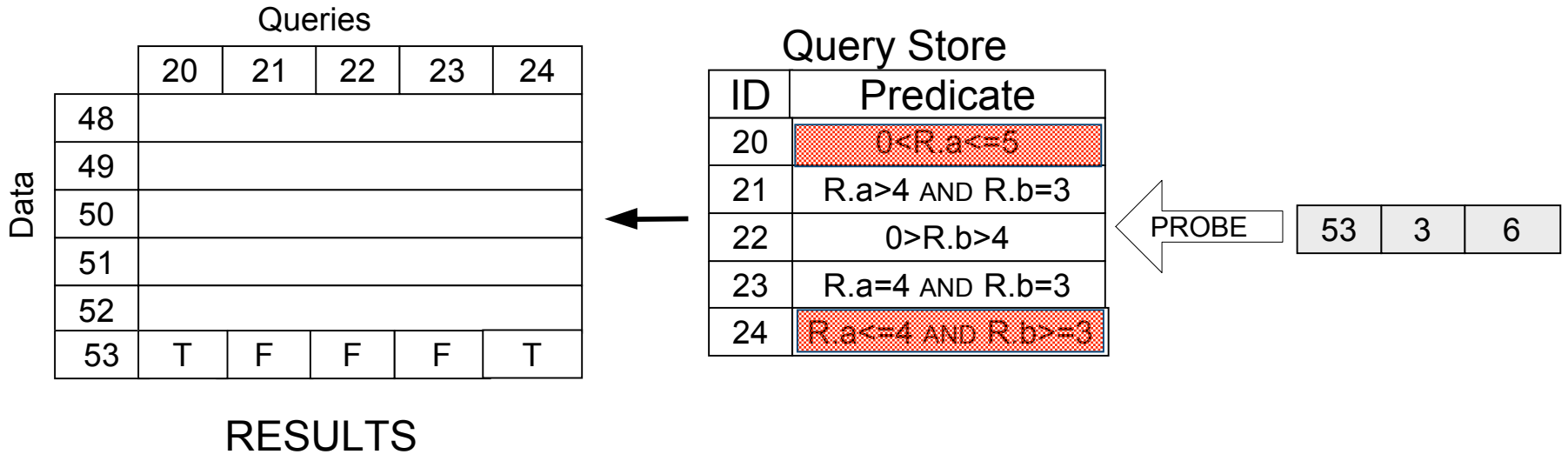
New Selection Spec (continued)



Selection – new data



Selection – new data (cont.)



Outline

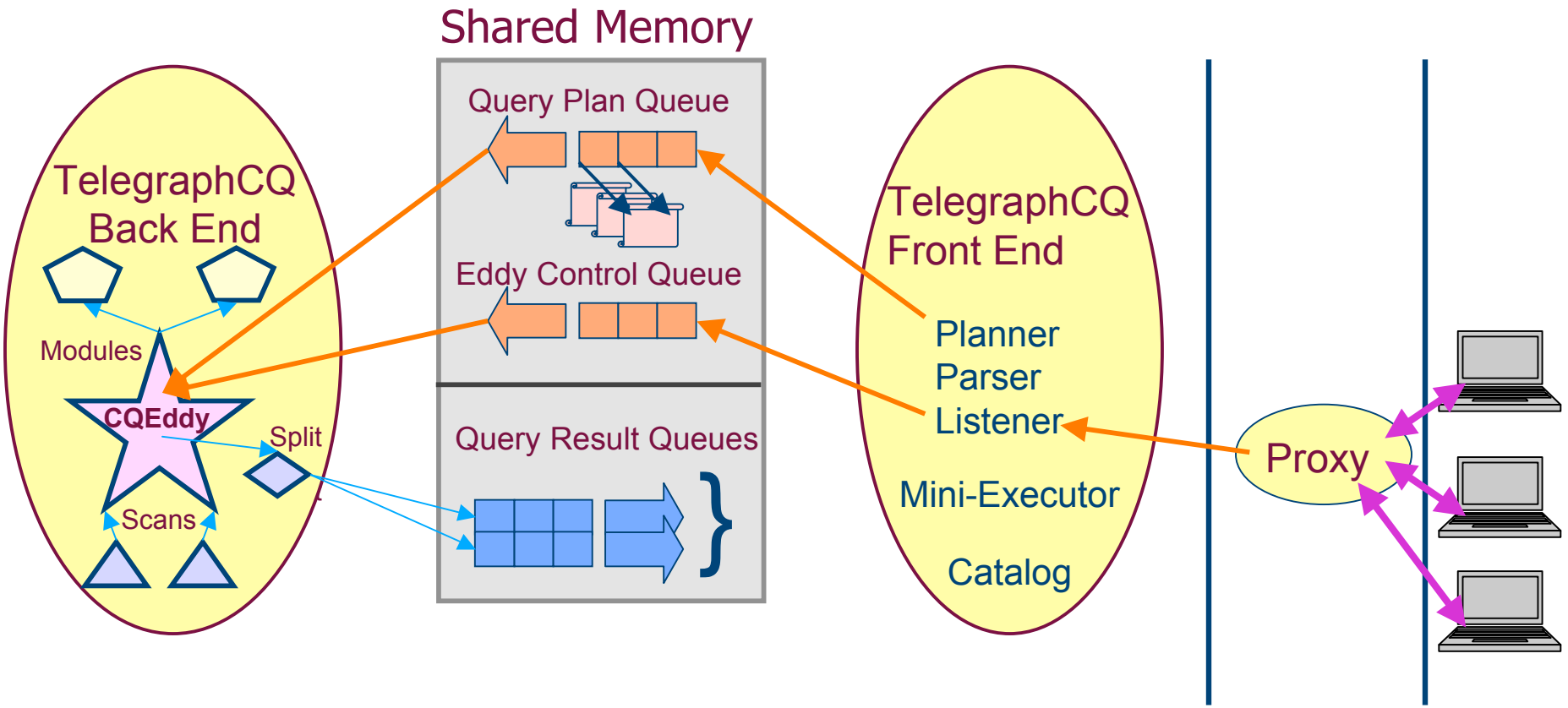
- Motivation and context
- Telegraph: basic technology
- The new TelegraphCQ system
- Stream semantics and language Issues
- Conclusions

TelegraphCQ - Shared CQ Processing [CIDR 2003]

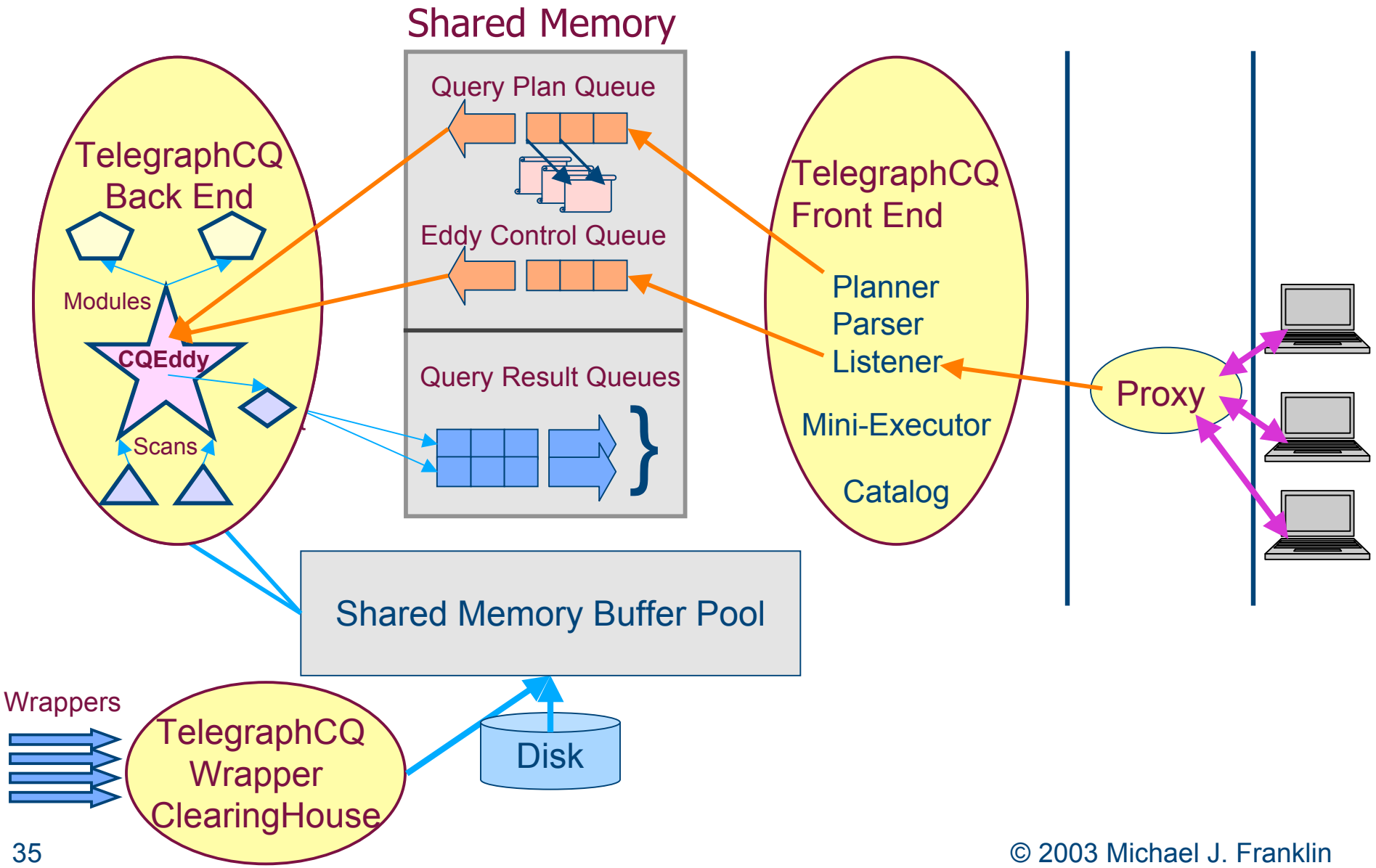
- **Our first Prototype(s) had issues:**
  - Much ado about Java
    - Java tools not great
    - We're in the business of moving tuples around
      - Fighting the memory manager at every step
  - Compiler Option per Thesis Topic

- **We chose to build TCQ inside of PostgreSQL**
  - Large thriving community of users
  - Useful features (UDFs,JDBC etc)
  - In-house experience
  - Proof that our flaky techniques could live in a real system.
- **Basic approach**
  - Components reused, e.g., semaphores, parser, planner
  - Components re-factored, e.g.,executor,access methods
  - LOC: 18K new (out of ~120K)

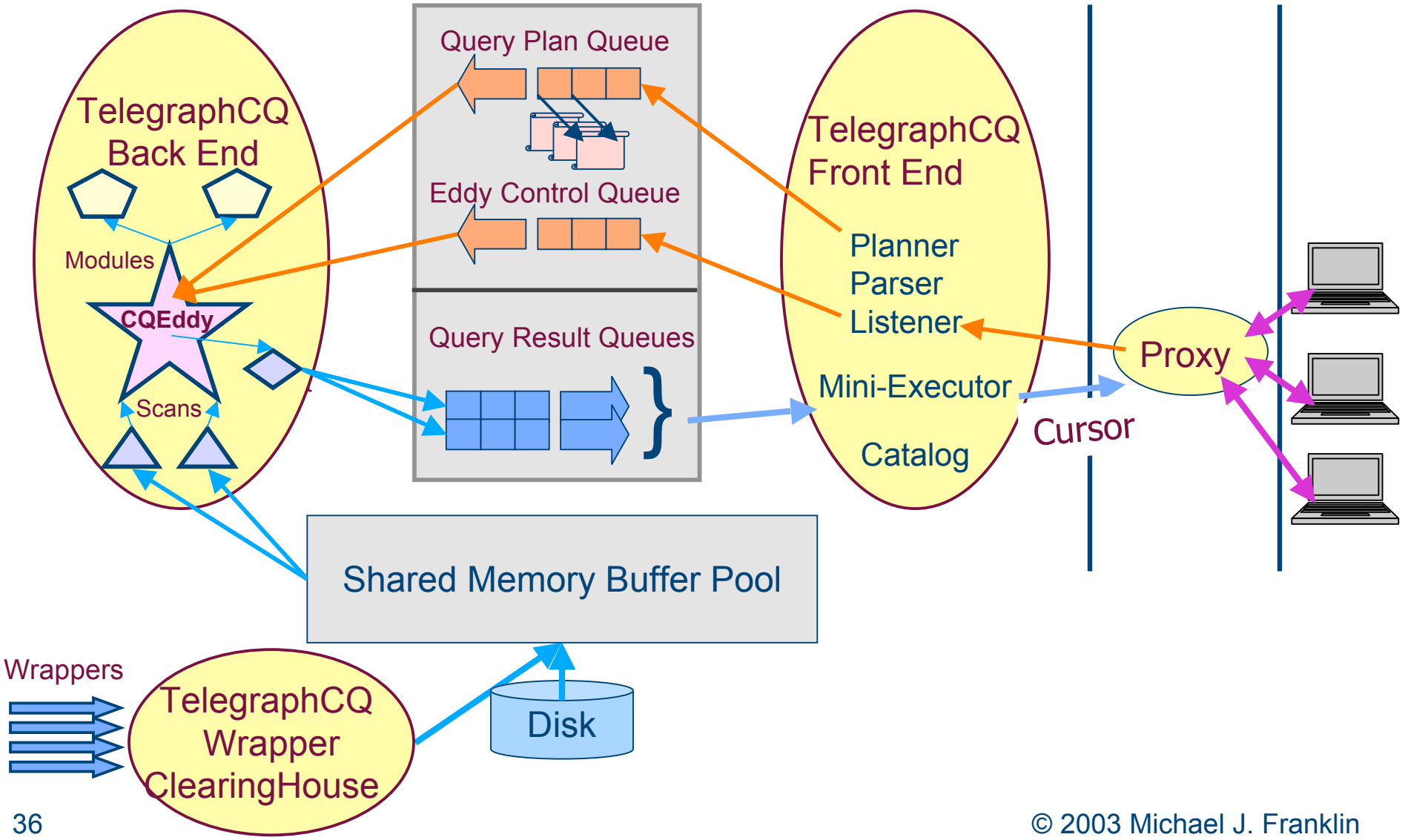
The TelegraphCQ Architecture



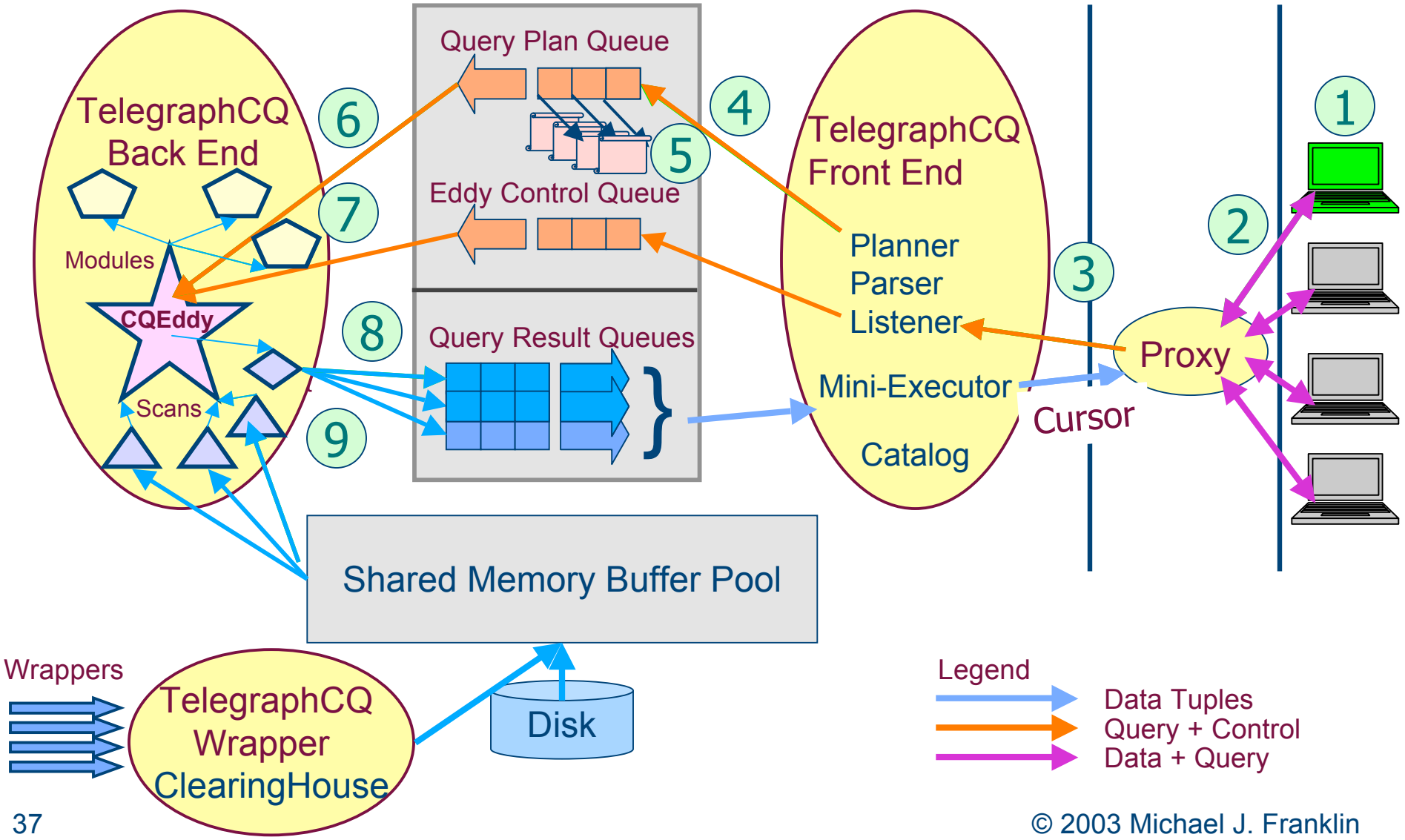
The TelegraphCQ Architecture



The TelegraphCQ Architecture



Dynamic Query Addition

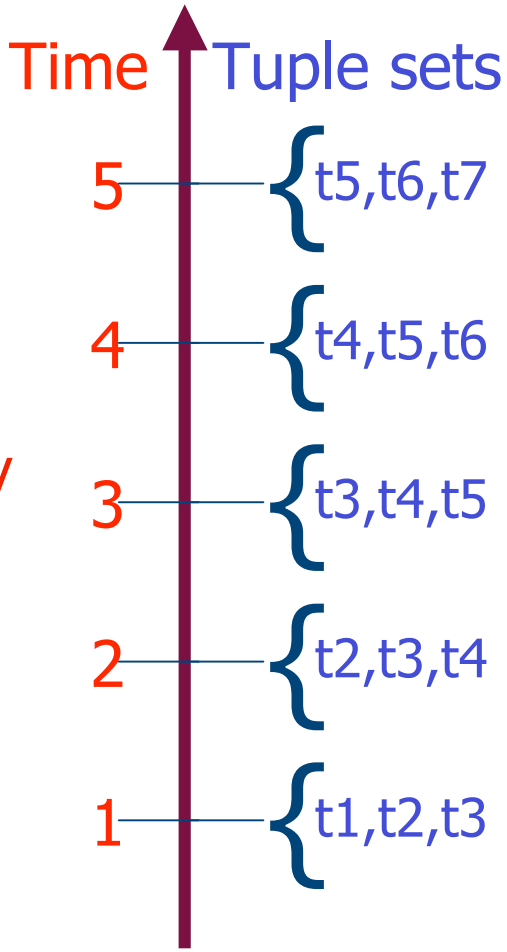


Outline

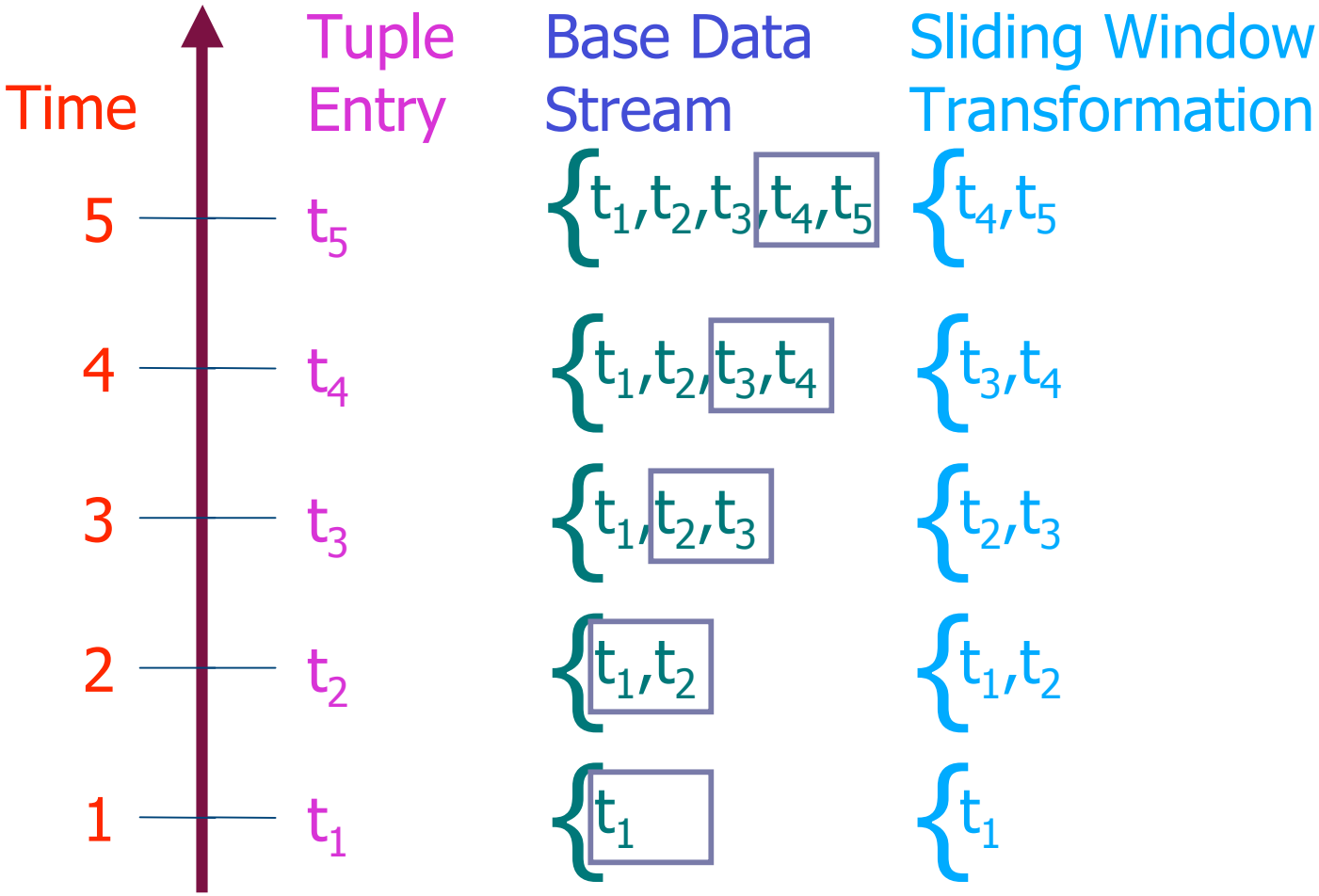
- Motivation and context
- Telegraph: basic technology
- The new TelegraphCQ system
- Stream semantics and language Issues
- Conclusions

Semantics of data streams

- **Different notions of data streams**
  - Ordered sequence of tuples
  - Bag of tuple/timestamp pairs **[STREAM]**
  - Mapping from time to sets of tuples
- **Data streams are unbounded**
  - Windows: vital to restrict data for a query
- **A stream can be transformed by:**
  - Moving a window across it
  - A window can be moved by
    - Shifting its extremities
    - Changing its size



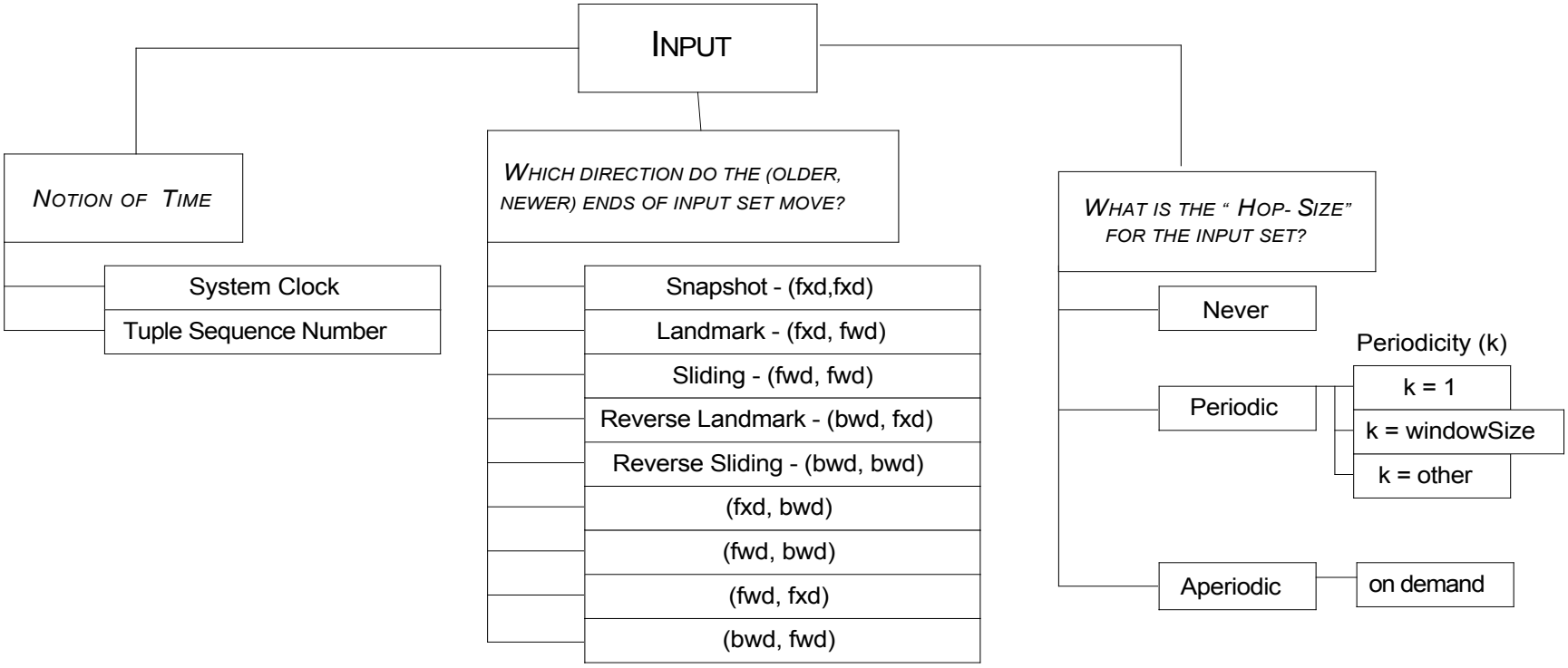
An example



The StreaQuel Language

- **An extension of SQL**
- **Operates exclusively on streams**
- **Is closed under streams**
- **Supports different ways to "create" streams**
  - Infinite time-stamped tuple sequence
  - Traditional stable relations
- **Flexible windows: sliding, landmark, and more**
- **Supports logical and physical time**
- **When used with a cursor mechanism, allows clients to do their own window-based processing.**
- **Target language for TelegraphCQ**

Classification of windowed queries



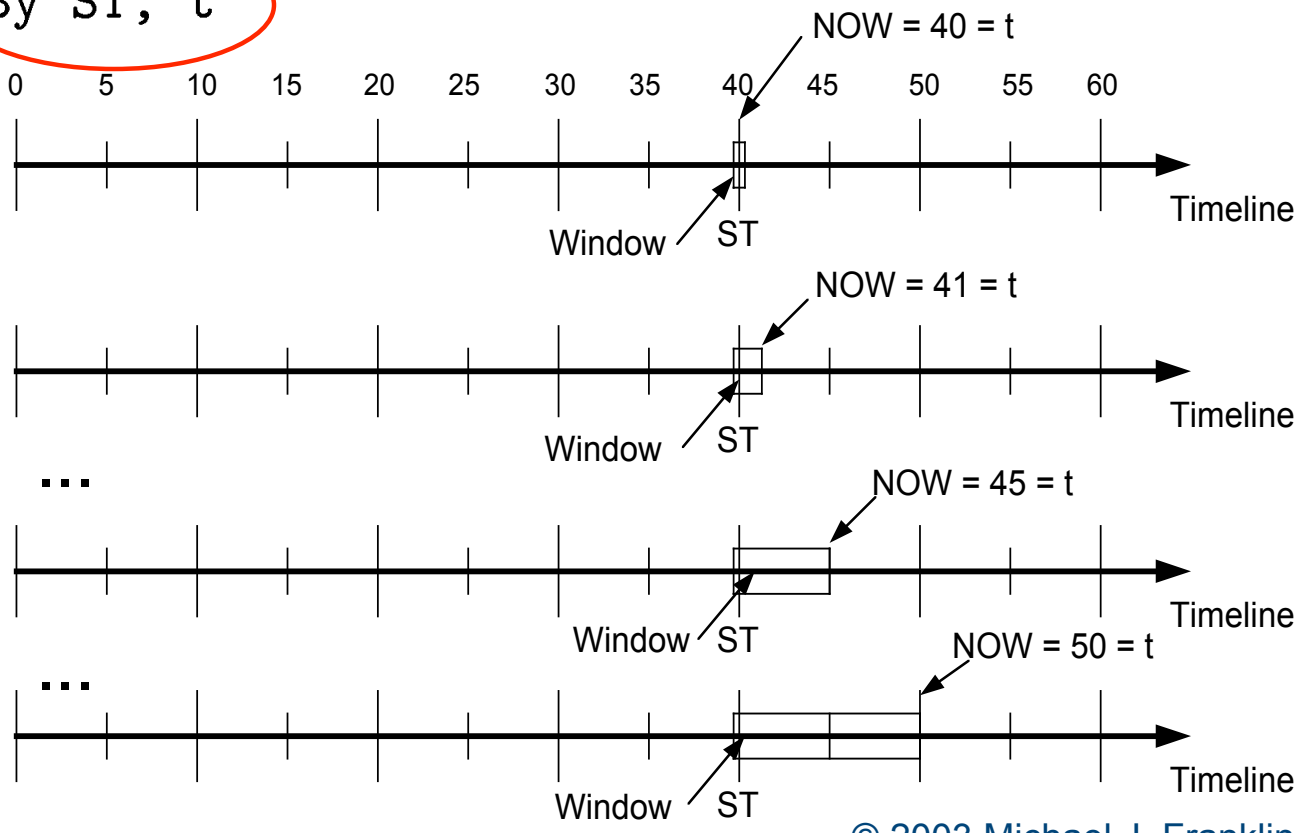
General Form of a StreaQuel Query

SELECT *projection_list*
FROM *from_list*
WHERE *selection_and_join_predicates*
ORDEREDBY
TRANSFORM…TO
WINDOW…BY

- **Windows can be applied to individual streams**
- **Window movement is expressed using a "for loop construct in the "transform" clause**
- **We're not completely happy with our syntax at this point.**

Example – Landmark query

```
Transform
    Stream1For(t = ST; t < ST + 10; t++)
            To Stream1(t)
Window
    Stream1 By ST, t
```



Outline

- Motivation and context
- Telegraph: basic technology
- The new TelegraphCQ system
- Stream semantics and language Issues
- Conclusions

Current Status - TelegraphCQ

- **What's running**
  - Shared joins with windows and aggregates
  - Archived/unarchived streams
- **PostgreSQL: Helped much more than expected**
  - Re-used a lot of code:
    - Expression evaluator, semaphores, parser, planner
  - Re-factored a fair bit:
    - Executor,Access Methods
    - Intermediate tuple formats
- **Obstacles that we faced**
  - No threading ☹
    - So far only wholly new code uses threads
    - Not yet been able to experiment with process model
    - Sharing – a few processes suffice anyway ?

What's next

- **Short term engineering stuff**
  - Ship an alpha release – aiming for end of this month
  - Begin performance evaluations
- **Ongoing research**
  - Interactions between storage and QoS
  - Cluster and distributed implementations
  - Adapting adaptivity
    - Move tuples in batches
    - Reduce frequency of plan changes
  - Egress operations - application connectivity
    - Pull vs push
  - Query overlap - multiple back ends

The TelegraphCQ Team

- **Students**
  - Sirish Chandrasekaran, Amol Deshpande, Sailesh Krishnamurthy, Sam Madden, Shankar Raman (emeritus), Fred Reiss, Mehul Shah
- **Faculty**
  - Mike Franklin and Joe Hellerstein
- **Professionals**
  - Owen Cooper and Wei Hong

(With help and input from the whole Berkeley database group.)

Conclusions

- **Dataflow and streaming are central to many emerging application areas.**
- **Adaptivity and Sharing are key requirements**
  - In TelegraphCQ sharing and adaptivity are

    Two sides of the same coin !
- **The PostgreSQL experience**
  - Saved tremendous time and effort
  - Enabled realistic system comparisons
  - Showed that our ideas are feasible in a real system
- **Ongoing work involves other streaming environments e.g. sensor networks and XML filtering.**

telegraph.cs.berkeley.edu