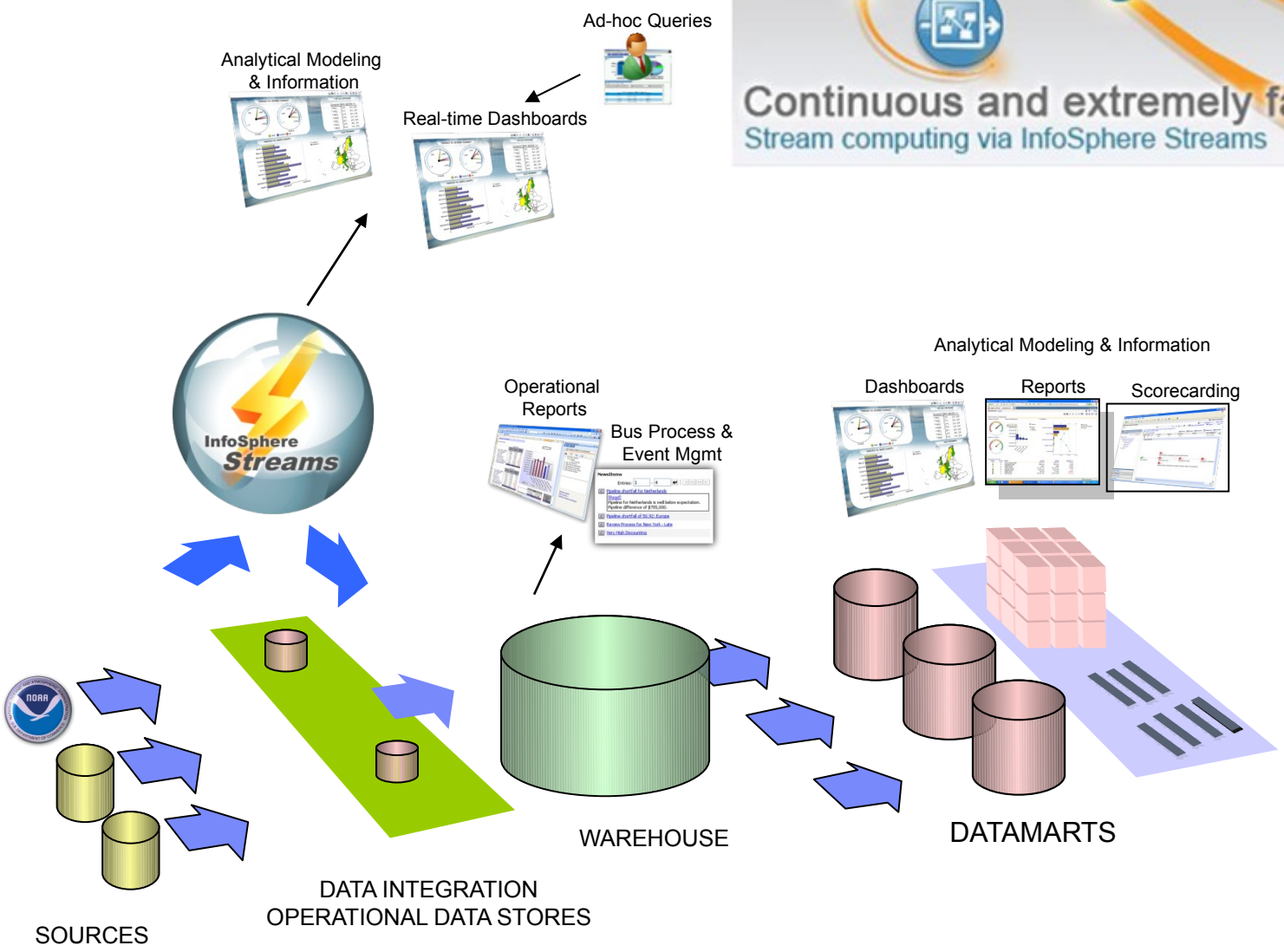
Knowledge Engineering for Planning-based Data-flow Composition

Mark D. Feblowitz, Anand Ranganathan, Anton V. Riabov, and Octavian Udrea
IBM T. J. Watson Research Center
Yorktown Heights, NY, USA

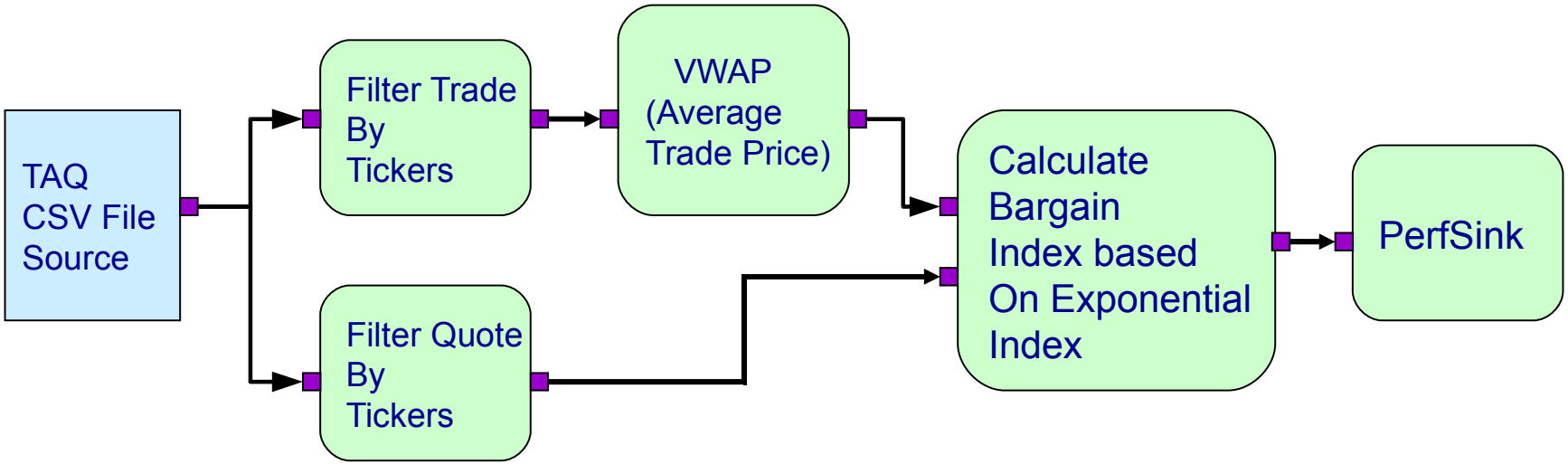
ICKEPS, 6/25/2012

Background: Distributed Stream Processing

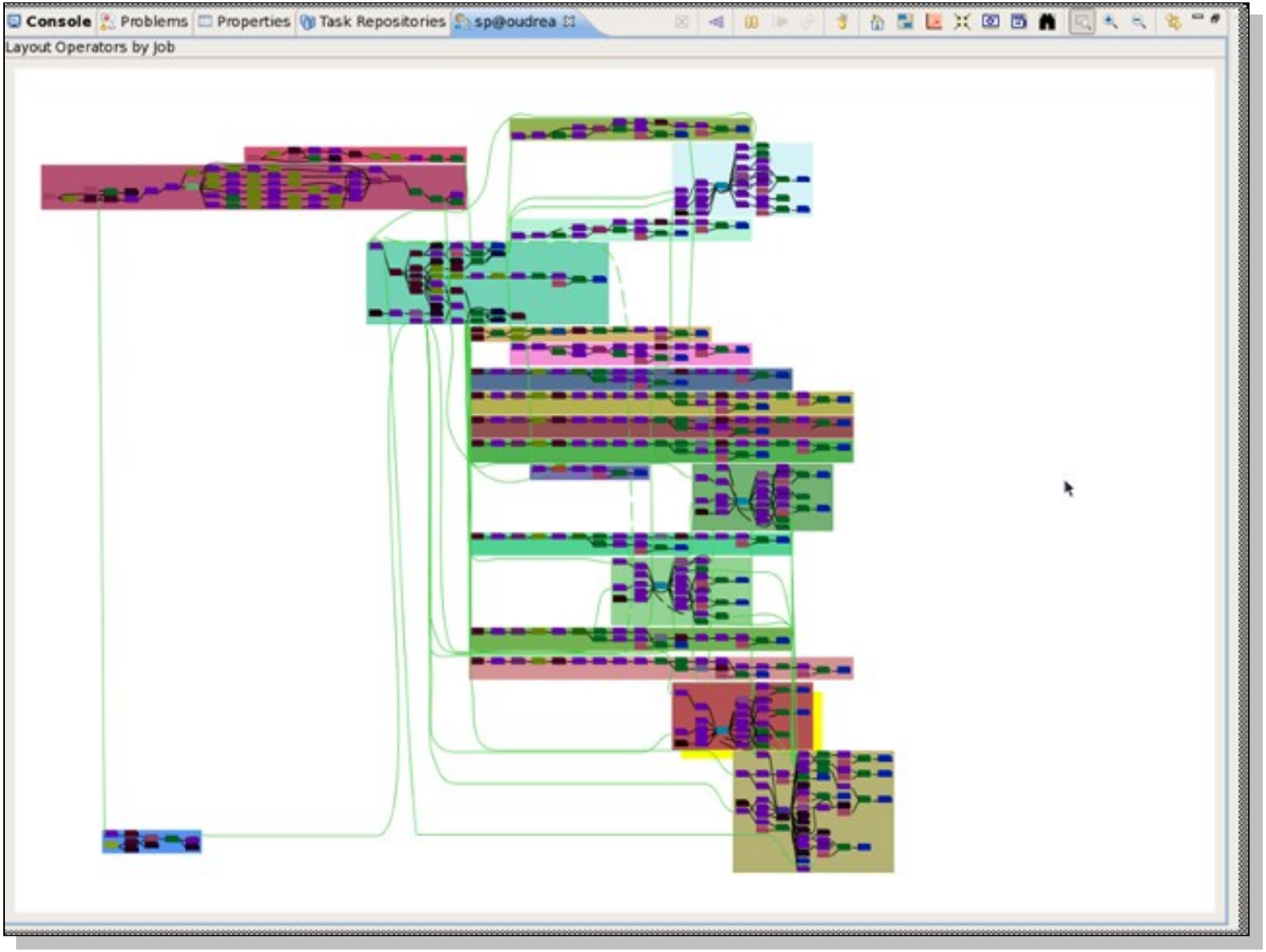
IBM



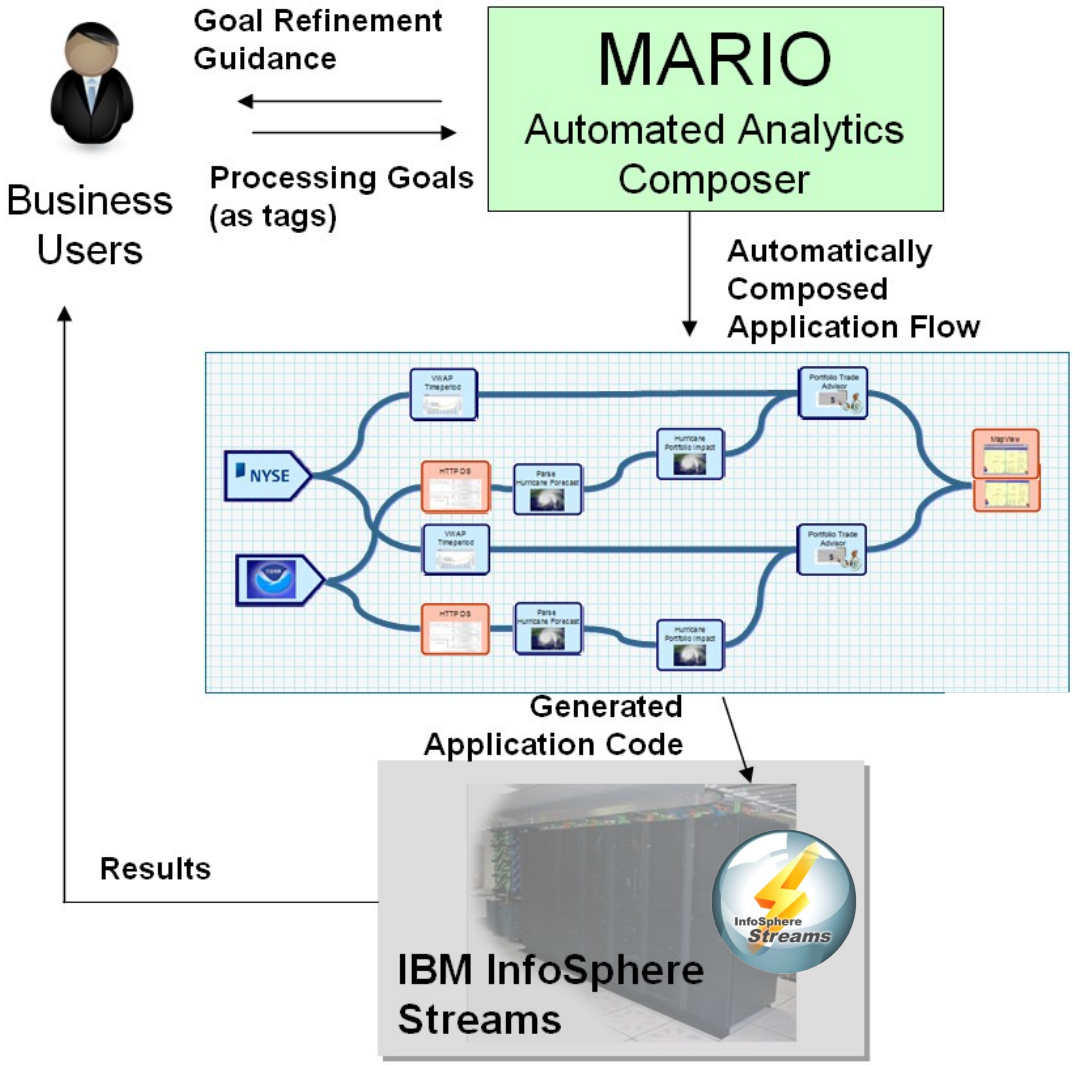
A Sample Stream Processing Application



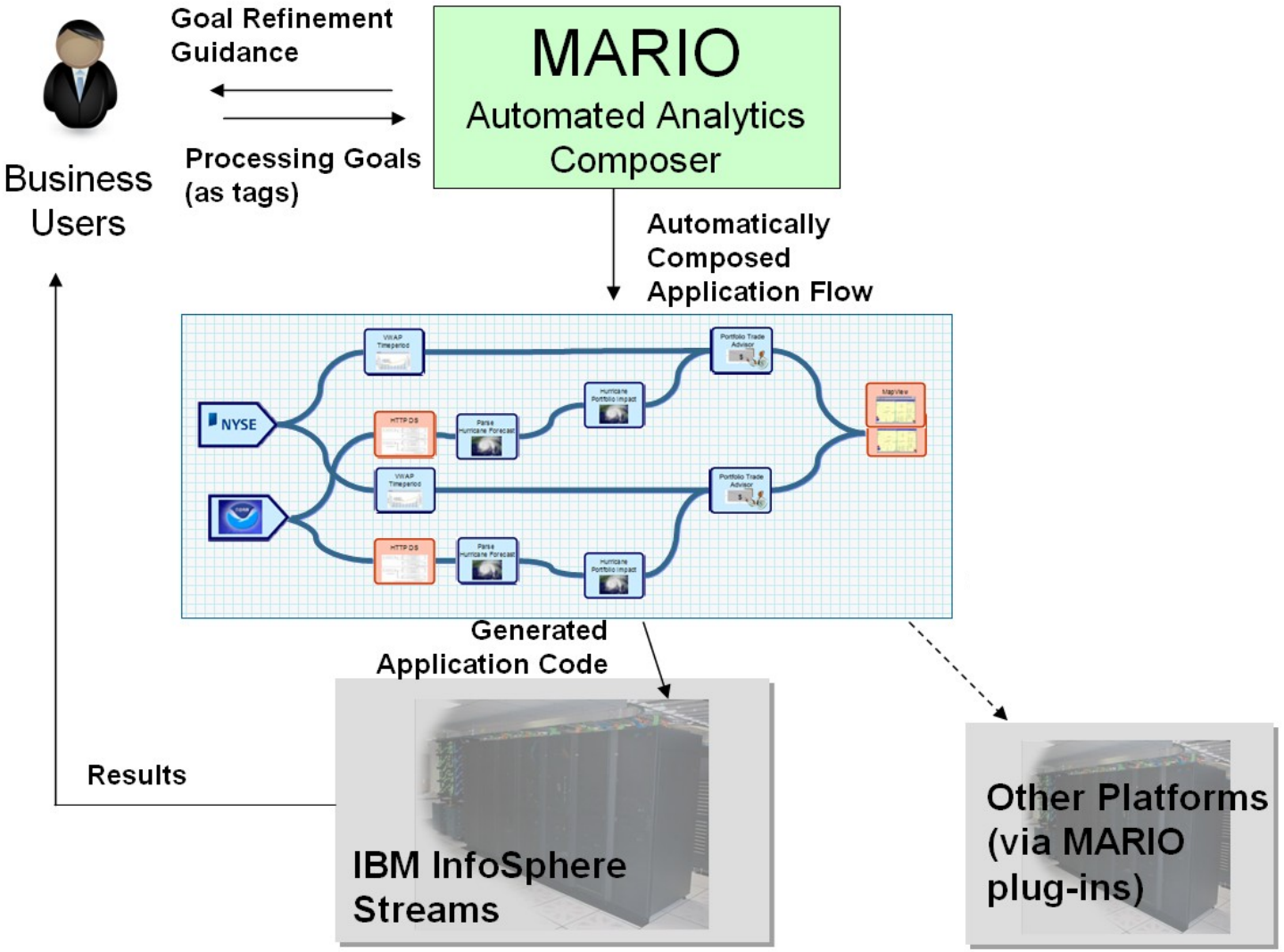
Stream Processing: A More Complex Example



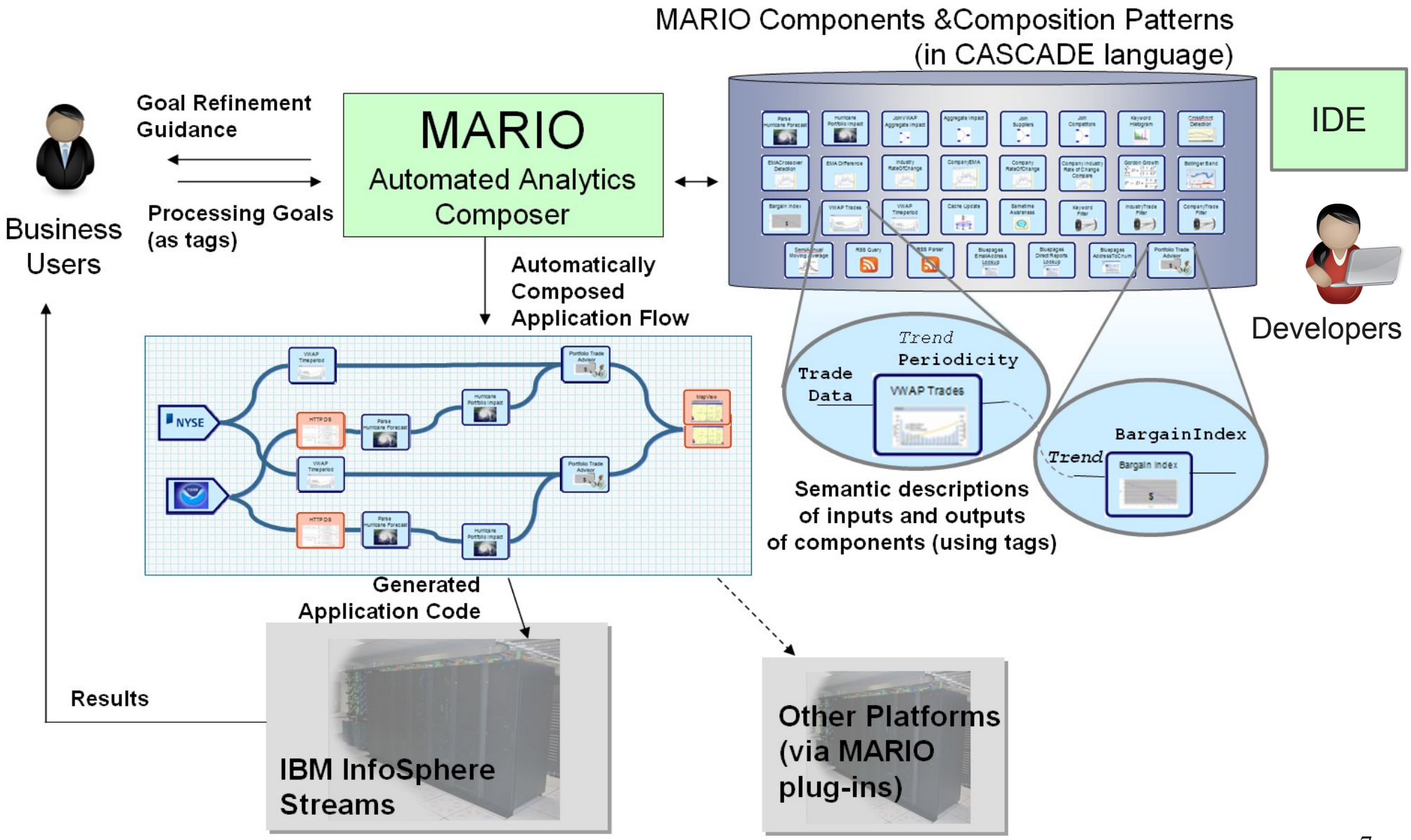
MARIO Planner and Cascade Language



MARIO Planner and Cascade Language



MARIO Planner and Cascade Language



Pilot Deployment

- Deployed by a customer, in a pilot deployment since 2009
  - Pilot includes Cascade IDE, MARIO, InfoSphere Streams
  - Developers use Eclipse IDE to describe components, composition patterns, etc
  - End-users compose, launch jobs and view results using MARIO Web-based interface

Related Work

- Mashups and Visual Programming (e.g., drag-and-drop) tools
  - For end-users, the interaction with MARIO is goal-centric – there is no flow editor in MARIO
  - Developers describe components and composition patterns in text, using Cascade language
- Web Service Composition (e.g., using planning)
  - Our approach can be used to compose web services in some scenarios, but there are differences from other approaches

Specifying Goals and Planning Domains

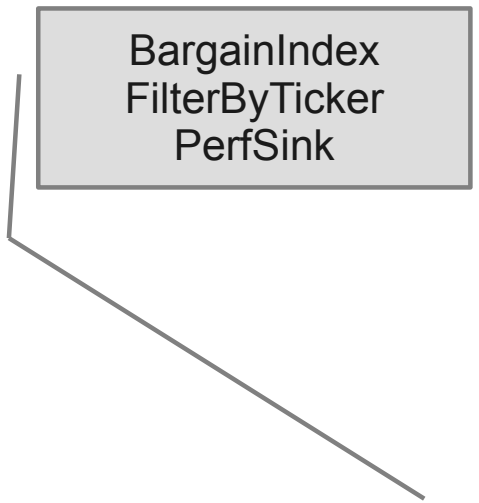
- In historical order, we have experimented with:
  - PDDL
  - SPPL [AAAI'05, ICAPS'06]
    - Exponentially faster performance compared to PDDL
  - OWL-based semantic annotations [ICWS'07]
    - Rich semantic descriptions
  - Tags and tag taxonomies [WWW'08, OOPSLA'08]
    - No OWL skills required to describe components and specify goals
  - Cascade [CIKM'09]
    - Modular structure, formally defined flow patterns

Specifying Goals and Planning Domains

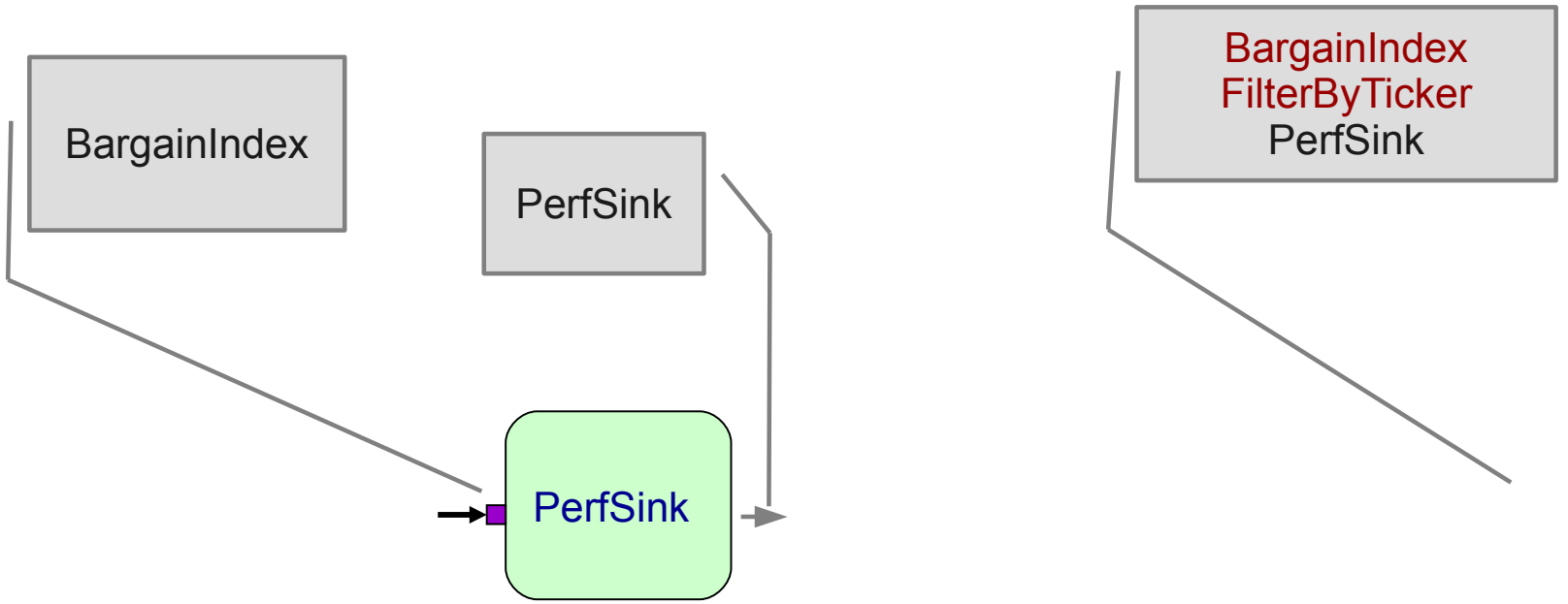
- Current implementation **includes**:
  - PDDL
  - **SPPL** [AAAI'05, ICAPS'06]
    - Exponentially faster performance compared to PDDL
  - OWL-based semantic annotations [ICWS'07]
    - Rich semantic descriptions
  - **Tags and tag taxonomies** [WWW'08, OOPSLA'08]
    - No OWL skills required to describe components and specify goals
  - **Cascade** [CIKM'09]
    - Modular structure, formally defined flow patterns

Example

Goal:

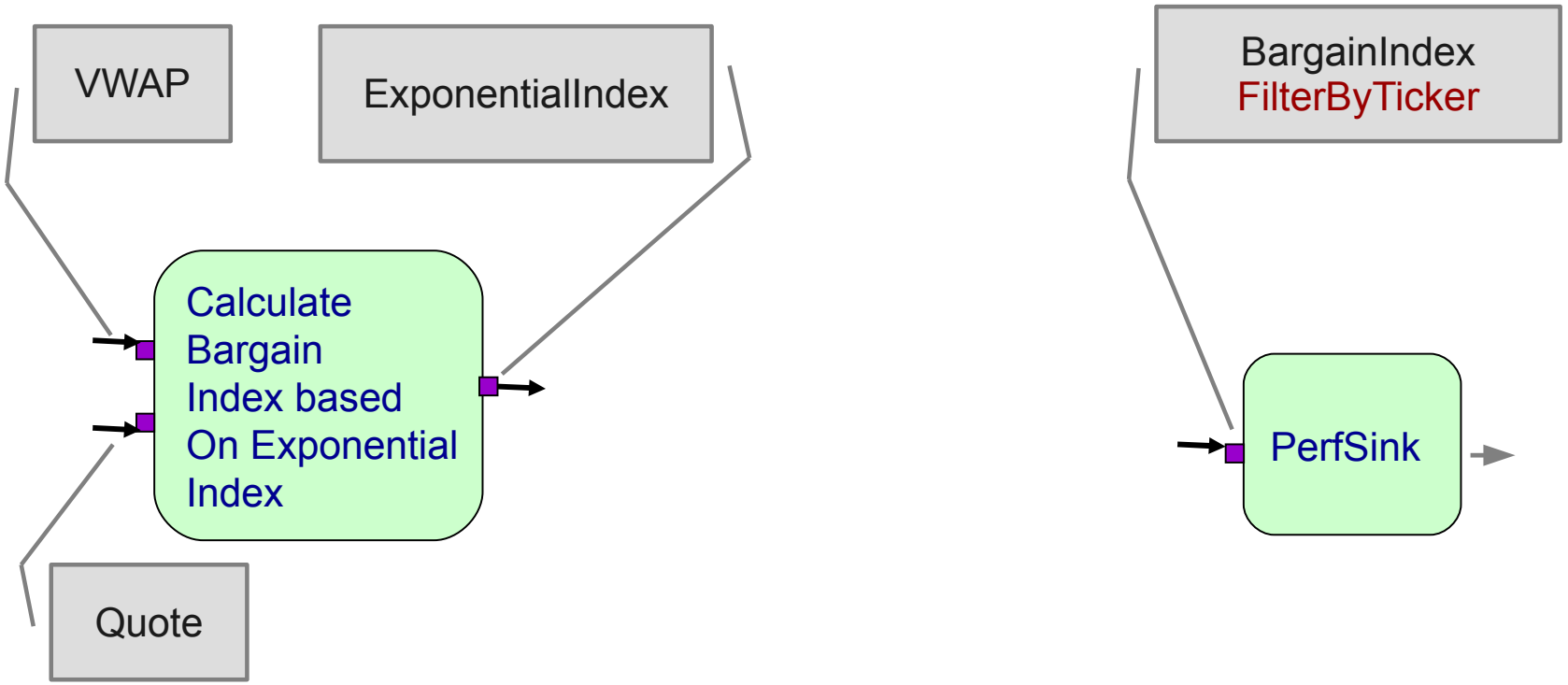


Example

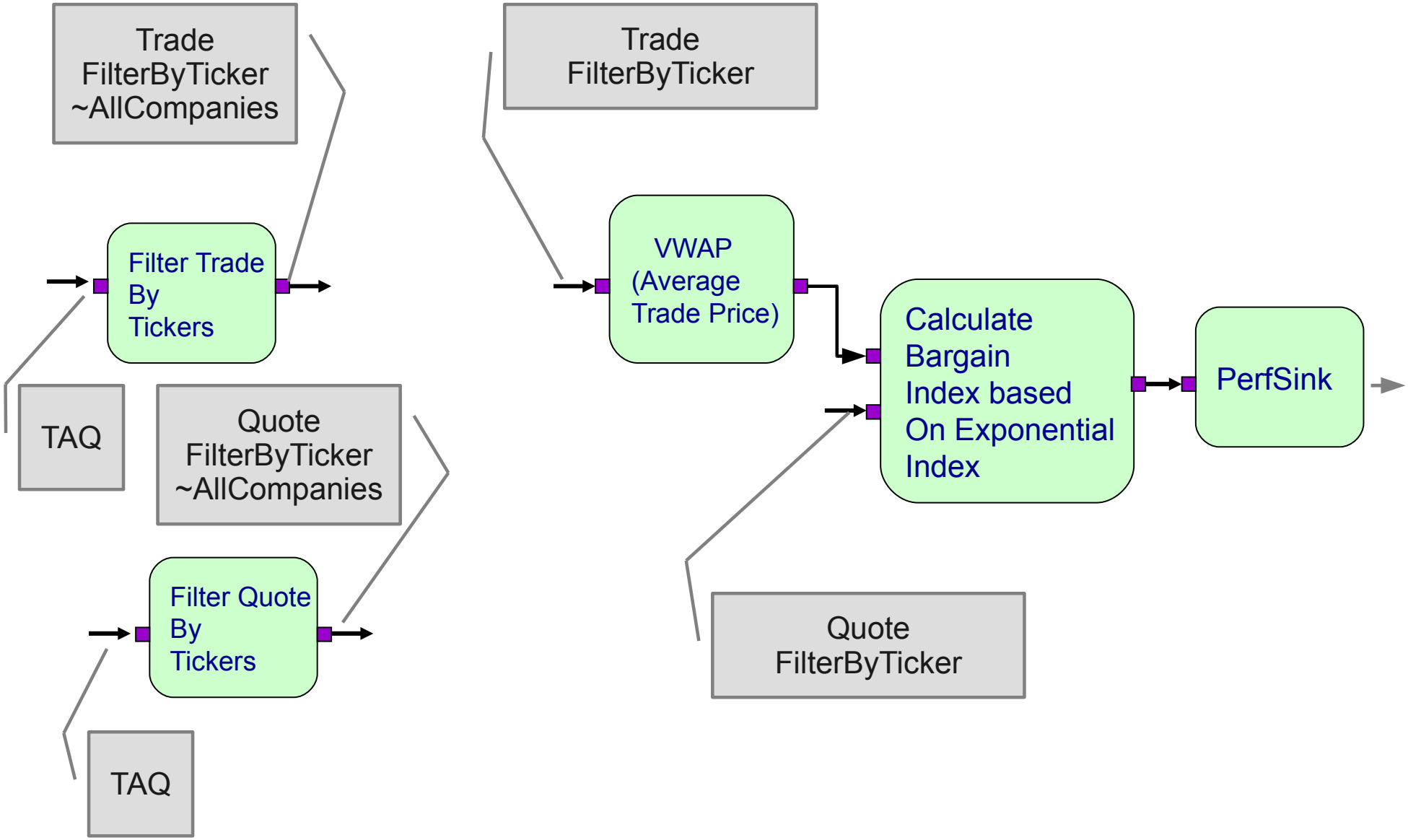

BargainIndex
PerfSink
PerfSink
BargainIndex
FilterByTicker
PerfSink

Example

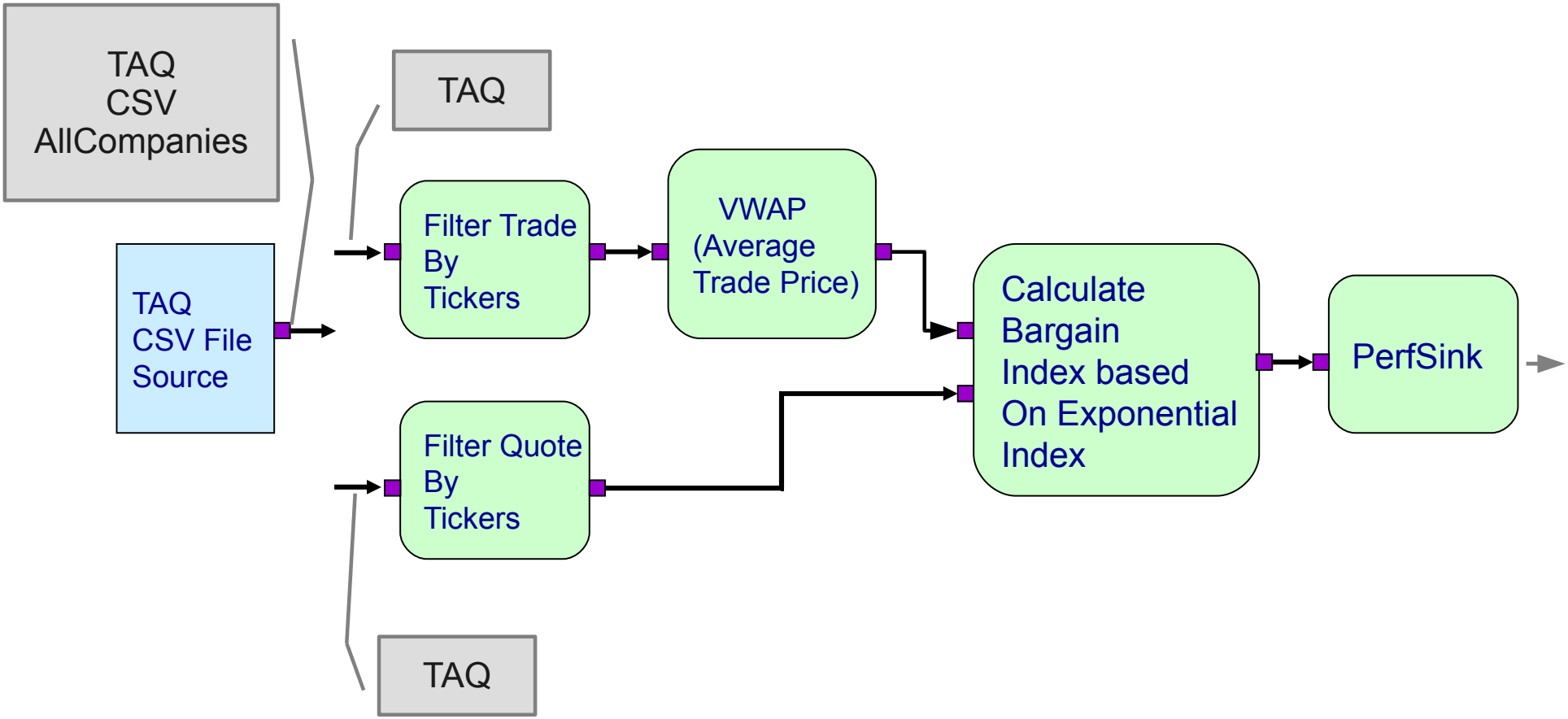
Taxonomy: ExponentialIndex → BargainIndex



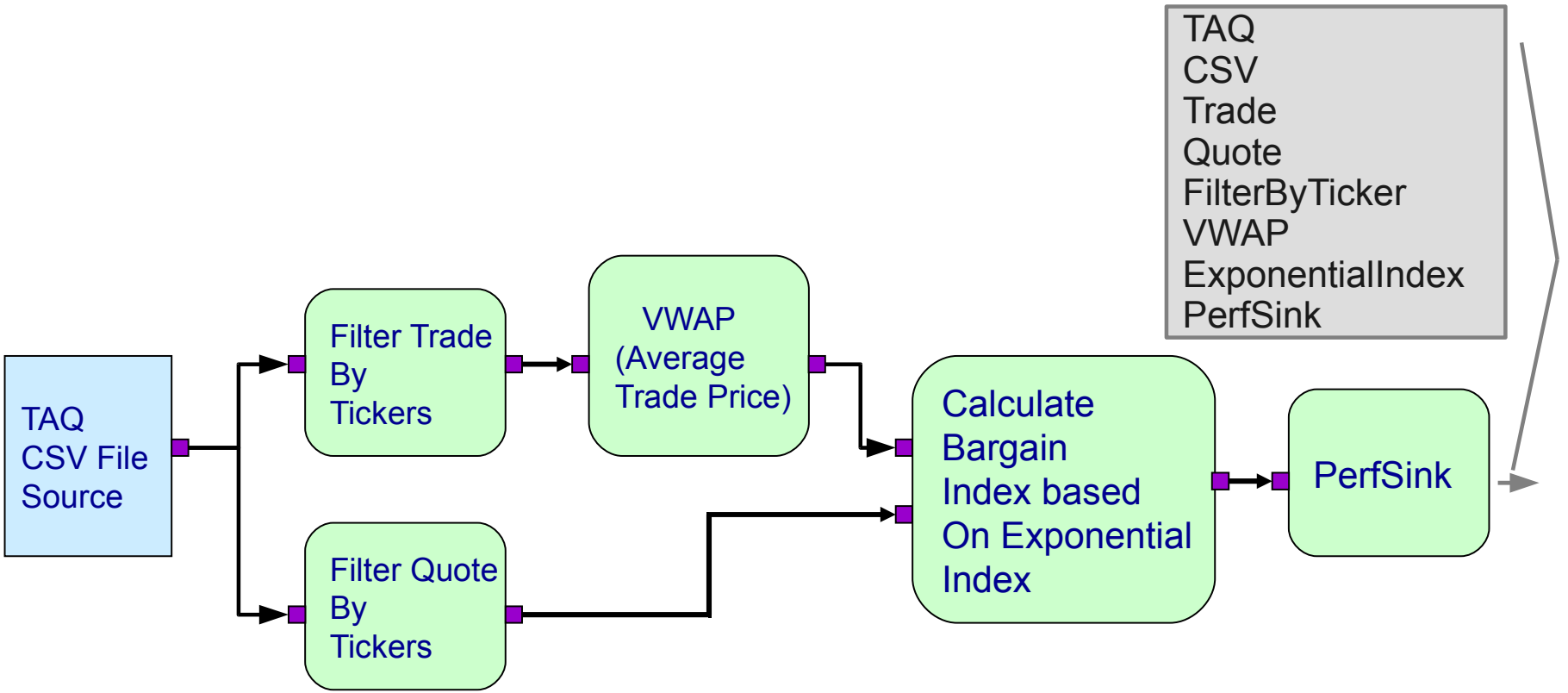
Example


Trade
FilterByTicker
~AllCompanies
Filter Trade
By
Tickers
TAQ
Quote
FilterByTicker
~AllCompanies
Filter Quote
By
Tickers
TAQ
Trade
FilterByTicker
VWAP
(Average
Trade Price)
Calculate
Bargain
Index based
On Exponential
Index
PerfSink
Quote
FilterByTicker

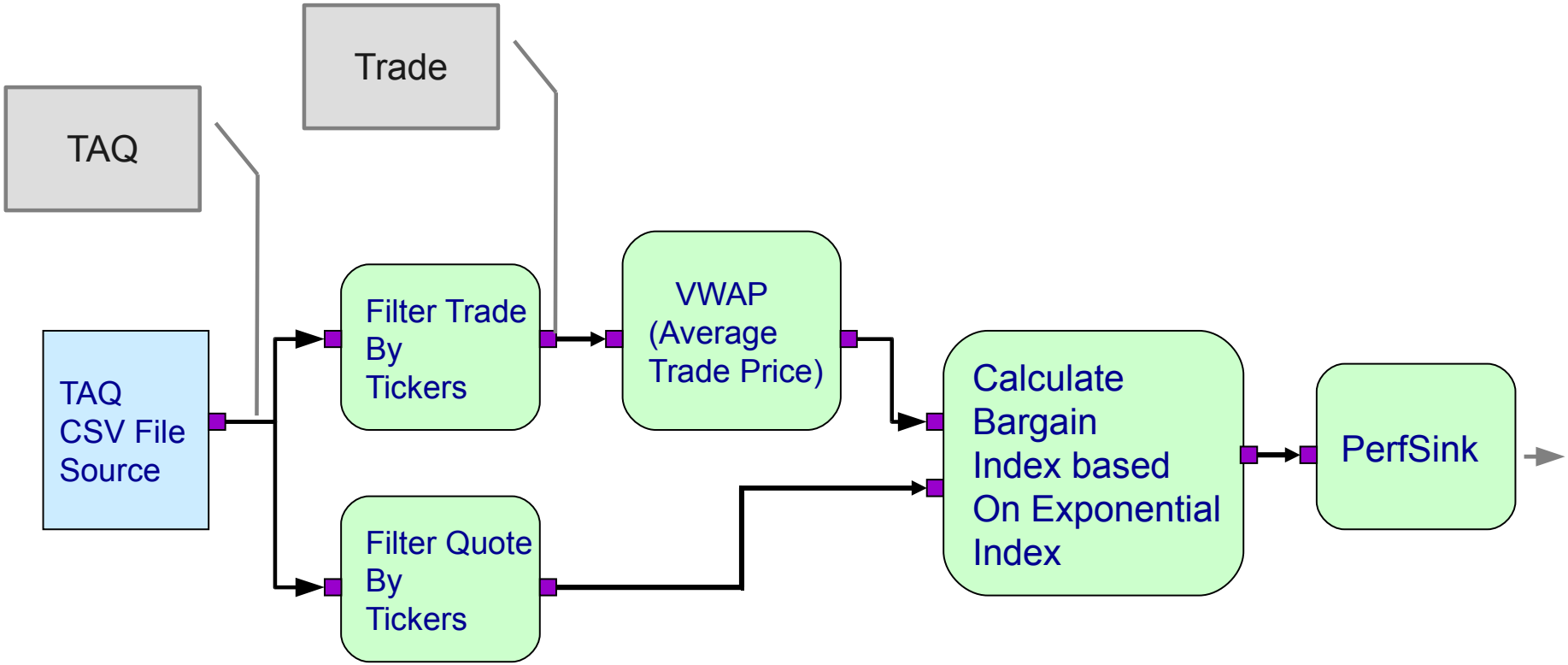
Example


TAQ
CSV
AllCompanies
TAQ
TAQ
CSV File
Source
Filter Trade
By
Tickers
VWAP
(Average
Trade Price)
Calculate
Bargain
Index based
On Exponential
Index
PerfSink
Filter Quote
By
Tickers
TAQ

Example

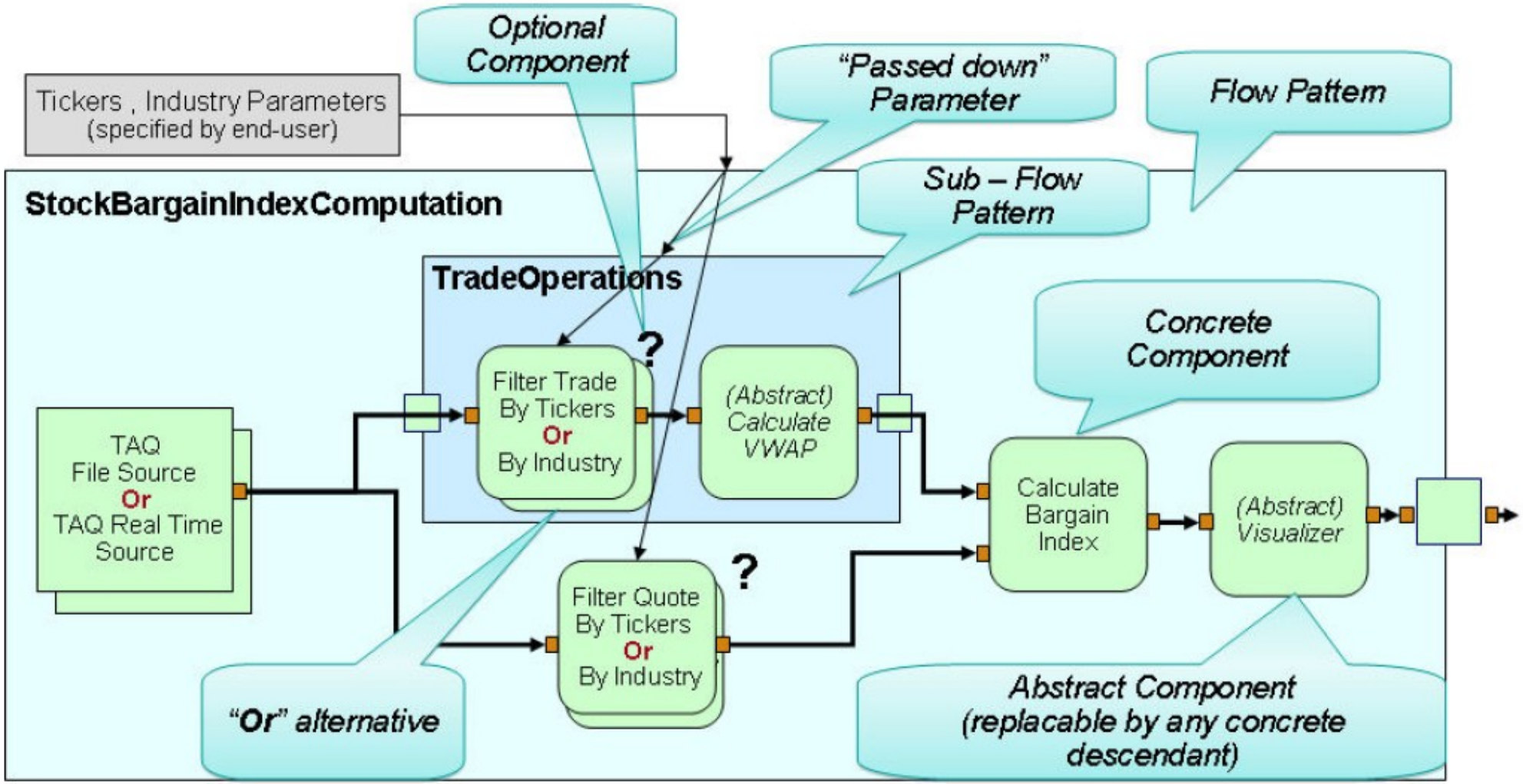

TAQ
CSV
Trade
Quote
FilterByTicker
VWAP
ExponentialIndex
PerfSink
TAQ
CSV File
Source
Filter Trade
By
Tickers
VWAP
(Average
Trade Price)
Filter Quote
By
Tickers
Calculate
Bargain
Index based
On Exponential
Index
PerfSink

Adding Another Filter?



How many new plans can be generated if we add one more component?

Cascade Flow Patterns



From Cascade to SPPL (and Back)

- Cascade compiler:
  - Parses and validates Cascade
  - Translates Cascade to input/output tag descriptions
- MARIO Tag-based Planner:
  - Translates tag component descriptions, goals and tag taxonomies to SPPL
  - Runs SPPL planner
  - Translates resulting plans into flows, annotates them with tags

Cascade Component Example

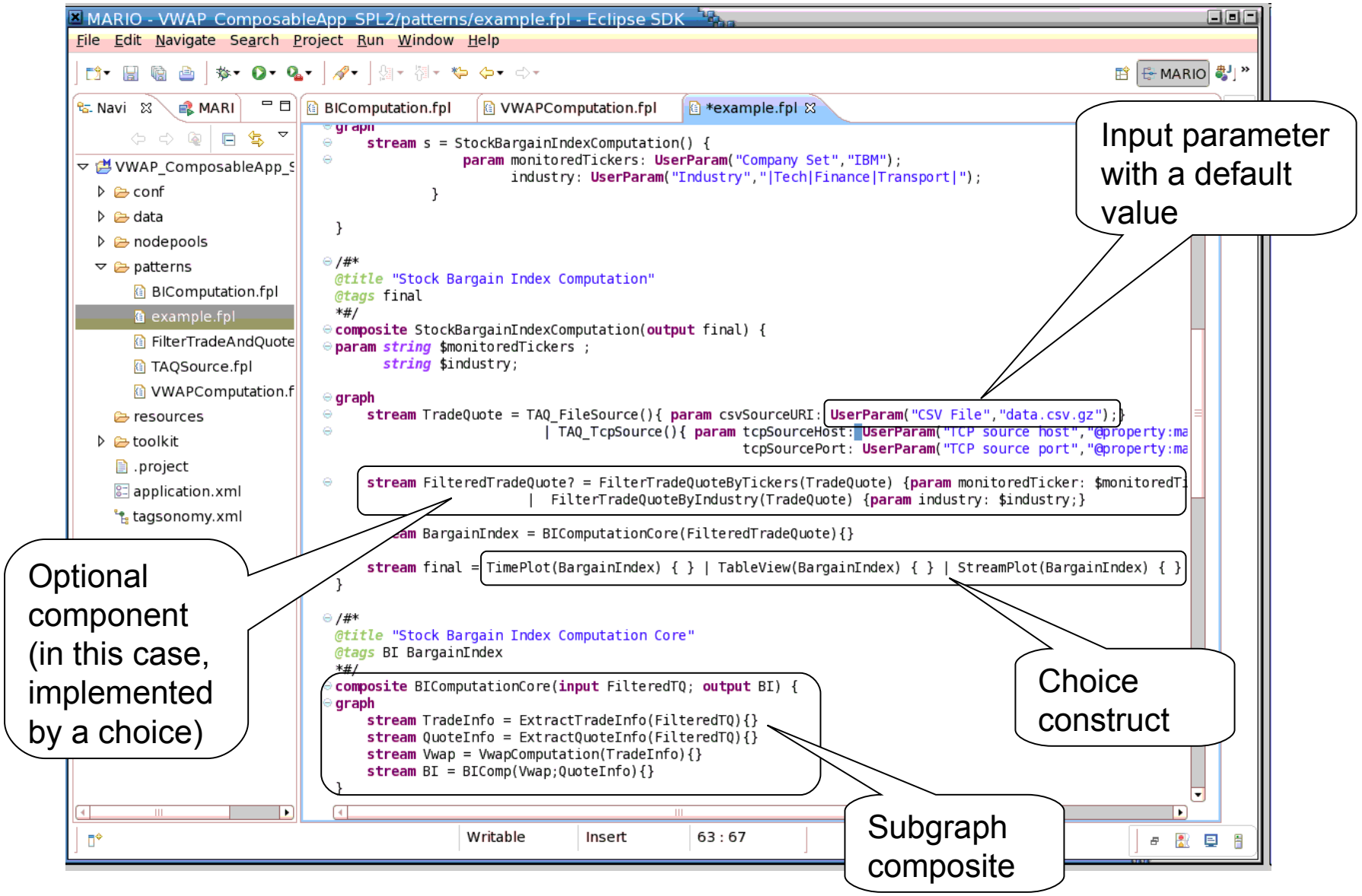
```
/#*
@type  "spl"
@title "Filter Trade and Quote By Tickers"
@tags TradeQuoteFilter ByMonitoredTickers ~AllCompanies OneCompany
*#/
component FilterTradeQuoteByTickers(output TradeQuoteFilter; input TradeQuote) {
        param string $monitoredTicker;
  /$
    stream <rstring ticker, rstring industry, rstring date, rstring time, rstring ttype,
            decimal64 price, decimal64 volume,  decimal64 vwap, decimal64 bidprice,
            decimal64 askprice,  decimal64 asksize> @TradeQuoteFilter@
     = Functor(@TradeQuote@) { param filter: ticker == "@monitoredTicker@"; }
  $/
}

composite StockBargainIndexComputation(output final) {
  param string $monitoredTickers ; string $industry;
  graph
  // ...
  stream FilteredTradeQuote? =
     FilterTradeQuoteByTickers(TradeQuote) {param monitoredTicker: $monitoredTickers;}
     |  FilterTradeQuoteByIndustry(TradeQuote) {param industry: $industry;}
  // ...
}
```

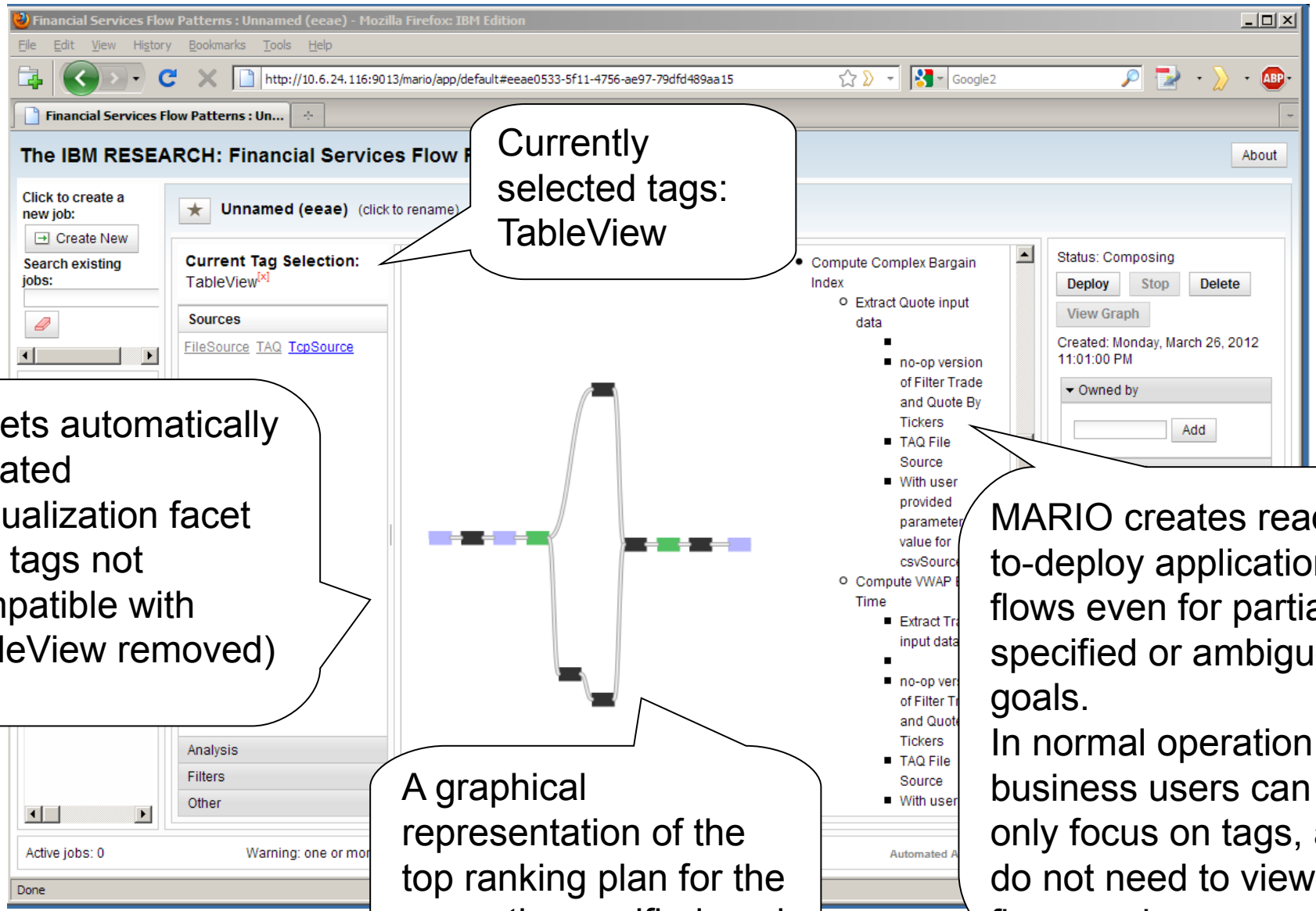
Generated SPPL

```
(:action FilterTradeQuoteByTickers--StockBargainIndexComputation-VWAP-FilteredTradeQuote-3950_c
        :singleton
        :parameters (  ?id1_p - __TradeQuote-StockBargainIndexComputation-VWAP_t
                       ?id2_p - __VWAP-FilteredTradeQuote-3950_t
                       ?id3_p - __StockBargainIndexComputation_t ?id4_p - _monitoredTicker_t)
        :cost ( -1 1)
        :precondition [TradeQuote]  (and   (T ?id1_p)    )
        :precondition [monitoredTicker]  (and   (T ?id2_p) (T ?id3_p) (T ?id4_p)     )
        :effect [TradeQuoteFilter]  (and
                (S ByMonitoredTickers_t) (S Complex_t)  (S OneCompany_t)
                (T __FilteredTradeQuote-StockBargainIndexComputation-VWAP_t)
                (not (S AllCompanies_t))    )
)
(:action Dummy-FilterTradeQuoteByTickers-StockBargainIndexComputation-VWAP-3951_c
        :singleton
        :parameters (  ?id1_p - __TradeQuote-StockBargainIndexComputation-VWAP_t)
        :cost ( -0.00001 1)
        :precondition [TradeQuote]  (and  (T ?id1_p)  )
        :effect [TradeQuoteFilter]  (and
                (T __FilteredTradeQuote-StockBargainIndexComputation-VWAP_t)
                (T __TradeQuote-StockBargainIndexComputation-VWAP_t)
        )
)
```

Eclipse-Based Cascade IDE



MARIO User Interface



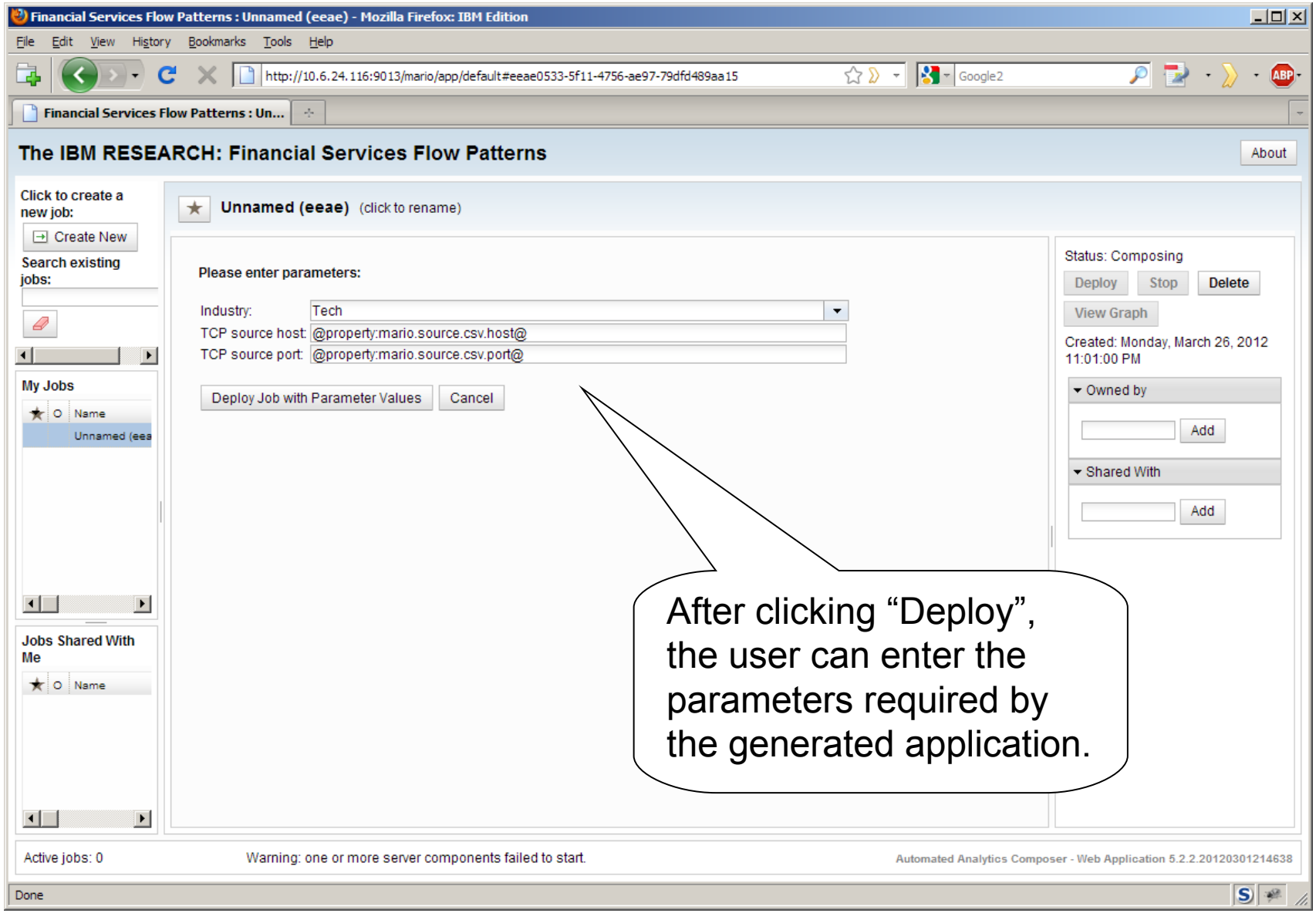
Currently selected tags: TableView

Facets automatically updated (Visualization facet and tags not compatible with TableView removed)

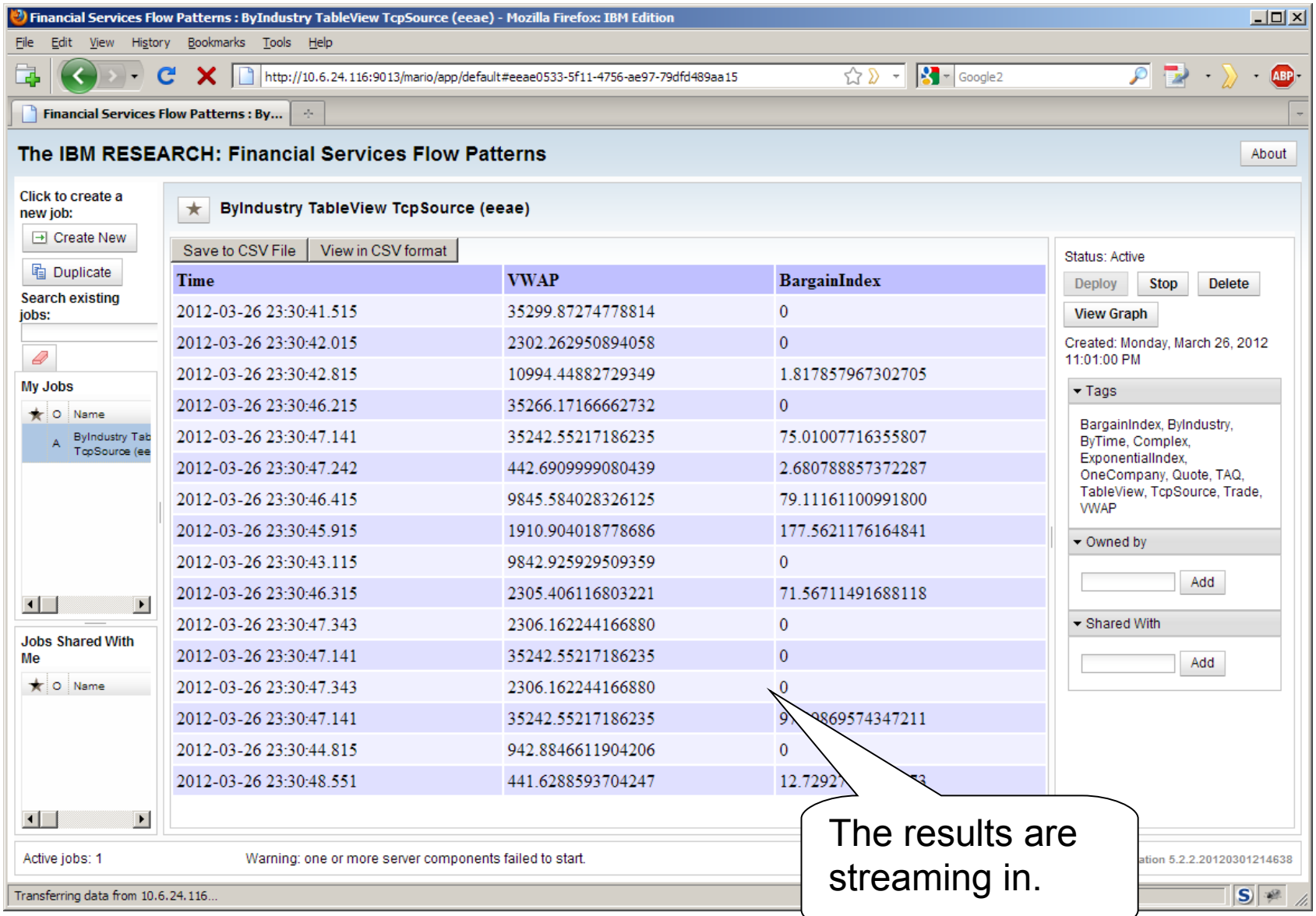
A graphical representation of the top ranking plan for the currently specified goal ("TableView")

MARIO creates ready-to-deploy application flows even for partially specified or ambiguous goals.
In normal operation, business users can only focus on tags, and do not need to view flow graphs.

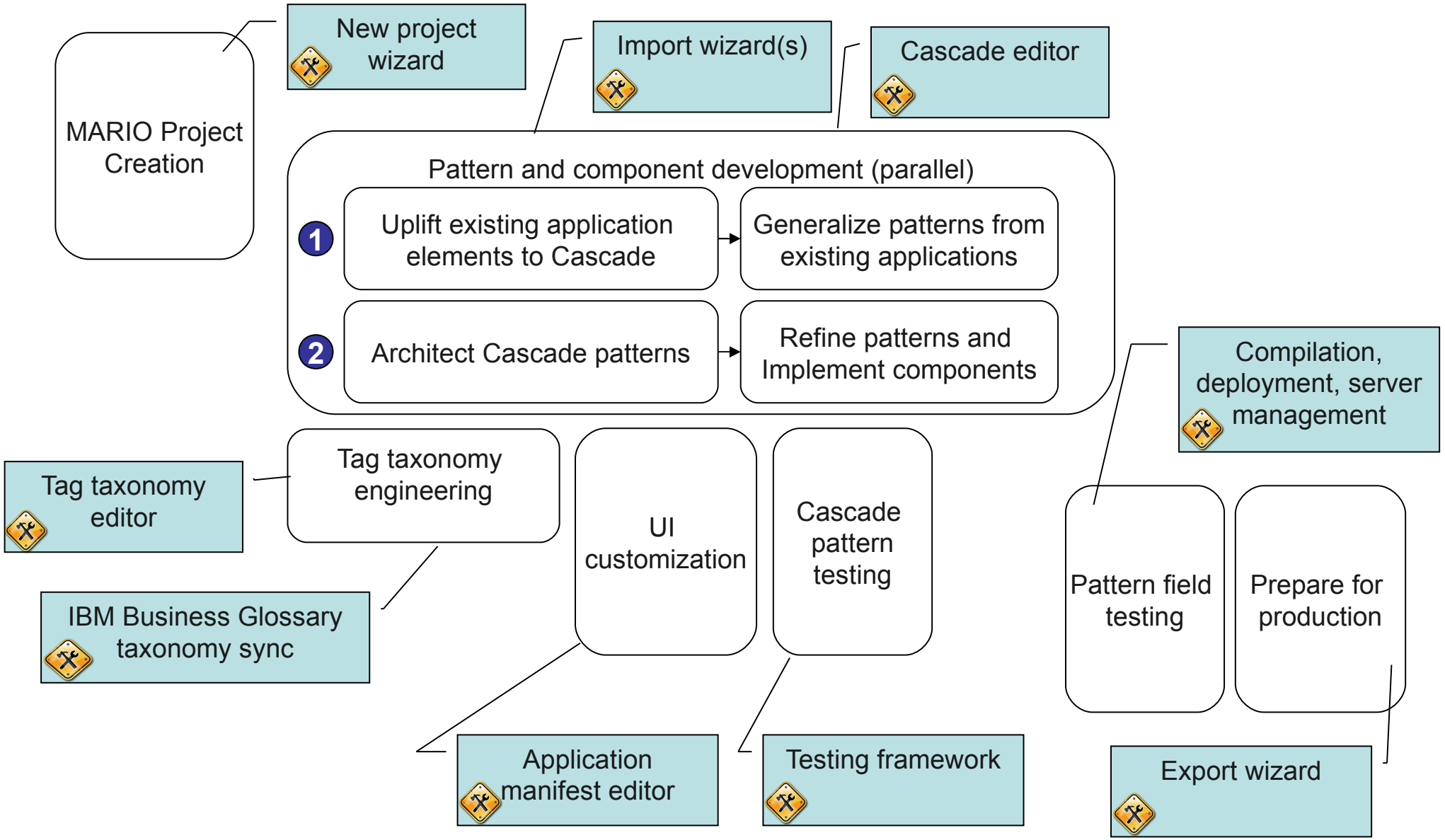
MARIO User Interface



MARIO User Interface



MARIO App Development Process and Tools



See Also

- Our paper comparing HTN and Cascade
  - In SPARK workshop tomorrow
- MARIO Demo
  - System Demos -- 8pm Wednesday