Philip Brown, Michael Haslam

predictum

JMP Discovery Summit 2012

Text Stream Processing in JMP

Outline

- What is Text Stream Processing
- Motivation
- Pattern Matching Functions in JMP
- TCP Sockets in JMP (Streams)
- Demo

What are Text Streams

- Textual data continuously arriving over time, usually in real time
- Often rapid, ordered sequence of text.
- Examples:
  - News articles, blogs, customer feedback/ratings.
  - Transactional Data, Sensor Data.
  - Social media (Twitter, Facebook), stock quotes

What is Text Stream Processing

- Essentially a form of Text Mining.
- Text mining often defined as:
  - "The discovery by computer of new, previously unknown information by automatically extracting information from different resources."[1]
  - Text mining is similar to data mining, except that where data mining involves analyzing structured data from databases, text mining can handle unstructured or semi-structured data such as full-text documents, emails and HTML files.
- Attempts to glean information from a continuous stream of unstructured or semi-structured textual data, through the identification and exploration of lexical patterns.

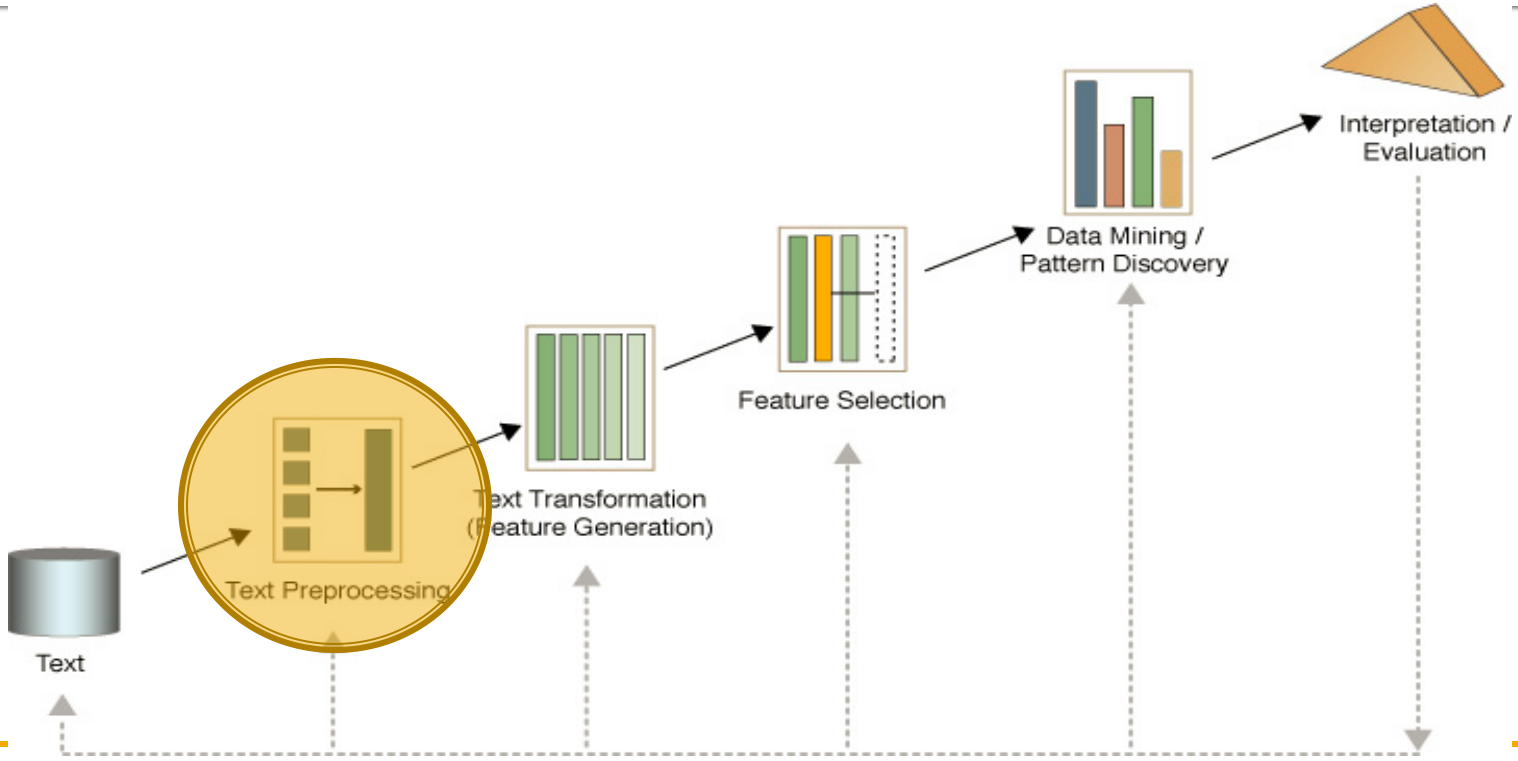
Text Stream Properties

- Produced at a high rate, often large amounts over time.
- Volume of data is too large to be stored.
- Able to be read only once or briefly due to the temporal aspect.
- Data is highly non-stationary (properties change over time).

Motivation for Text Stream Processing

- Information Extraction
  - Identifies key words or phrases and relationships within text using pattern matching.
- Topic Tracking
  - Using a pre-defined set of topics, users are notified when news articles relating to those topics are encountered
- Summarization and Categorization
- Sentiment Analysis
- Social Network Analysis
- Cases where a reaction time is important
  - Stock Exchange
  - Fraud detection

Text Mining Process



Pattern Matching Functions in JMP

- Two categories of functions.
  - Pattern Matching (actually only one such function)
  - Pattern Constructors
- THE pattern matching function:
  - `Pat Match(<source string>, <pattern>, <optional args>)`
    - <source string>: Any character based string or ***string stream***.
    - <pattern>: Any ***pattern object*** created with the pattern constructors.

Pattern Constructors

- Can be divided loosely into the following categories
  - String (string arguments)
  - "Command" (no arguments)
  - Positional (numeric arguments)
  - Assignment (variable name arguments)
  - Operator (pattern arguments)
- Patterns can be built using combinations of the pattern constructors.

Pattern Constructors: Numeric/Positional

Pat Pos(*n*): Match *n* steps into the source string from the left.

Pat R Pos(*n*): Match *n* steps into the source string from the right.

Pat Tab(*n*): Matches any character from o to *n* steps into the source string from the left.

Pat R Tab(*n*): Matches any character from o to ***m*** steps into the source string until ***n*** steps from the end.

Pat Len(*n*): Match an arbitrary *n* characters within the source string.

Pattern Constructors: String

Pat String(*string*) : Match the **entire** *string.*

Pat Any(*string*): Match **any single** character in the *string.*

Pat Not Any(*string*) : Match **any single** character NOT in the *string.*

Pat Span(*string*) : Match **one or more** characters in the *string.*

Pat Break(*string*): Match **any single** character in the *string* and halts the pattern matcher.

Pat Regex(*string*): Match the quoted regex expression.

Examples

```
p = "12346" + Pat RPos(2);
Pat Match( "1234600", p );

p = "12346" + Pat Len(3);
Pat Match( "12346000", p );
```

Pattern Constructors: Command

Pat Arb(): Represents any arbitrary character. (Result can be assigned to a variable)

Pat Rem(): Represents the remainder of the source string.

Pat Fail(): Represents an immediate FAIL to the pattern matcher

Pat Abort(): An immediate ABORT (regardless if passing or failing)

Pat Succeed(): An immediate PASS.

Pat Fence(): "One-way PASS". Represents and immediate PASS when pattern matcher is moving forward. Fails when pattern matcher tries to move backward.

Examples

```
patt = PatPos(2) + PatArb() + PatRTab(3) >>
someVar + PatAt(here) + PatRem()>>rest;
```


Pattern Constructors: Assignment

Pat At(*varname* **):** Assign the current position in the source string that the pattern matcher is at, to *varname*.

Pat Immediate(**pattern,** *varname*): Assign result of **pattern** (command/operator) to *varname*.

Pat Conditional(**pattern,** *varname*): If **pattern** succeeds, assign result to *varname*.

Pattern Constructors: Operator

Pat Arb No(p): Match pattern p, an arbitrary number of times (zero or more).

Pat Repeat(p, min, max, <GREEDY, RELUCTANT>): Match pattern p a specific amount of times.

Pat Altern(p1, p2,…,pn): Match pattern p1 OR p2 OR…pn (p1 | p2 | … | pn)

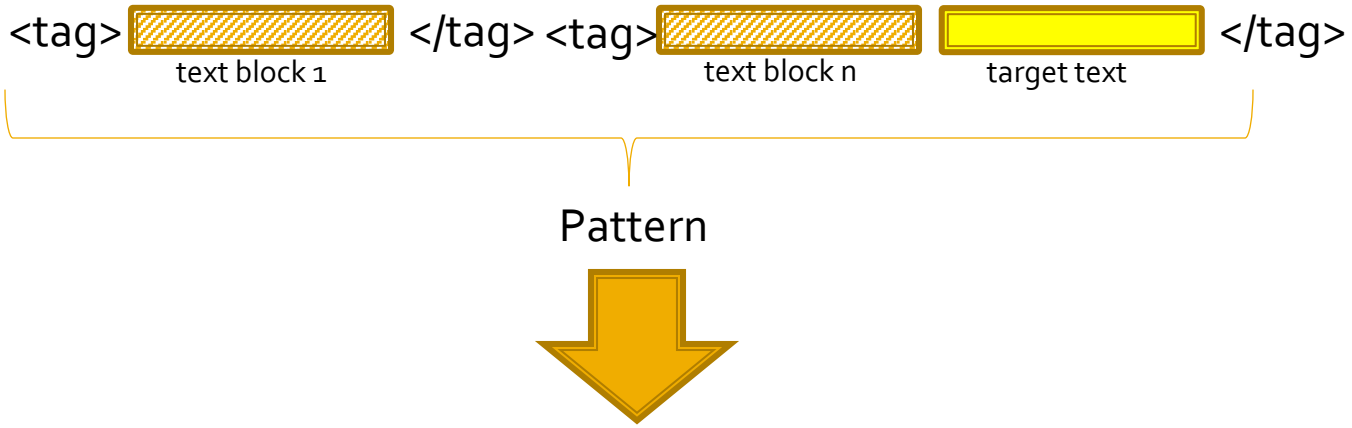
Pat Concat(p1,p2,…pn): Concat patterns p1, p2,..pn (p1 + p2 + … + pn)

Pat Test(Expr): Denote as a Pat Succeed() if result of Expr is TRUE. Otherwise Pat Fail().

Examples

```
patt = PatImmediate(PatAny("aeiou"), thisVowel) +
PatArb()>>thisPart + PatLen(2) >? lastPart;
```


Pattern Construction



patternObj =
PatString("<tag>") + PatArb() + "</tag>" + "<tag>" + ("this" | "that") +
PatArb() >> targetText + </tag>

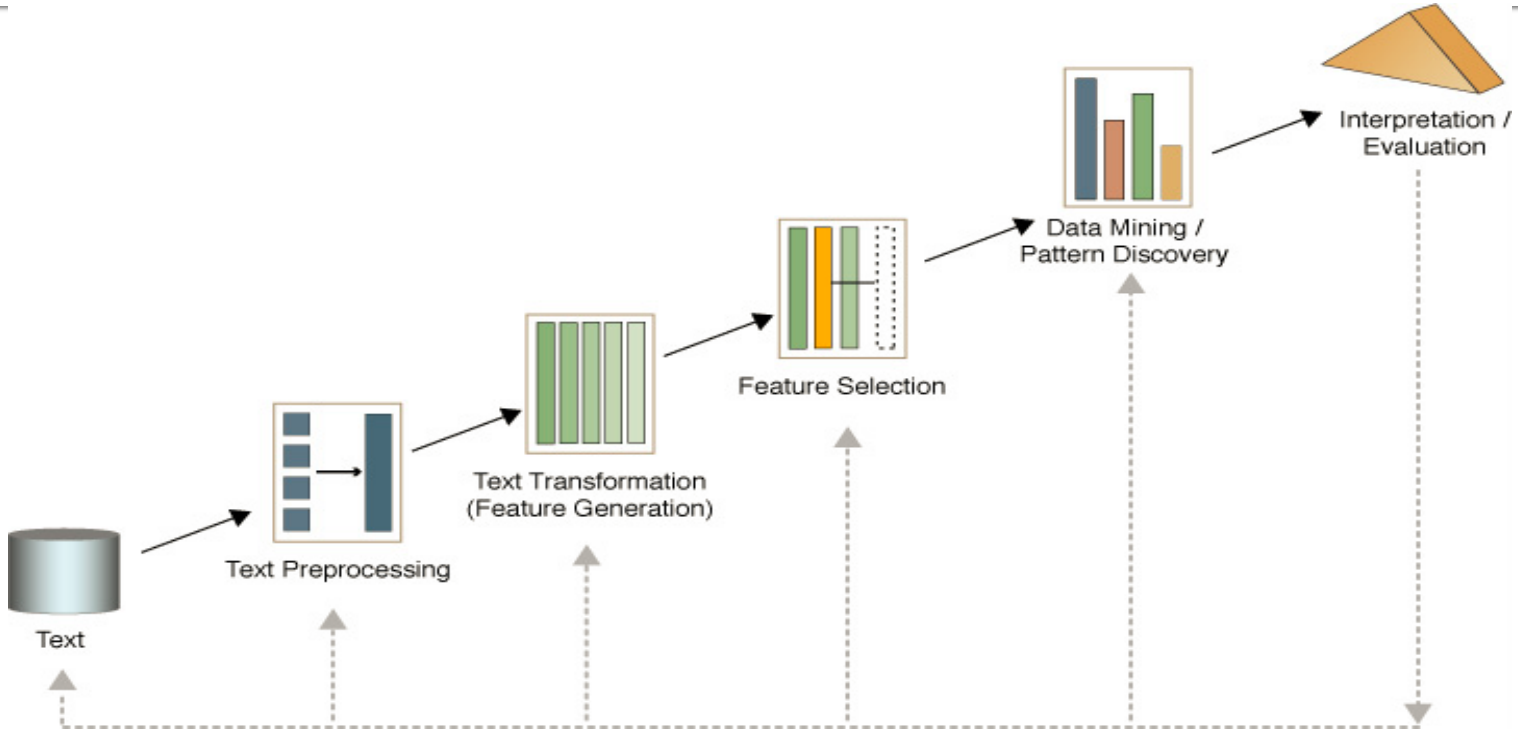
Pattern Matching Objectives

- Look for certain markers, categorize and parse
- Filter by language/ grammer.
- Interpret sentiment (word analysis)

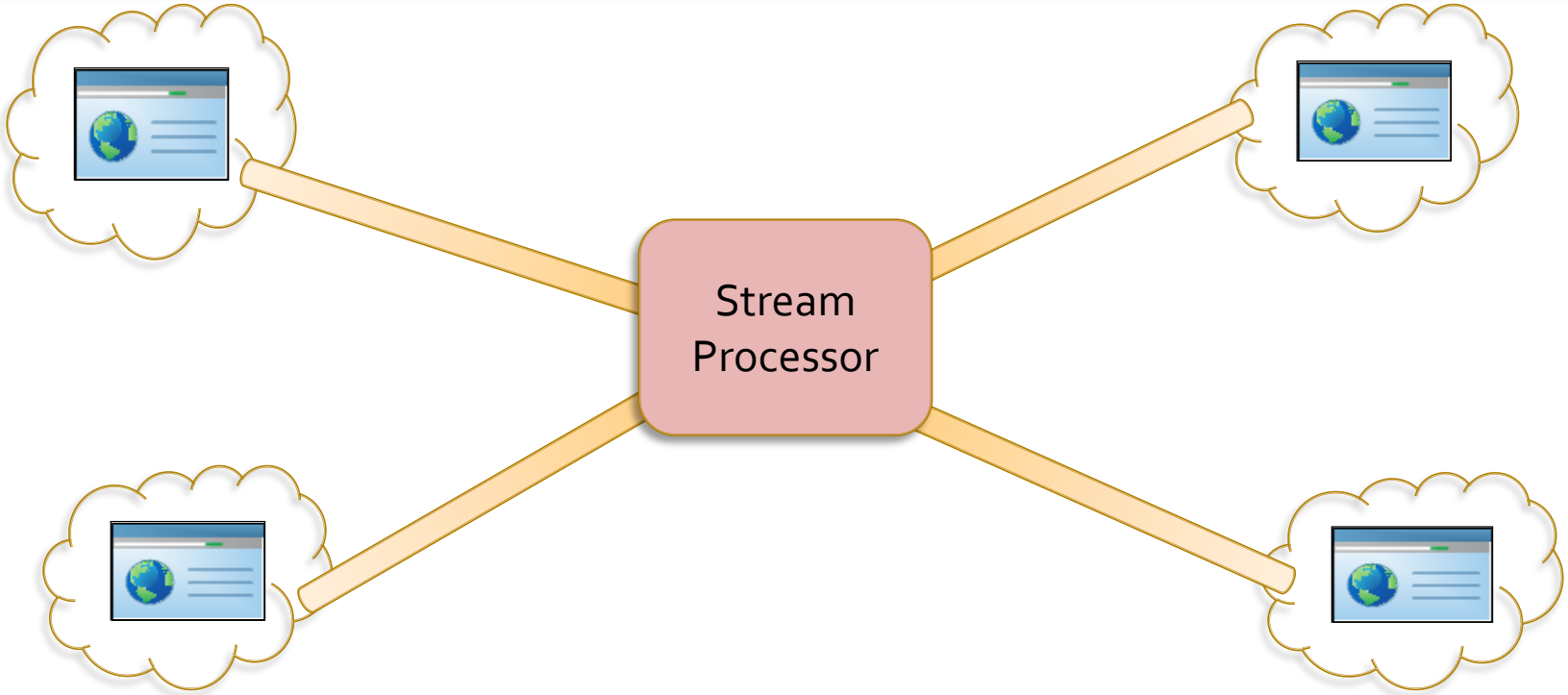
Demonstration

Parsing HTML Encoded JMP Report using pattern matching

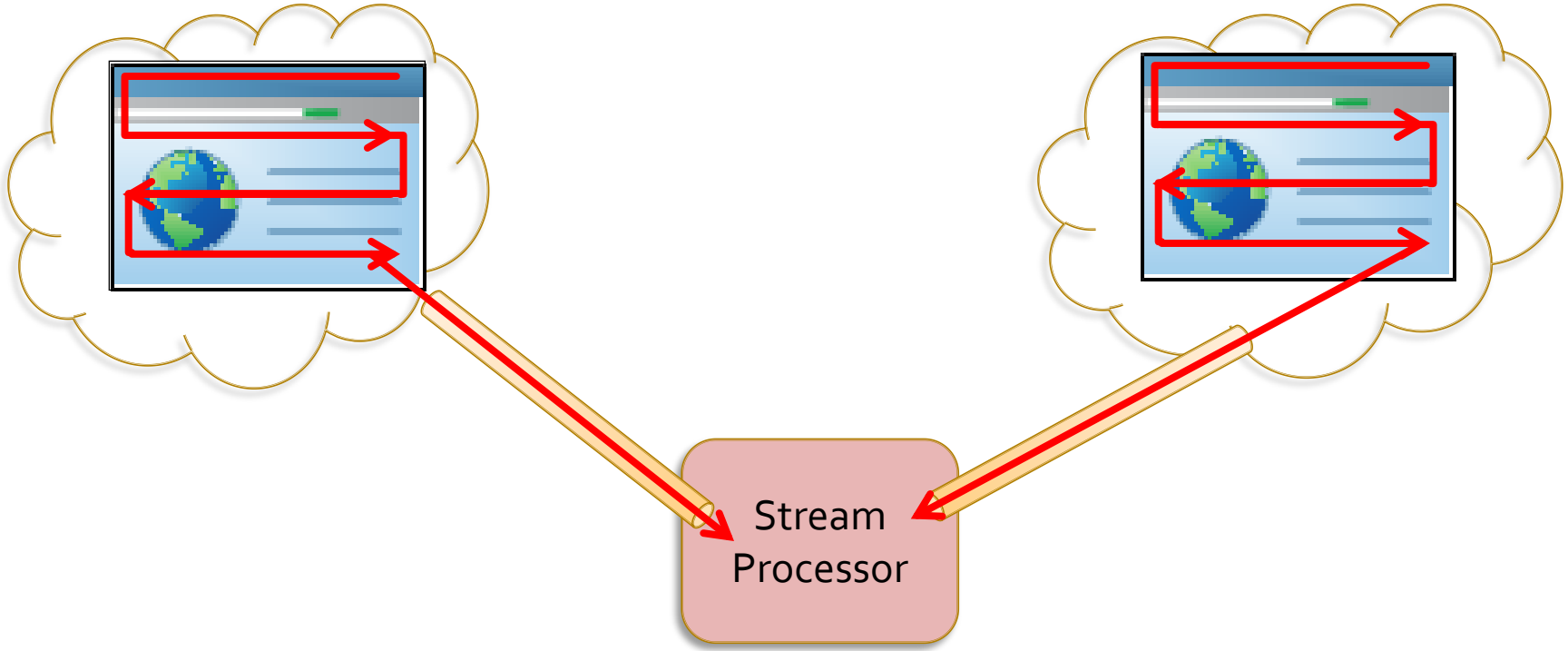
Text Mining Process



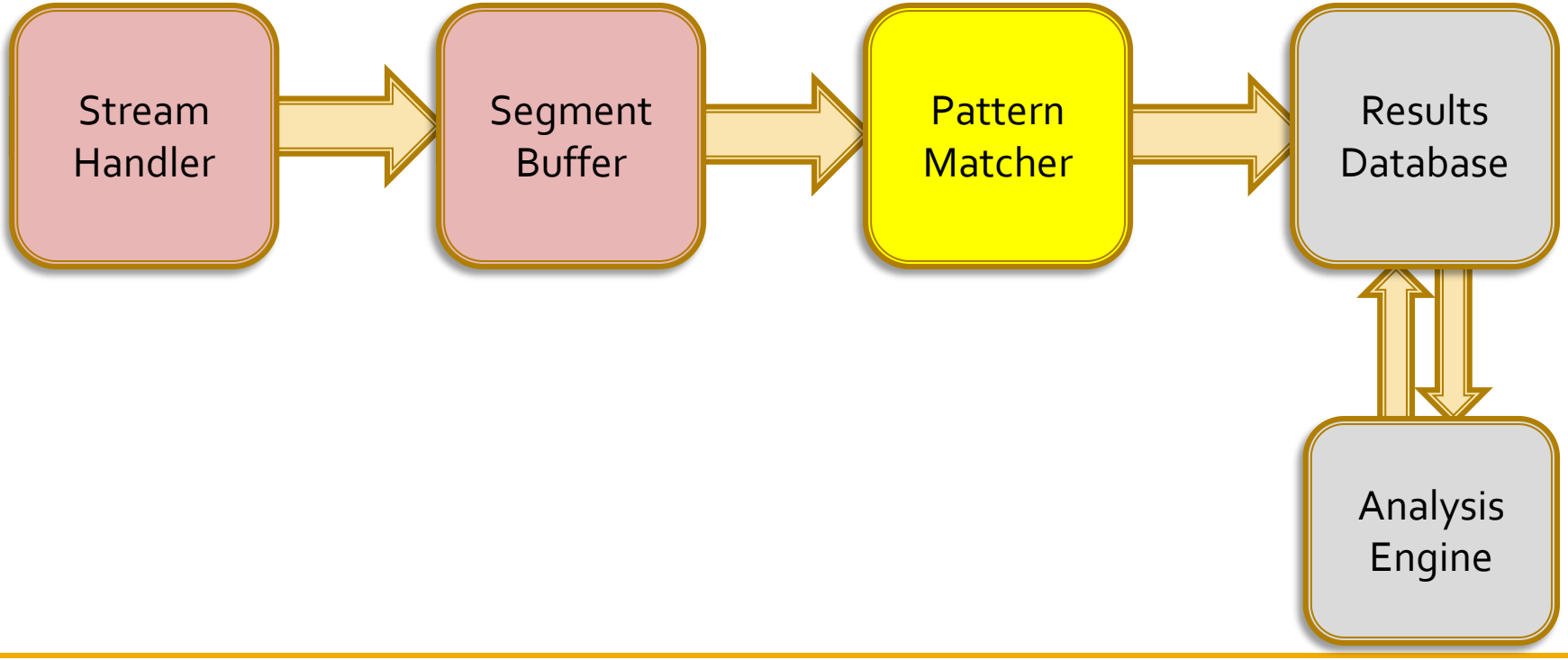
Streaming via TCP socket



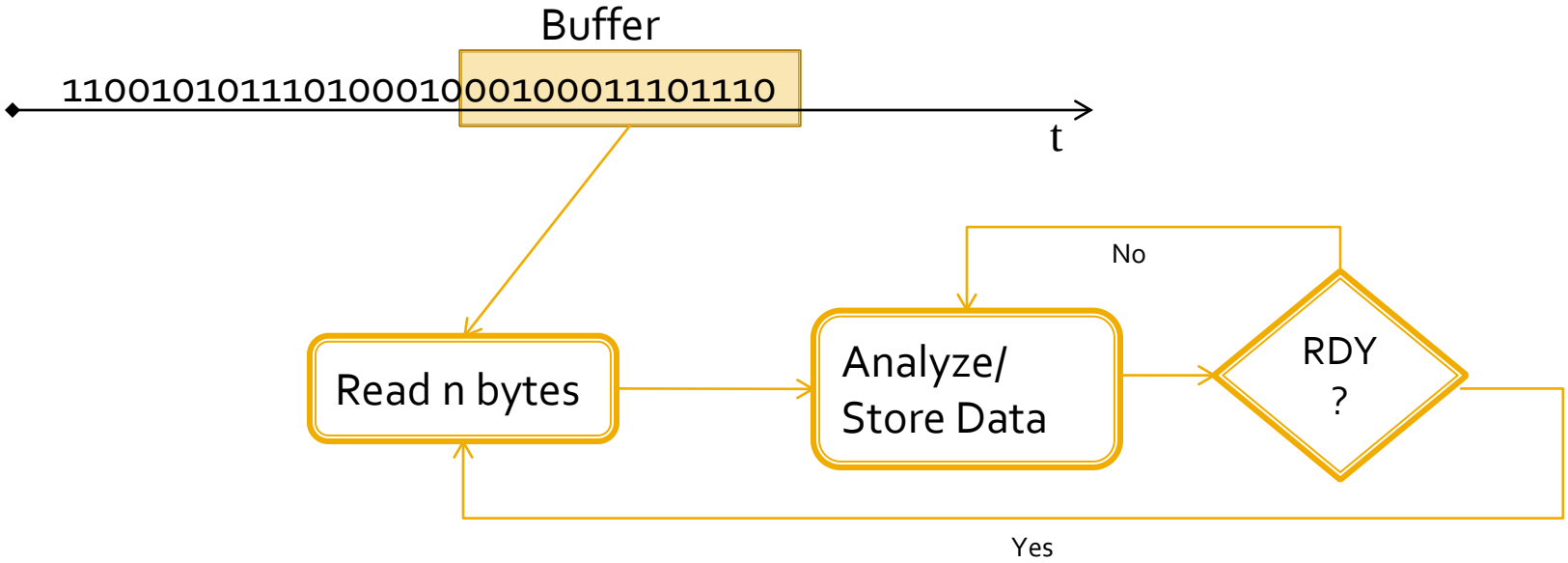
Streaming via TCP socket



Overview of Stream Processor



Stream Processor - Detail



Demonstration

Processing live TCP stream with JMP's pattern matching functions.