LinkedIn

Quasar: A High-Performance Scoring and Ranking Library

May 20, 2019



Fei Chen

Senior Engineering Manager

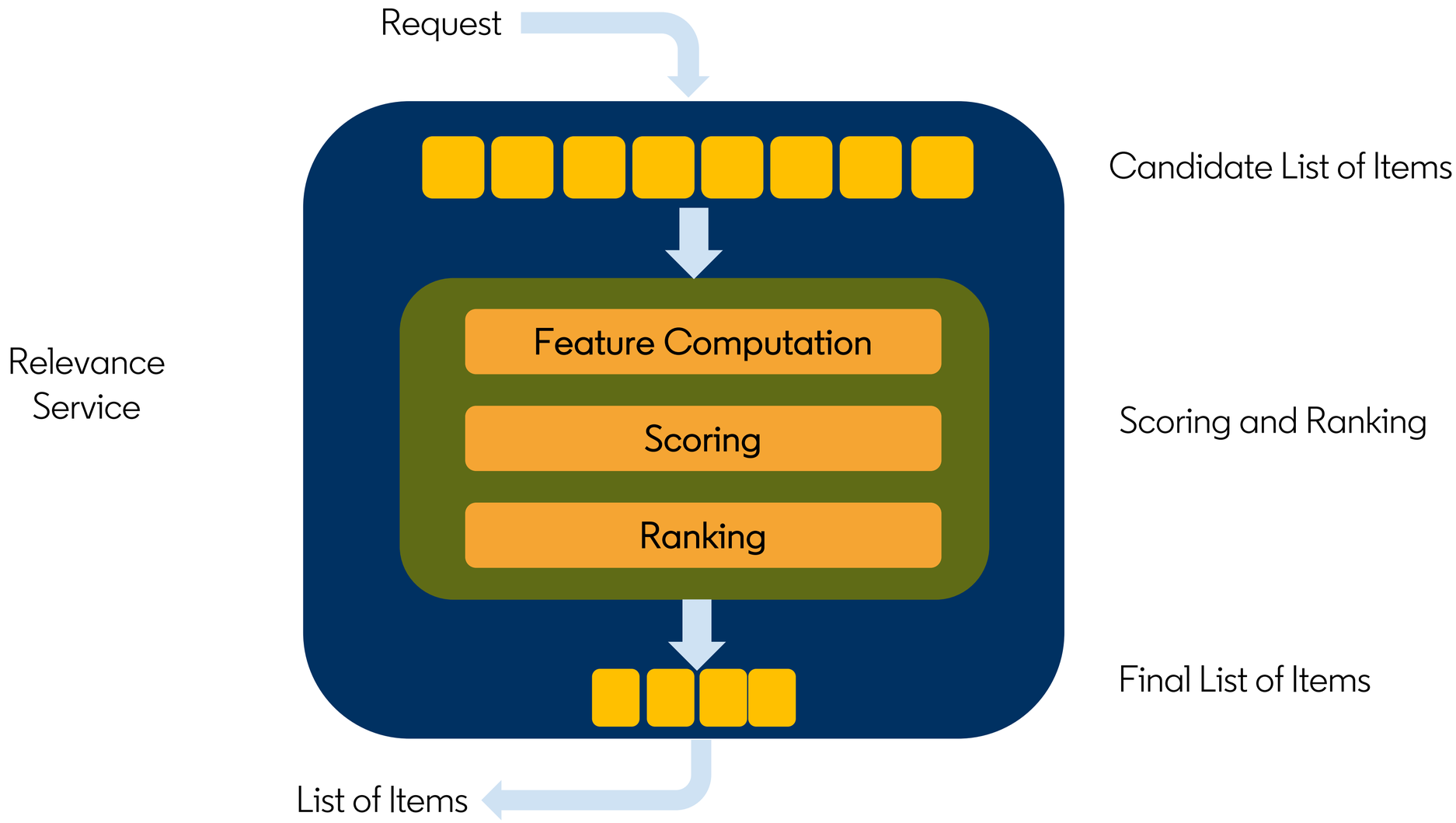


Andris Birkmanis

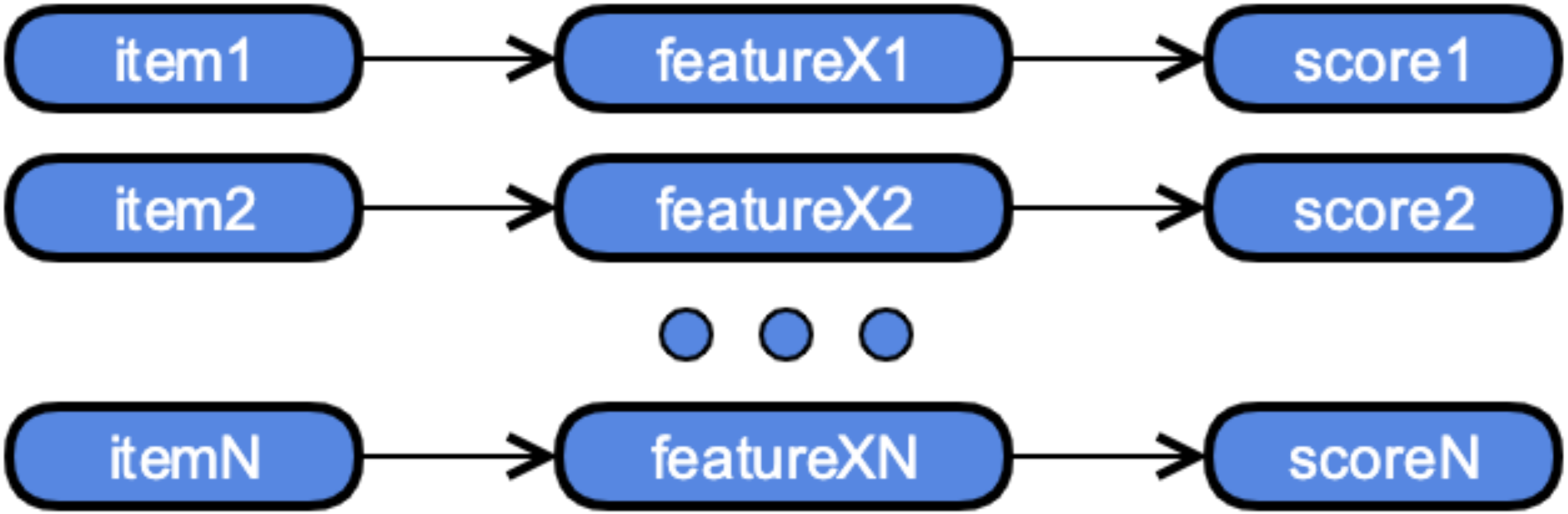
Staff Software Engineer

Background

Scoring and Ranking Paradigm



Feature Computation and Scoring

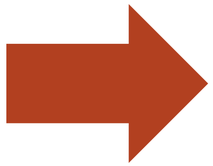


Spreadsheet Metaphor

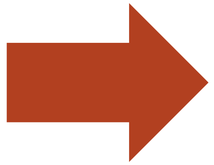
Item	FeatureX	FeatureY	Score
item1	0.3	-0.5	0.7
item2	0.9	0.5	0.8
…	…	…	…

Ranking

Item	X	Y	S1
item1	0.1	0.3	0.5
item2	-0.3	0.2	0.8
item3	-0.4	0.1	0.6
item4	0.3	0.5	0.9
item5	0.3	0.1	0.7
item6	0.1	-0.2	0.5



TOP 3 BY S1



Item	X	Y	S1
item2	-0.3	0.2	0.8
item4	0.3	0.5	0.9
item5	0.3	0.1	0.7

Why Quasar?

Prior to Quasar

- Product-specific solutions
- Highly optimized for each product
- Little sharing/reuse across products

Business Challenges

Scale ML efforts when

- # of products increases
- # of engineers increases
- Model complexity increases

Key Business Metrics

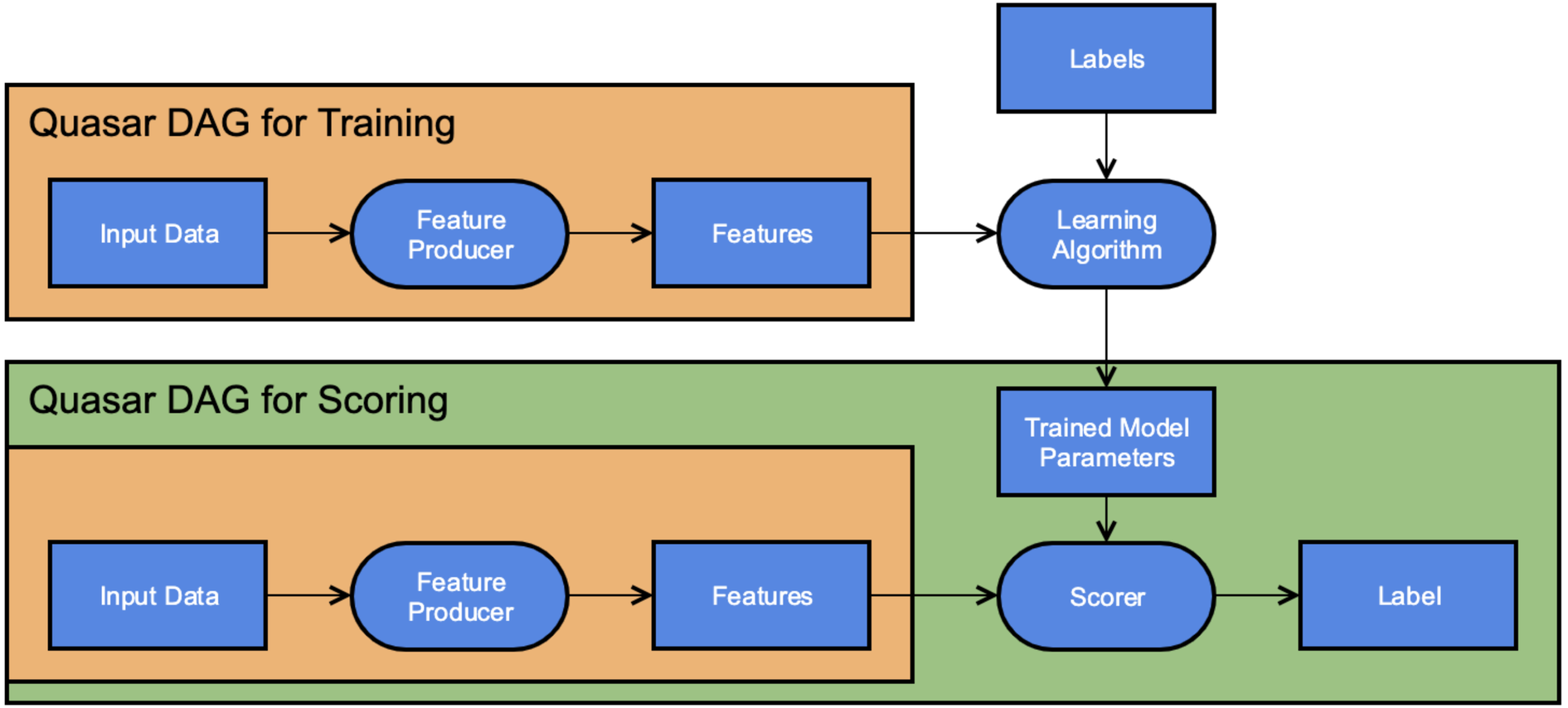
- Productivity
- System efficiency

A Ubiquitous Library for

- Easily sharing modeling efforts
- Model portability
- Expressiveness and composability
- Performance tuned for various system constraints

Portability

Portability and Consistency



DSL

Quasar DSL by An Example

1. Problem

- Generate an engaging feed for a member.

2. Input & Output

Input

- Member profile
- List of all posts by member's connections

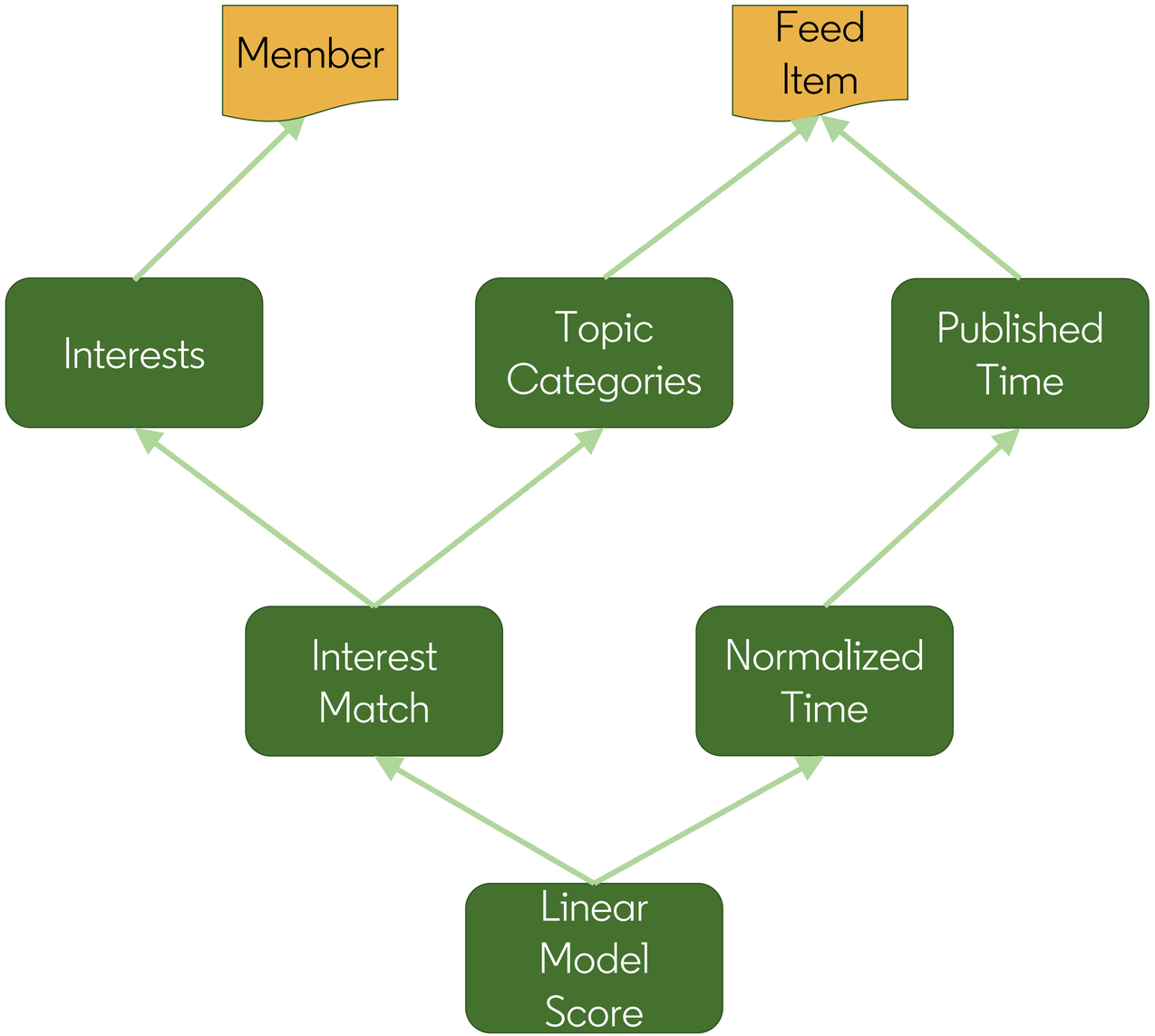
Output

- Top posts ordered by relevance

3. Relevance Model

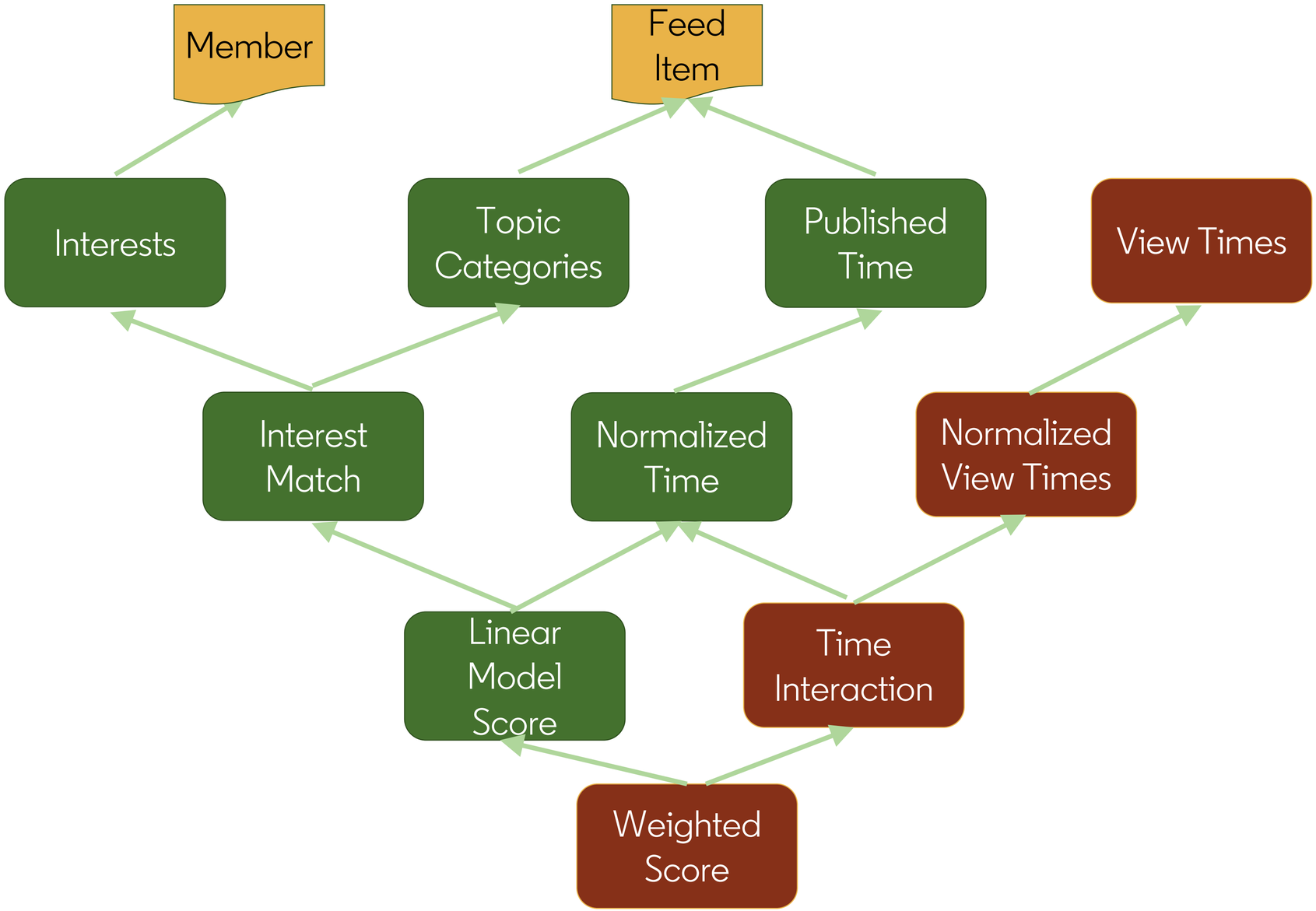
- Do member's interests match post topics?
- Is the post fresh?

Quasar Scoring DAG of the Example

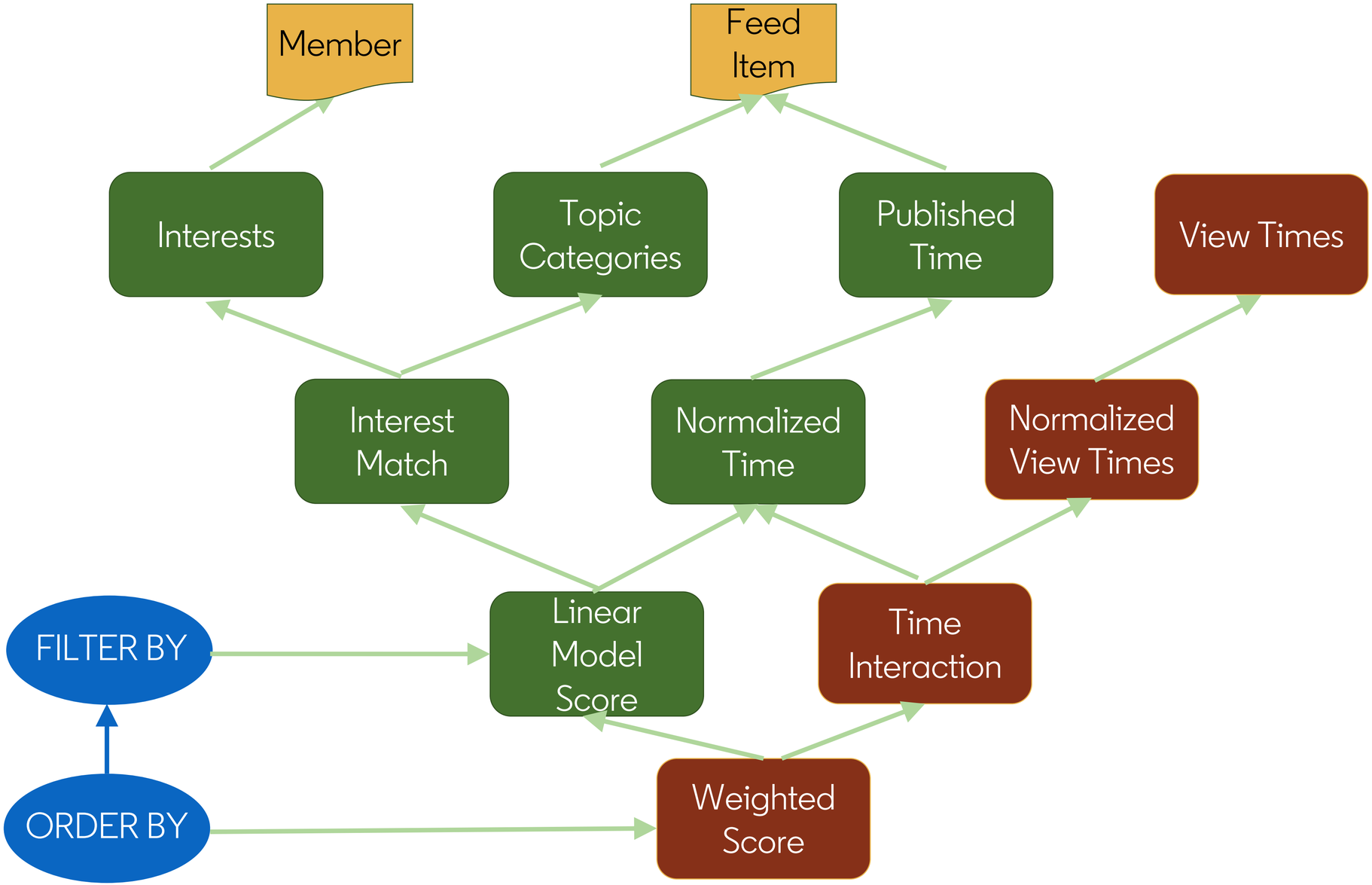


- Expressive
  - Arithmetic and logical expressions
  - Commonly used scoring models
- Strongly Typed
  - LONG, FLOAT, DOUBLE, STRING,
  - Dense/Sparse Vectors and Tensors
- Declarative
  - Specification order != execution order
  - Immutable types

DSL for Model Composition: Ensembles

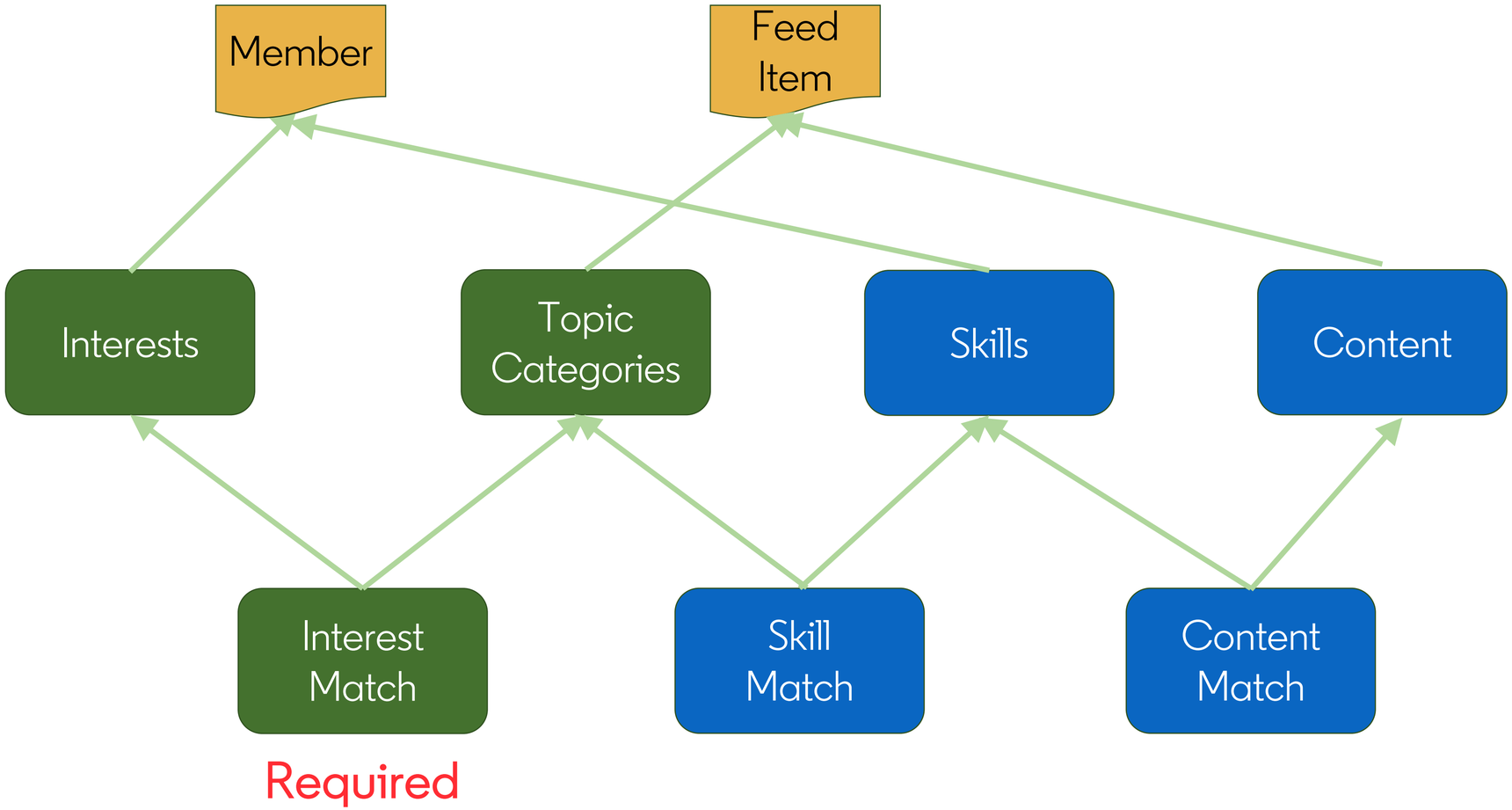


DSL for Model Composition: Multi-Passes

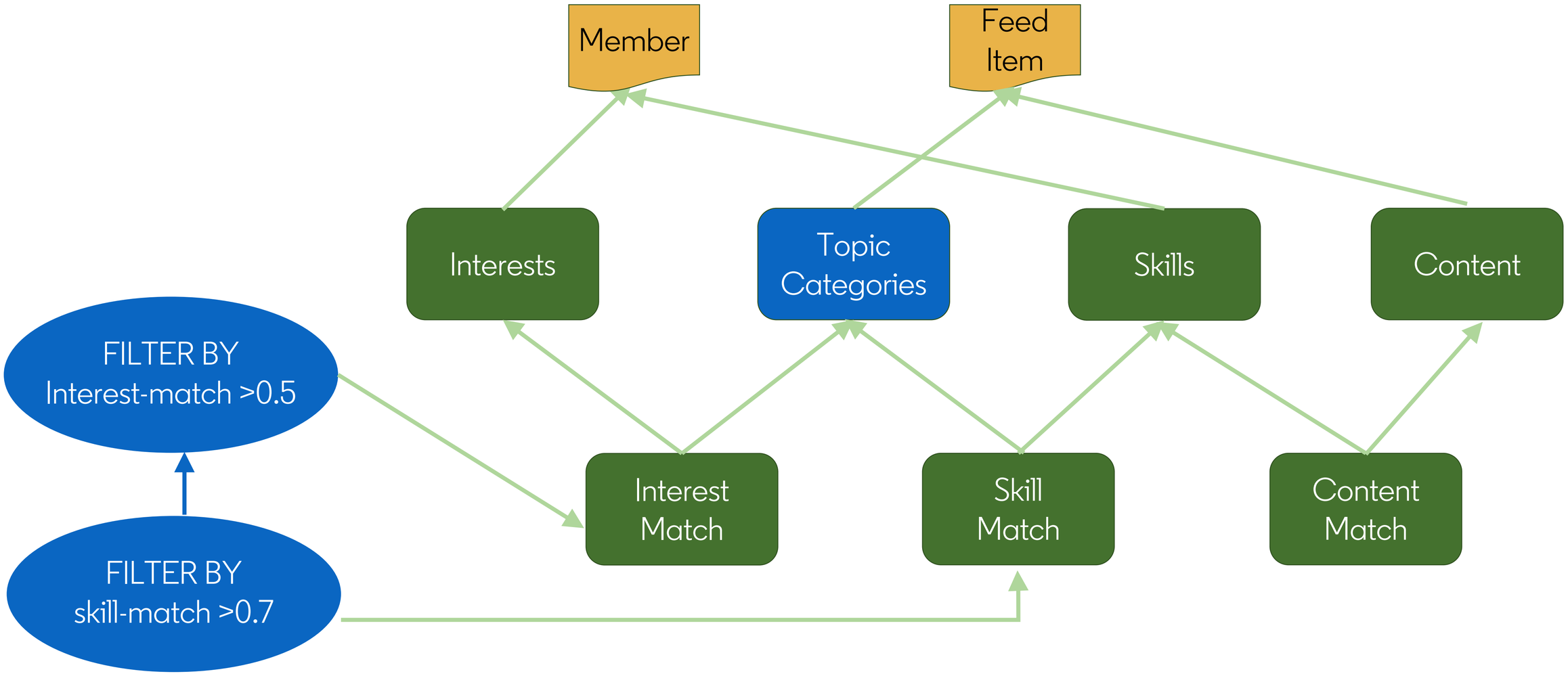


Engine Optimization

Skipping Unneeded Feature Computation



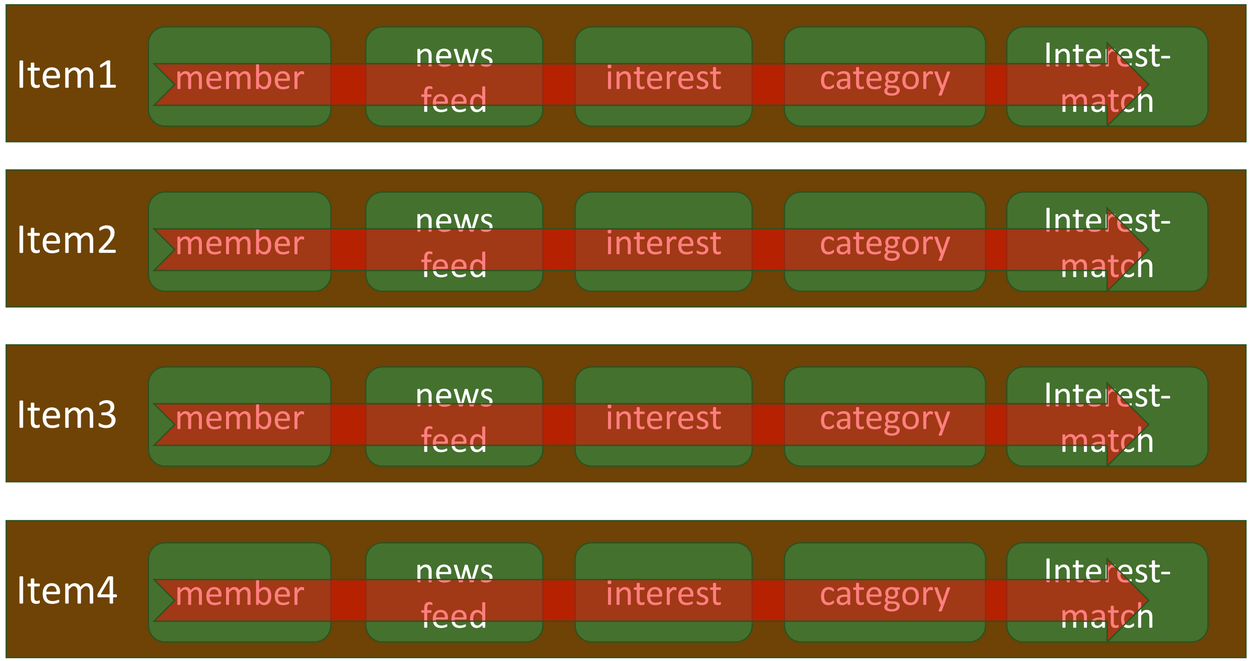
Multi-pass Ranking: Avoid Repeating Computation



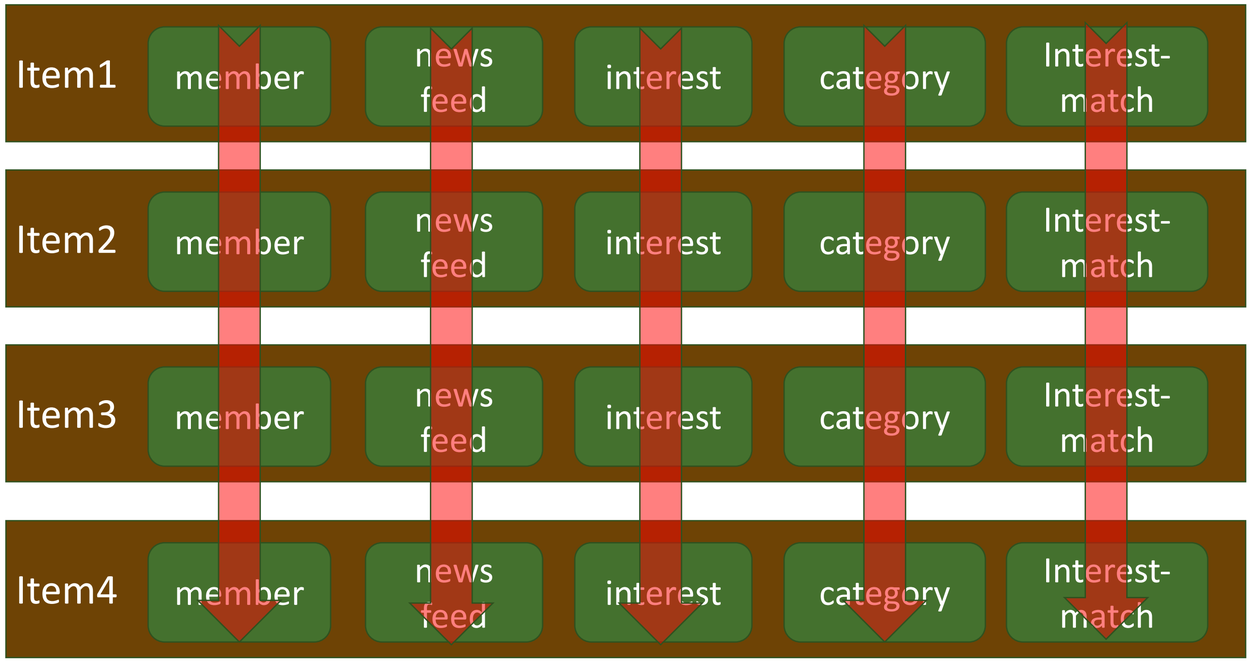
Batch-Column Order Evaluation

Compute Features with Code Locality

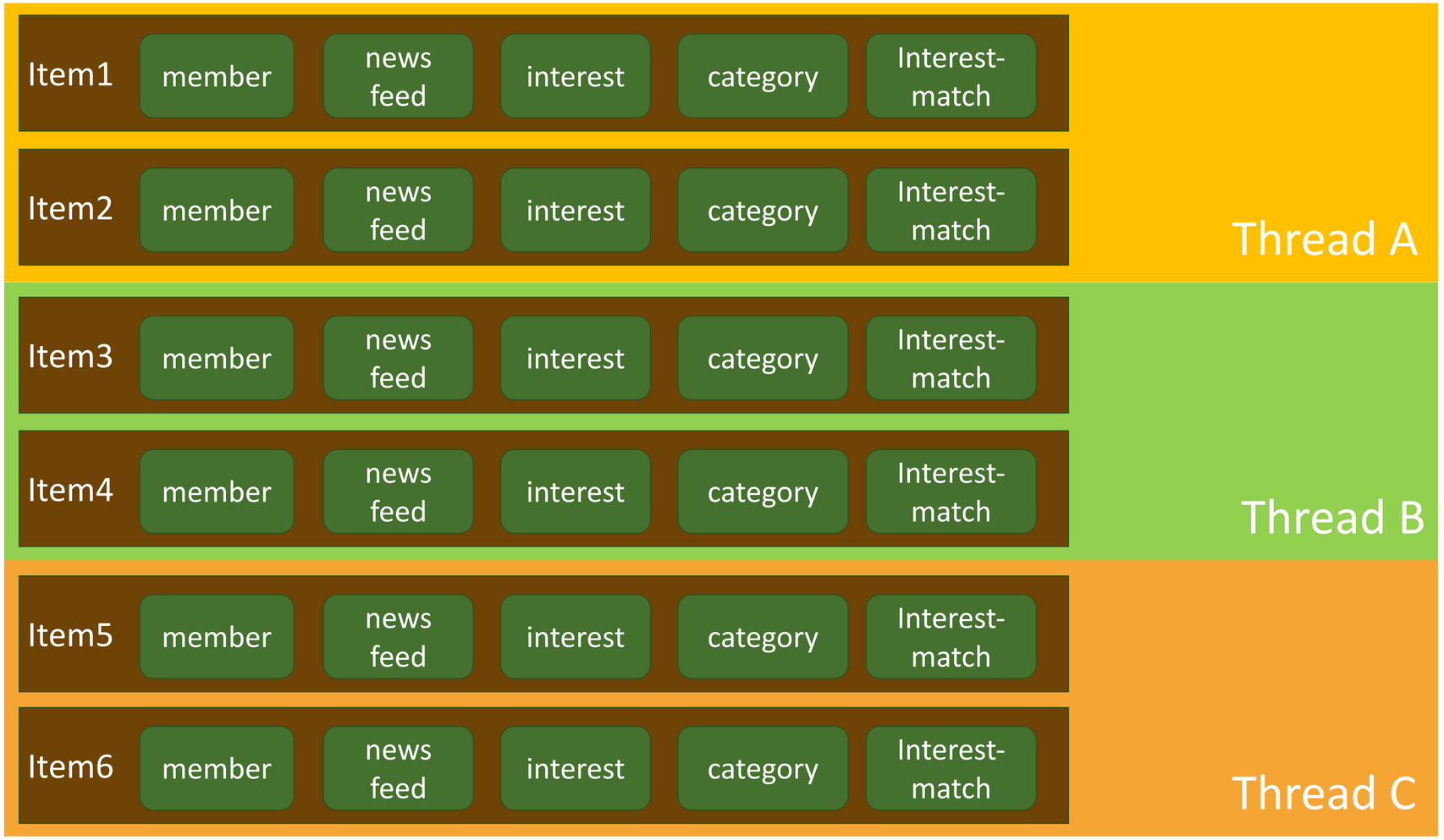
Row-order Feature Computation Flow



Column-order Feature Computation Flow



Engine Optimization: Multi-threaded Evaluation

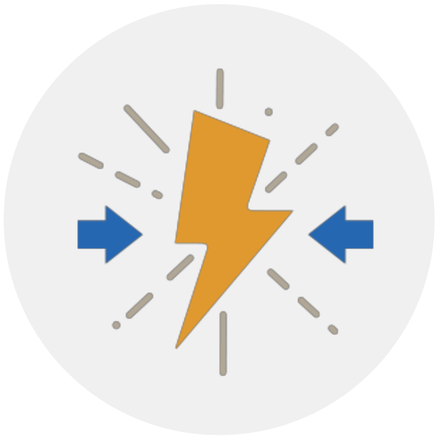


Post-adoption Challenges

Fragmentation Again



Planned a single solution



UDFs still fragmented



New strategy?

Feature Type Standardization

- Reducing degrees of freedom for types for features
  - Relations/Tensors
- Small composable operations
  - Relational algebra/Linear algebra/Einstein notation
- Automatic specialization and fusion of operations in codegen
- Stronger metadata management
  - Categorical and other dimensions

- Still leave ability to use Java UDFs as a fallback or for experimentation

Example of the new DSL

```
// Map operation
TENSOR[ActionType] ][VerbType][ObjectType]:FLOAT activity_log<i><j><k> = log(activity<i><j><k> + 1.0);

// Interaction
TENSOR[ActorType][VerbType][ObjectType]:FLOAT activity<i><j><k> = actor<i> * verb<j> * object<k>

// Join to get a subset
TENSOR[ActorType][VerbType][ObjectType]:FLOAT interstingActivities<i><j><k> = activity<i><j><k>
* importantVerbs<j>;

// Lookup operation
FLOAT recent_viral_activity = activity_recent[VIRAL];
TENSOR[ActorType][ObjectType]:FLOAT likeActivities<i><k> = activityType<i>[LIKE]<k>;
```

Example of the new DSL, cont.

```
// Dot Product with explicit types
TENSOR[ActivityType]:FLOAT activities1 = ...;
TENSOR[ActivityType]:FLOAT activities2 = ...;
FLOAT similarity += activities1<i> * activities2<i>;

// Dot Product with inferred types
val activities1 = ...;
val activities2 = ...;
val similarity += activities1<i> * activities2<i>;
```

Future Directions

- Improving reusability via modules
- Enriching the standard feature type (nested relational?)
- Improving optimizations in codegen
- Backends other than JVM?
- Opensourcing

Thank you