Querying and Creating Visualizations by Analogy

Carlos E. Scheidegger, Huy T. Vo, David Koop,
Juliana Freire, Cláudio T. Silva

SCI Institute, School of Computing
University of Utah

Outline

- Provenance **reuse**
  - We have all this rich metadata - let's use it
- Query-by-example
- Visualization by Analogy
- (VisTrails intro)
  - **Transparent provenance tracking**

Related Work

- Visualization Systems and Libraries
  - AVS, DX, SCIRun, VTK
- History tracking and formalisms
  - Jankun-Kelly et al's pset-calculus
  - Kreuseler et al, VDM history
  - Brodlie's et al's GRASPARC
  - VisTrails

Provenance

- The “pedigree” of an artifact
  - Where did it come from? Who held it?

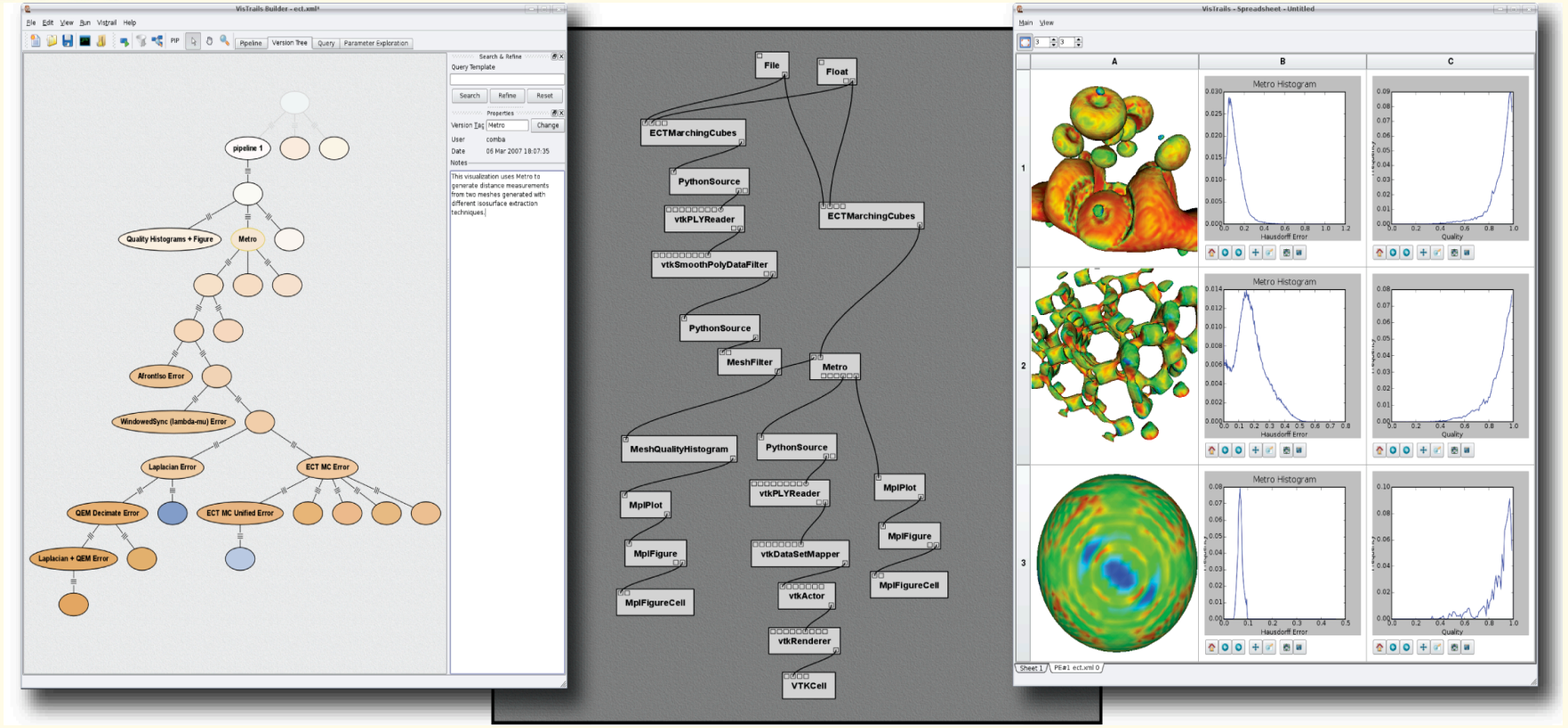
From the Tour: The Beginnings of Impressionist Landscape
Object 5 of 7

Provenance

From the artist 1890 to (Durand-Ruel, Paris and New York); sold 17 March 1892 to Adolphe A. Tavernier, Paris;[1] (Tavernier sale, Galerie Georges Petit, Paris, 6 March 1900, no. 74, as *Rue à Sèvres*); purchased by Berhend, Paris.[2] (Wildenstein & Co., London, New York and Paris).[3] Capt. Edward H. Molyneux [1894-1974], Paris, by 1948;[4] sold 15 August 1955 to Ailsa Mellon Bruce [1901-1969], New York; bequest 1970 to NGA.

[1]Dates of purchase and sale by Durand-Ruel according to François Daulte, *Alfred Sisley*, Paris, 1959, no. 44. [2]Annotated copy of sales catalogue in NGA curatorial files. [3]According to François Daulte, *Alfred Sisley*, Paris, 1959, no. 44. According to letter dated 18 June 1999, in NGA curatorial files, Wildenstein & Co. has no record of ever having owned this painting. [4]Lent by Molyneux to the 1948 Paris exhibition, *Huite Siècles*

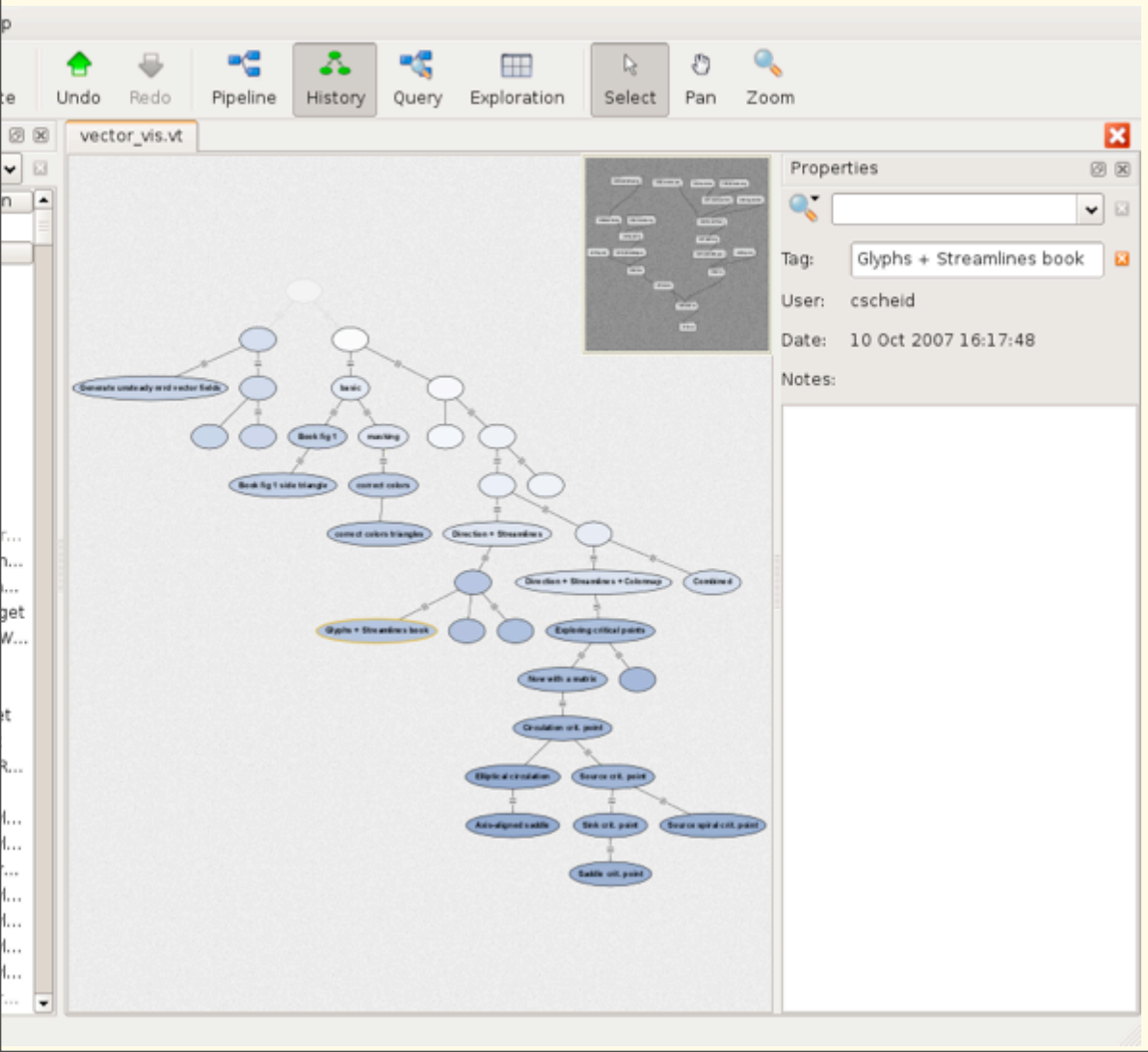
Provenance in VisTrails



- **Process** provenance
- How was this visualization created?

www.vistrails.org

Version Tree



- Persistent
- Transparent
- **Reuse**
  - Can we do better than just presenting?

www.vistrails.org

Why not query languages?


```
Select ExecutableWorkflowId, Execution_Event.ExecutionId, Event.EventId,
Execution_Event.ExecutableWorkflow_ExecutableActivityId
from Execution, Execution_Event, Event, Event_Property_Value, Property, Va
where Value=Cast('C:\TEMP\atlas-x.gif' as binary) and
Event_Property_Value.PropertyId=Property.PropertyId and Event_Property_Va
and Event.EventId=Event_Property_Value.EventId and Execution_Event.EventI
Execution_Event.ExecutionId=Execution.ExecutionId;
```

Why not query languages?


```
Select ExecutableWorkflowId, Execution_Event.ExecutionId, Event.EventId,
Execution_Event.ExecutableWorkflow_ExecutableActivityId
from Execution, Execution_Event, Event, Event_Property_Value, Property, Va
where Value=Cast('C:\TEMP\atlas-x.gif' as binary) and
Event_Property_Value.PropertyId=Property.PropertyId and Event_Property_Va
and Event.EventId=Event_Property_Value.EventId and Execution_Event.EventI
Execution_Event.ExecutionId=Execution.ExecutionId;
```

wf{*}: upstream(x) union x where
x.module = “SoftMean” and executed
(x) and y in upstream(x) and
y.module = “AlignWarp” and
y.parameter(“model”) = “12”

Why not query languages?

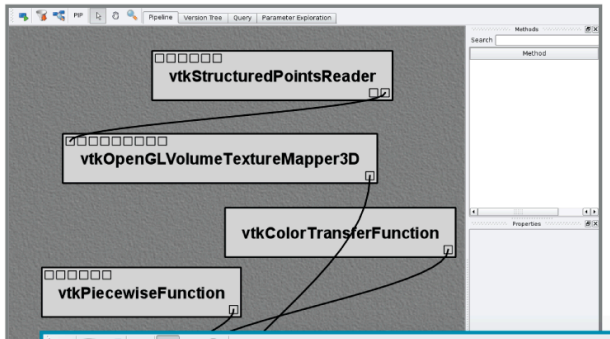
```
Select ExecutableWorkflowId, Execution_Event.ExecutionId, Event.EventId,
Execution_Event.ExecutableWorkflow_ExecutableActivityId
from Execution, Execution_Event, Event, Event_Property_Value, Property, Va
where Value=Cast('C:\TEMP\atlas-x.gif' as binary) and
Event_Property_Value.PropertyId=Property.PropertyId and Event_Property_Va
and Event.EventId=Event_Property_Value.EventId and Execution_Event.EventI
Execution_Event.ExecutionId=Execution.ExecutionId;
```

This is still only mildly better than straight SQL... Does not expose mapping to relational schema

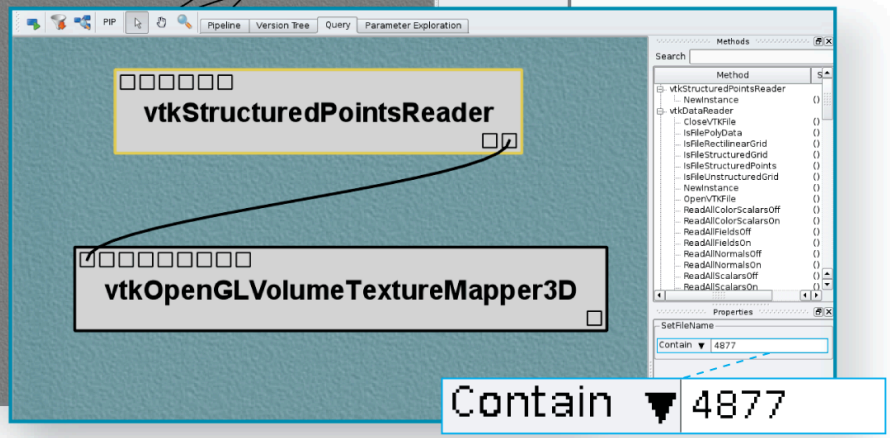
```
wf{*}: upstream(x) union x where
x.module = “SoftMean” and executed
(x) and y in upstream(x) and
y.module = “AlignWarp” and
y.parameter(“model”) = “12”
```

Query-by-Example

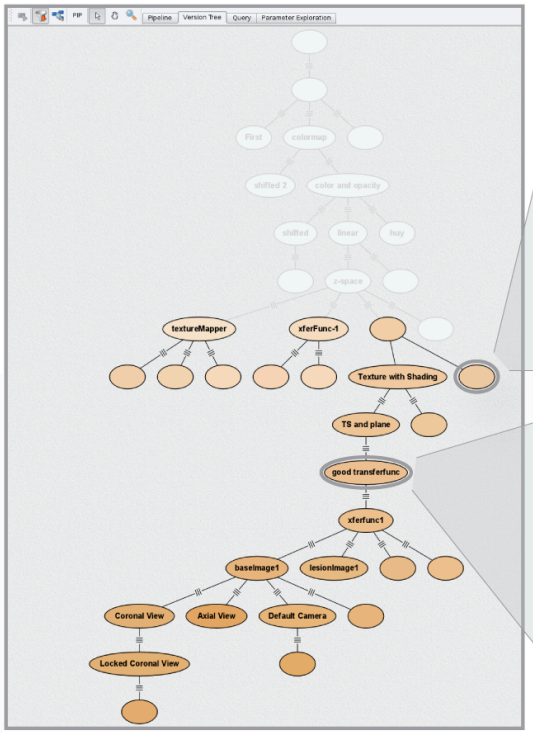
Pipeline Interface

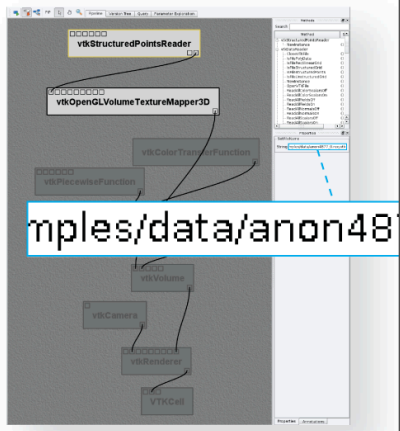


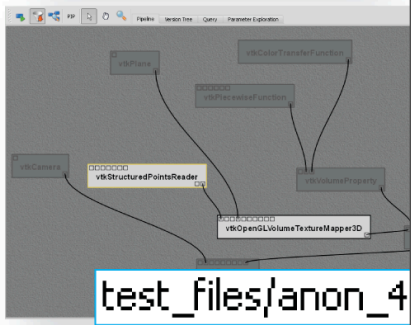
Query Interface



Query Result

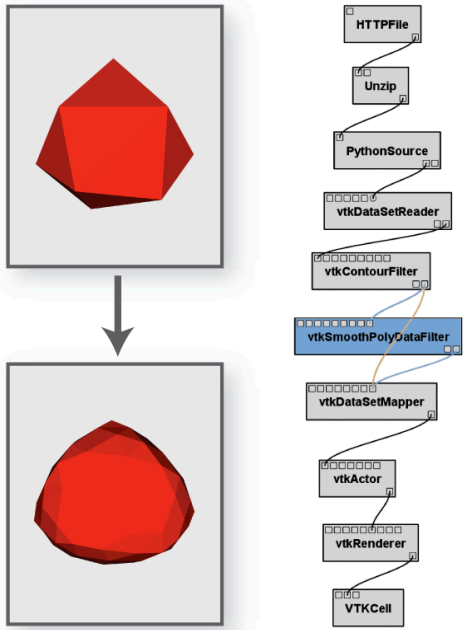




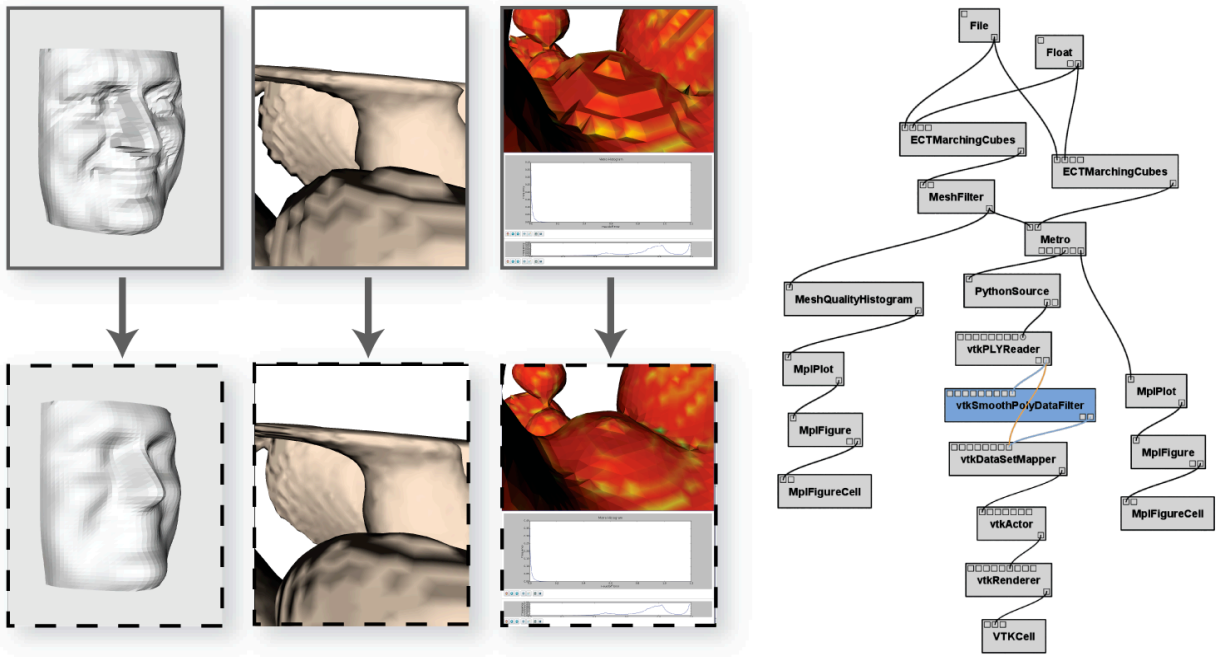


- Do not teach the user new forms of interaction!

Visualization by Analogy



Analogy Template



Automatically constructed visualizations

- Create new visualizations by saying "do as they did"
  - Specify **what**, not **how**

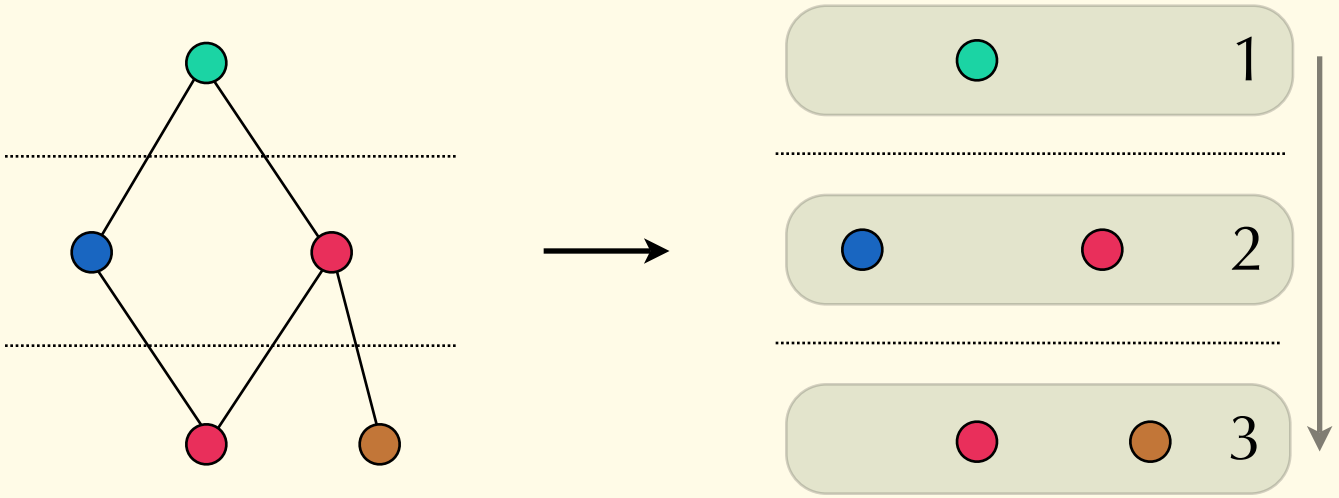
www.vistrails.org

Query-by-Example

- Trivially reducible from MAX-CLIQUE
  - ... and MAX-CLIQUE is NP-Complete
  - ... and MAX-CLIQUE is fundamentally hard to approximate
- Solution: algorithm tailored to problem domain

Query-by-Example

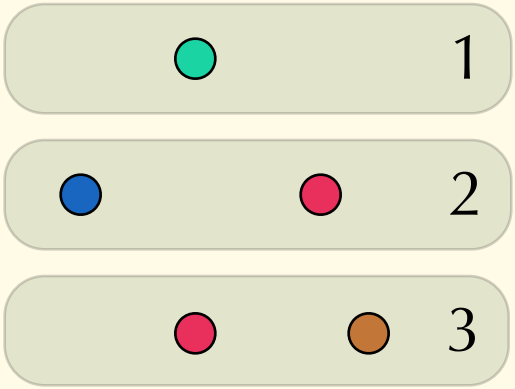
- Split every subgraph in topologically sorted layers
  - Ok, since all pipelines are DAGs in VisTrails



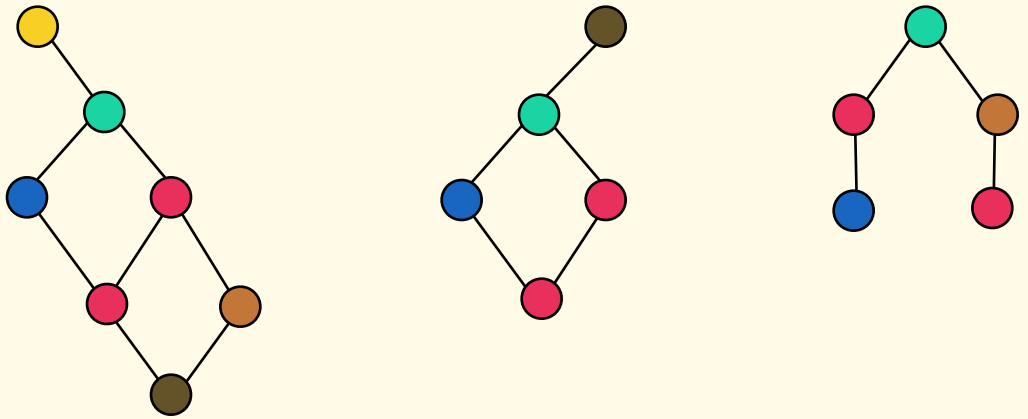
Query-by-Example

- Now search for layers that are connected in the same way in the database

Query

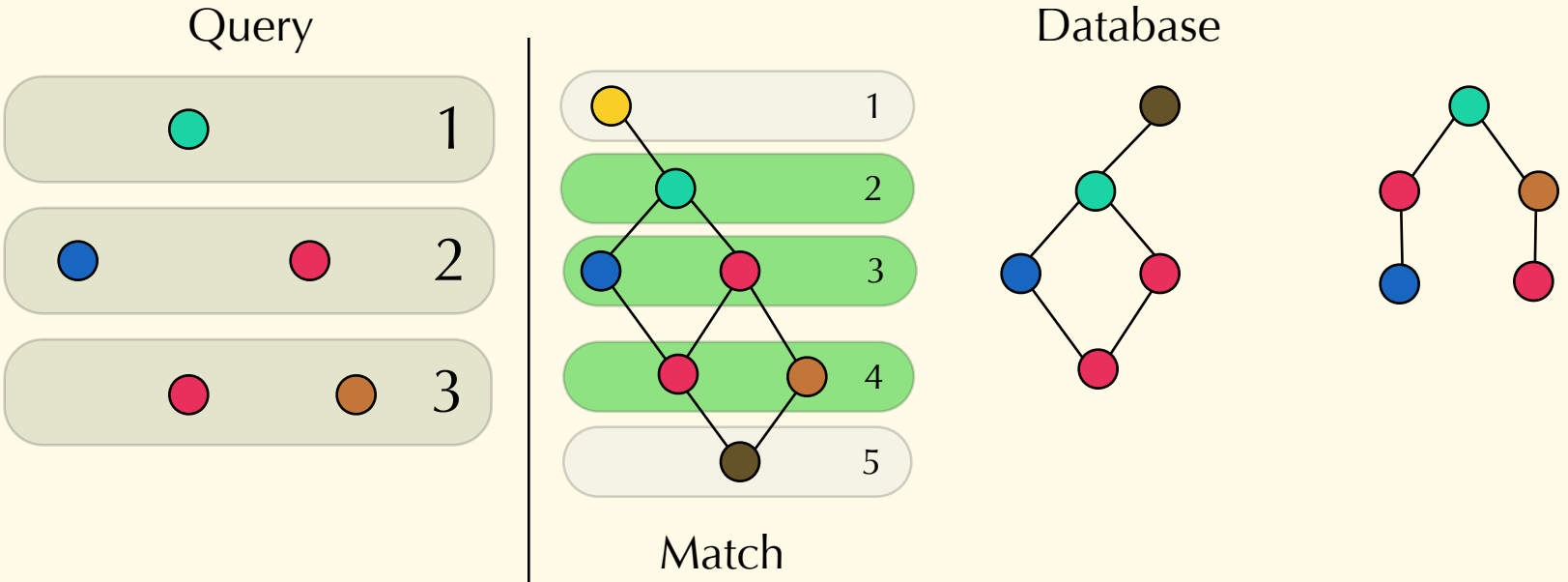


Database



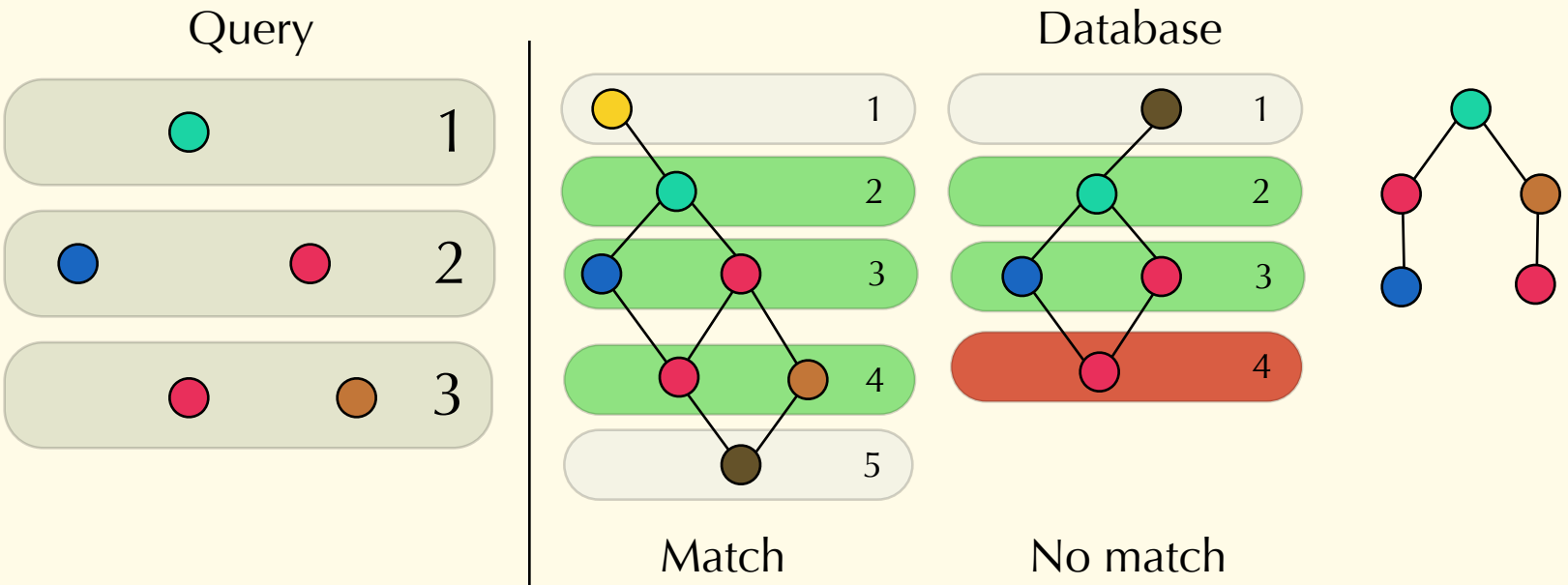
Query-by-Example

- Now search for layers that are connected in the same way in the database



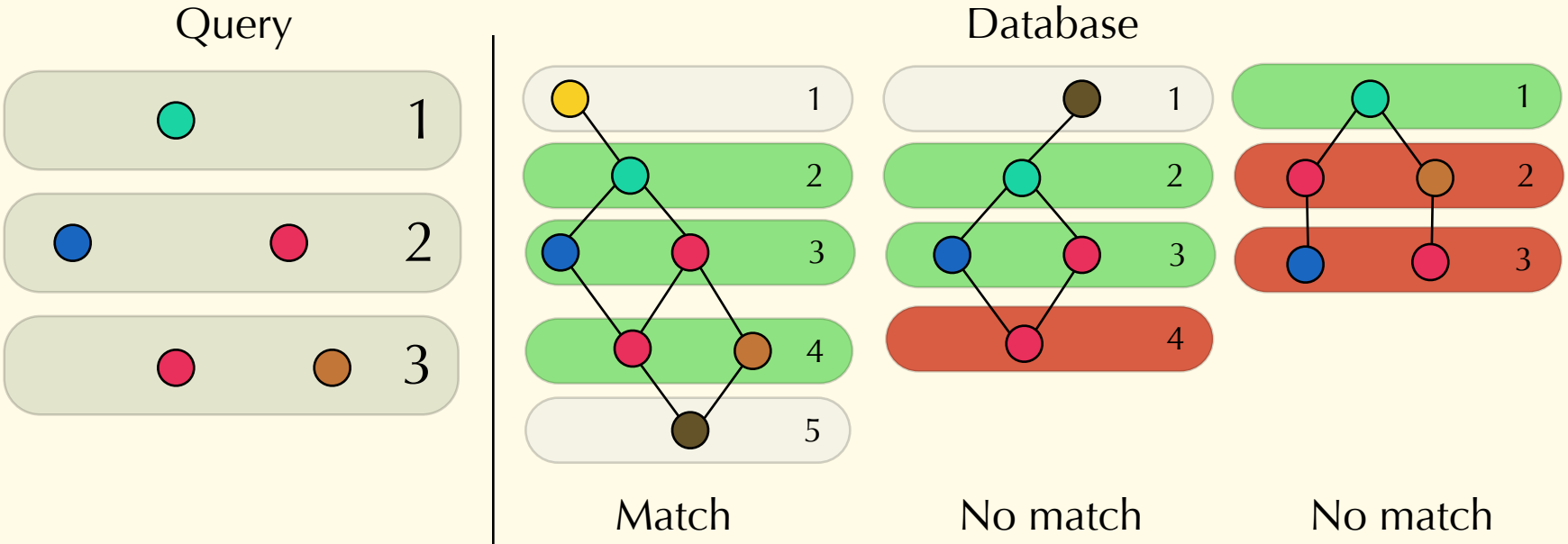
Query-by-Example

- Now search for layers that are connected in the same way in the database



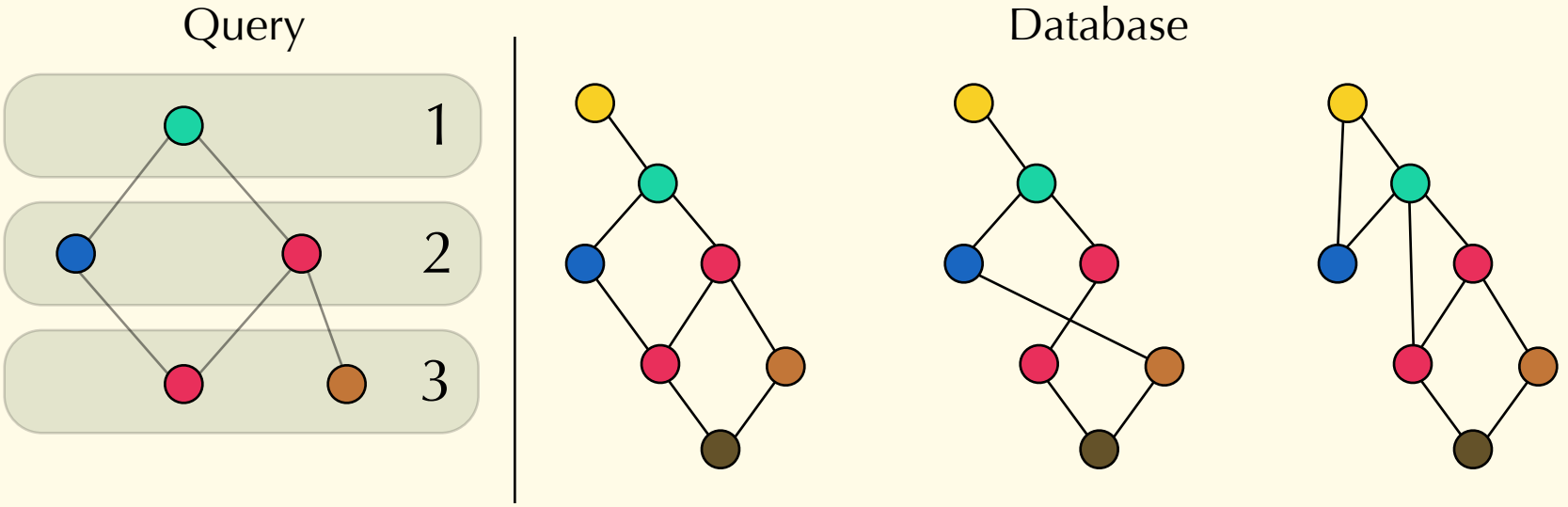
Query-by-Example

- Now search for layers that are connected in the same way in the database



Query-by-Example

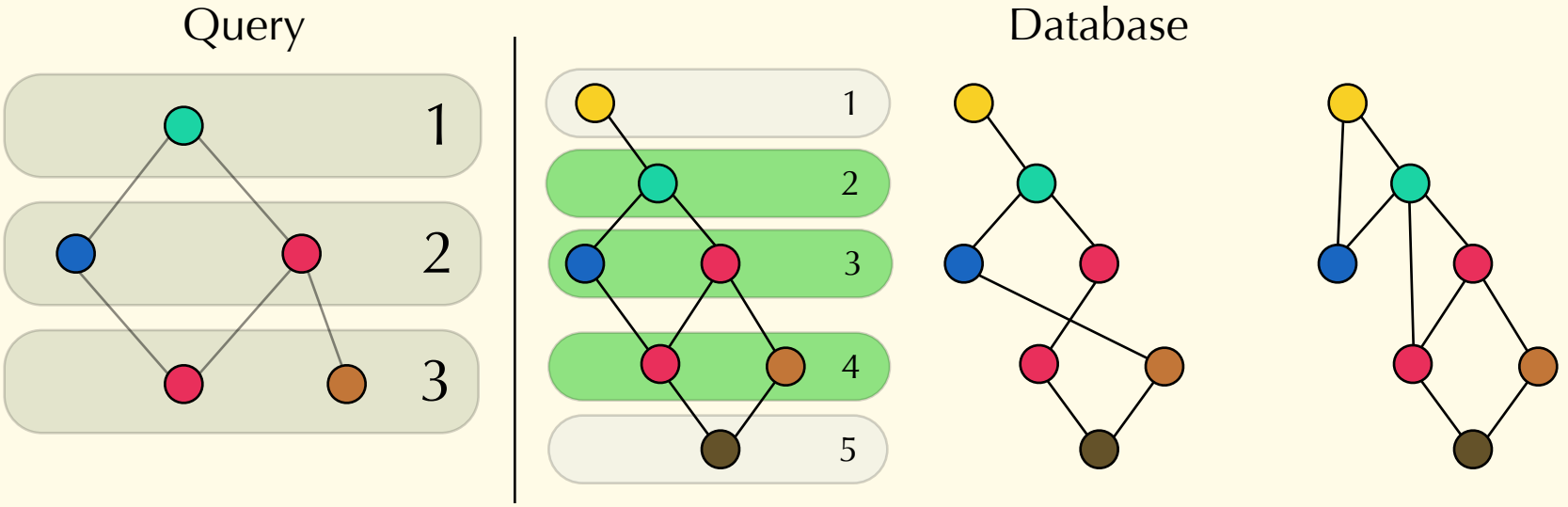
- Might return false positives - it ignores the particular connectivity between topological layers



- Not too harmful - most modules cannot connect to one another

Query-by-Example

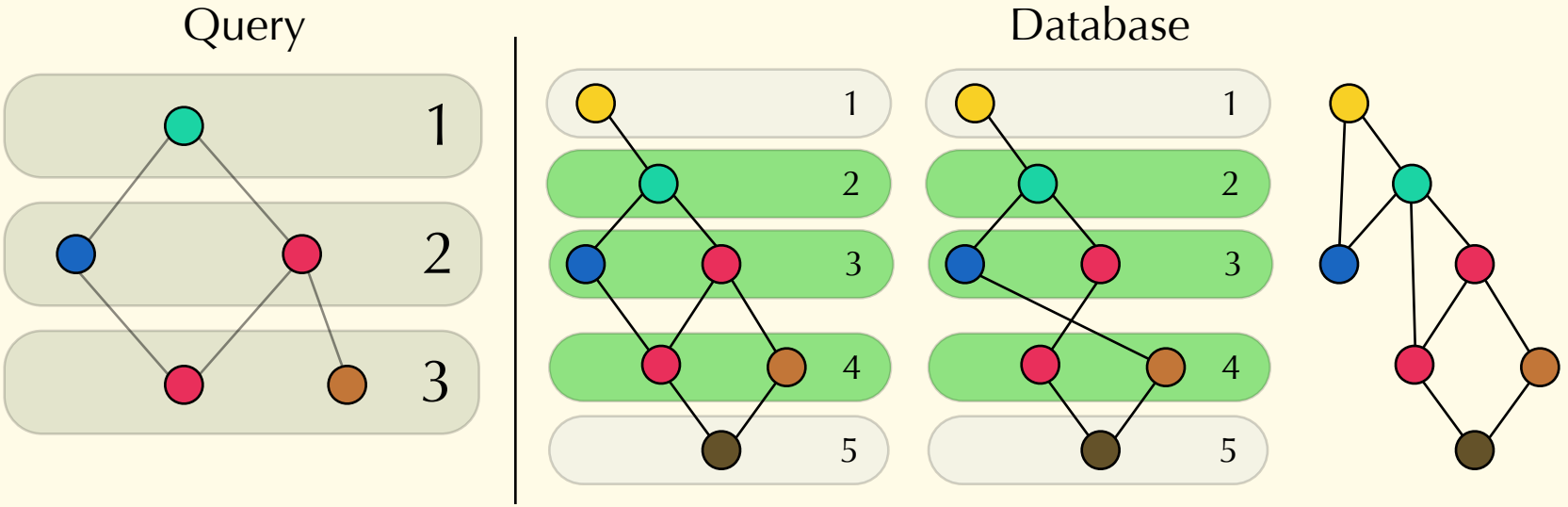
- Might return false positives - it ignores the particular connectivity between topological layers



- Not too harmful - most modules cannot connect to one another

Query-by-Example

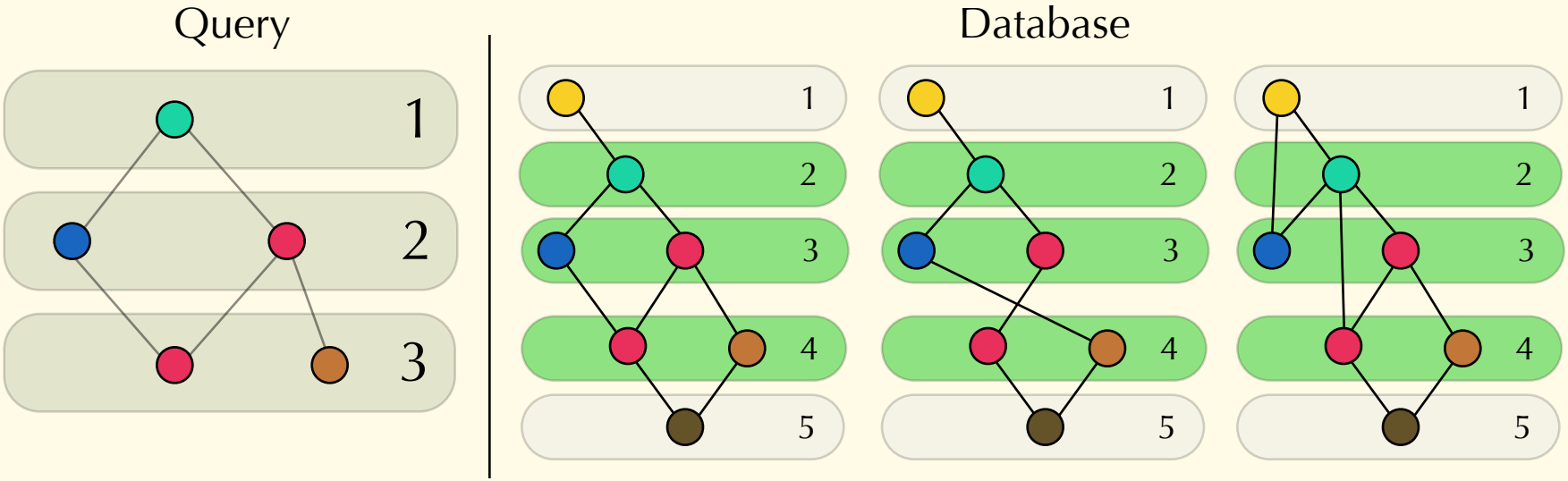
- Might return false positives - it ignores the particular connectivity between topological layers



- Not too harmful - most modules cannot connect to one another

Query-by-Example

- Might return false positives - it ignores the particular connectivity between topological layers



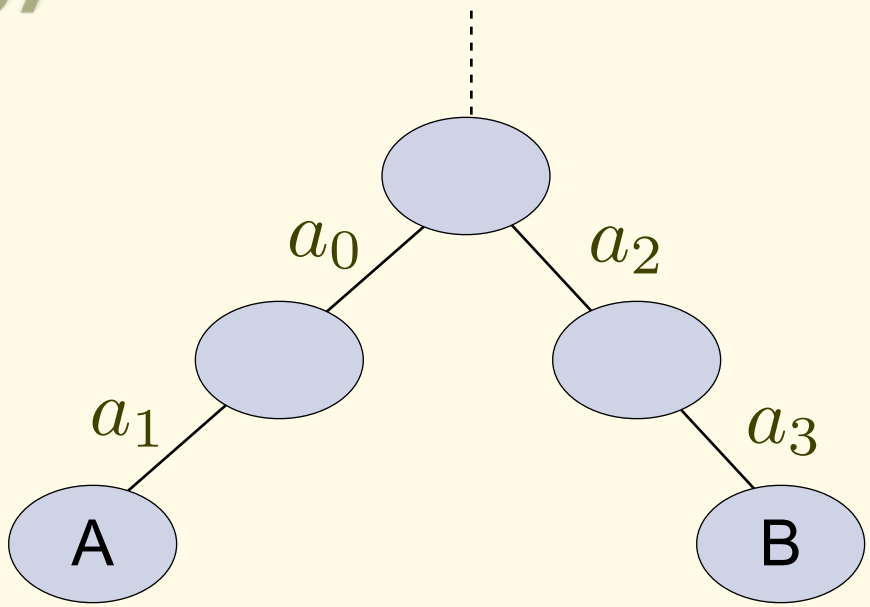
- Not too harmful - most modules cannot connect to one another

QBE Demo

Vistrail diffs

- A version tree stores a set of **actions**
  - Each action is a function on the set of all possible visualizations: $\mathcal{V} \rightarrow \mathcal{V}$
- $a_n \circ a_{n-1} \circ a_{n-2} \cdots \circ a_0$
- We can use those to determine the difference between visualizations
- Moving up, then down the version tree

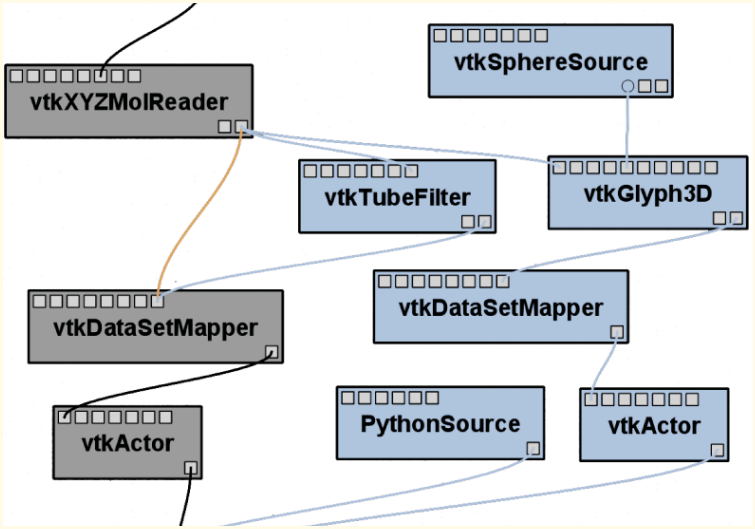
Vistrail diffs

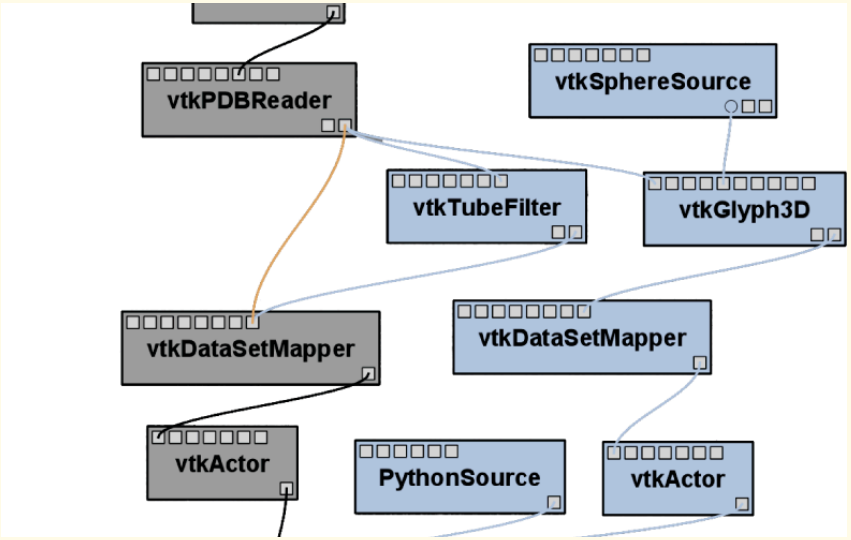


- Action to go from A to B is $a_3 \circ a_2 \circ a_0^{-1} \circ a_1^{-1}$

Visualization by Analogy

- A diff is a template: reapply it elsewhere





- How do we match two pipelines?

Algorithm Overview

- Compute the difference $\delta_{ab} = \Delta(p_a, p_b)$
- Compute the map $\mathsf{map}_{ac} = \mathsf{map}(p_a, p_c)$
- Apply map_{ac} to δ_{ab} $\delta^{\star}_{cb} = \mathsf{map}_{ac}(\delta_{ab})$
- Compute the new pipeline $p_d = \delta^{\star}_{cb}(p_c)$

Visualization by Analogy

- Simplest version is again reducible from MAX-CLIQUE
- We will now use a probabilistic argument to create a Markov chain

How does it work?

- Module compatibility: **prior**
  - $f : M^2 \rightarrow [0, 1]$
  - **Independent** of graph topology
- Probability of match between a pair
  - **Dependent** of graph topology
  - Linear combination of probability of match in the neighborhood pairs and data
- This is a Markov chain!

How does it work?

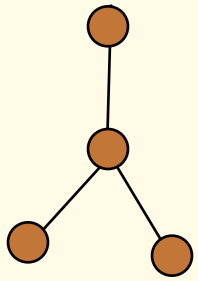
- Graph product G of the two input graphs
  - each vertex in G represents a possible match
  - similarity is then defined as

$$\pi = \alpha A(G)\pi + (1-\alpha)c(G)$$
$$= M_G\pi$$

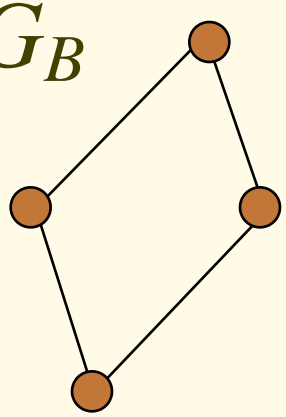
- π is an eigenvector of M_G
  - It is the limit distribution of the transition matrix

How does it work?

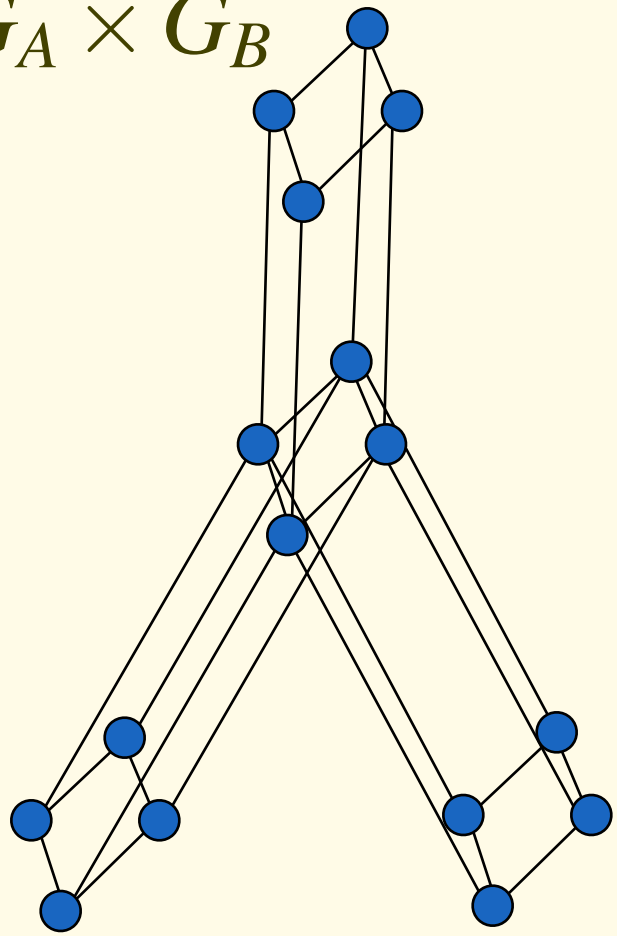
G_A



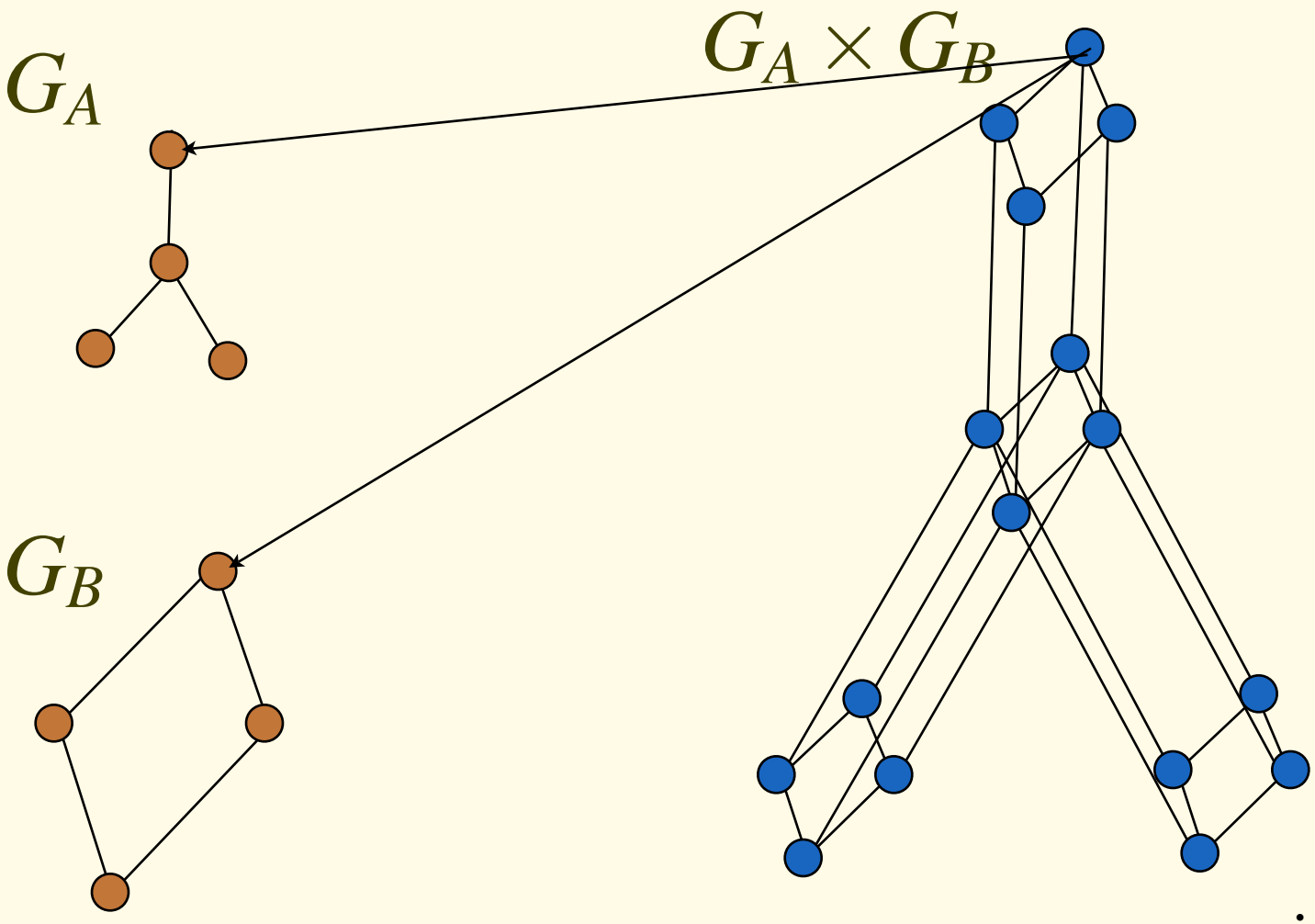
G_B



$G_A \times G_B$

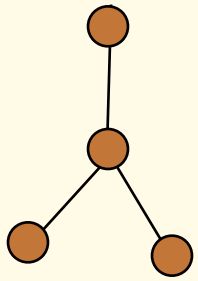


How does it work?

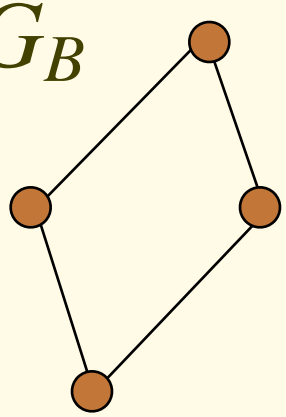


How does it work?

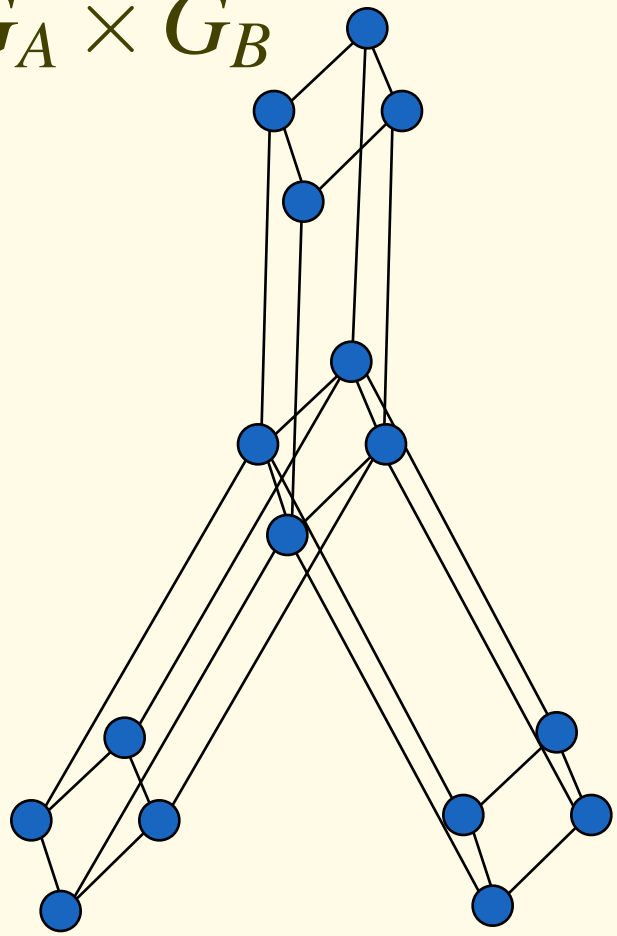
G_A



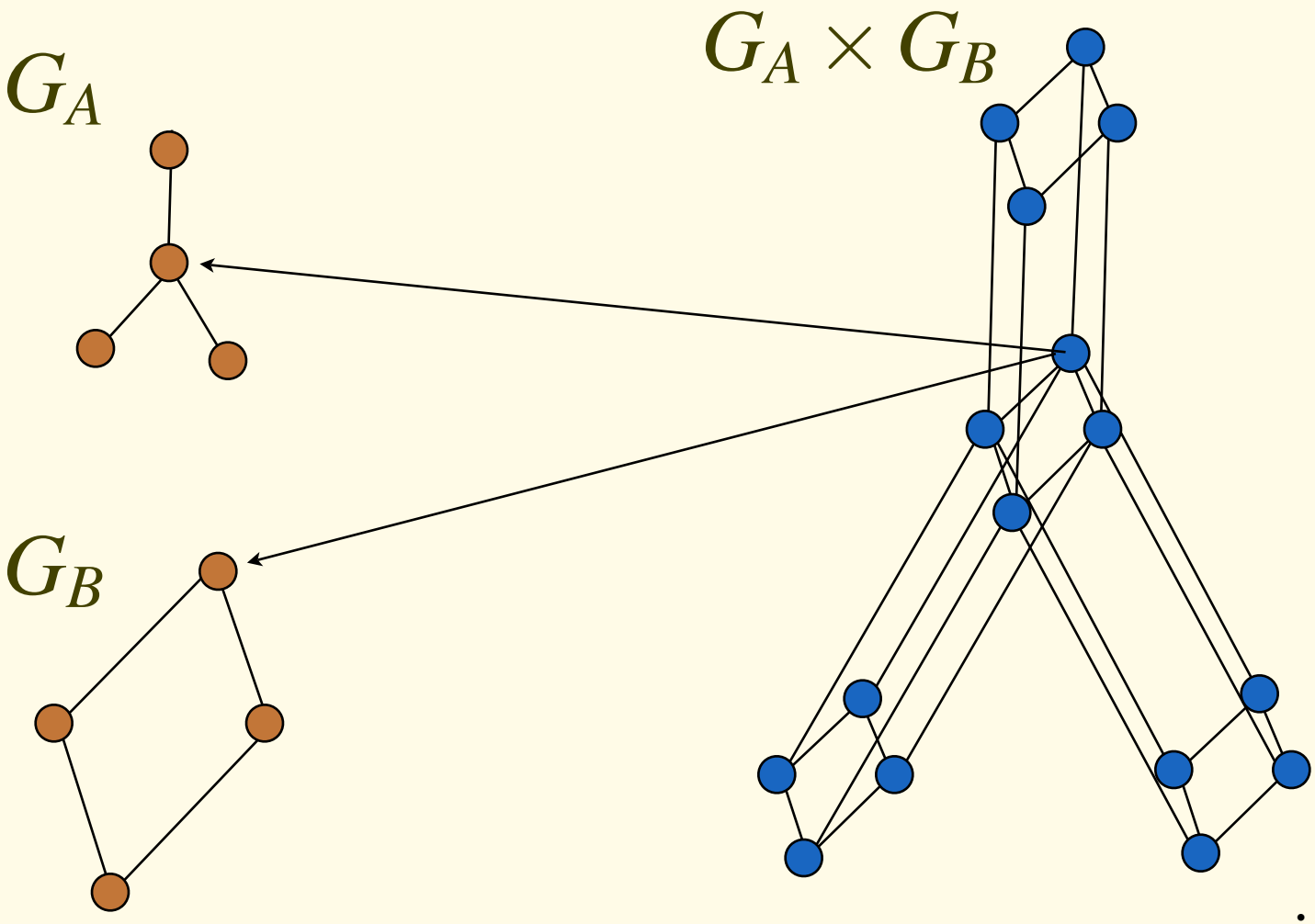
G_B



$G_A \times G_B$

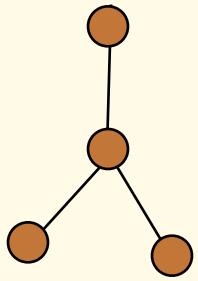


How does it work?

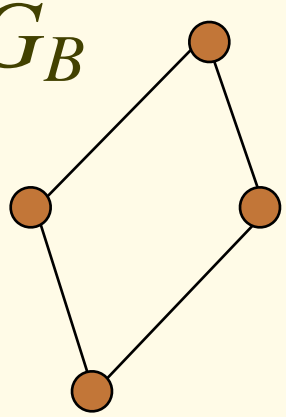


How does it work?

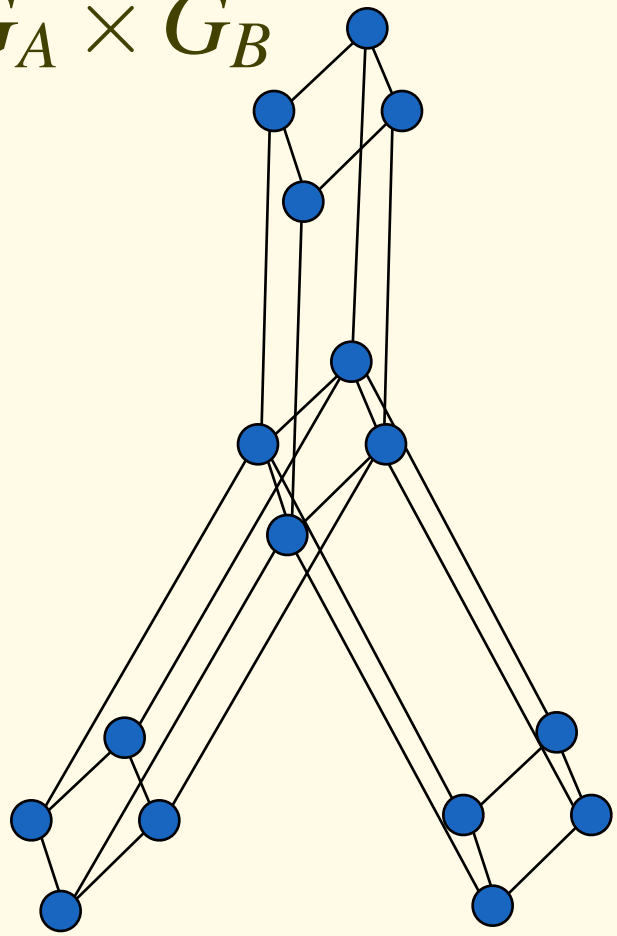
G_A



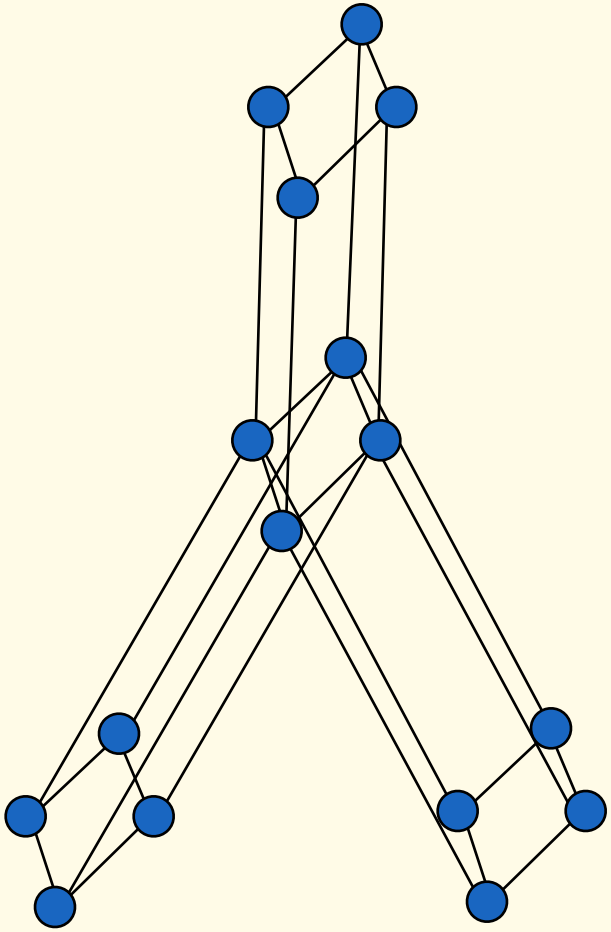
G_B



$G_A \times G_B$

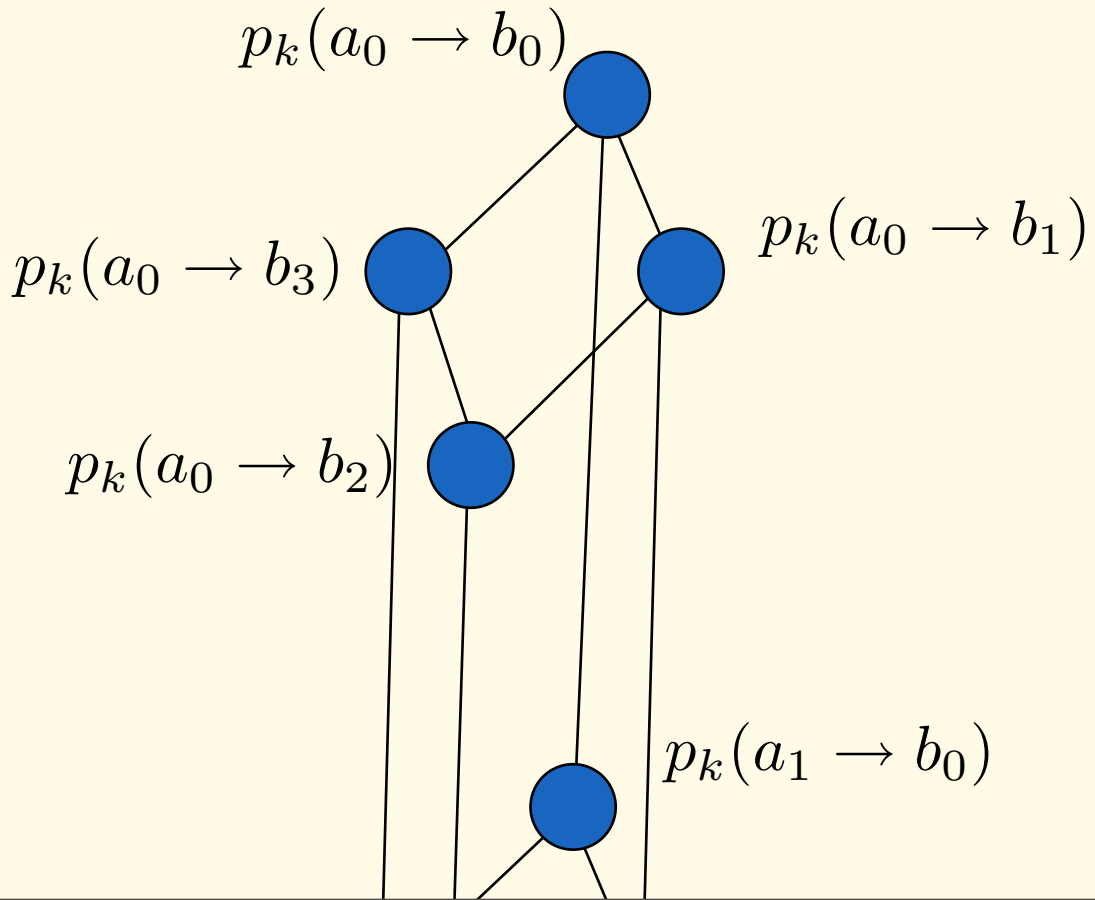


How does it work?



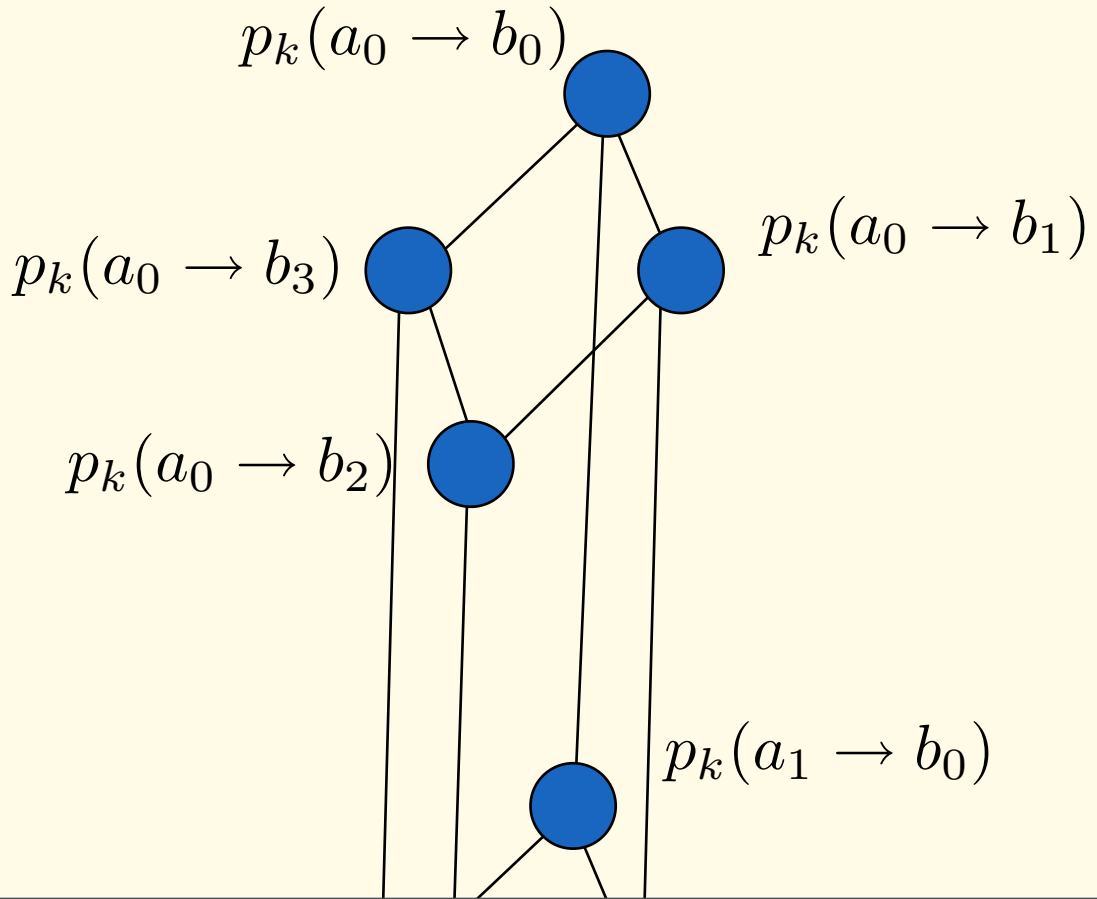
Each node is assigned some initial value. (It doesn't matter which, as long as the values sum to one!)

How does it work?



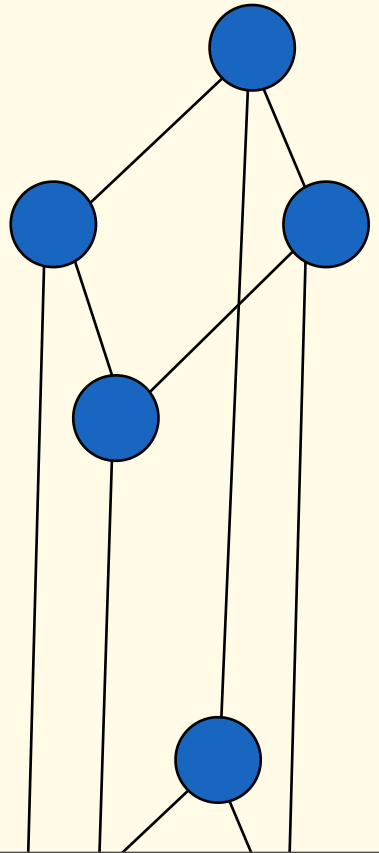
How does it work?

$$p_{k+1}(a_0 \to b_0) = (1-\alpha)c(a_0, b_0) + \alpha/3 \quad (p_k(a_0 \to b_3) + p_k(a_0 \to b_1) + p_k(a_1 \to b_0))$$



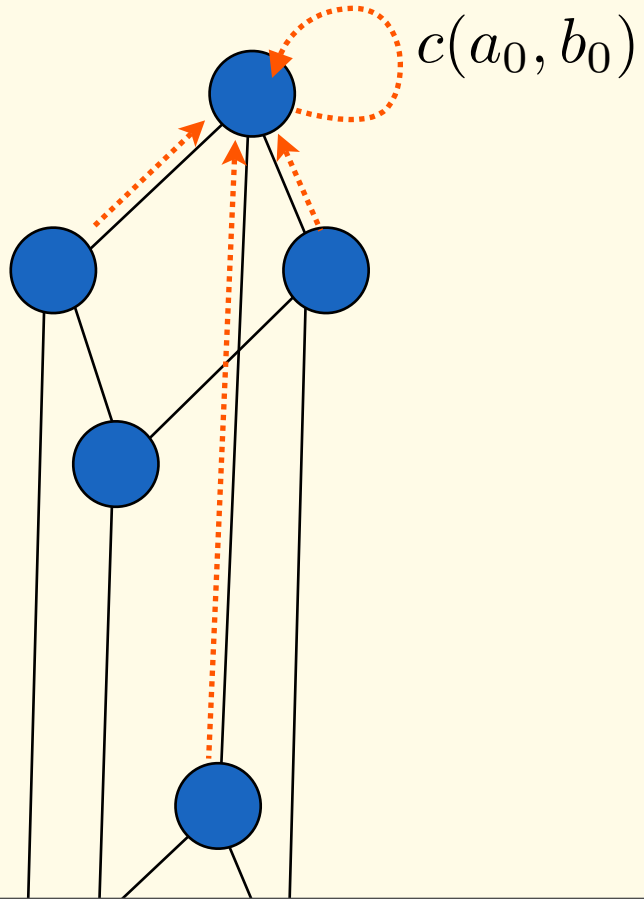
How does it work?

$$p_{k+1}(a_0 \to b_0) = (1-\alpha)c(a_0, b_0) + \alpha/3 \quad (p_k(a_0 \to b_3) + p_k(a_0 \to b_1) + p_k(a_1 \to b_0))$$



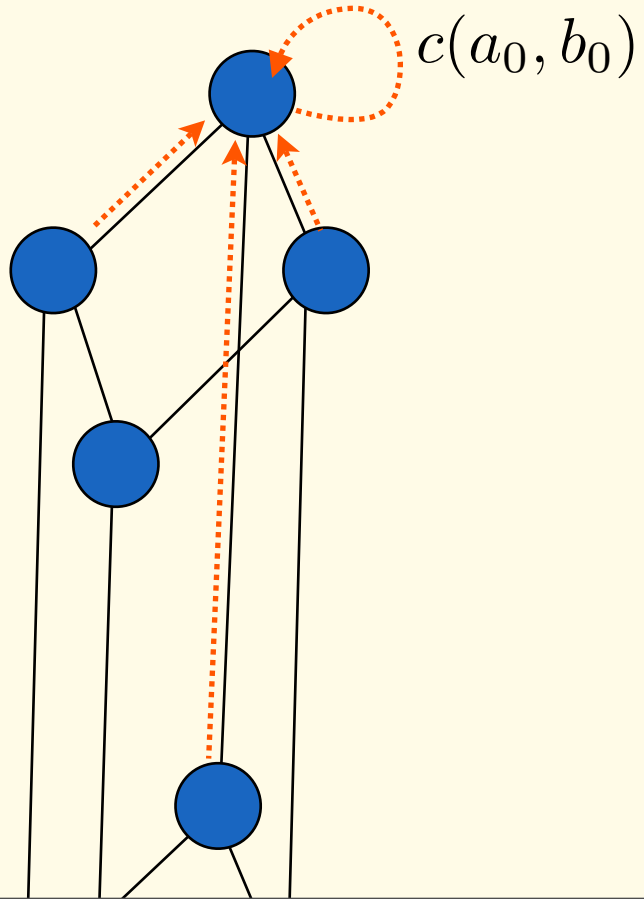
How does it work?

$$p_{k+1}(a_0 \to b_0) = (1-\alpha)c(a_0, b_0) + \alpha/3 \quad (p_k(a_0 \to b_3) + p_k(a_0 \to b_1) + p_k(a_1 \to b_0))$$



How does it work?

$$p_{k+1}(a_0 \to b_0) = (1-\alpha)c(a_0, b_0) + \alpha/3 \quad (p_k(a_0 \to b_3) + p_k(a_0 \to b_1) + p_k(a_1 \to b_0))$$



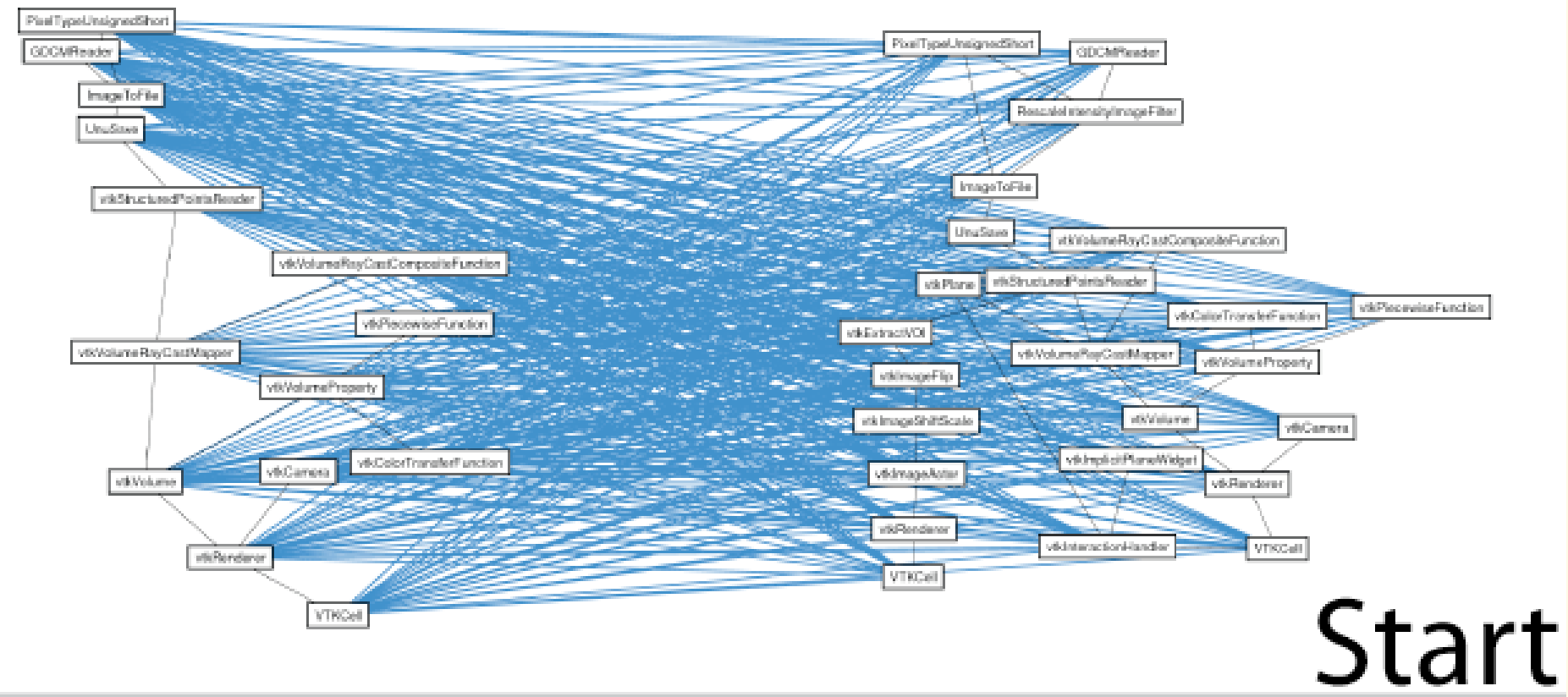
Do it for all nodes,
until convergence

How does it work?

- π is defined over graph product
- For each module in the second pipeline, pick maximal value of π on first pipeline: this is the match

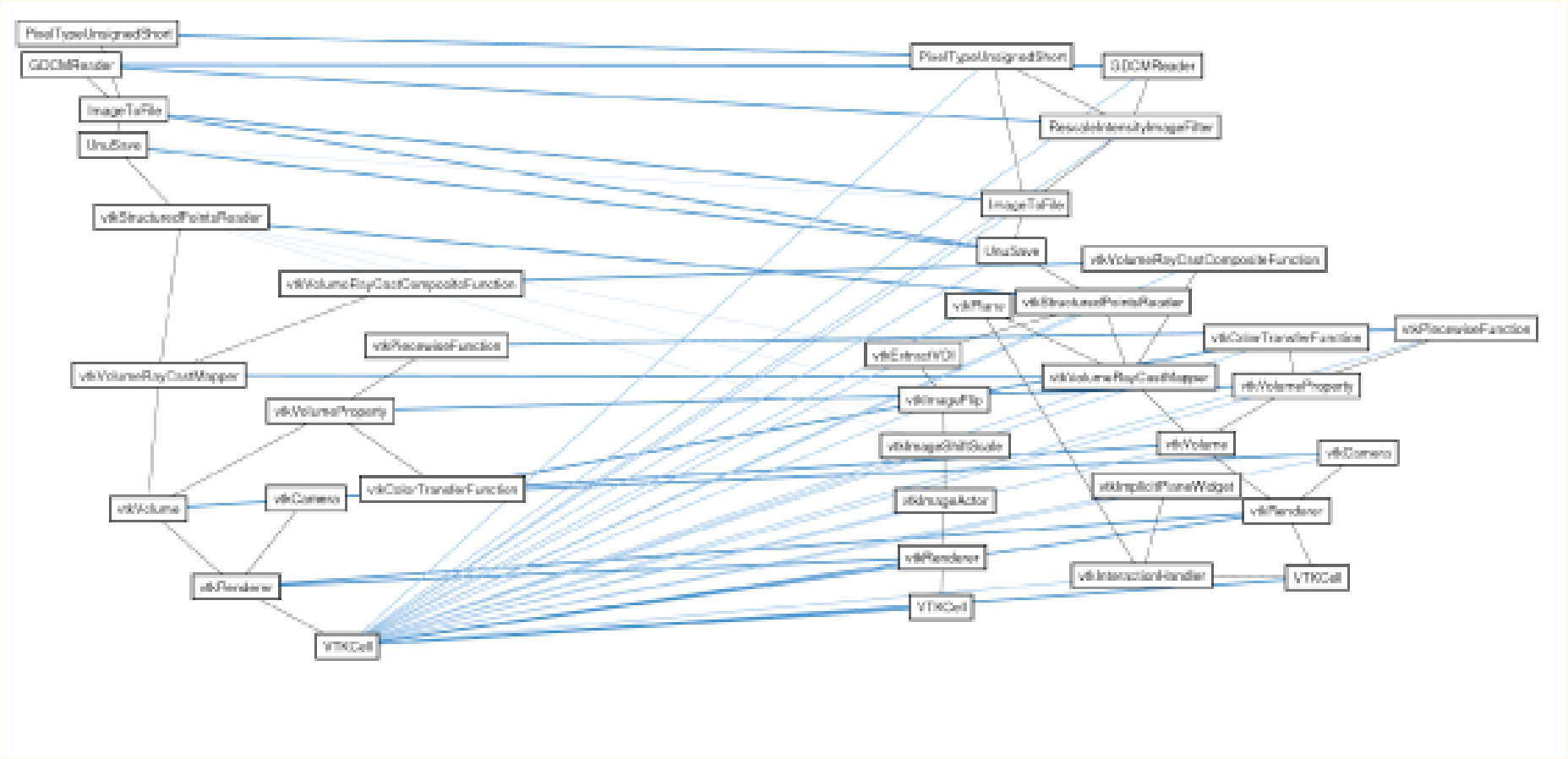
- Many others possible

The matching algorithm



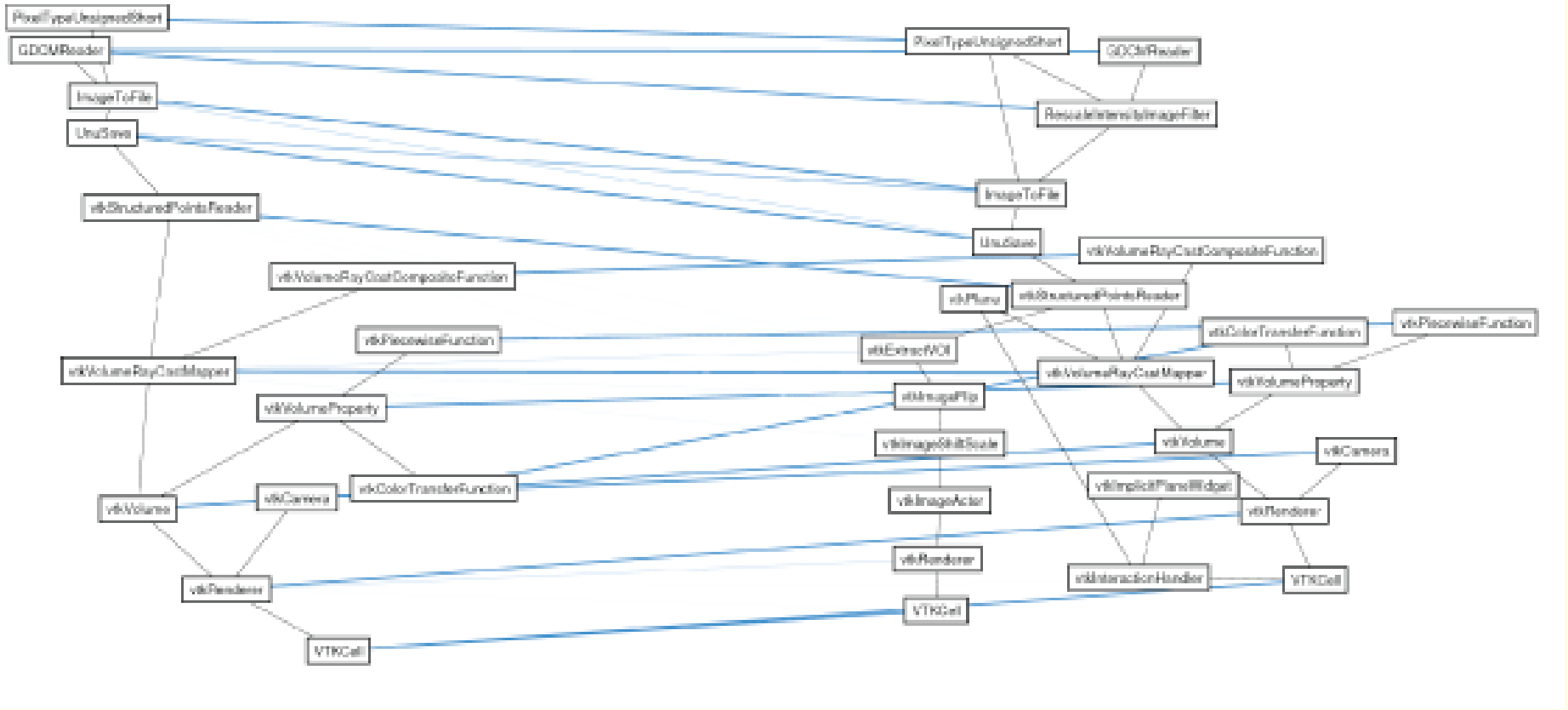
www.vistrails.org

The matching algorithm



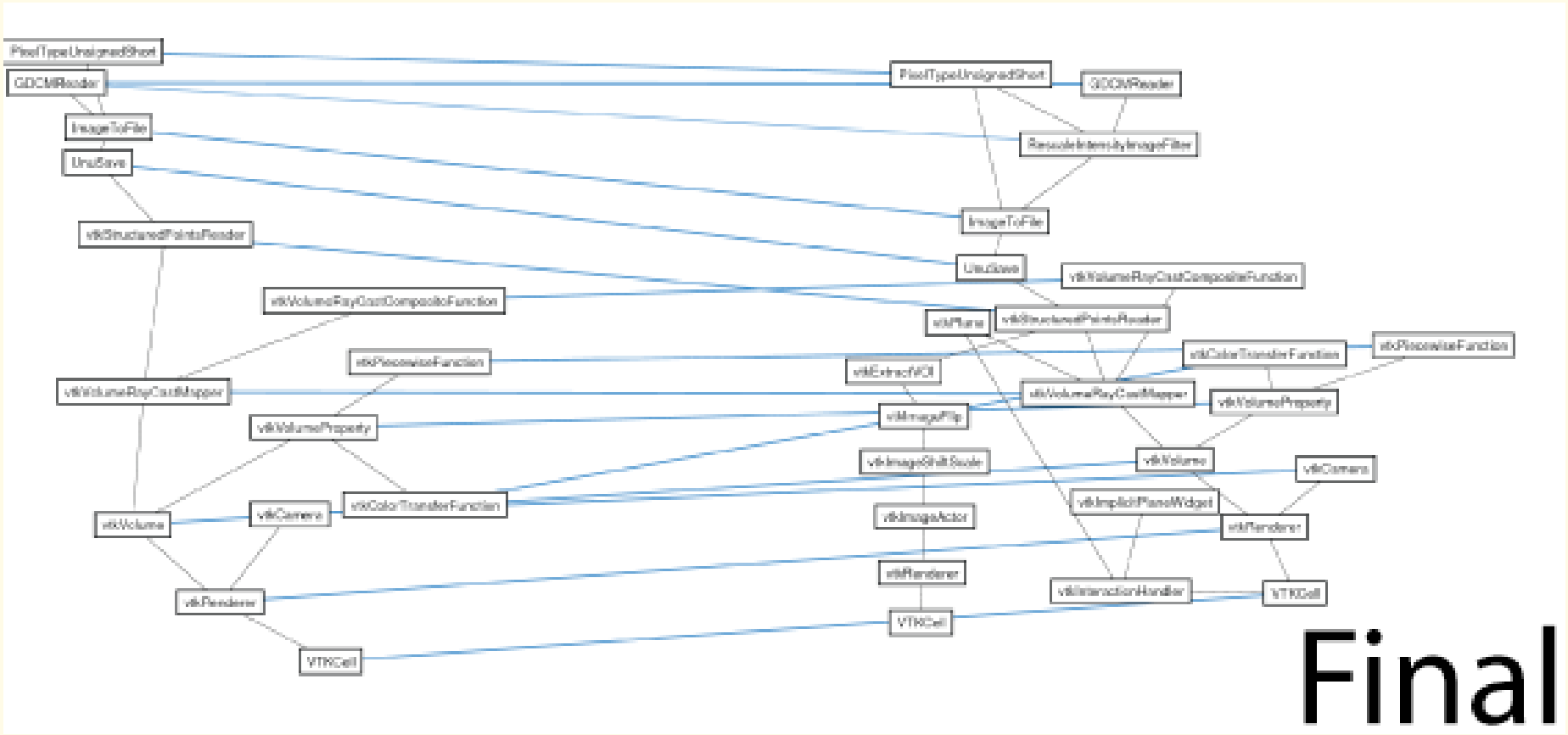
www.vistrails.org

The matching algorithm

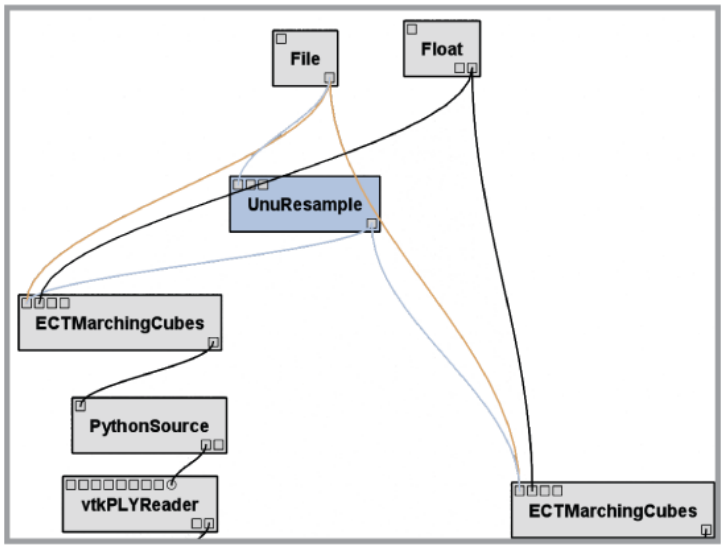


www.vistrails.org

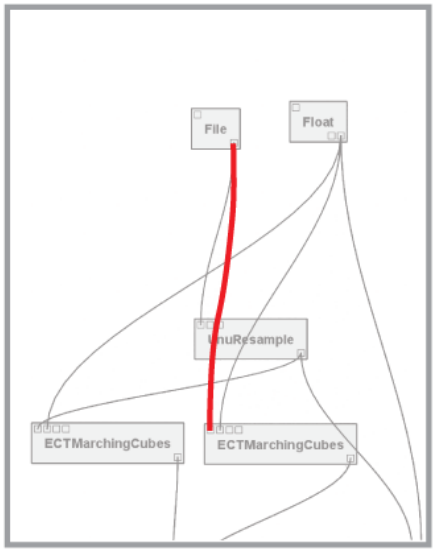
The matching algorithm



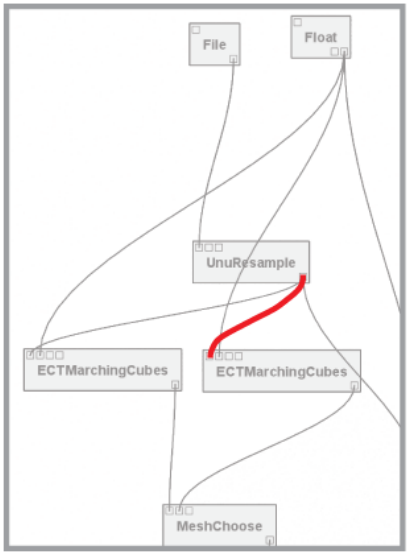
Failure Modes



Defined analogy



Produced result



Expected result

- Analogies are not fool-proof

Case study

- Creating a complex visualization out of simple ones

- (demo)

Discussion

- If your system can encode actions as functions on the space of objects of interest, store these explicitly
- That will be your "version tree" - everything else is just the same

- Easy to incorporate domain-specific knowledge in analogies: change $A(G)$ and $c(G)$

Acknowledgments

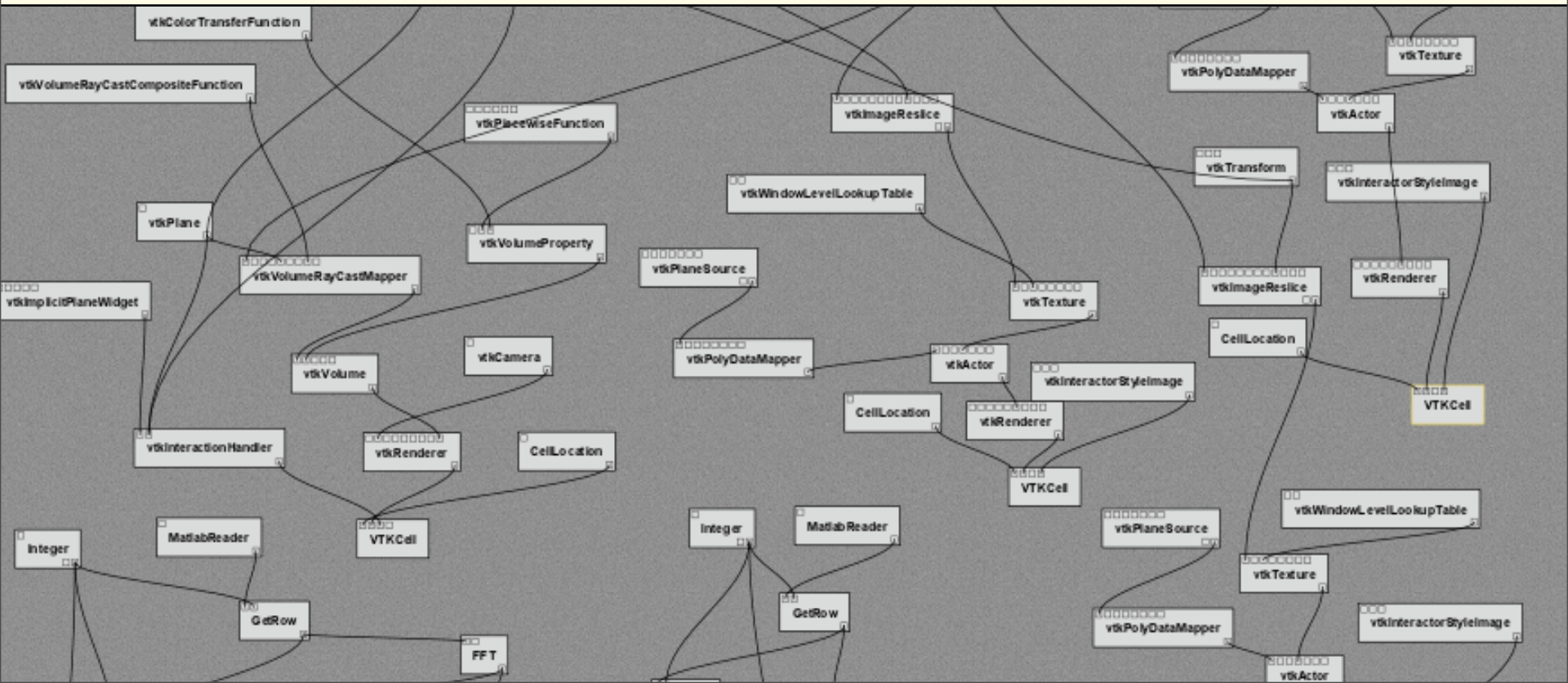
- Sarang Joshi, Suresh Venkatasubramanian, Erik Anderson, João Comba
- VisTrails dev team
- Many open source packages and devs: VTK, SciPy, teem, matplotlib
- VisTrails is open source! **http://www.vistrails.org**
  - **Shameless plug: Visit the SCI booth!**
- NSF, DOE, IBM Faculty Award

Thank you!

- Questions?

Too much data

- We are better off with visualization systems than without - but it's still pretty messy



Video

Creating and Querying Visualizations by Analogy

SCI Institute, School of Computing

UNIVERSITY OF UTAH