SVS Release + Relevant Info

Aaron Mininger

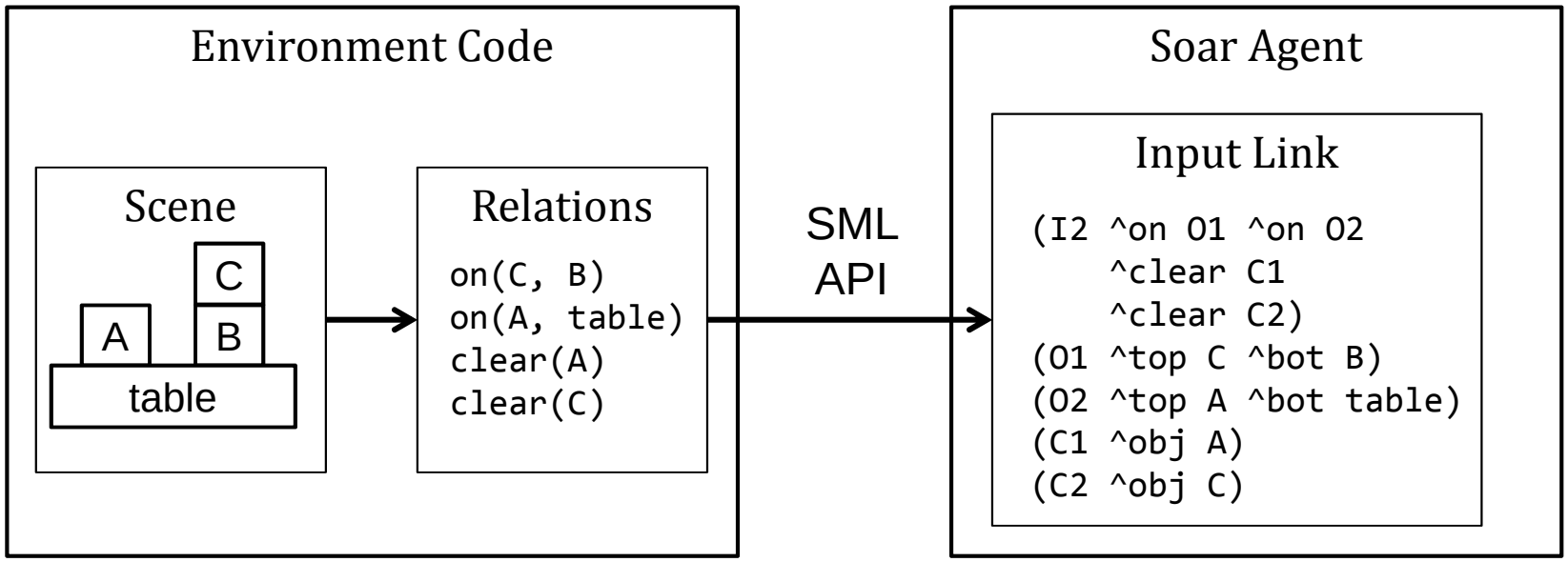
Overview

- Provides a general framework for Soar to reason about continuous environments
- Environment state is represented as 3D scene graph
- Has working memory interface similar to EpMem and Smem

```
(S1 ^svs S3)
(S3 ^spatial-scene S4 ^command C3)
```

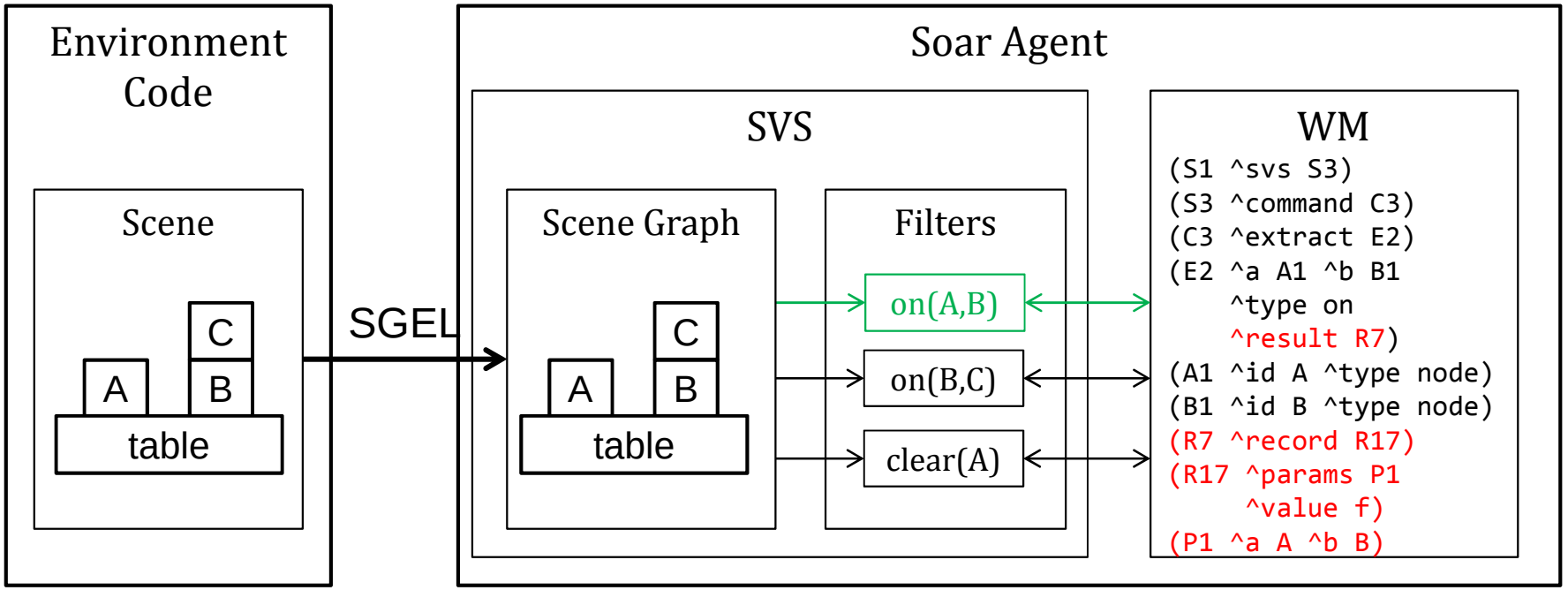
Typical Environment Setup

- Environment reports state with task-specific representation
- All possibly important relations are reported all the time



With SVS

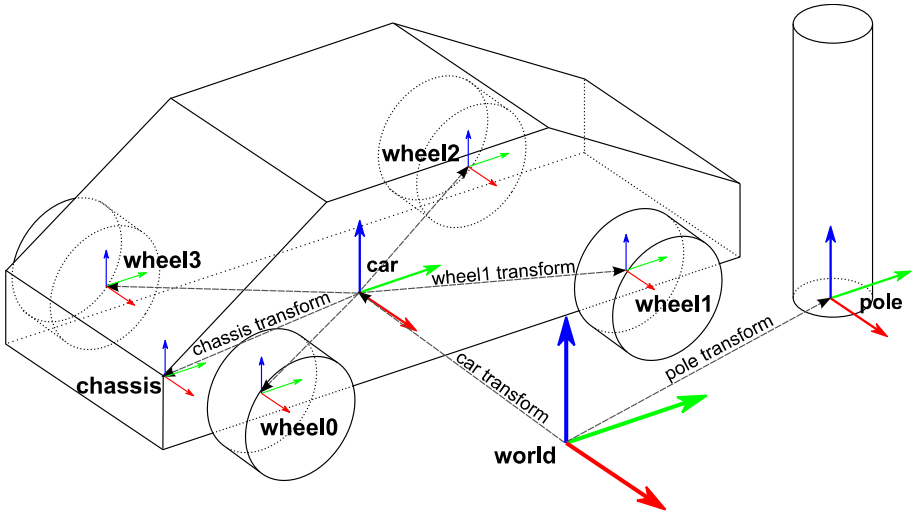
- Environment reports state with task-agnostic language (Scene Graph Edit Language)
- Agent queries only relations of interest
- Relations fixed across environments

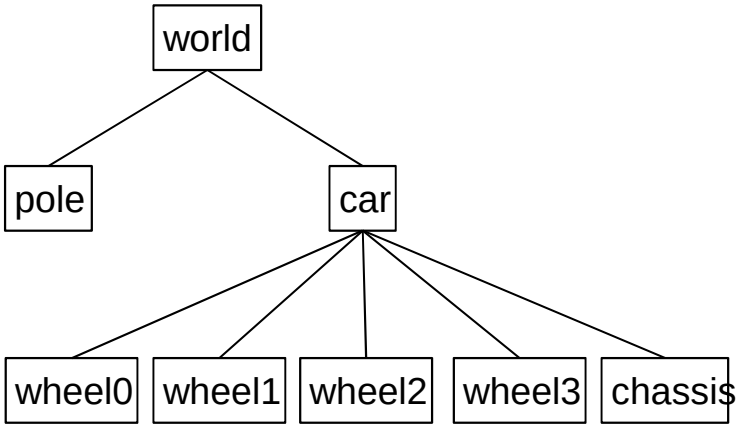


Scene Graph

- Organizes objects as tree of nodes
- Child nodes are a part of the parent node
  - Group nodes
  - Geometry nodes
- Each node as position, rotation, transform
- Copied to each substate

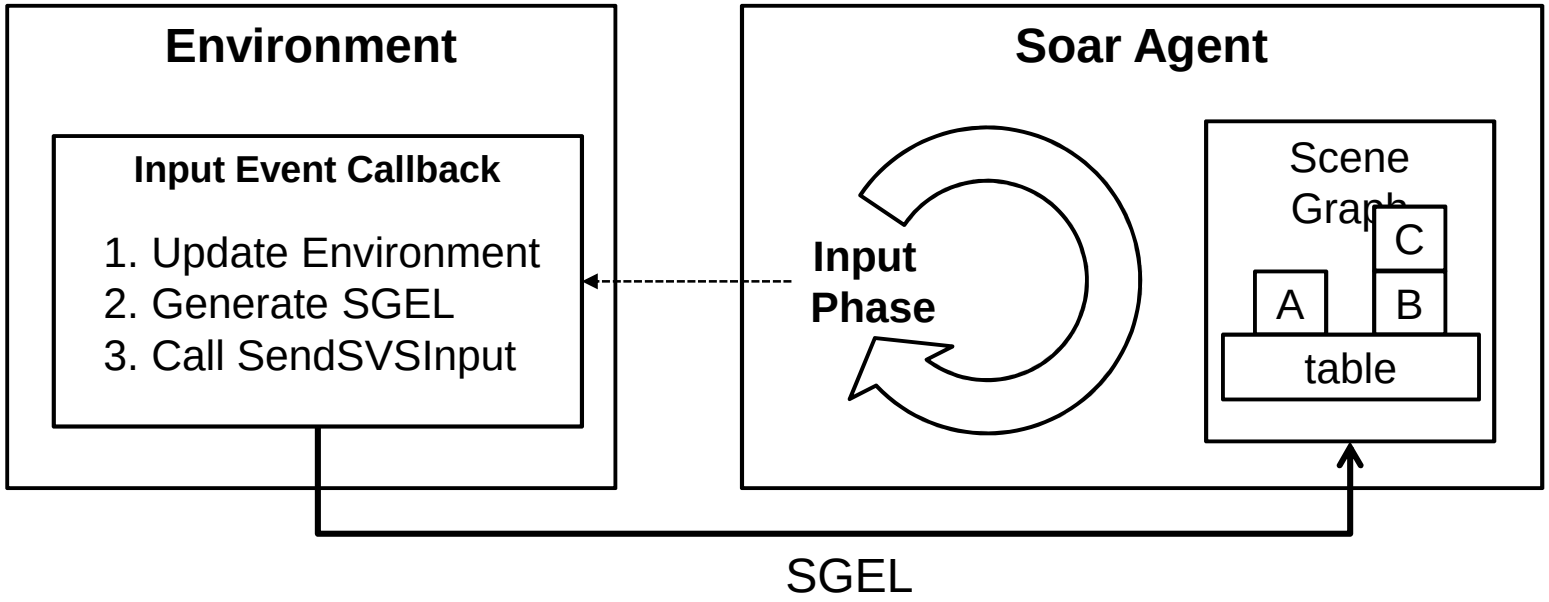
```
(S1 ^svs S3)
(S3 ^command C3 ^spatial-scene S4)
(S4 ^id world ^child C1 ^child C2)
(C1 ^id pole)
(C2 ^id car ^child C3 ^child C4
    ^child C5 ^child C6 ^child C7)
(C3 ^id wheel0)
(C4 ^id wheel1)
(C5 ^id wheel2)
(C6 ^id wheel3)
(C7 ^id chassis)
```



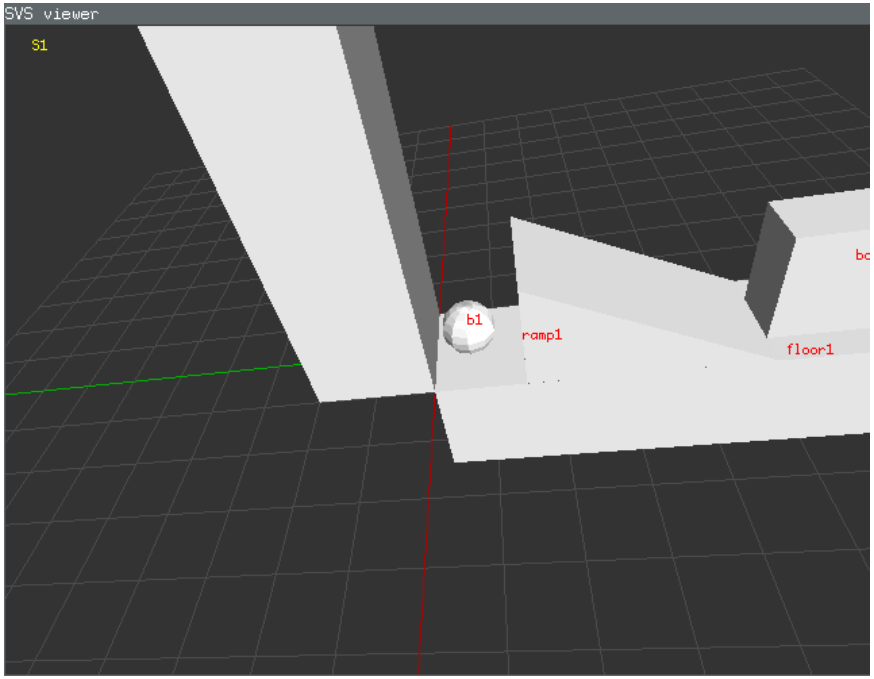


SML Interface

- SML environment updates SVS scene graph once per decision cycle via Agent::SendSVSInput
- Commands are text strings in the Scene Graph Edit Language



SVS Viewer



- Displays scene graph contents
- Separate program that SVS communicates with via TCP sockets
- Run program

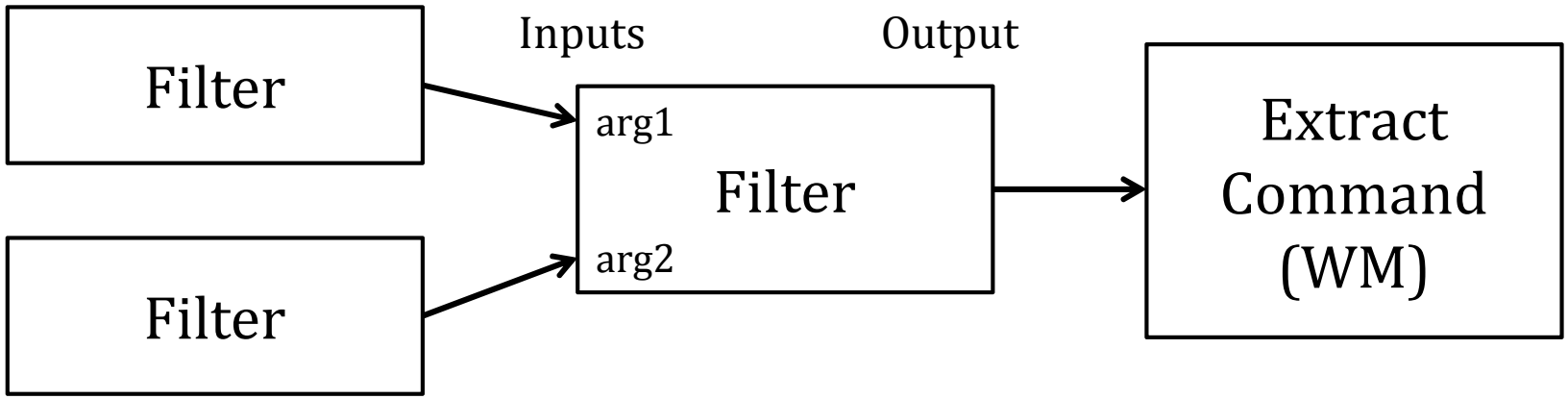
  ```
  svs_viewer -s PORT
  ```

- Tell SVS to connect

  ```
  svs connect_viewer PORT
  ```

Filters

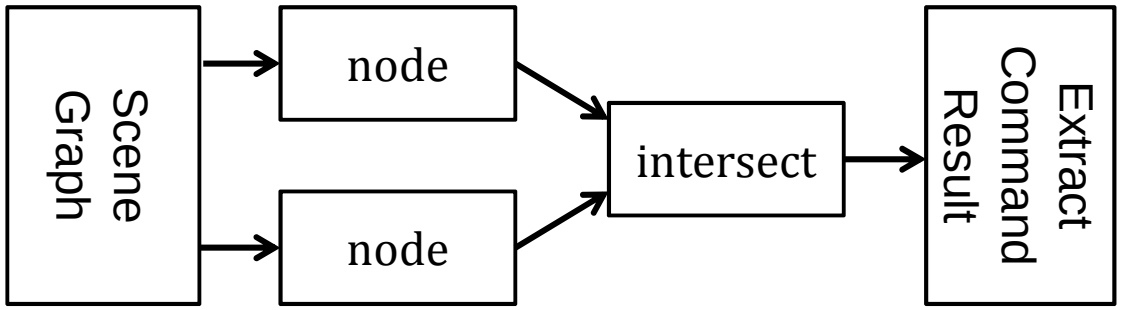
- Transforms continuous information from scene graph into symbolic information in working memory
  - Implements spatial relations, among other things
- Can be combined into pipeline
- Caches results and avoids recomputation when possible



Working Memory Interface

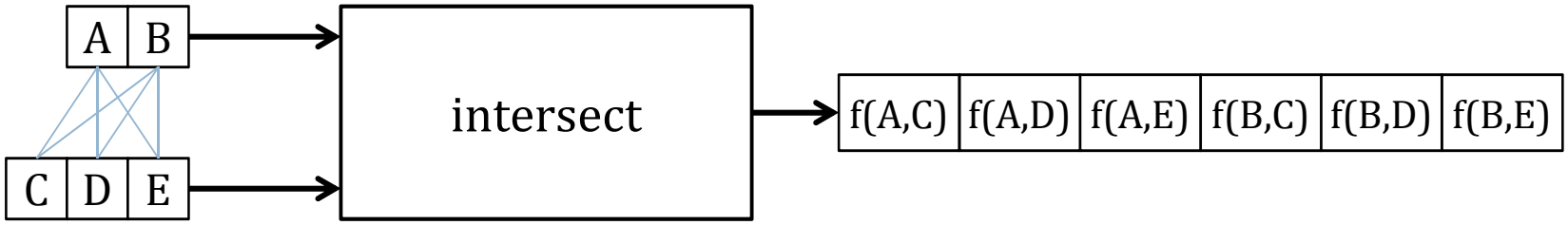

```
(S1 ^svs S3)
(S3 ^command C3 ^spatial-scene S4)
(C3 ^extract E2)
(E2 ^a A1 ^b B1 ^type intersect)
(A1 ^id b1 ^type node)
(B1 ^id b2 ^type node)
```

```
(S1 ^svs S3)
(S3 ^command C3 ^spatial-scene S4)
(C3 ^extract E2)
(E2 ^a A1 ^b B1 ^type intersect ^result R7
    ^status success)
(A1 ^id b1 ^type node ^status success)
(B1 ^id b2 ^type node ^status success)
(R7 ^record R17)
(R17 ^params P1 ^value f)
(P1 ^a b1 ^b b2)
```



Multiple Inputs to Filters

- Sometimes you want to run a filter on many objects
  - Example: Does X intersect anything?
  - It's annoying and inefficient to make a filter for every pair of objects
- Filters can take multiple objects as inputs and outputs
- Different combination methods:
  - Cartesian product (most common)
  - Grouped
  - Flattened



Some Built-in Filters

Filter	Parameters	Type	Output type
node	id	map	node
all_nodes	none	special	node
[xyz]-greater-than	a, b	map	boolean
[xyz]-less-than	a, b	map	boolean
[xyz]-aligned-than	a, b	map	boolean
on-top	a, b	map	boolean
intersect	a, b	map	boolean
distance	a, b	map	float
closest	a, b	rank	node
smaller-than	a, b	map	boolean

Commands

- Agent issues SVS commands via command link
- “add_node”, “property” commands performed on substate scene graph copies allows for look-ahead search

extract	Show the results of filter pipelines in working memory
add_node	Add a new node (generated by filter pipeline) into the scene graph
property	Change the position, rotation, scaling of existing objects
project	Find position that satisfies spatial relation to an object

Release Info

SVS is Coming!

- Upcoming release of Soar 9.4 by August 1
- Includes an addition to the Soar Manual
- New Soar Tutorial

Conclusion

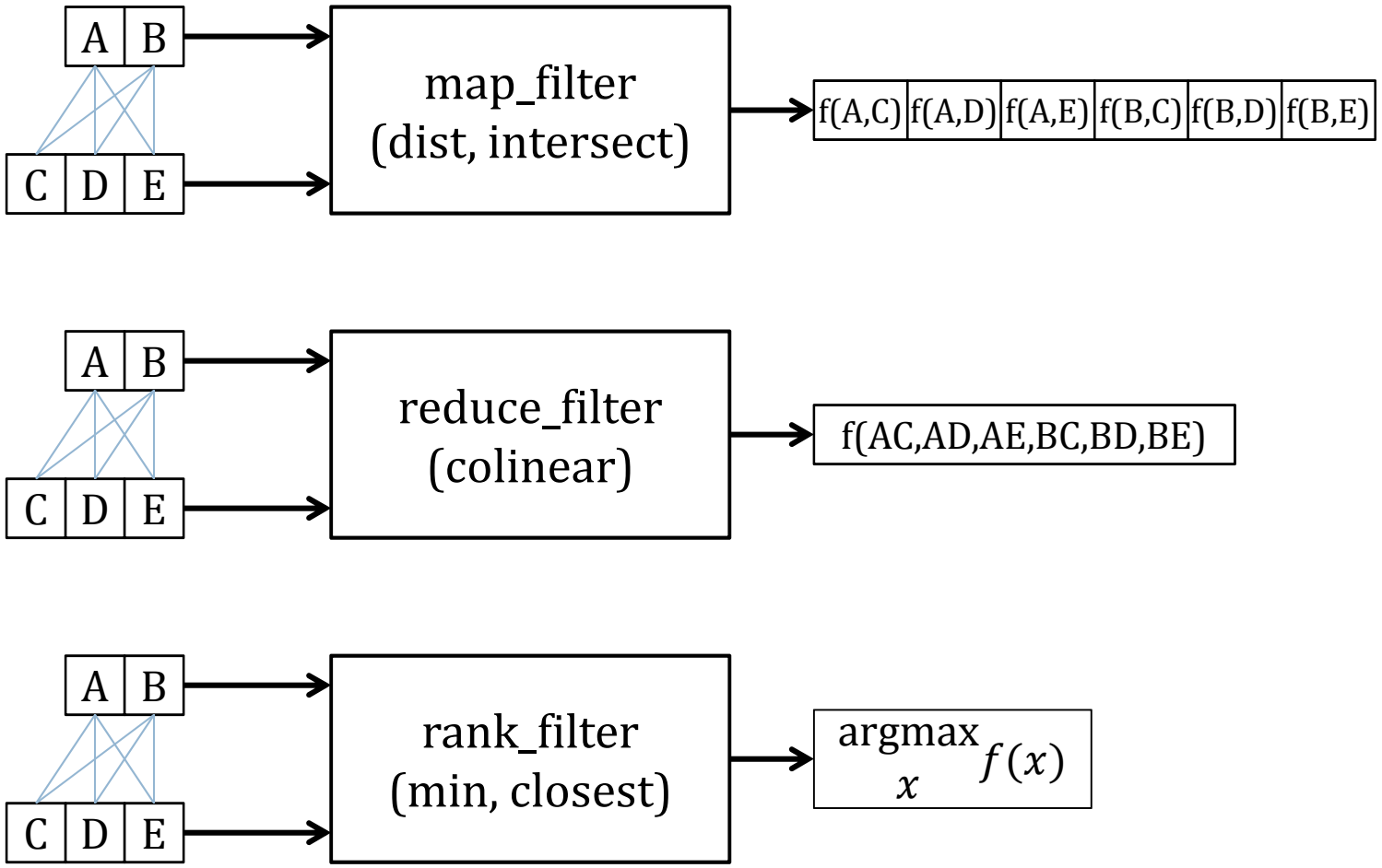
Nuggets

- Filter pipelines are expressive and general
- Works on all OSes
- Doesn't add any library dependencies to Soar
- Used extensively in the Rosie project

Coal

- Missing some functionality from old SVS
- Trades performance for simplicity

Filter Subclasses



Credits

- SVS theory was developed by Sam Wintermute
  - Wintermute, S. *Imagery in Cognitive Architecture: Representation and Control at Multiple Levels of Abstraction*. Cognitive Systems Research, 19-20,1-29.
- Soar Visual Imagery (SVI) was developed by Scott Lathrop
  - Lathrop, S.D., and Laird, J.E. (2007). *Towards Incorporating Visual Imagery into a Cognitive Architecture.* Proceedings of the Eighth International Conference on Cognitive Modeling. Ann Arbor, MI.

Writing a New Filter

- Write new class inheriting from an existing filter subclass (probably typed_map_filter)
- Register the new filter with the filter_table
- Recompile Soar

```
class custom_filter : public typed_map_filter<bool>
{
public:
    custom_filter(Symbol *root, soar_interface *si, filter_input *input, scene *scn)
    : typed_map_filter<bool>(root, si, input) // call superclass constructor
    {}

    bool compute(const filter_params *params, bool adding, bool &out, bool &changed)
    {
        // do your computation here
    };
};
```

Scene Graph Edit Language

- `a NAME TYPE PARENT [GEOMETRY] [TRANSFORM]`

  Add object to the scene graph
- `d NAME`

  Delete object from scene graph
- `c NAME [GEOMETRY] [TRANSFORM]`

  Change object geometry and/or transform
- `p NAME PROPERTY VALUE`

  Set custom property
- Geometries
  - Ball: `b RADIUS`
  - Convex polyhedron: `v X1 Y1 Z1 X2 Y2 Z2 ...`
- `Transforms`
  - `[p X Y Z] [r X Y Z] [s X Y Z]`