Making Astrometric Solver Tractable through In-Situ Visual Analytics

This work is financially supported
by the German Aerospace Agency
(Deutsches Zentrum für Luft- und Raumfahrt e.V., DLR)
through grant 50OD2201

Konstantin Ryabinin,
konstantin.riabinin@uni-heidelberg.de

Wolfgang Löffler,
loeffler@ari.uni-heidelberg.de,

Olga Erokhina,
olga.erokhina@uni-heidelberg.de

Gerasimos Sarras,
gerasimos.sarras@uni-heidelberg.de

Michael Biermann,
biermann@ari.uni-heidelberg.de

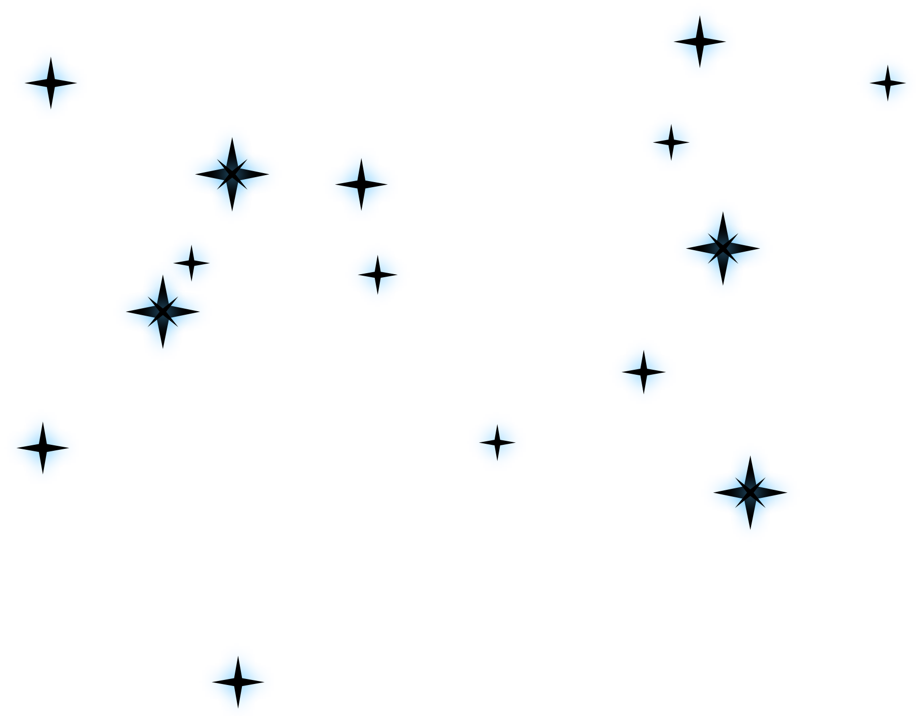
Singapore – 2025

Space Astrometry and Big Data

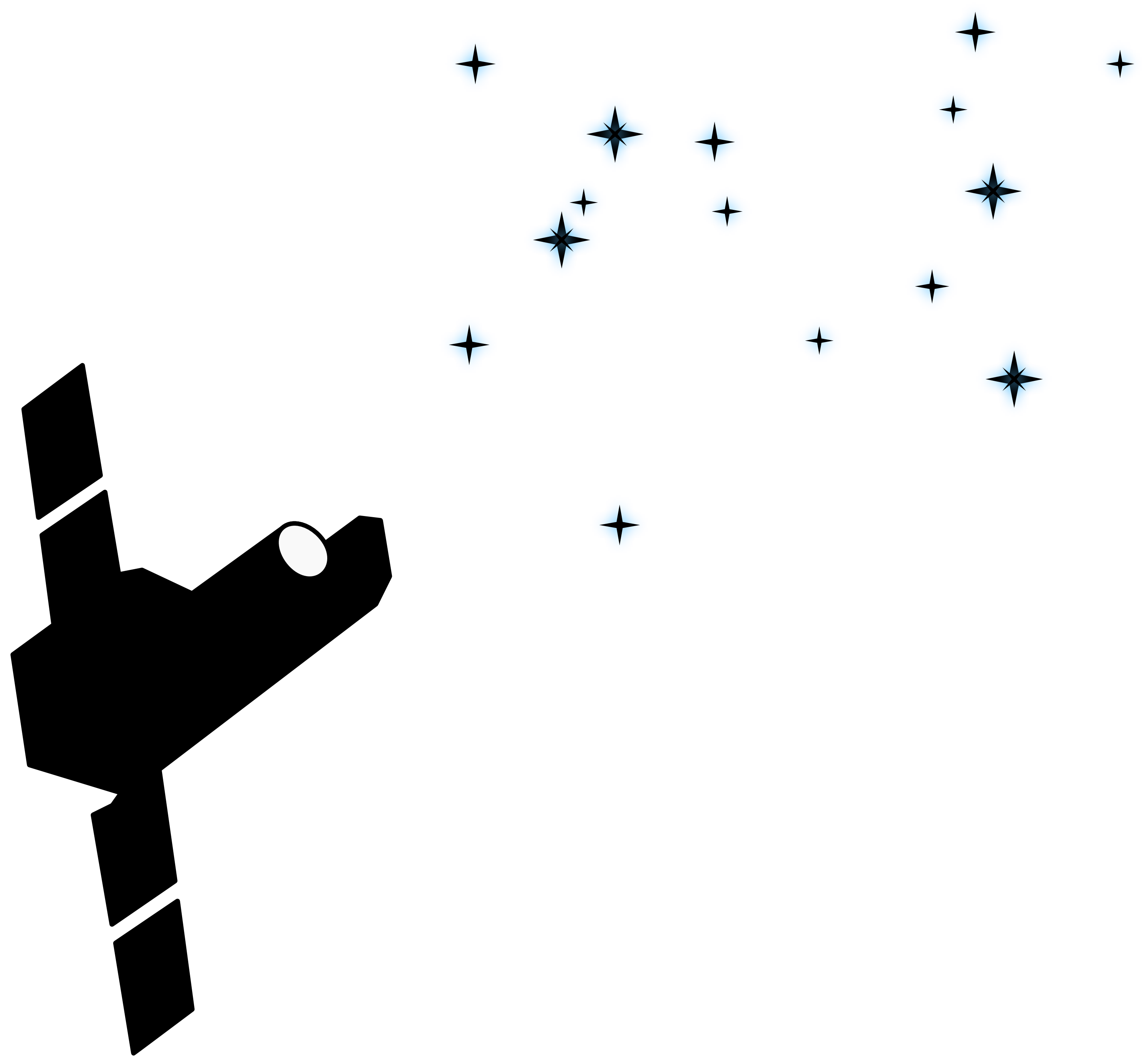
Space Astrometry reveals the stellar parameters from observations made by satellite telescopes

Space Astrometry and Big Data

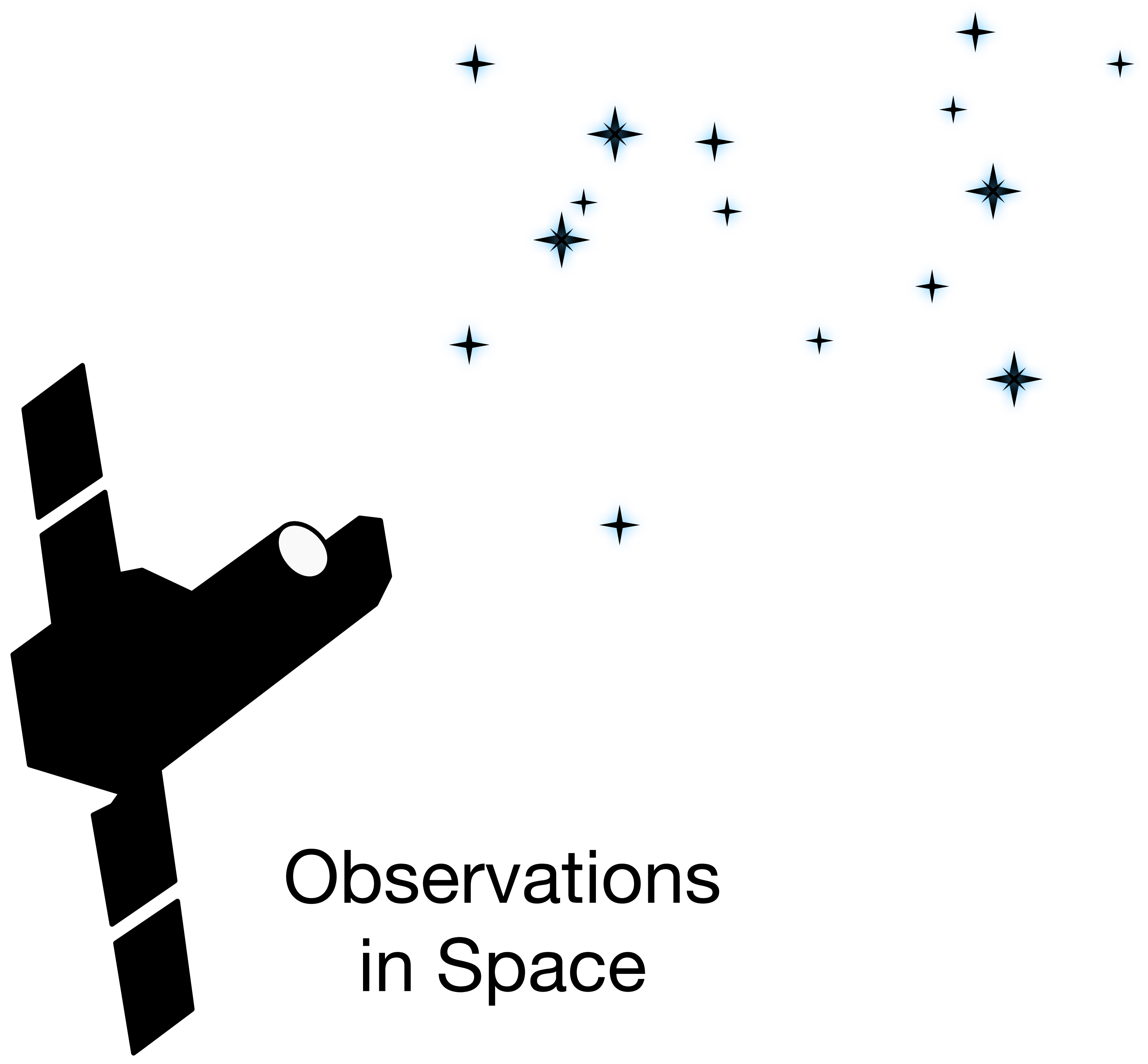
Space Astrometry reveals the stellar parameters from observations made by satellite telescopes



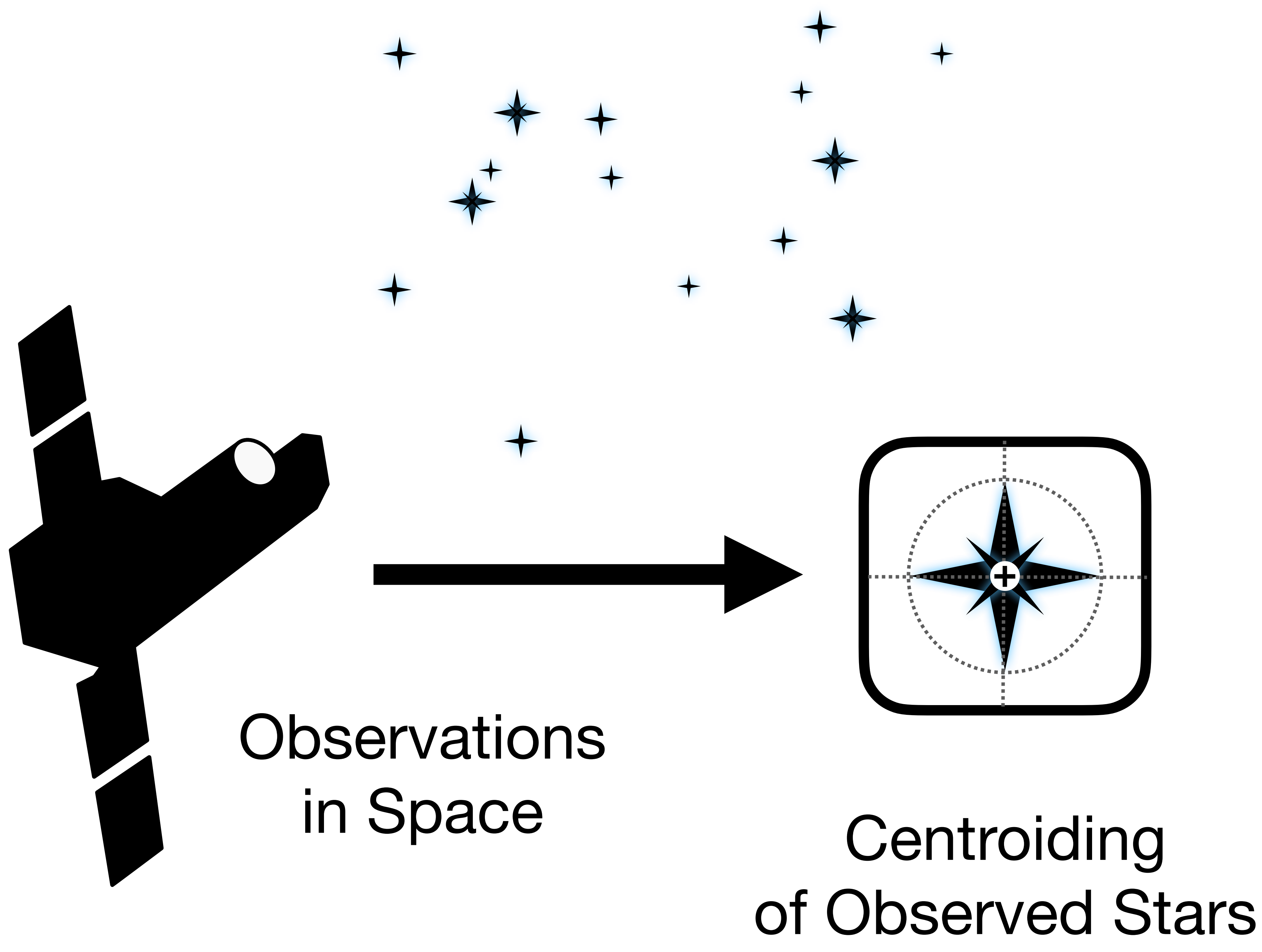
Space Astrometry reveals the stellar parameters from observations made by satellite telescopes



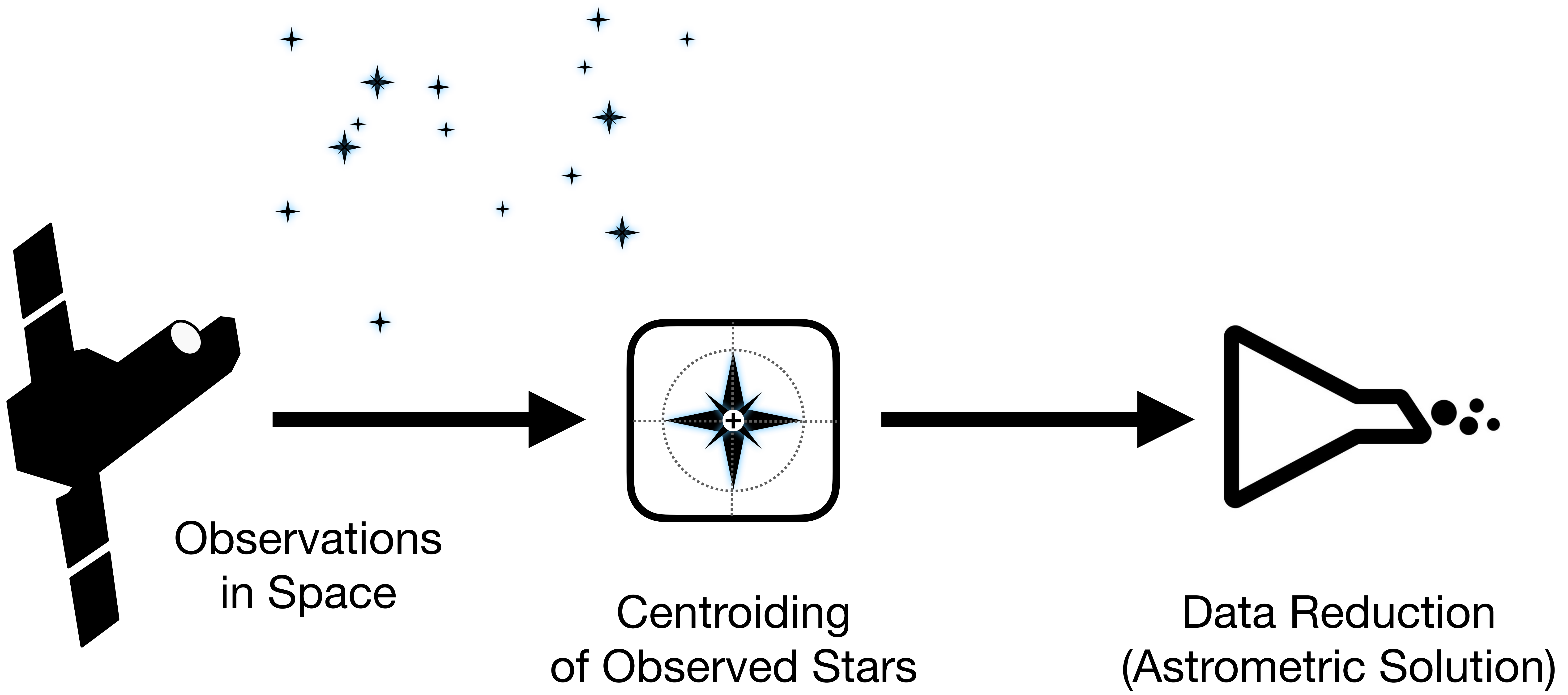
Space Astrometry reveals the stellar parameters from observations made by satellite telescopes



Space Astrometry reveals the stellar parameters from observations made by satellite telescopes

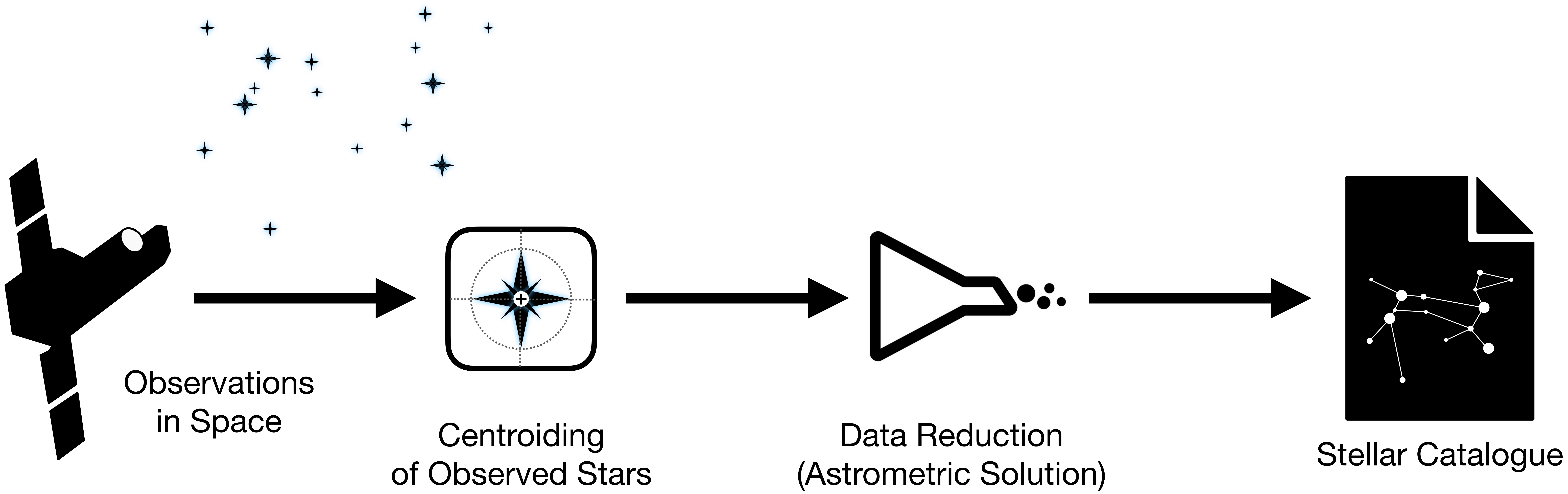


Space Astrometry reveals the stellar parameters from observations made by satellite telescopes

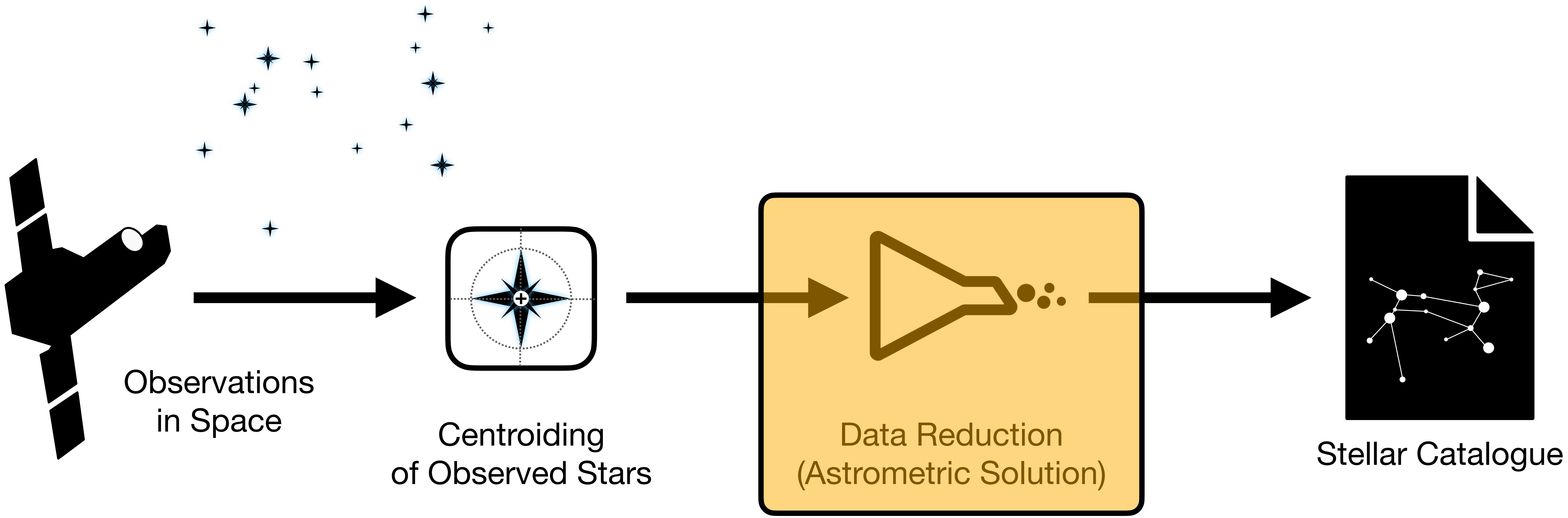


Space Astrometry and Big Data

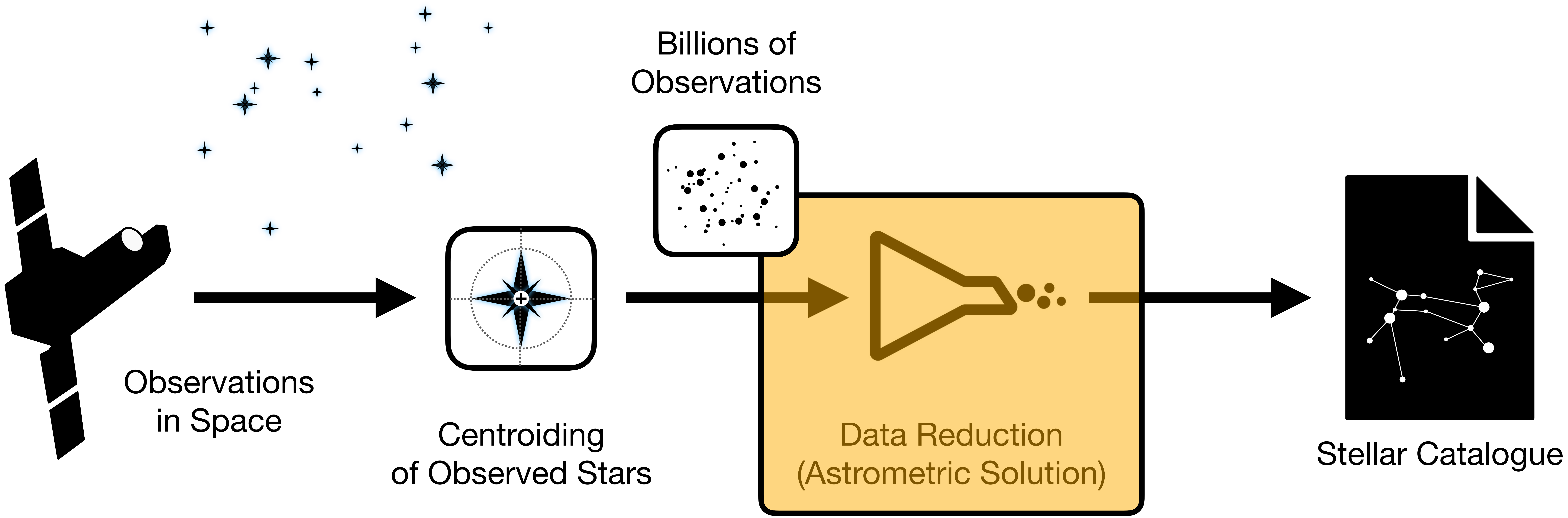
Space Astrometry reveals the stellar parameters from observations made by satellite telescopes



Space Astrometry reveals the stellar parameters from observations made by satellite telescopes

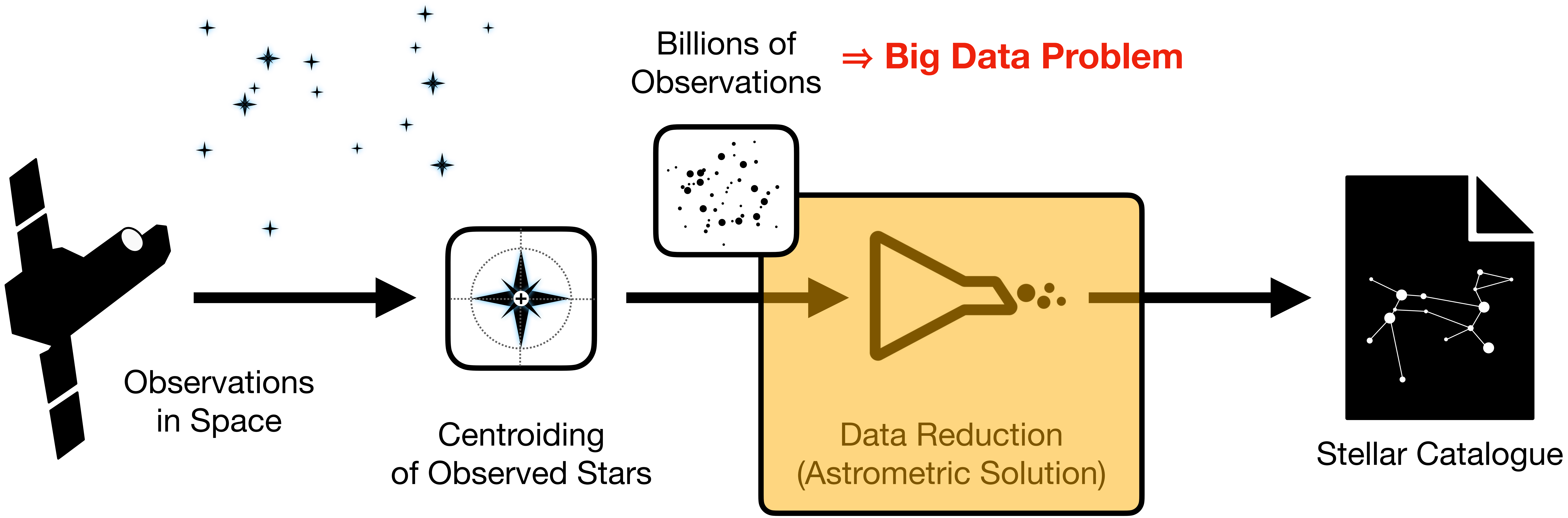


Space Astrometry reveals the stellar parameters from observations made by satellite telescopes

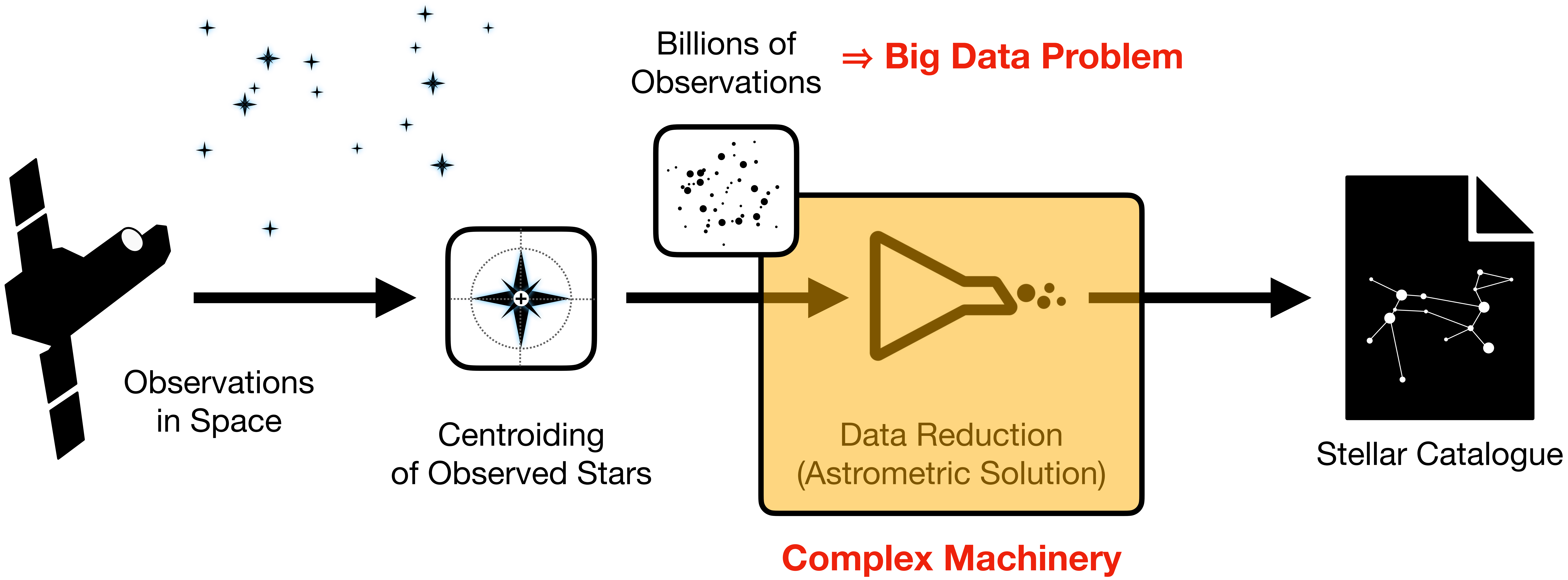


Space Astrometry and Big Data

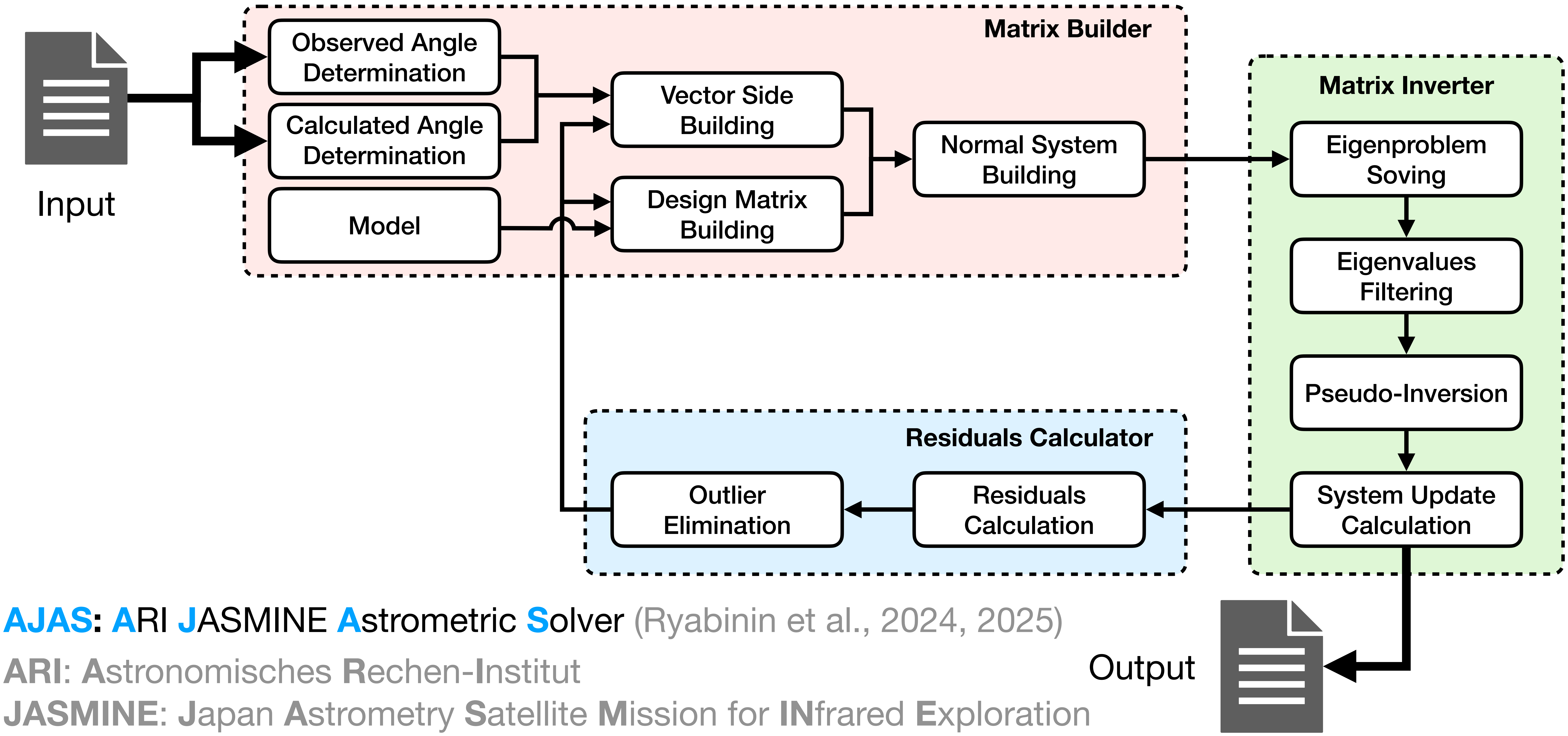
Space Astrometry reveals the stellar parameters from observations made by satellite telescopes



Space Astrometry reveals the stellar parameters from observations made by satellite telescopes



Complex Machinery of Data Reduction in AJAS

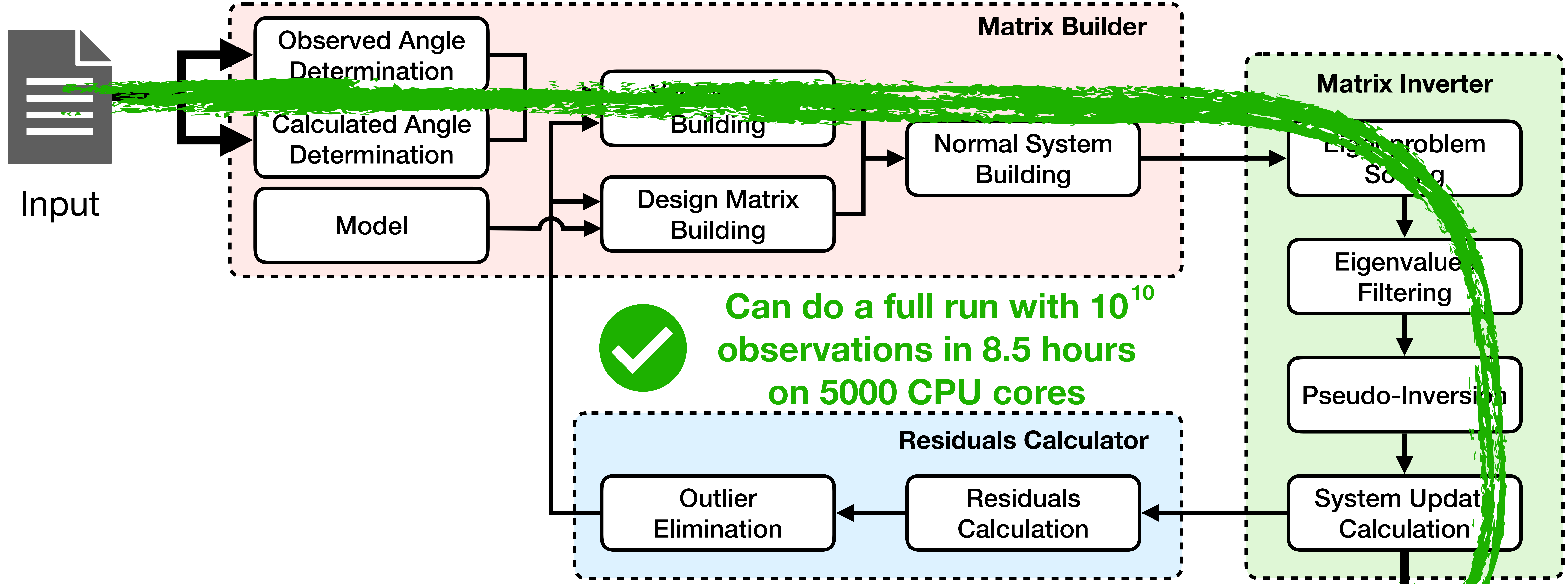


AJAS: **A**RI **J**ASMINE **A**strometric **S**olver (Ryabinin et al., 2024, 2025)

ARI: **A**stronomisches **R**echen-**I**nstitut

JASMINE: **J**apan **A**strometry **S**atellite **M**ission for **IN**frared **E**xploration

Complex Machinery of Data Reduction in AJAS

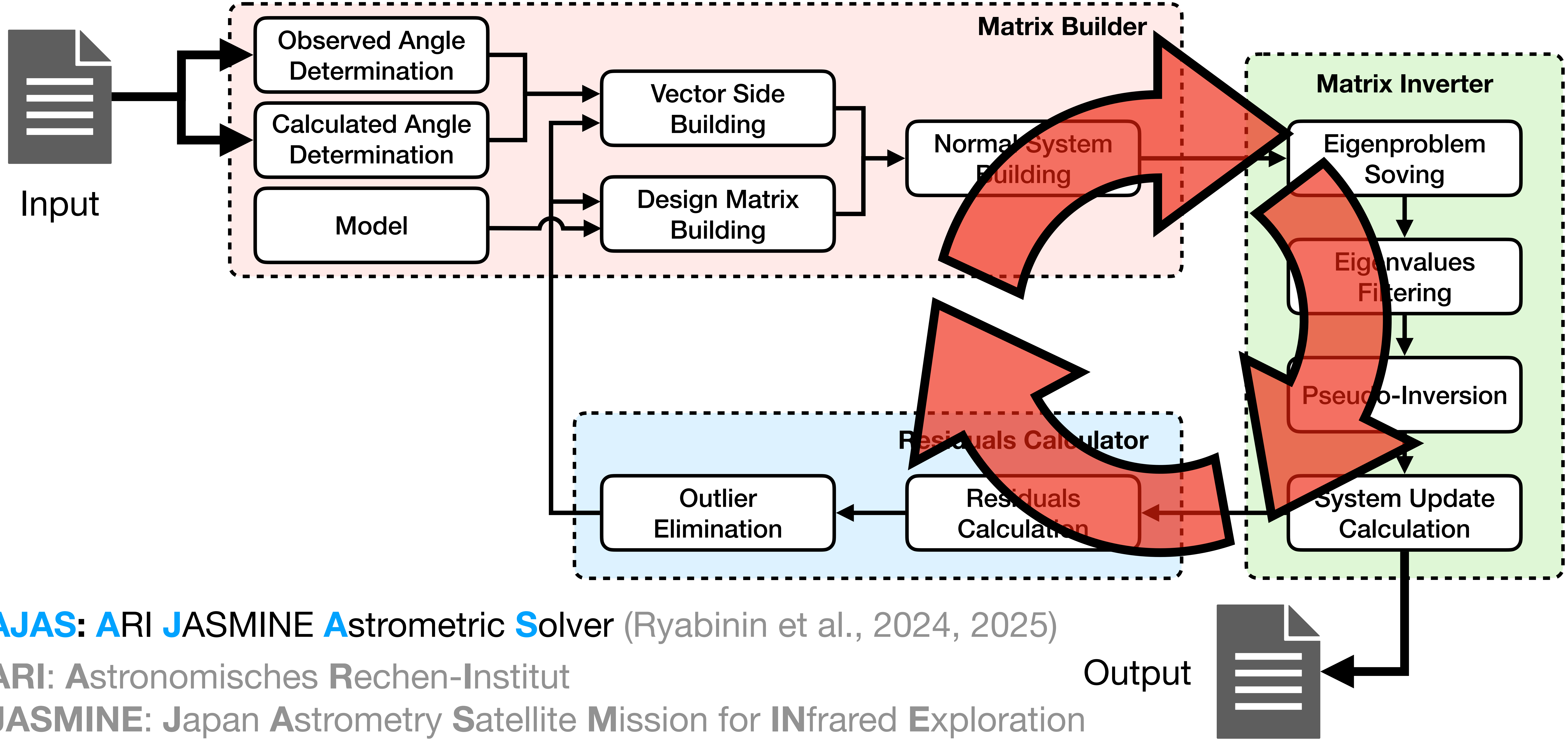


AJAS: **A**RI **J**ASMINE **A**strometric **S**olver (Ryabinin et al., 2024, 2025)

ARI: **A**stronomisches **R**echen-**I**nstitut

JASMINE: **J**apan **A**strometry **S**atellite **M**ission for **IN**frared **E**xploration

Complex Machinery of Data Reduction in AJAS

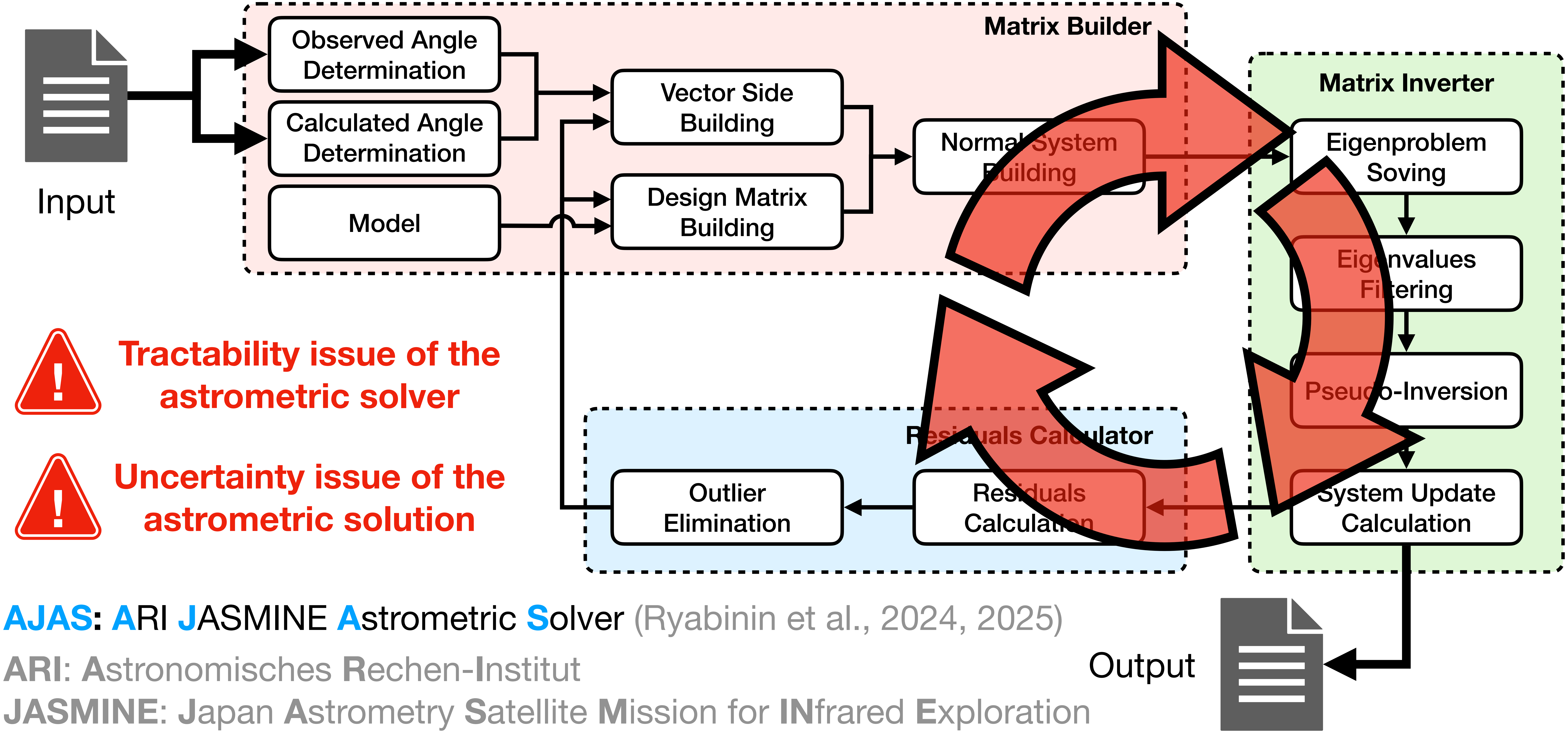


AJAS: **A**RI **J**ASMINE **A**strometric **S**olver (Ryabinin et al., 2024, 2025)

ARI: **A**stronomisches **R**echen-**I**nstitut

JASMINE: **J**apan **A**strometry **S**atellite **M**ission for **IN**frared **E**xploration

Complex Machinery of Data Reduction in AJAS



Tractability issue of the astrometric solver

Uncertainty issue of the astrometric solution

AJAS: **A**RI **J**ASMINE **A**strometric **S**olver (Ryabinin et al., 2024, 2025)

ARI: **A**stronomisches **R**echen-**I**nstitut

JASMINE: **J**apan **A**strometry **S**atellite **M**ission for **IN**frared **E**xploration

In-Situ = { in the place, where + at the time, while } the data are being processed

Tractability issue of the astrometric solver

Uncertainty issue of the astrometric solution

Tackle by

SciVi Ontology-Driven Visual Analytics Platform

(Ryabinin et al., 2013–2023)

SciVi = ∑ Control Modules + ∑ Operators

(Ryabinin et al., 2013–1023)

SciVi = ∑ Control Modules + ∑ Operators

Operator = Implementation + Ontology

Implementation → Own, Third-Party

Ontology → Lightweight: $O = \{ R, T \}$

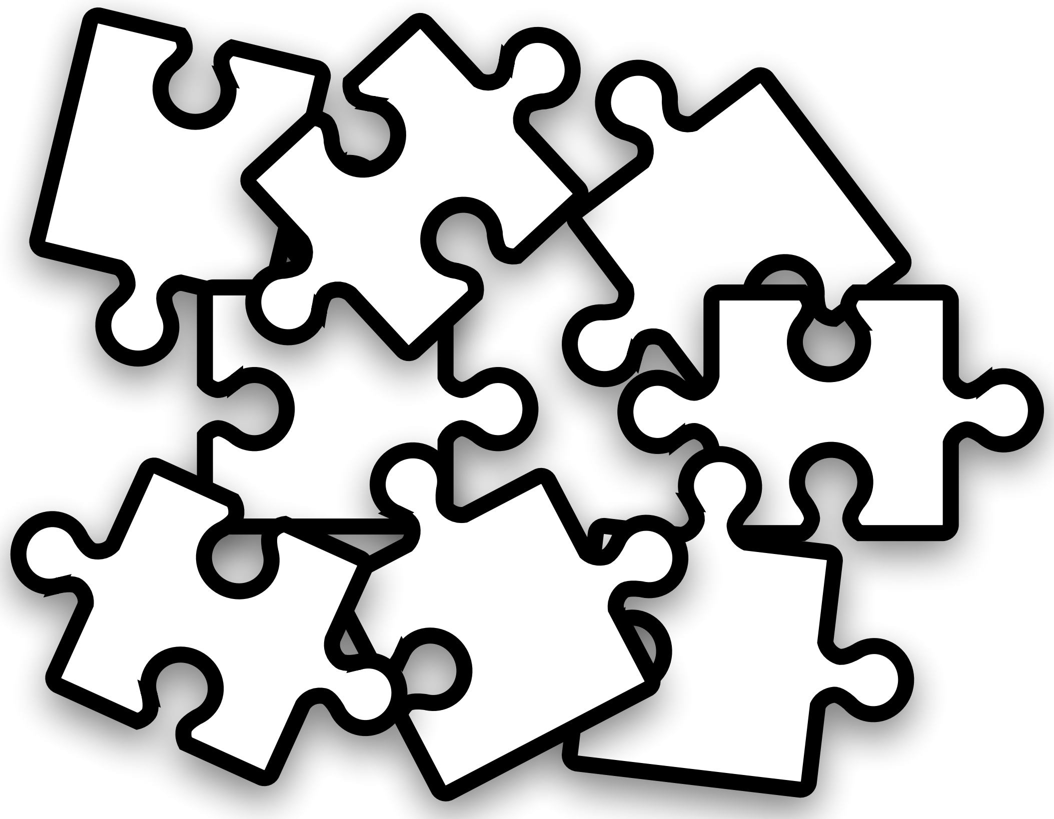
(Ryabinin et al., 2013–1023)

SciVi = ∑ Control Modules + ∑ Operators

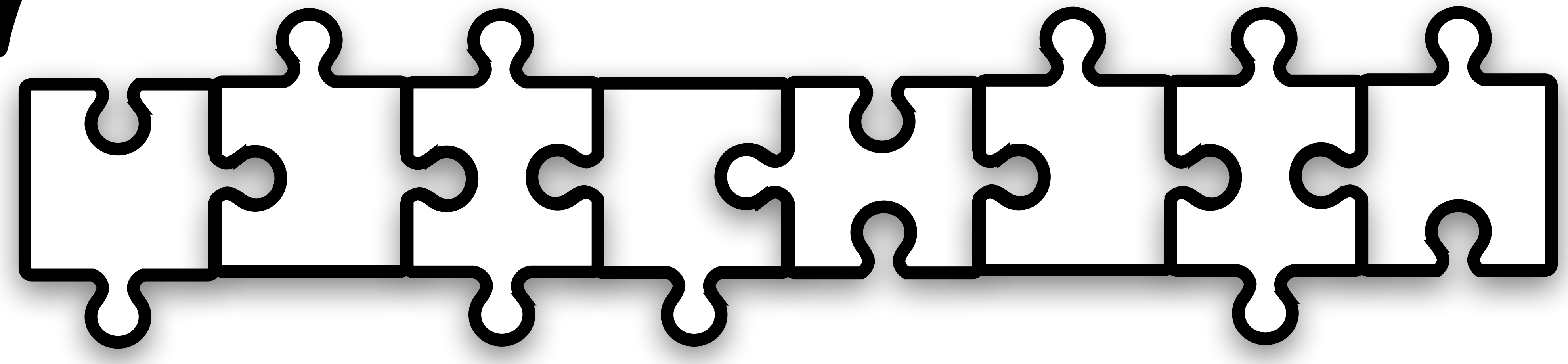
Operator = Implementation + Ontology

Implementation: Own, Third-Party

Ontology: Lightweight: $O = \{ R, T \}$



SciVi allows building pipelines of operators

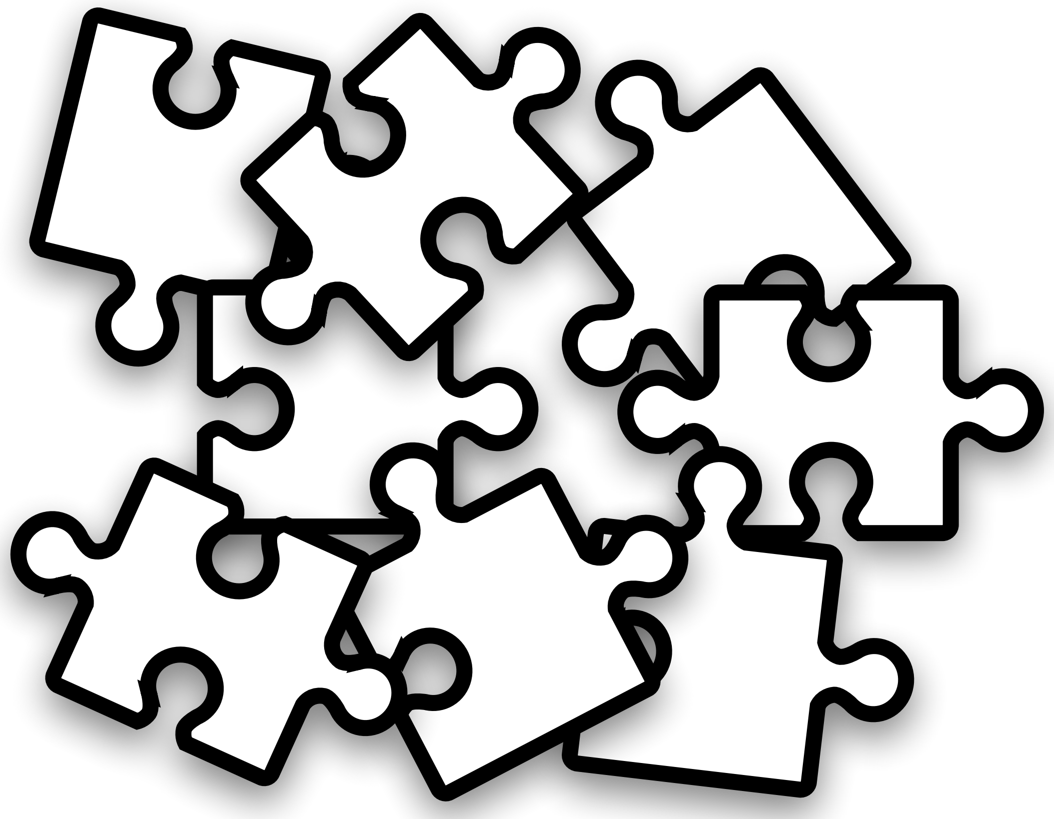


(Ryabinin et al., 2013–1023)

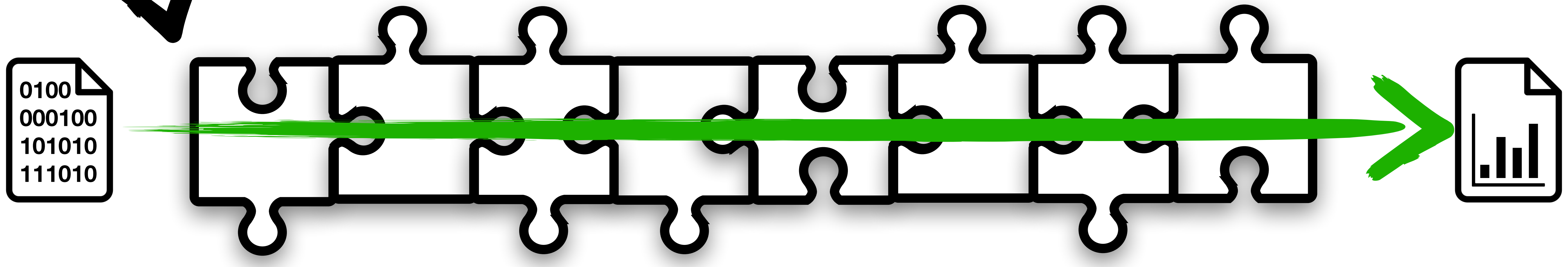
SciVi = ∑ Control Modules + ∑ Operators

Operator = Implementation + Ontology

Own Third-Party Lightweight: $O = \{ R, T \}$



SciVi allows building pipelines of operators



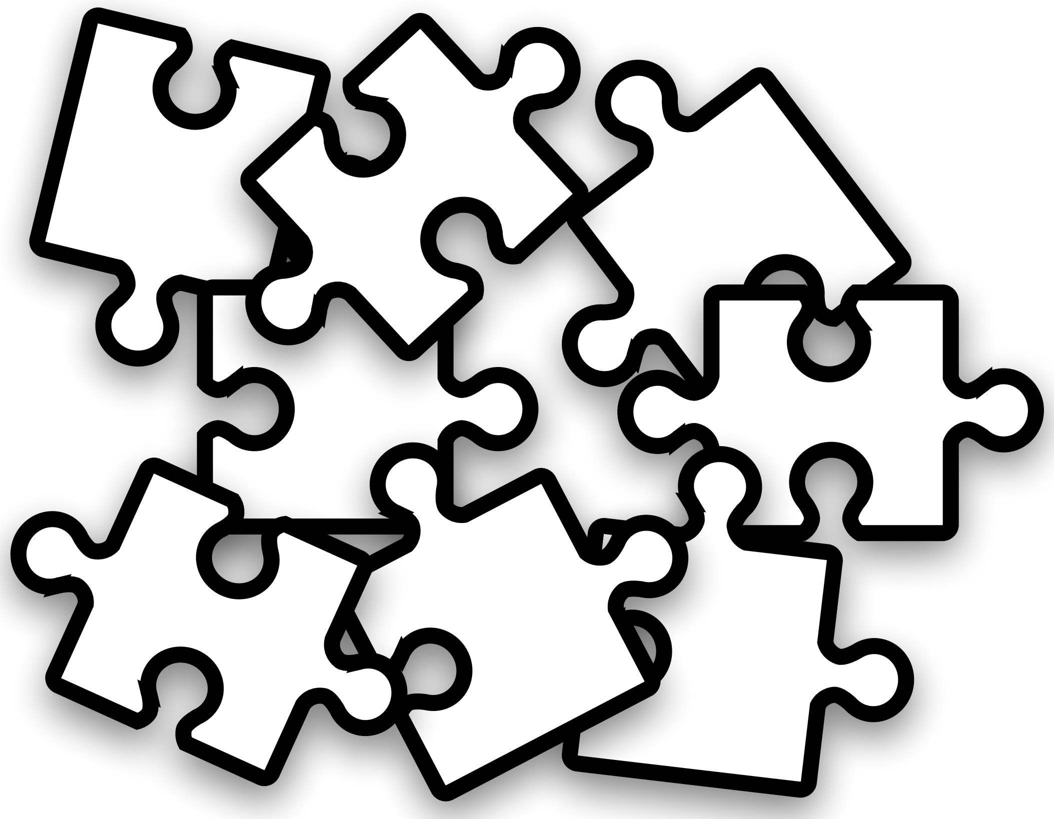
(Ryabinin et al., 2013–1023)

SciVi = ∑ Control Modules + ∑ Operators

Operator = Implementation + Ontology

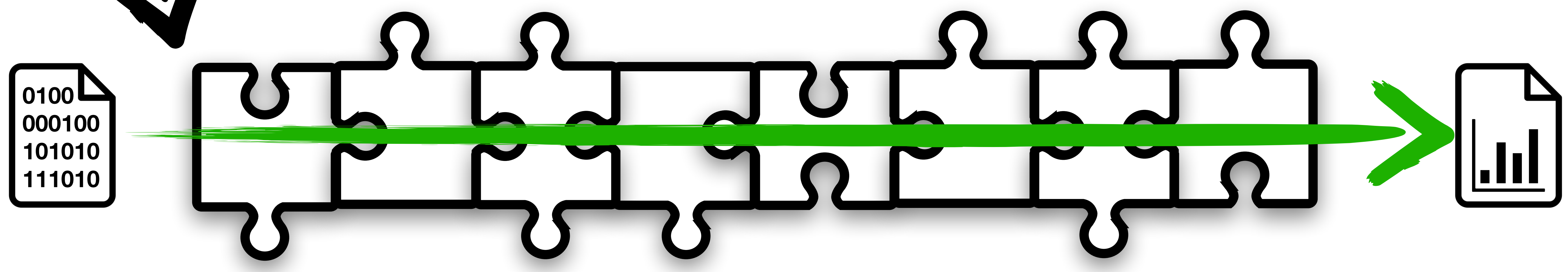
Implementation: Own, Third-Party

Ontology: Lightweight: $O = \{ R, T \}$



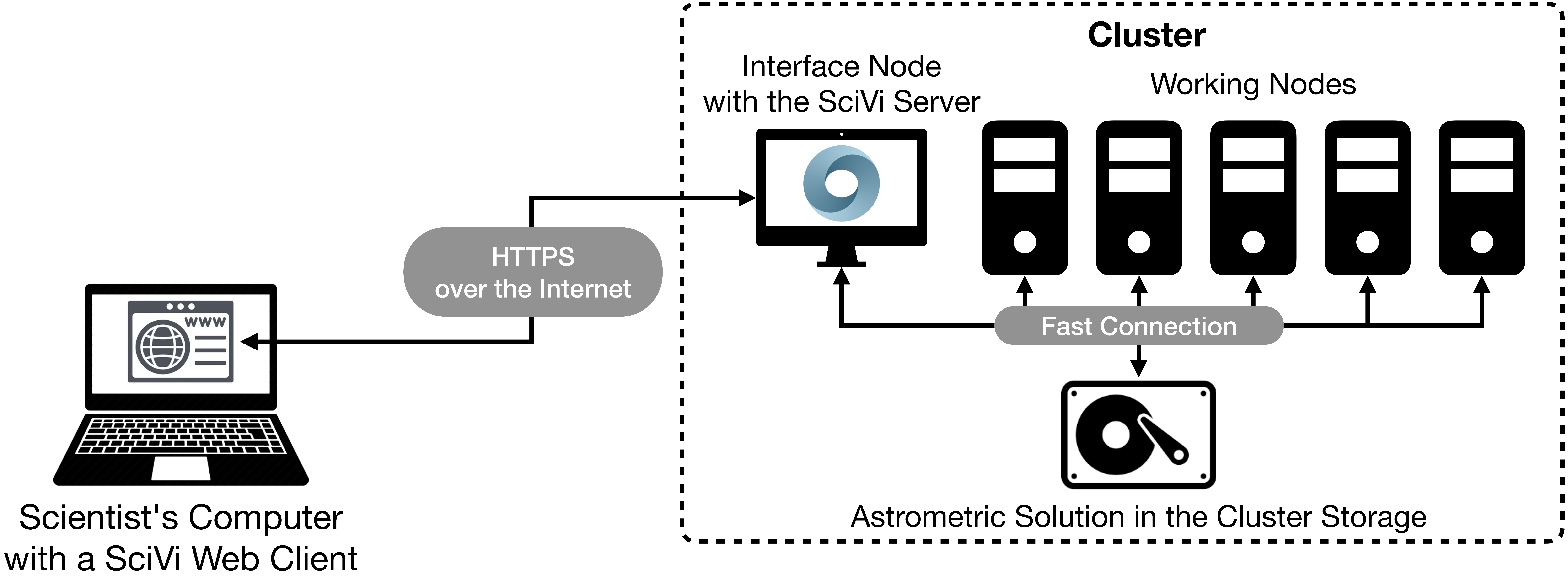
SciVi allows building pipelines of operators

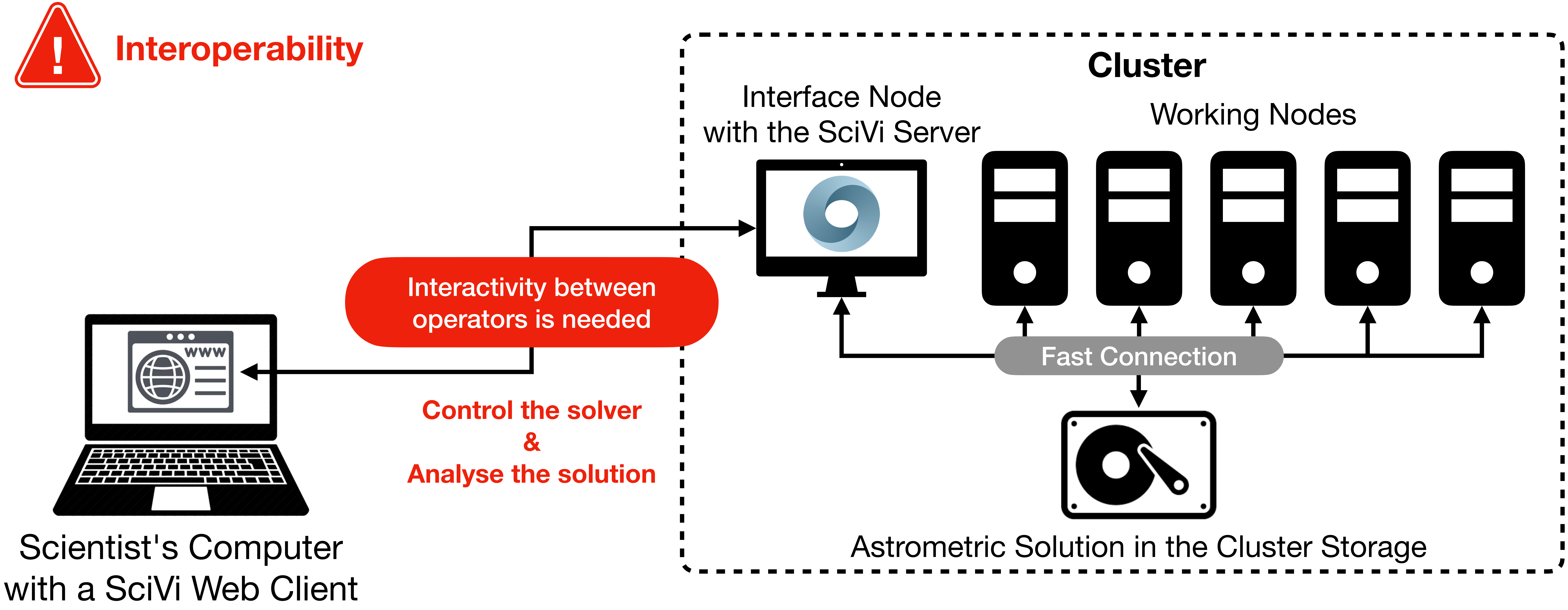
Corresponding GUI is generated automatically



(Ryabinin et al., 2013–1023)

AJAS + SciVi = ♥

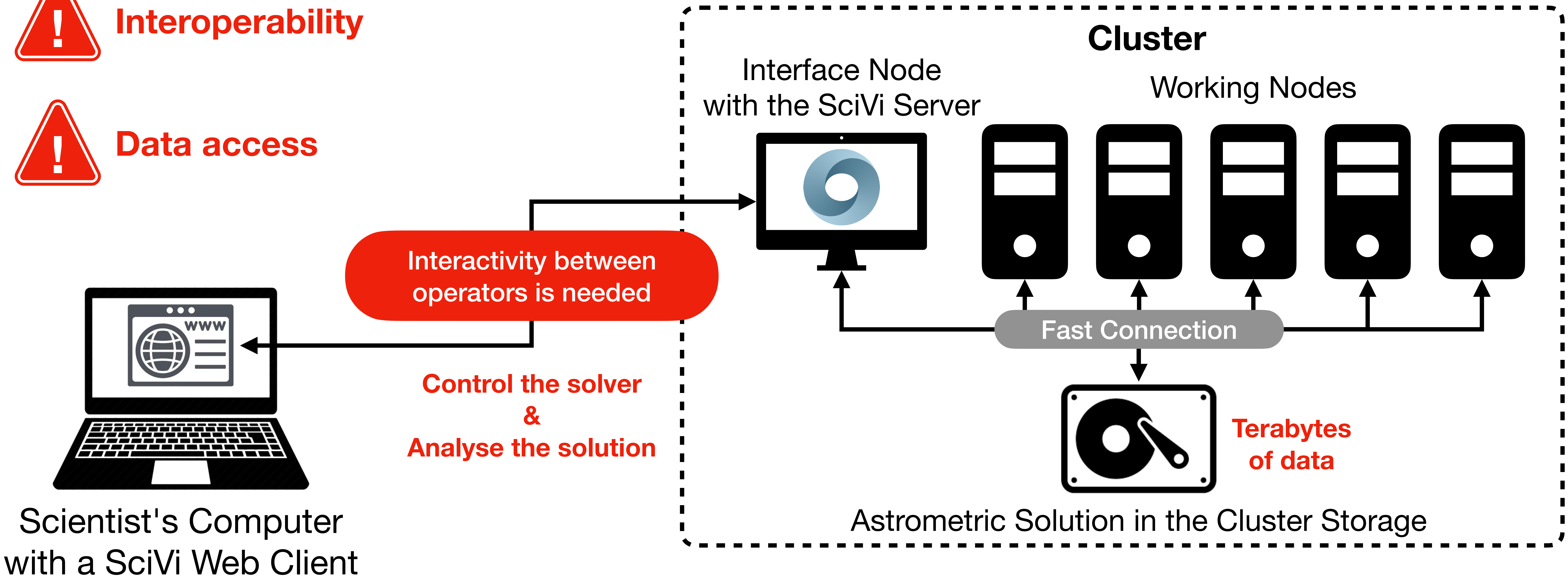

Cluster
Interface Node
with the SciVi Server
Working Nodes
HTTPS
over the Internet
Fast Connection
Astrometric Solution in the Cluster Storage
Scientist's Computer
with a SciVi Web Client



Interoperability
Cluster
Interface Node
with the SciVi Server
Working Nodes
Interactivity between
operators is needed
Fast Connection
Control the solver
&
Analyse the solution
Astrometric Solution in the Cluster Storage
Scientist's Computer
with a SciVi Web Client

AJAS + SciVi = ♥

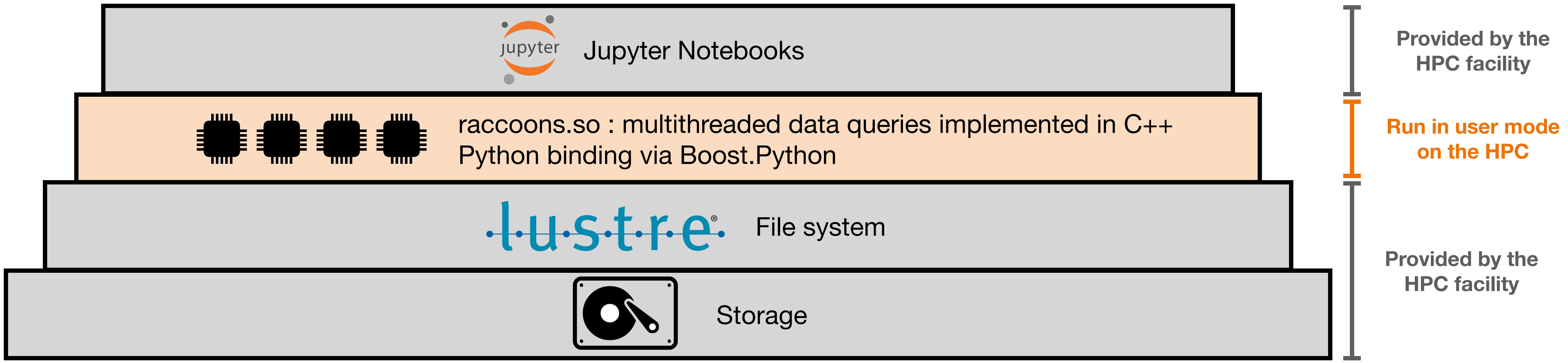
- **Interoperability**
- **Data access**



RACCOONS: **R**apid **ACC**ess **O**perations **O**n **N**umerical **S**olutions

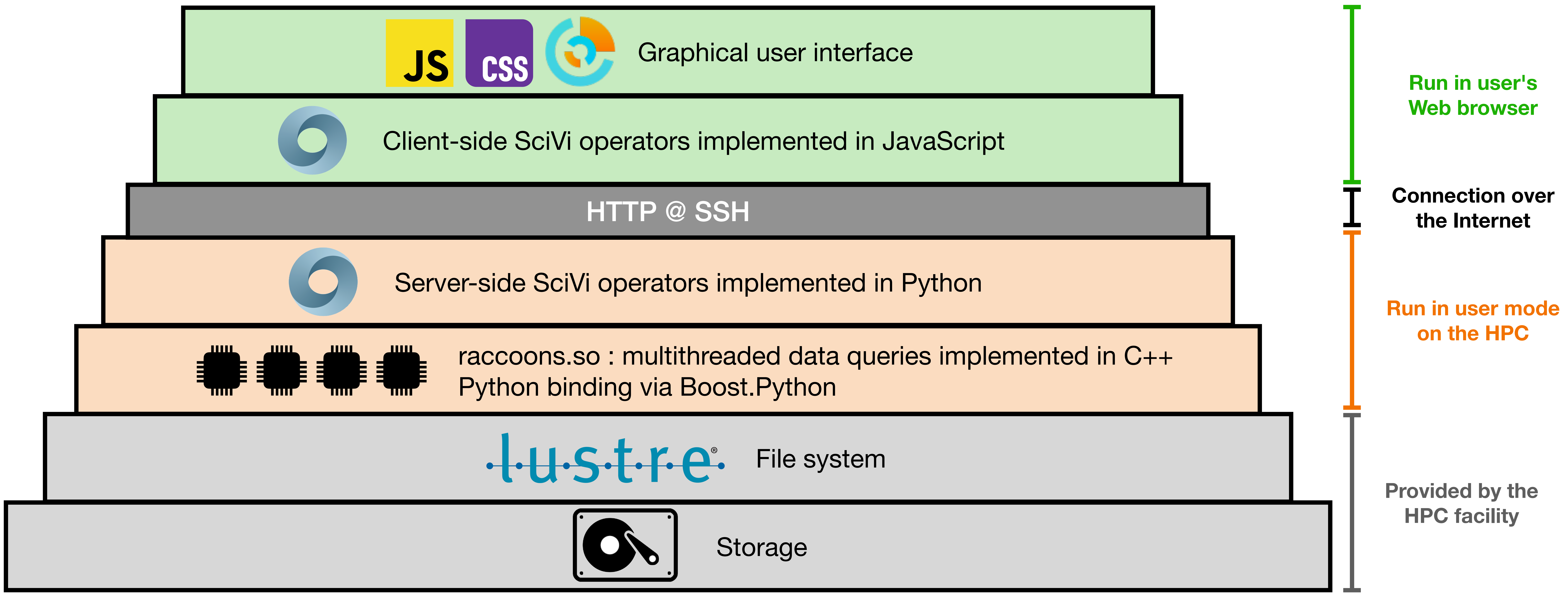
RACCOONS: **R**apid **ACC**ess **O**perations **O**n **N**umerical **S**olutions

Inspired by the wild neighbours who created their nest in the roof truss of the ARI building

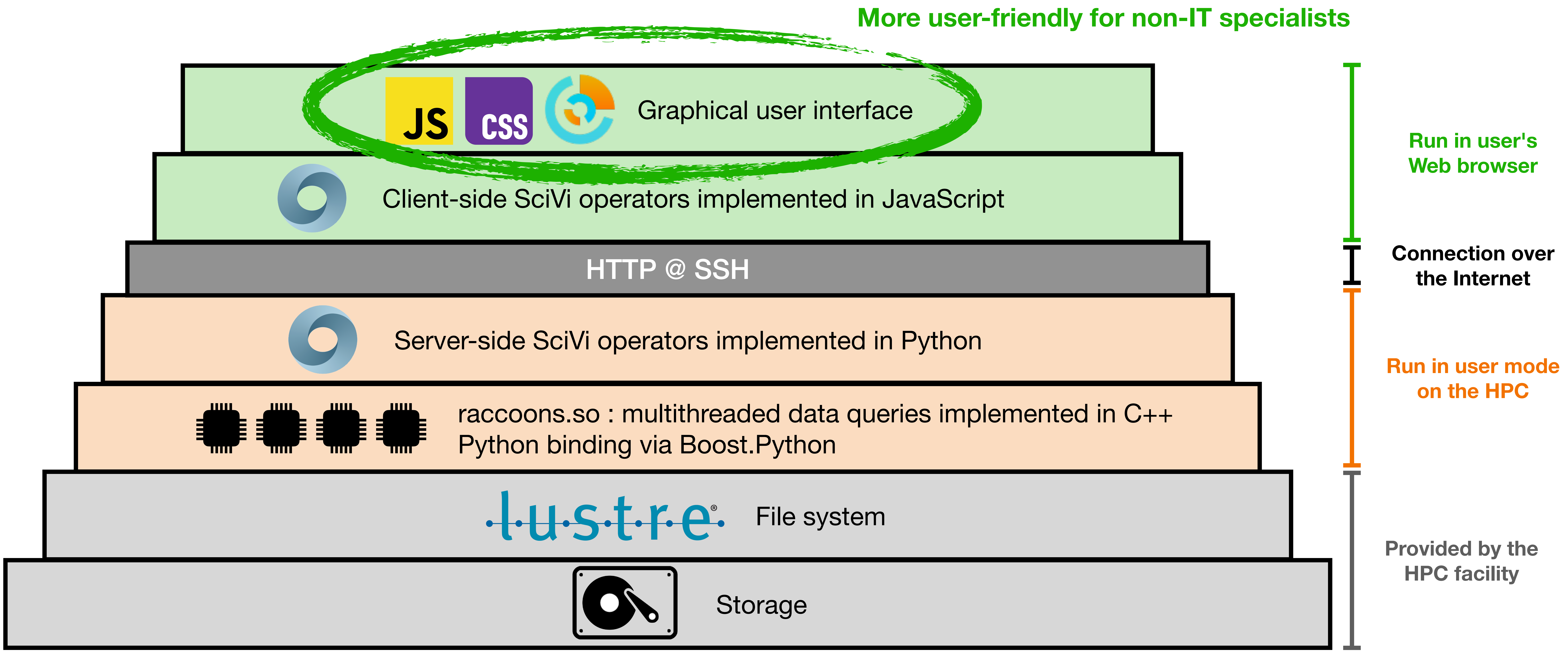


jupyter
Jupyter Notebooks
raccoons.so : multithreaded data queries implemented in C++
Python binding via Boost.Python
lustre
File system
Storage
Provided by the HPC facility
Run in user mode on the HPC
Provided by the HPC facility

RACCOONS: SciVi Binding



RACCOONS: SciVi Binding



Python code:

obs.py

```python
import raccoons

def get_obs(path: str) -> list:
    engine = raccoons.Engine(path)
    query = raccoons.MapReduceQuery(
        num_groups = engine.mission.number_of_stars,
        condition = "{ return observation.starID; }",
        selector = "{ return 1; }",
        reduction = "{ slot += value; }")
    engine.forEachObservation([ query ])
    return query.histogram(num_bins = 50)
```

RACCOONS Example: histogram of observations / star

Python code:

obs.py

```python
import raccoons

def get_obs(path: str) -> list:
    engine = raccoons.Engine(path)
    query = raccoons.MapReduceQuery(
        num_groups = engine.mission.number_of_stars,
        condition = "{ return observation.starID; }",
        selector = "{ return 1; }",
        reduction = "{ slot += value; }")
    engine.forEachObservation([ query ])
    return query.histogram(num_bins = 50)
```

- condition → Match the data
- selector → Extract the data
- reduction → Reduce the data

RACCOONS Example: histogram of observations / star

Python code:

obs.py

```python
import raccoons

def get_obs(path: str) -> list:
    engine = raccoons.Engine(path)
    query = raccoons.MapReduceQuery(
        num_groups = engine.mission.number_of_stars,
        condition = "{ return observation.starID; }",
        selector = "{ return 1; }",
        reduction = "{ slot += value; }")
    engine.forEachObservation([ query ])
    return query.histogram(num_bins = 50)
```

Wrapped as C++ lambda functions

Match the data → `[](const void *object) { return (Index)((Observation *)object)->starID(); }`

Extract the data → `[](const void *object) { return 1.0; }`

Reduce the data → `[](double value, double &slot) { slot += value; }`

Python code:

obs.py

```python
import raccoons

def get_obs(path: str) -> list:
    engine = raccoons.Engine(path)
    query = raccoons.MapReduceQuery(
        num_groups = engine.mission.number_of_stars,
        condition = "{ return observation.starID; }",
        selector = "{ return 1; }",
        reduction = "{ slot += value; }")
    engine.forEachObservation([ query ])
    return query.histogram(num_bins = 50)
```

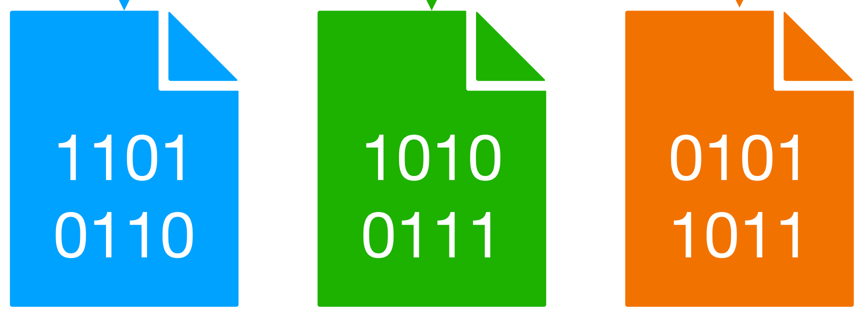
Wrapped as C++ lambda functions

Match the data → `[](const void *object) { return (Index)((Observation *)object)->starID(); }`

Extract the data → `[](const void *object) { return 1.0; }`

Reduce the data → `[](double value, double &slot) { slot += value; }`

JIT-compiled by LLVM

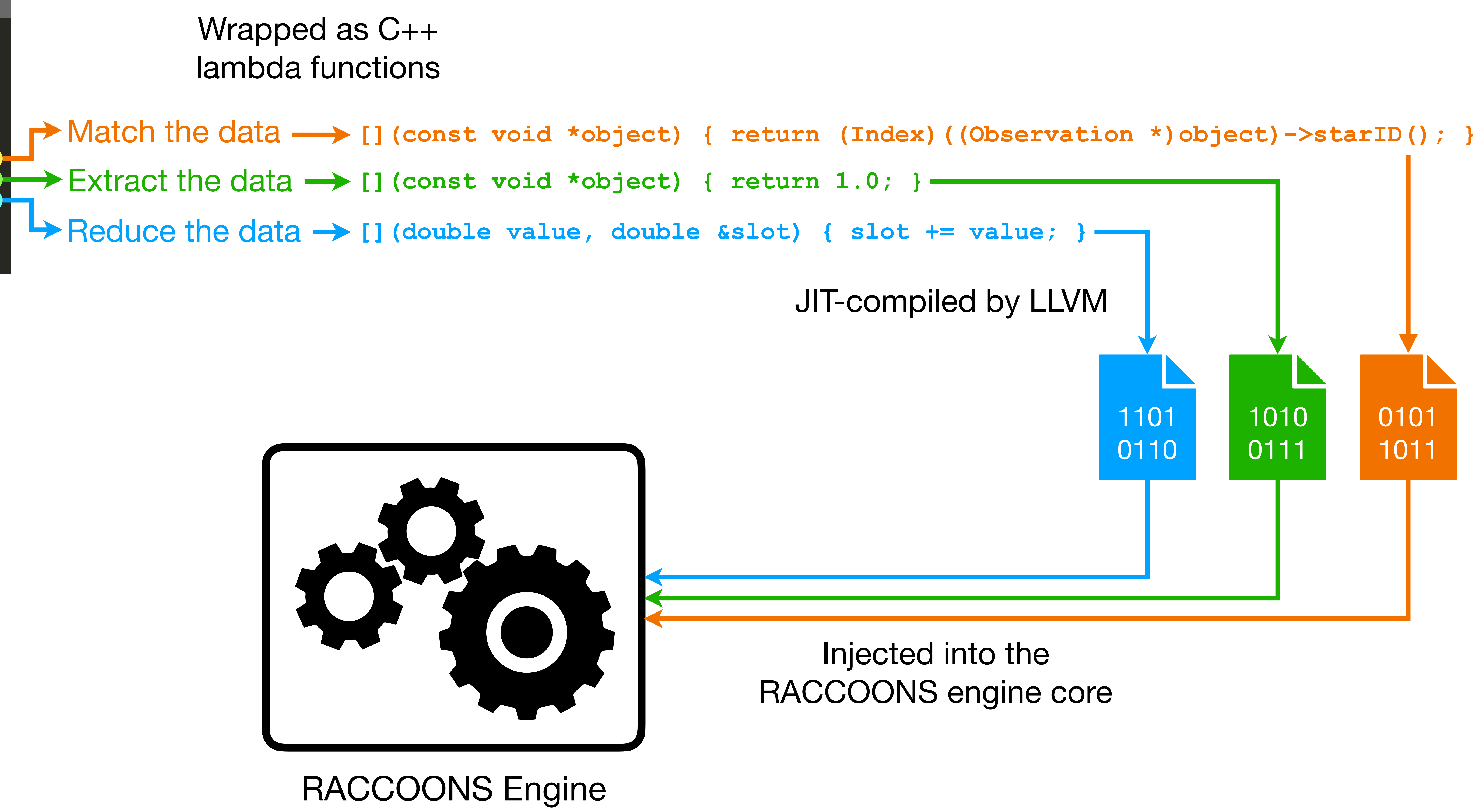


RACCOONS Example: histogram of observations / star

Python code:

```python
import raccoons

def get_obs(path: str) -> list:
    engine = raccoons.Engine(path)
    query = raccoons.MapReduceQuery(
        num_groups = engine.mission.number_of_stars,
        condition = "{ return observation.starID; }",
        selector = "{ return 1; }",
        reduction = "{ slot += value; }")
    engine.forEachObservation([ query ])
    return query.histogram(num_bins = 50)
```



RACCOONS and SciVi Interoperability

Python code:

obs.py

```python
import raccoons

def get_obs(path: str) -> list:
    engine = raccoons.Engine(path)
    query = raccoons.MapReduceQuery(
        num_groups = engine.mission.number_of_stars,
        condition = "{ return observation.starID; }",
        selector = "{ return 1; }",
        reduction = "{ slot += value; }")
    engine.forEachObservation([ query ])
    return query.histogram(num_bins = 50)
```

JavaScript code:

obs.js

```javascript
async function plotObs(path) {
    const obs = await get_obs(path);
    NChart3DLib().then(mdl => {
        const chartConfig = {
            cartesianSystem:
            {
                xAxis:
                {
                    caption: { text: "Observations / Source" },
                    valueMask: "%.0f"
                },
                yAxis:
                {
                    caption: { text: "Log(Amount)" },
                    isLogarithmic: true,
                    valueMask: "%.1e"
                }
            },
            series:
            [
                {
                    type: "column",
                    brush: "#60cce8",
                    borderBrush: "#000000",
                    borderThickness: 1,
                    points:
                    {
                        type: "xy",
                        data: obs
                    }
                }
            ]
        };
        const chart = new mdl.NChart("obsPlotCanvas");
        chart.loadJSON(JSON.stringify(chartConfig));
    });
}
```

RACCOONS and SciVi Interoperability

Python code:

obs.py

```python
import raccoons

def get_obs(path: str) -> list:
    engine = raccoons.Engine(path)
    query = raccoons.MapReduceQuery(
        num_groups = engine.mission.number_of_stars,
        condition = "{ return observation.starID; }",
        selector = "{ return 1; }",
        reduction = "{ slot += value; }")
    engine.forEachObservation([ query ])
    return query.histogram(num_bins = 50)
```

Looks like a function call...

JavaScript code:

obs.js

```javascript
async function plotObs(path) {
    const obs = await get_obs(path);
    NChart3DLib().then(mdl => {
        const chartConfig = {
            cartesianSystem:
            {
                xAxis:
                {
                    caption: { text: "Observations / Source" },
                    valueMask: "%.0f"
                },
                yAxis:
                {
                    caption: { text: "Log(Amount)" },
                    isLogarithmic: true,
                    valueMask: "%.1e"
                }
            },
            series:
            [
                {
                    type: "column",
                    brush: "#60cce8",
                    borderBrush: "#000000",
                    borderThickness: 1,
                    points:
                    {
                        type: "xy",
                        data: obs
                    }
                }
            ]
        };
        const chart = new mdl.NChart("obsPlotCanvas");
        chart.loadJSON(JSON.stringify(chartConfig));
    });
}
```

RACCOONS and SciVi Interoperability

Python code:

obs.py

```python
import raccoons

def get_obs(path: str) -> list:
    engine = raccoons.Engine(path)
    query = raccoons.MapReduceQuery(
        num_groups = engine.mission.number_of_stars,
        condition = "{ return observation.starID; }",
        selector = "{ return 1; }",
        reduction = "{ slot += value; }")
    engine.forEachObservation([ query ])
    return query.histogram(num_bins = 50)
```

Looks like a function call...

It is function call!

JavaScript code:

obs.js

```javascript
async function plotObs(path) {
    const obs = await get_obs(path);
    NChart3DLib().then(mdl => {
        const chartConfig = {
            cartesianSystem:
            {
                xAxis:
                {
                    caption: { text: "Observations / Source" },
                    valueMask: "%.0f"
                },
                yAxis:
                {
                    caption: { text: "Log(Amount)" },
                    isLogarithmic: true,
                    valueMask: "%.1e"
                }
            },
            series:
            [
                {
                    type: "column",
                    brush: "#60cce8",
                    borderBrush: "#000000",
                    borderThickness: 1,
                    points:
                    {
                        type: "xy",
                        data: obs
                    }
                }
            ]
        };
        const chart = new mdl.NChart("obsPlotCanvas");
        chart.loadJSON(JSON.stringify(chartConfig));
    });
}
```

RACCOONS and SciVi Interoperability

Python code:

obs.py

```python
import raccoons

def get_obs(path: str) -> list:
    engine = raccoons.Engine(path)
    query = raccoons.MapReduceQuery(
        num_groups = engine.mission.number_of_stars,
        condition = "{ return observation.starID; }",
        selector = "{ return 1; }",
        reduction = "{ slot += value; }")
    engine.forEachObservation([ query ])
    return query.histogram(num_bins = 50)
```

JavaScript code:

obs.js

```javascript
async function plotObs(path) {
    const obs = await get_obs(path);
    NChart3DLib().then(mdl => {
        const chartConfig = {
            cartesianSystem:
            {
                xAxis:
                {
                    caption: { text: "Observations / Source" },
                    valueMask: "%.0f"
                },
                yAxis:
                {
                    caption: { text: "Log(Amount)" },
                    isLogarithmic: true,
                    valueMask: "%.1e"
                }
            },
            series:
            [
                {
                    type: "column",
                    brush: "#60cce8",
                    borderBrush: "#000000",
                    borderThickness: 1,
                    points:
                    {
                        type: "xy",
                        data: obs
                    }
                }
            ]
        };
        const chart = new mdl.NChart("obsPlotCanvas");
        chart.loadJSON(JSON.stringify(chartConfig));
    });
}
```

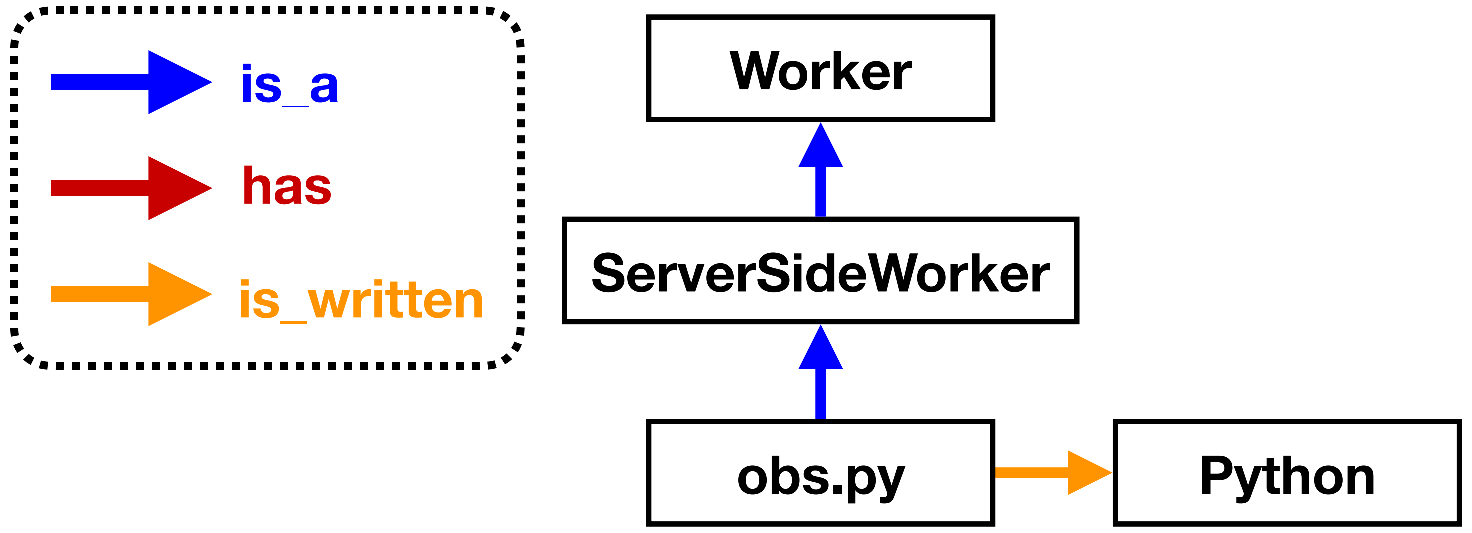
RACCOONS and SciVi Interoperability

Python code:

obs.py

```python
import raccoons

def get_obs(path: str) -> list:
    engine = raccoons.Engine(path)
    query = raccoons.MapReduceQuery(
        num_groups = engine.mission.number_of_stars,
        condition = "{ return observation.starID; }",
        selector = "{ return 1; }",
        reduction = "{ slot += value; }")
    engine.forEachObservation([ query ])
    return query.histogram(num_bins = 50)
```



JavaScript code:

obs.js

```javascript
async function plotObs(path) {
    const obs = await get_obs(path);
    NChart3DLib().then(mdl => {
        const chartConfig = {
            cartesianSystem:
            {
                xAxis:
                {
                    caption: { text: "Observations / Source" },
                    valueMask: "%.0f"
                },
                yAxis:
                {
                    caption: { text: "Log(Amount)" },
                    isLogarithmic: true,
                    valueMask: "%.1e"
                }
            },
            series:
            [
                {
                    type: "column",
                    brush: "#60cce8",
                    borderBrush: "#000000",
                    borderThickness: 1,
                    points:
                    {
                        type: "xy",
                        data: obs
                    }
                }
            ]
        };
        const chart = new mdl.NChart("obsPlotCanvas");
        chart.loadJSON(JSON.stringify(chartConfig));
    });
}
```

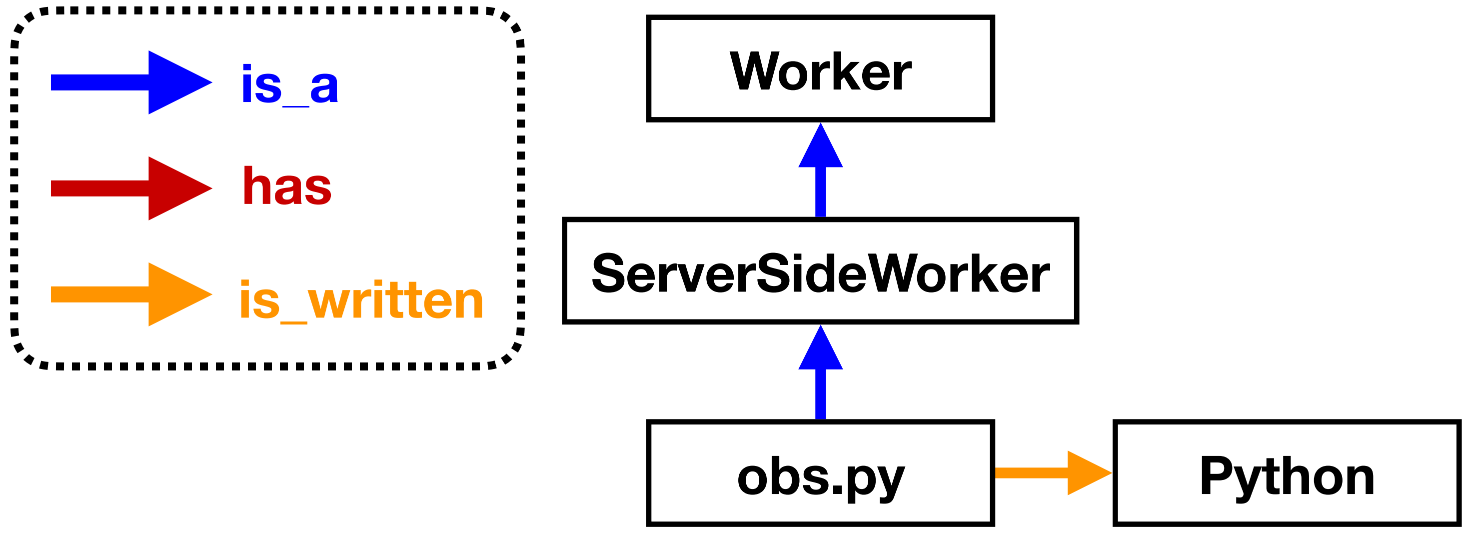
RACCOONS and SciVi Interoperability

Python code:

obs.py

```python
import raccoons

def get_obs(path: str) -> list:
    engine = raccoons.Engine(path)
    query = raccoons.MapReduceQuery(
        num_groups = engine.mission.number_of_stars,
        condition = "{ return observation.starID; }",
        selector = "{ return 1; }",
        reduction = "{ slot += value; }")
    engine.forEachObservation([ query ])
    return query.histogram(num_bins = 50)
```



JavaScript code:

obs.js

```javascript
async function plotObs(path) {
    const obs = await get_obs(path);
    NChart3DLib().then(mdl => {
        const chartConfig = {
            cartesianSystem:
            {
                xAxis:
                {
                    caption: { text: "Observations / Source" },
                    valueMask: "%.0f"
                },
                yAxis:
                {
                    caption: { text: "Log(Amount)" },
                    isLogarithmic: true,
                    valueMask: "%.1e"
                }
            },
            series:
            [
                {
                    type: "column",
                    brush: "#60cce8",
                    borderBrush: "#000000",
                    borderThickness: 1,
                    points:
                    {
                        type: "xy",
                        data: obs
                    }
                }
            ]
        };
        const chart = new mdl.NChart("obsPlotCanvas");
        chart.loadJSON(JSON.stringify(chartConfig));
    });
}
```

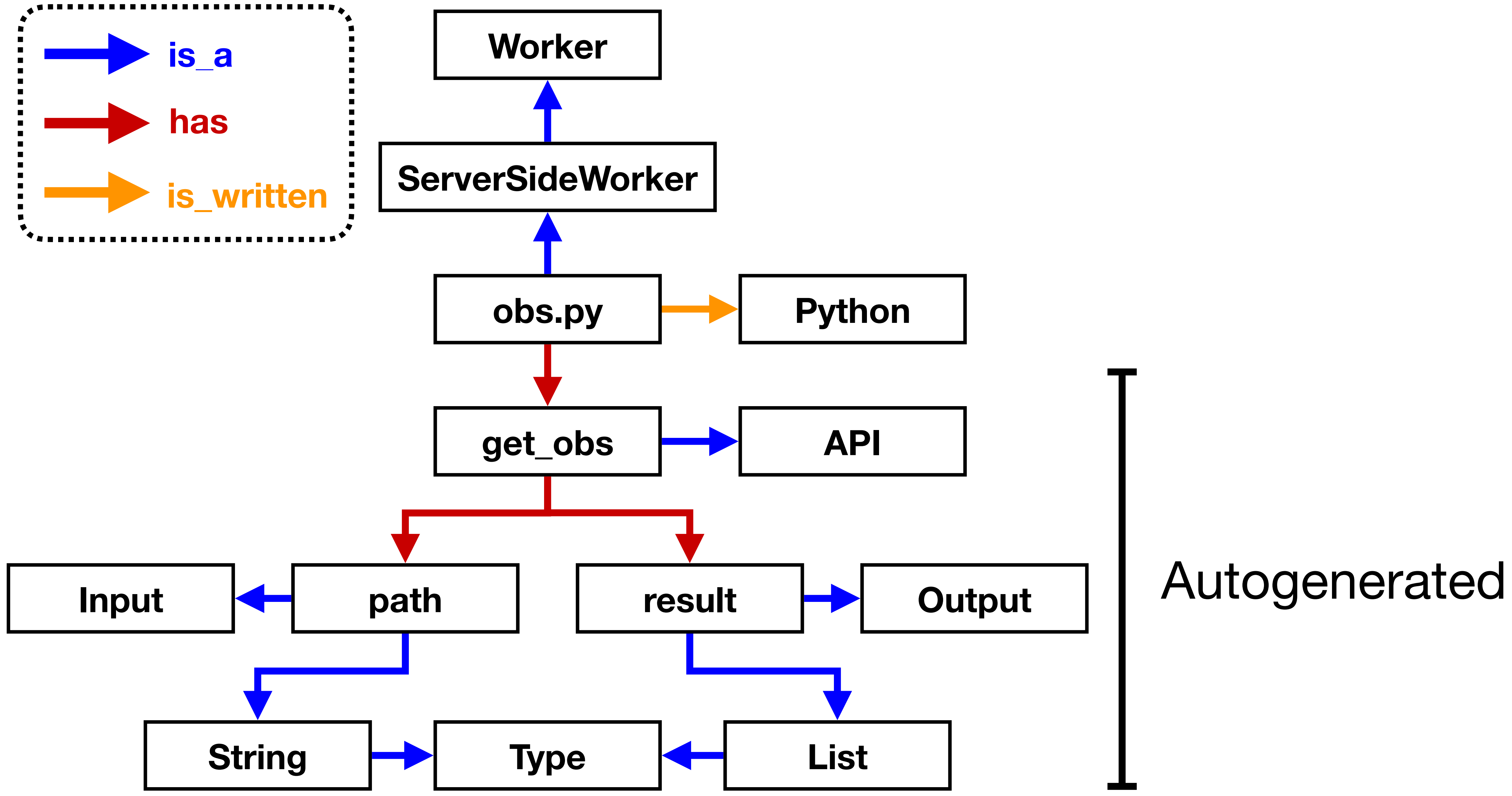
RACCOONS and SciVi Interoperability

Python code:

obs.py

```python
import raccoons

def get_obs(path: str) -> list:
    engine = raccoons.Engine(path)
    query = raccoons.MapReduceQuery(
        num_groups = engine.mission.number_of_stars,
        condition = "{ return observation.starID; }",
        selector = "{ return 1; }",
        reduction = "{ slot += value; }")
    engine.forEachObservation([ query ])
    return query.histogram(num_bins = 50)
```



JavaScript code:

obs.js

```javascript
async function plotObs(path) {
    const obs = await get_obs(path);
    NChart3DLib().then(mdl => {
        const chartConfig = {
            cartesianSystem:
            {
                xAxis:
                {
                    caption: { text: "Observations / Source" },
                    valueMask: "%.0f"
                },
                yAxis:
                {
                    caption: { text: "Log(Amount)" },
                    isLogarithmic: true,
                    valueMask: "%.1e"
                }
            },
            series:
            [
                {
                    type: "column",
                    brush: "#60cce8",
                    borderBrush: "#000000",
                    borderThickness: 1,
                    points:
                    {
                        type: "xy",
                        data: obs
                    }
                }
            ]
        };
        const chart = new mdl.NChart("obsPlotCanvas");
        chart.loadJSON(JSON.stringify(chartConfig));
    });
}
```

RACCOONS and SciVi Interoperability

Python code:

obs.py

```python
import raccoons

def get_obs(path: str) -> list:
    engine = raccoons.Engine(path)
    query = raccoons.MapReduceQuery(
        num_groups = engine.mission.number_of_stars,
        condition = "{ return observation.starID; }",
        selector = "{ return 1; }",
        reduction = "{ slot += value; }")
    engine.forEachObservation([ query ])
    return query.histogram(num_bins = 50)

```

Ontology-Driven FFI

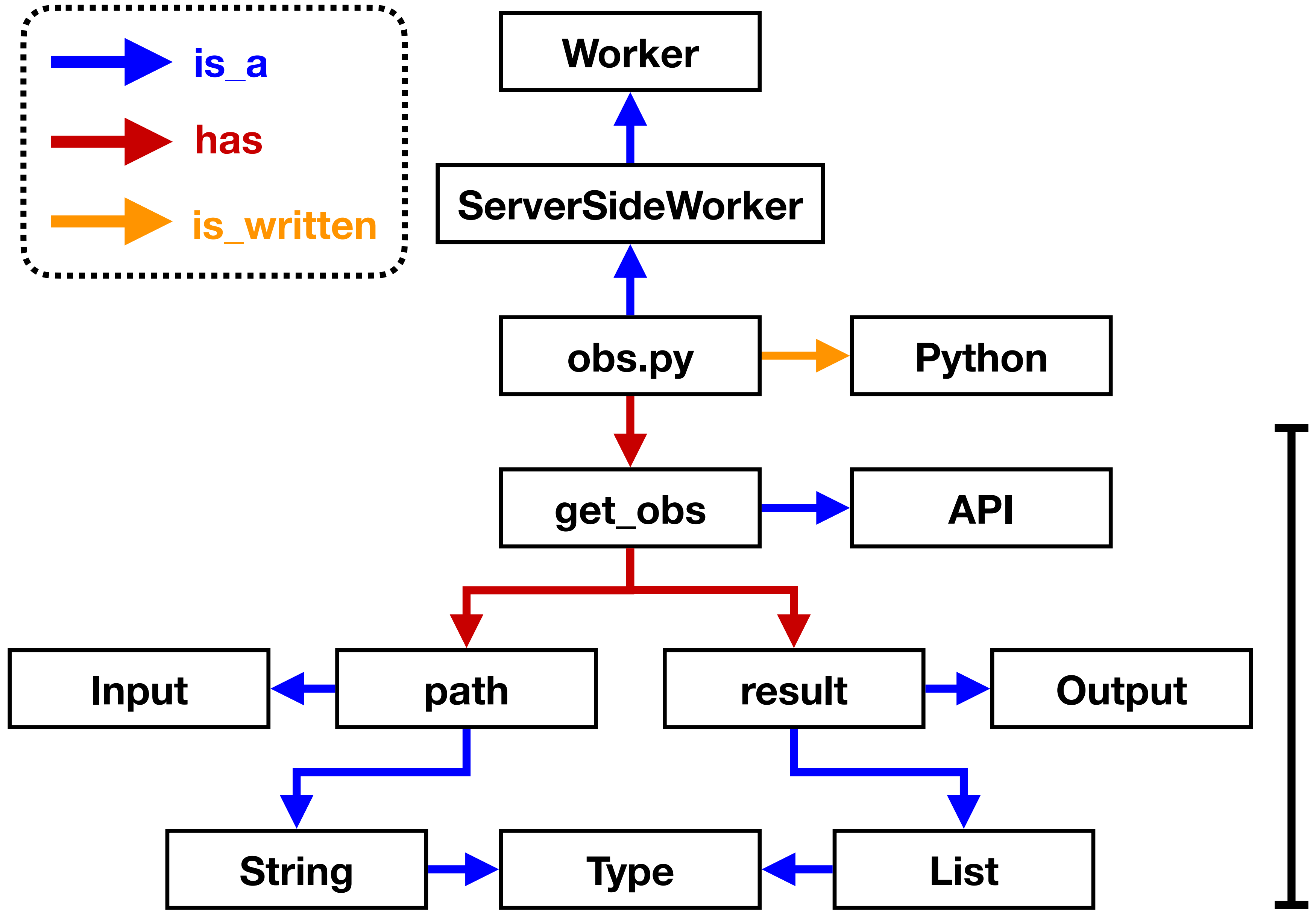
Automatic Marshaling

JavaScript code:

obs.js

```javascript
async function plotObs(path) {
    const obs = await get_obs(path);
    NChart3DLib().then(mdl => {
        const chartConfig = {
            cartesianSystem:
            {
                xAxis:
                {
                    caption: { text: "Observations / Source" },
                    valueMask: "%.0f"
                },
                yAxis:
                {
                    caption: { text: "Log(Amount)" },
                    isLogarithmic: true,
                    valueMask: "%.1e"
                }
            },
            series:
            [
                {
                    type: "column",
                    brush: "#60cce8",
                    borderBrush: "#000000",
                    borderThickness: 1,
                    points:
                    {
                        type: "xy",
                        data: obs
                    }
                }
            ]
        };
        const chart = new mdl.NChart("obsPlotCanvas");
        chart.loadJSON(JSON.stringify(chartConfig));
    });
}

```



Autogenerated

RACCOONS and SciVi Interoperability

Python code:

obs.py

```python
import raccoons

def get_obs(path: str) -> list:
    engine = raccoons.Engine(path)
    query = raccoons.MapReduceQuery(
        num_groups = engine.mission.number_of_stars,
        condition = "{ return observation.starID; }",
        selector = "{ return 1; }",
        reduction = "{ slot += value; }")
    engine.forEachObservation([ query ])
    return query.histogram(num_bins = 50)

```

Ontology-Driven FFI

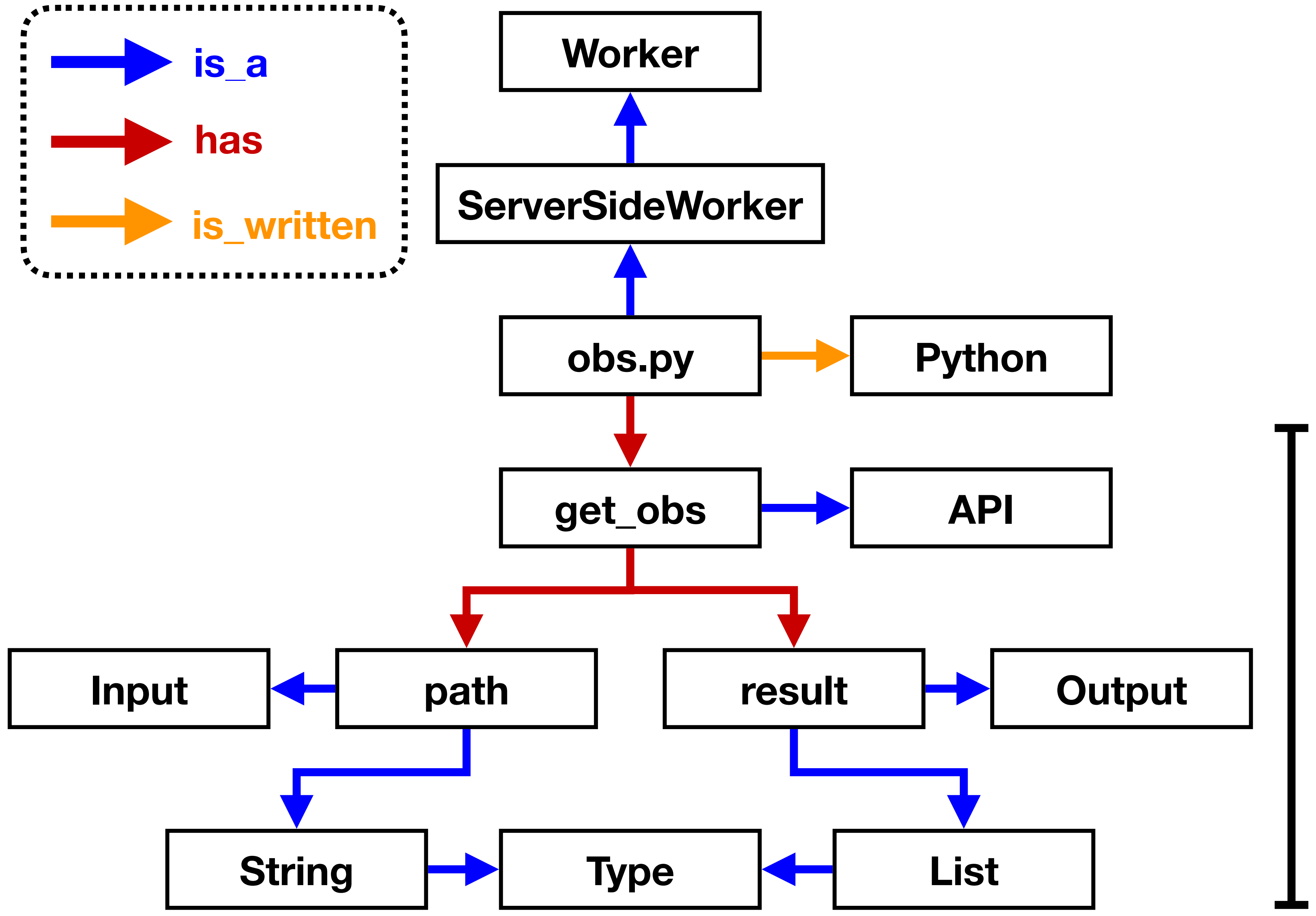
Automatic Marshaling

JavaScript code:

obs.js

```javascript
async function plotObs(path) {
    const obs = await get_obs(path);
    ...Lib().then(mdl => {
        const chartConfig = {
            cartesianSystem:
            {
                xAxis:
                {
                    caption: { text: "Observations / Source" },
                    valueMask: "%.0f"
                },
                yAxis:
                {
                    caption: { text: "Log(Amount)" },
                    isLogarithmic: true,
                    valueMask: "%.1e"
                }
            },
            series:
            [
                {
                    type: "column",
                    brush: "#60cce8",
                    borderBrush: "#000000",
                    borderThickness: 1,
                    points:
                    {
                        type: "xy",
                        data: obs
                    }
                }
            ]
        };
        const chart = new mdl.NChart("obsPlotCanvas");
        chart.loadJSON(JSON.stringify(chartConfig));
    });
}

```



Autogenerated

RACCOONS and SciVi Interoperability

Python code:

obs.py

```python
import raccoons

def get_obs(path: str) -> list:
    engine = raccoons.Engine(path)
    query = raccoons.MapReduceQuery(
        num_groups = engine.mission.number_of_stars,
        condition = "{ return observation.starID; }",
        selector = "{ return 1; }",
        reduction = "{ slot += value; }")
    engine.forEachObservation([ query ])
    return query.histogram(num_bins = 50)
```

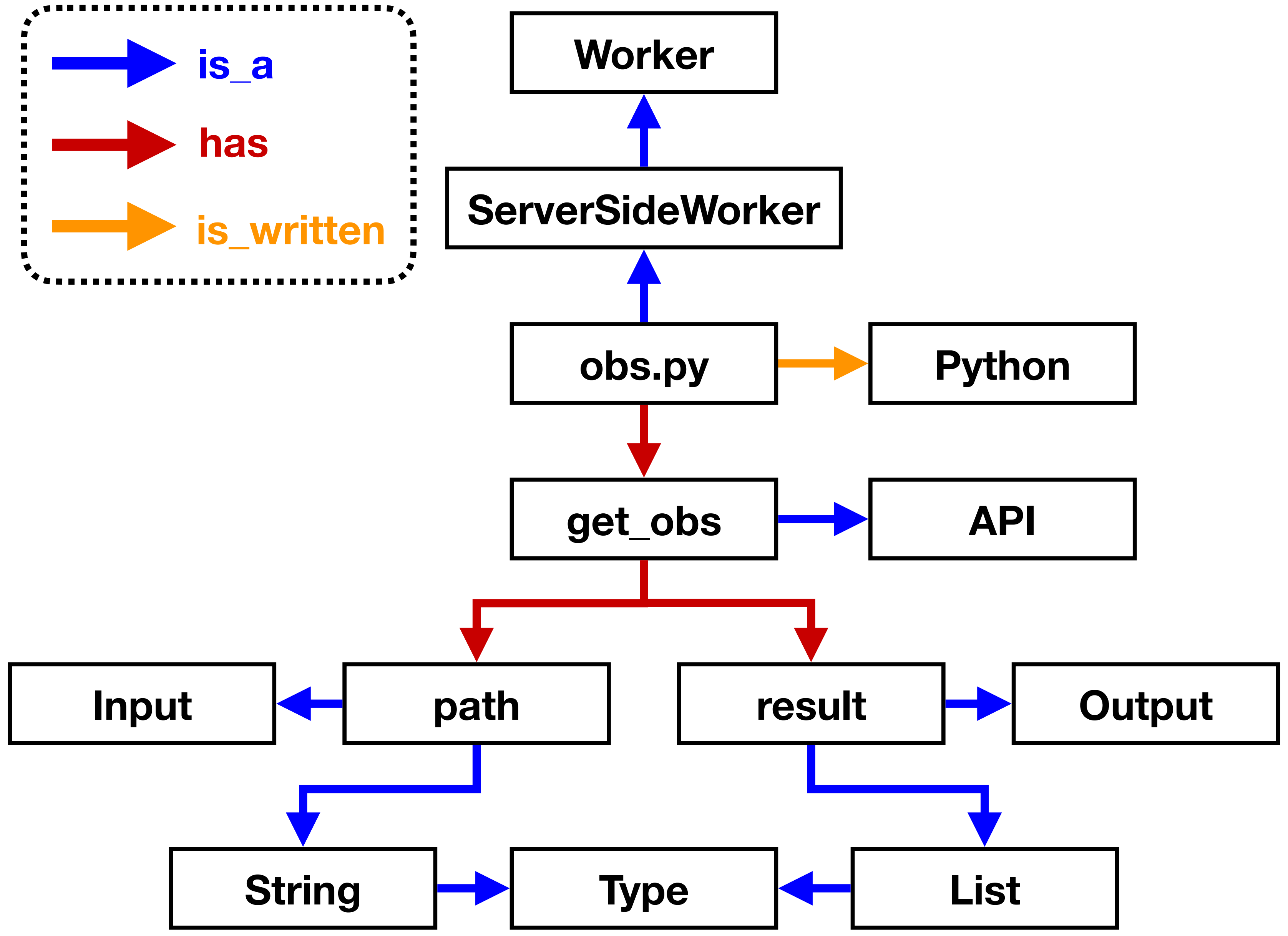
Ontology-Driven FFI

Automatic Marshaling

JavaScript code:

obs.js

```javascript
async function plotObs(path) {
    const obs = await get_obs(path);
    [illegible]Lib().then(mdl => {
        const chartConfig = {
            cartesianSystem:
            {
                xAxis:
                {
                    caption: { text: "Observations / Source" },
                    valueMask: "%.0f"
                },
                yAxis:
                {
                    caption: { text: "Log(Amount)" },
                    isLogarithmic: true,
                    valueMask: "%.1e"
                }
            },
            series:
            [
                {
                    type: "column",
                    brush: "#60cce8",
                    borderBrush: "#000000",
                    borderThickness: 1,
                    points:
                    {
                        type: "xy",
                        data: obs
                    }
                }
            ]
        };
        const chart = new mdl.NChart("obsPlotCanvas");
        chart.loadJSON(JSON.stringify(chartConfig));
    });
}
```



Autogenerated

RACCOONS and SciVi Interoperability

Python code:

obs.py

```python
import raccoons

def get_obs(path: str) -> list:
    engine = raccoons.Engine(path)
    query = raccoons.MapReduceQuery(
        num_groups = engine.mission.number_of_stars,
        condition = "{ return observation.starID; }",
        selector = "{ return 1; }",
        reduction = "{ slot += value; }")
    engine.forEachObservation([ query ])
    return query.histogram(num_bins = 50)

```

Ontology-Driven FFI

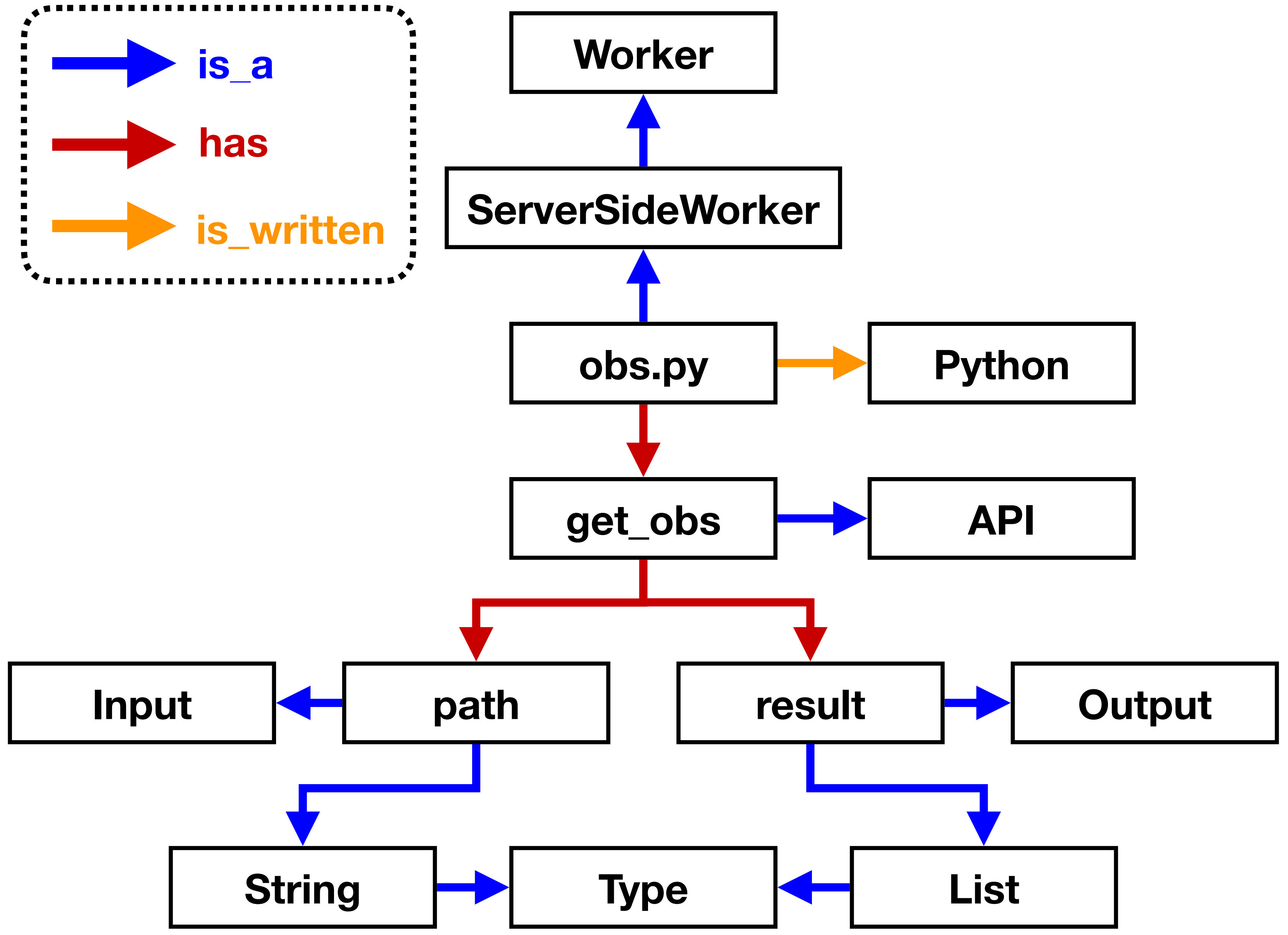
Automatic Marshaling

JavaScript code:

obs.js

```javascript
async function plotObs(path) {
    const obs = await get_obs(path);
    [illegible]Lib().then(mdl => {
        const chartConfig = {
            cartesianSystem:
            {
                xAxis:
                {
                    caption: { text: "Observations / Source" },
                    valueMask: "%.0f"
                },
                yAxis:
                {
                    caption: { text: "Log(Amount)" },
                    isLogarithmic: true,
                    valueMask: "%.1e"
                }
            },
            series:
            [
                {
                    type: "column",
                    brush: "#60cce8",
                    borderBrush: "#000000",
                    borderThickness: 1,
                    points:
                    {
                        type: "xy",
                        data: obs
                    }
                }
            ]
        };
        const chart = new mdl.NChart("obsPlotCanvas");
        chart.loadJSON(JSON.stringify(chartConfig));
    });
}

```



Autogenerated

NChart3D

https://nchart3d.com/

RACCOONS and SciVi Interoperability

Python code:

obs.py

```python
import raccoons

def get_obs(path: str) -> list:
    engine = raccoons.Engine(path)
    query = raccoons.MapReduceQuery(
        num_groups = engine.mission.number_of_stars,
        condition = "{ return observation.starID; }",
        selector = "{ return 1; }",
        reduction = "{ slot += value; }")
    engine.forEachObservation([ query ])
    return query.histogram(num_bins = 50)
```

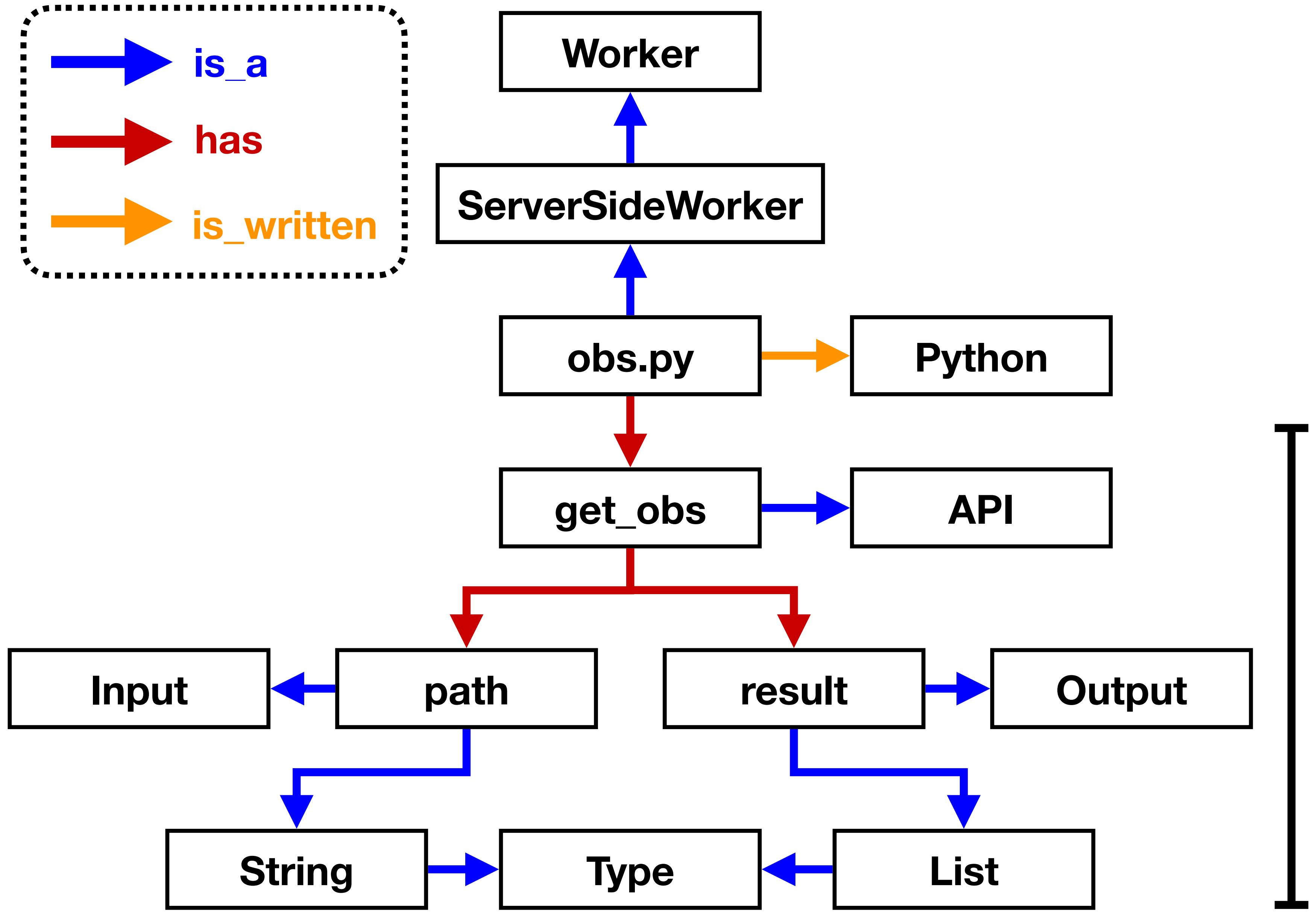
Ontology-Driven FFI

Automatic Marshaling

JavaScript code:

obs.js

```javascript
async function plotObs(path) {
    const obs = await get_obs(path);
    [illegible]Lib().then(mdl => {
        const chartConfig = {
            cartesianSystem:
            {
                xAxis:
                {
                    caption: { text: "Observations / Source" },
                    valueMask: "%.0f"
                },
                yAxis:
                {
                    caption: { text: "Log(Amount)" },
                    isLogarithmic: true,
                    valueMask: "%.1e"
                }
            },
            series:
            [
                {
                    type: "column",
                    brush: "#60cce8",
                    borderBrush: "#000000",
                    borderThickness: 1,
                    points:
                    {
                        type: "xy",
                        data: obs
                    }
                }
            ]
        };
        const chart = new mdl.NChart("obsPlotCanvas");
        chart.loadJSON(JSON.stringify(chartConfig));
    });
}
```



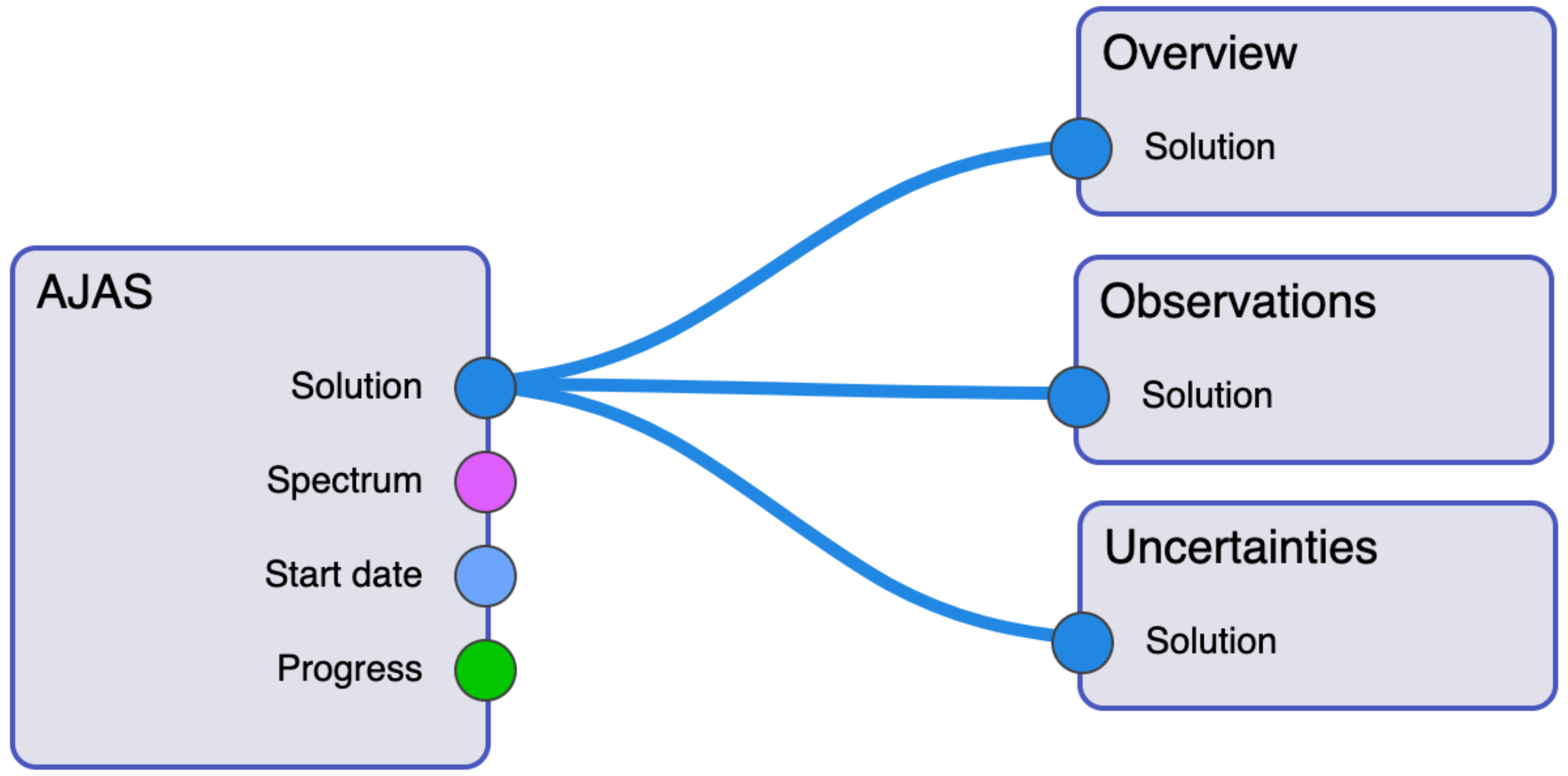
Autogenerated

- Written in C++
- Compiled with Emscripten for Web
- Uses GPU for rendering
- Can handle 1M+ data points and remains interactive

NChart3D

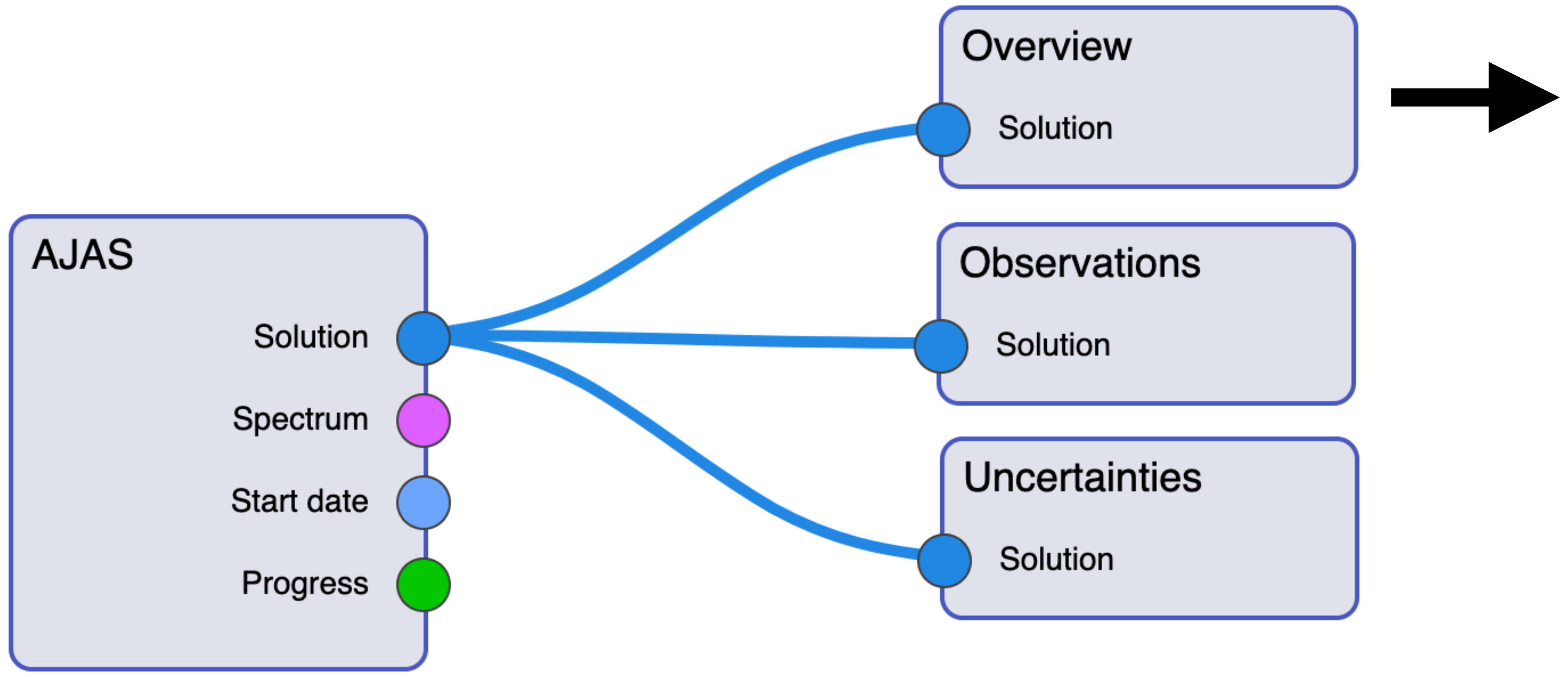
https://nchart3d.com/

Dataflow editor generated by SciVi



Solution Analysis & Uncertainty Assessment

Dataflow editor generated by SciVi



Interactive visualization in SciVi

Mission Overview

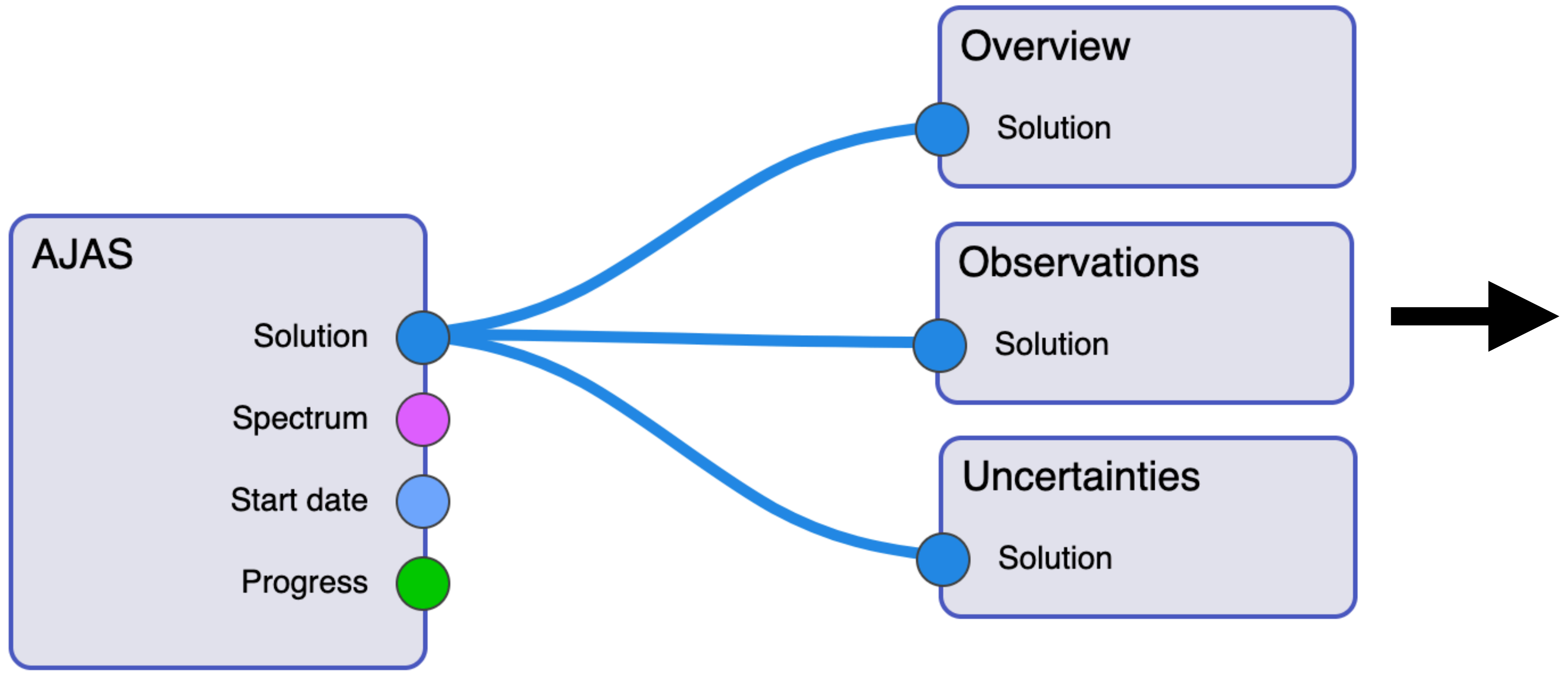
1.	Number of sources	11271
2.	Number of Gaia priors	5684
3.	Number of exposures	25200
4.	Number of detectors	4
5.	Number of astrometric parameters	2
6.	Number of calibration orders	6
7.	Number of units of each order	[25200,25200,1,1,1,1]
8.	Number of 2D-observations	14132556
9.	Number of unknowns in the system	627486
10.	Number of equations in the system	28265112
11.	Size of reduced normal matrix of the system	22686

AJAS Statistics

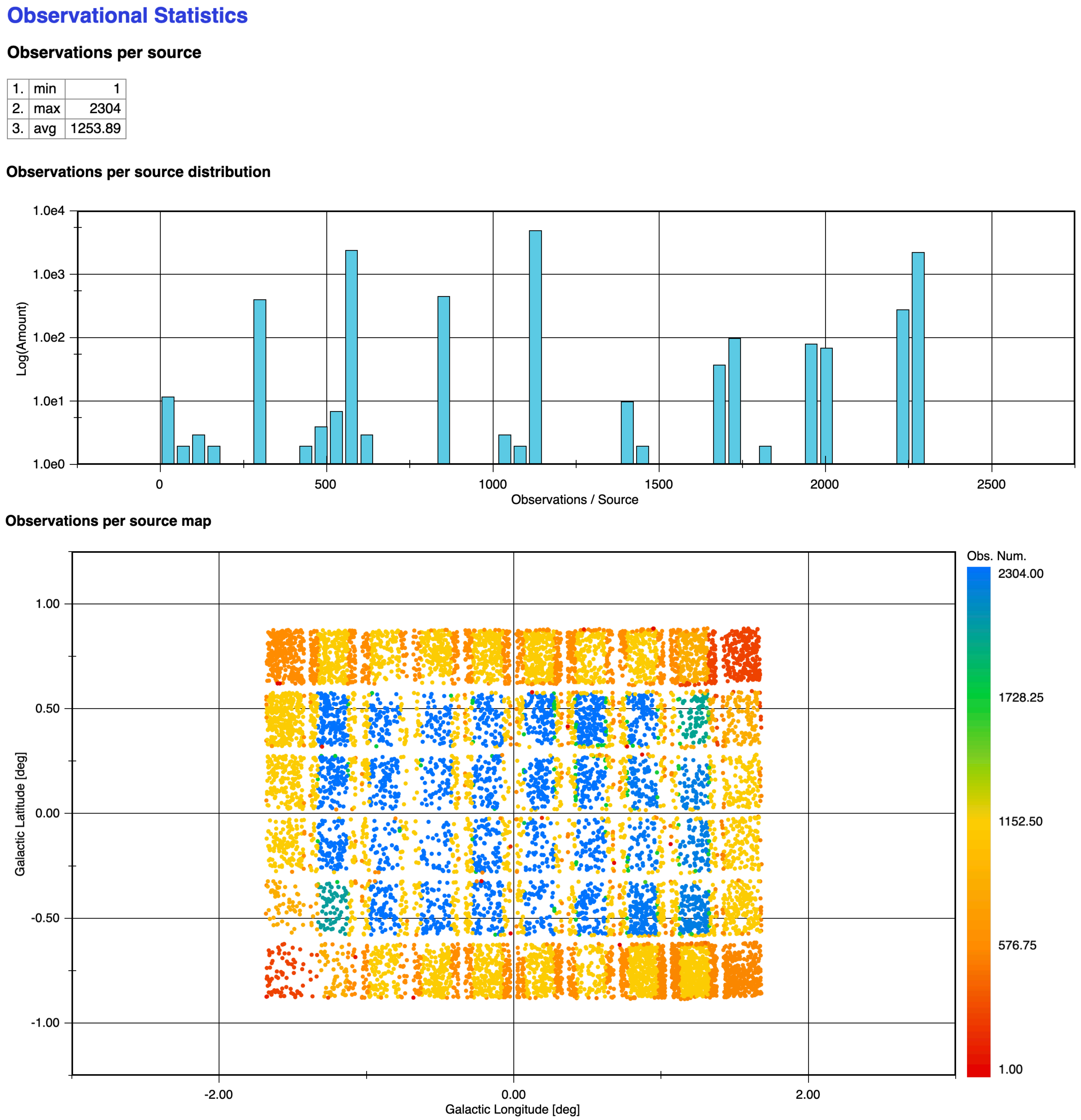
1.	AJAS Git hash	7418a21f672dc6234a26e6a7a372d2bedcf6755b
2.	Process grid	2 × 2
3.	Number of building threads per process	1
4.	Number of summation threads per process	1
5.	Matrix building time	00:00:57.661
6.	Eigenproblem solving time	00:07:15.376
7.	Eigenvalues filtering time	00:00:00.0
8.	Pseudoinverse matrix calculation time	00:01:04.866
9.	Backsubstitution and resuduials calculation time	00:00:00.513
10.	Total solving time	00:09:18.416

Solution Analysis & Uncertainty Assessment

Dataflow editor generated by SciVi

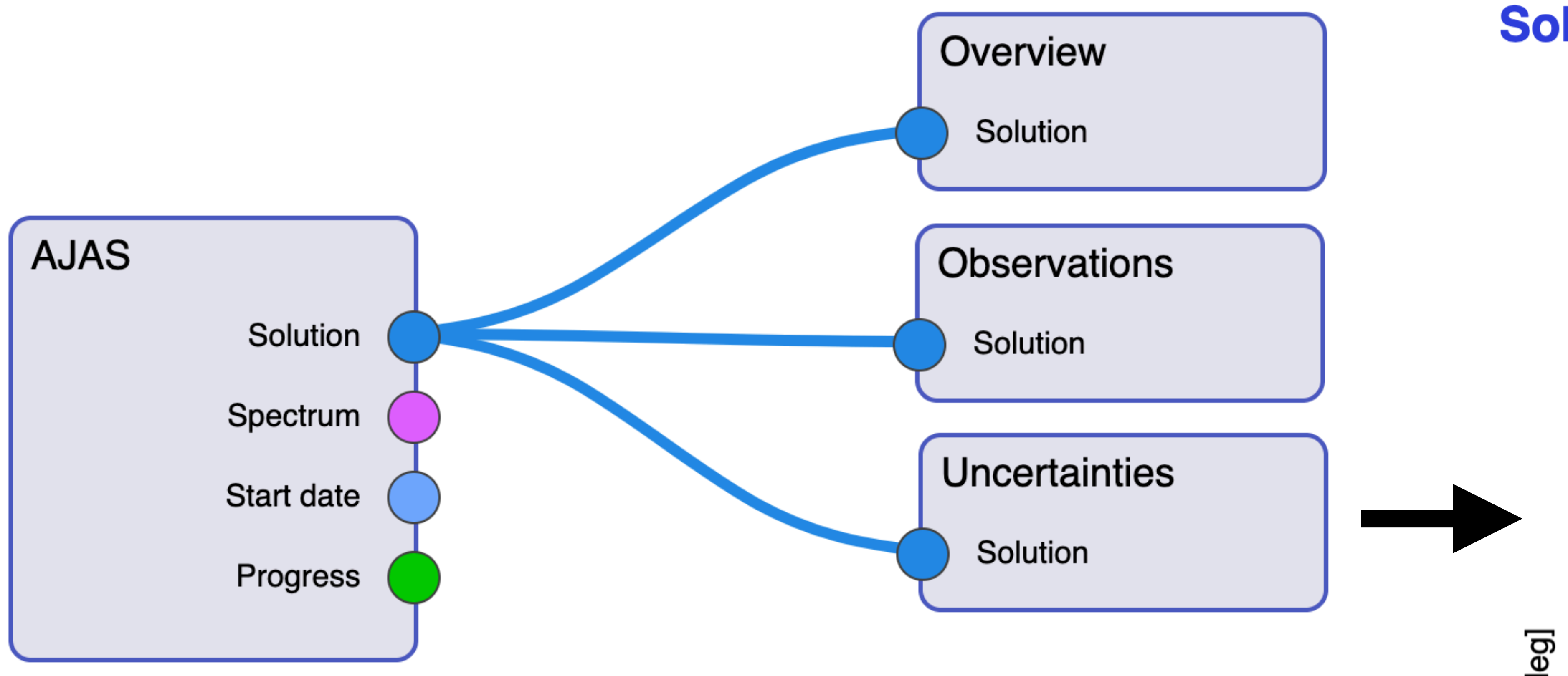


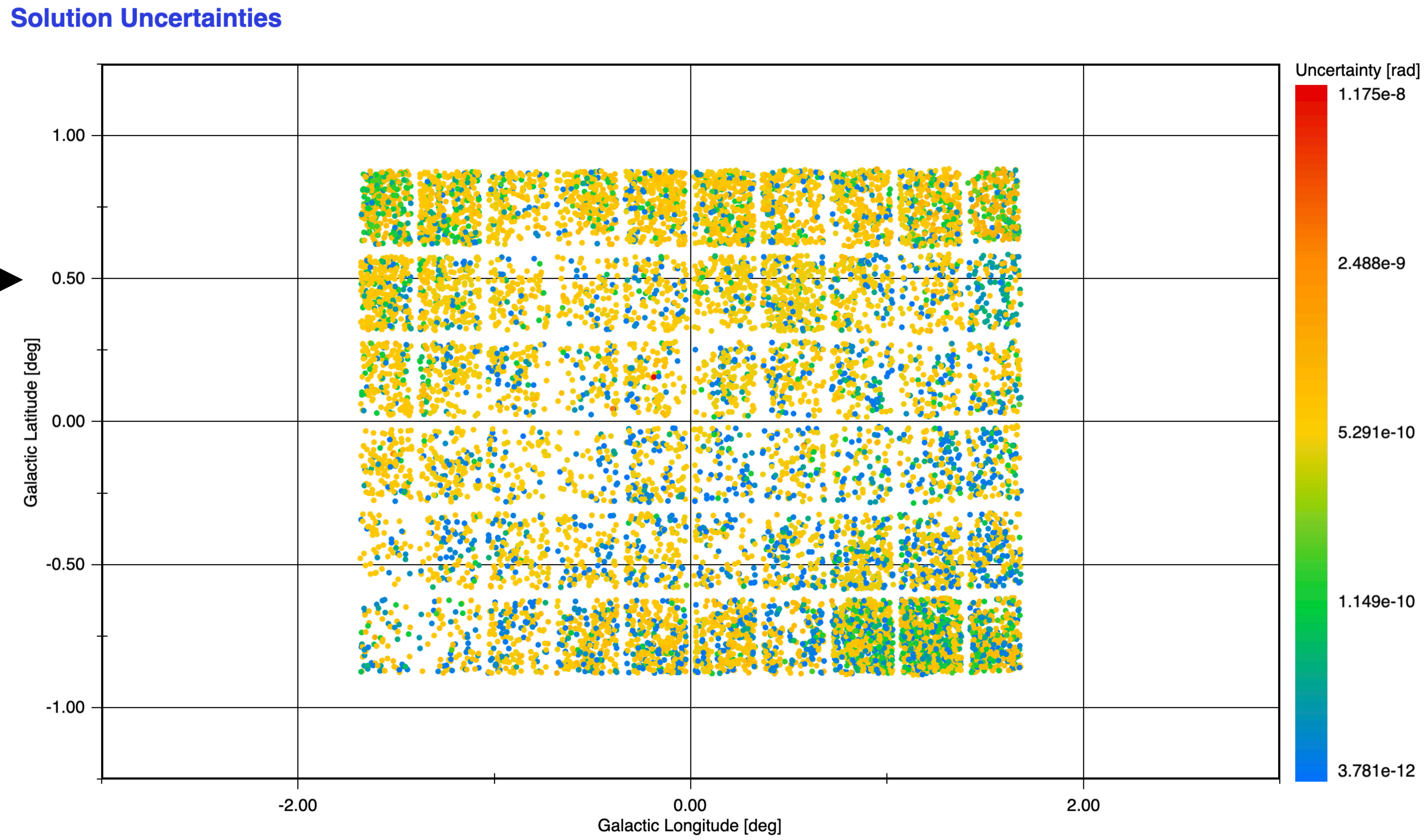
Interactive visualization in SciVi



Dataflow editor generated by SciVi

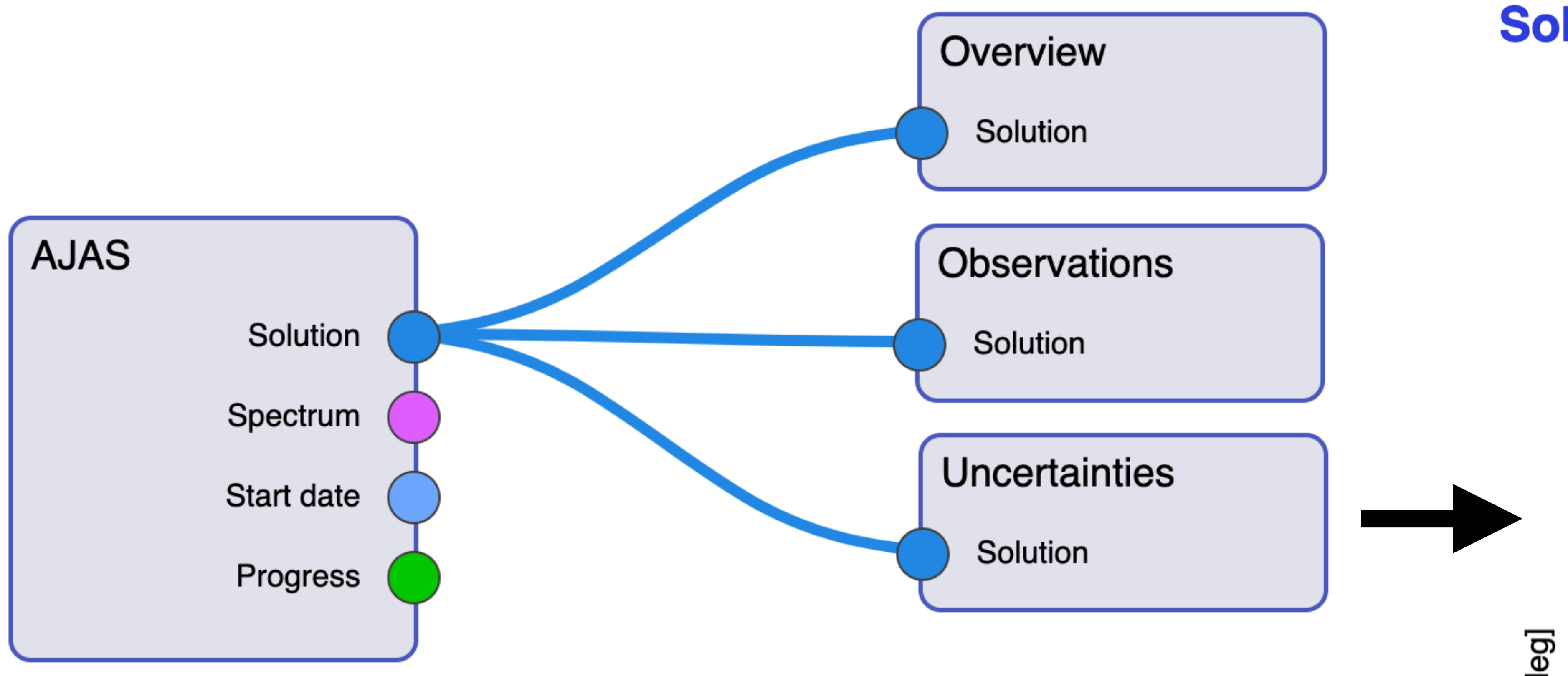
Interactive visualization in SciVi



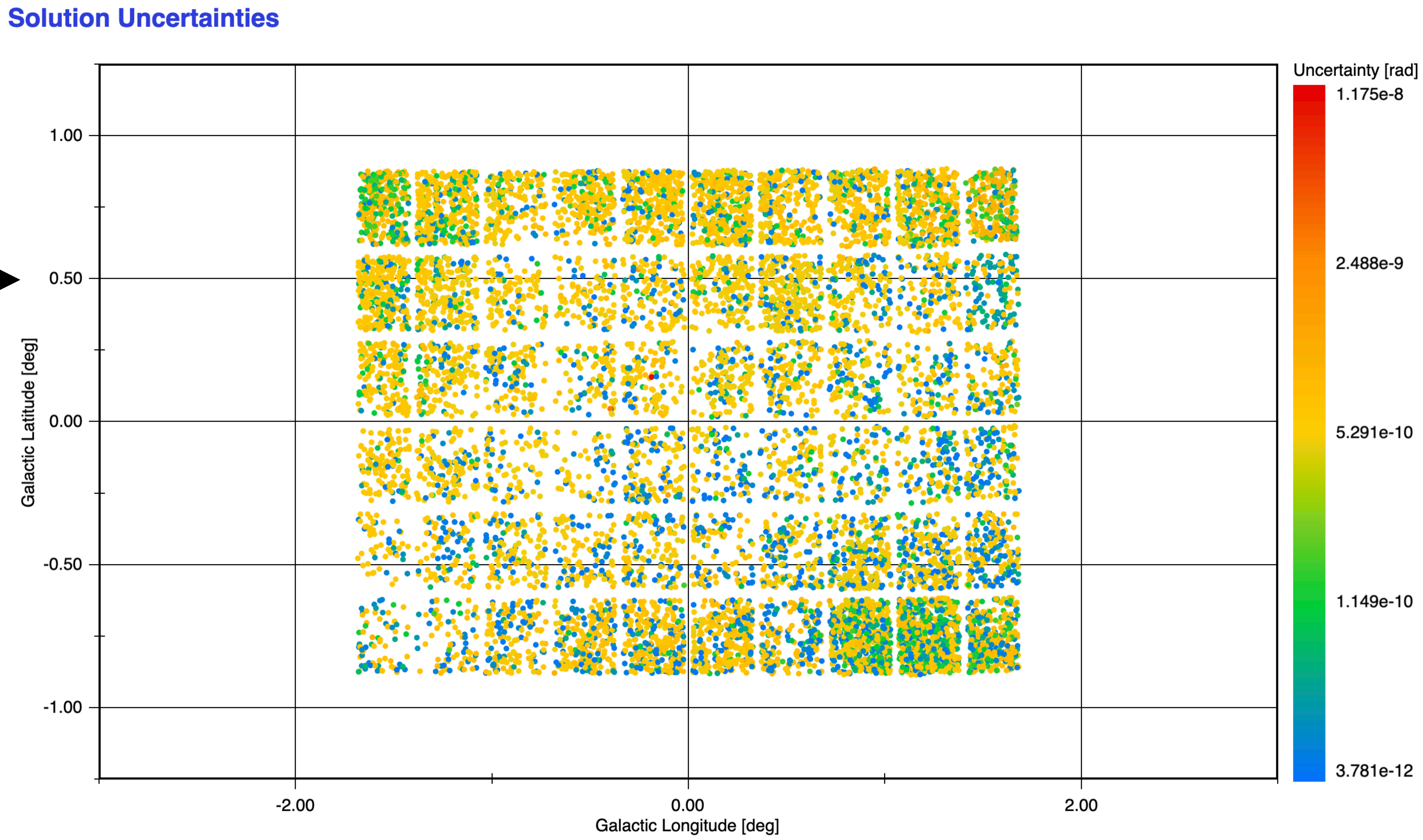


Solution Analysis & Uncertainty Assessment

Dataflow editor generated by SciVi



Interactive visualization in SciVi

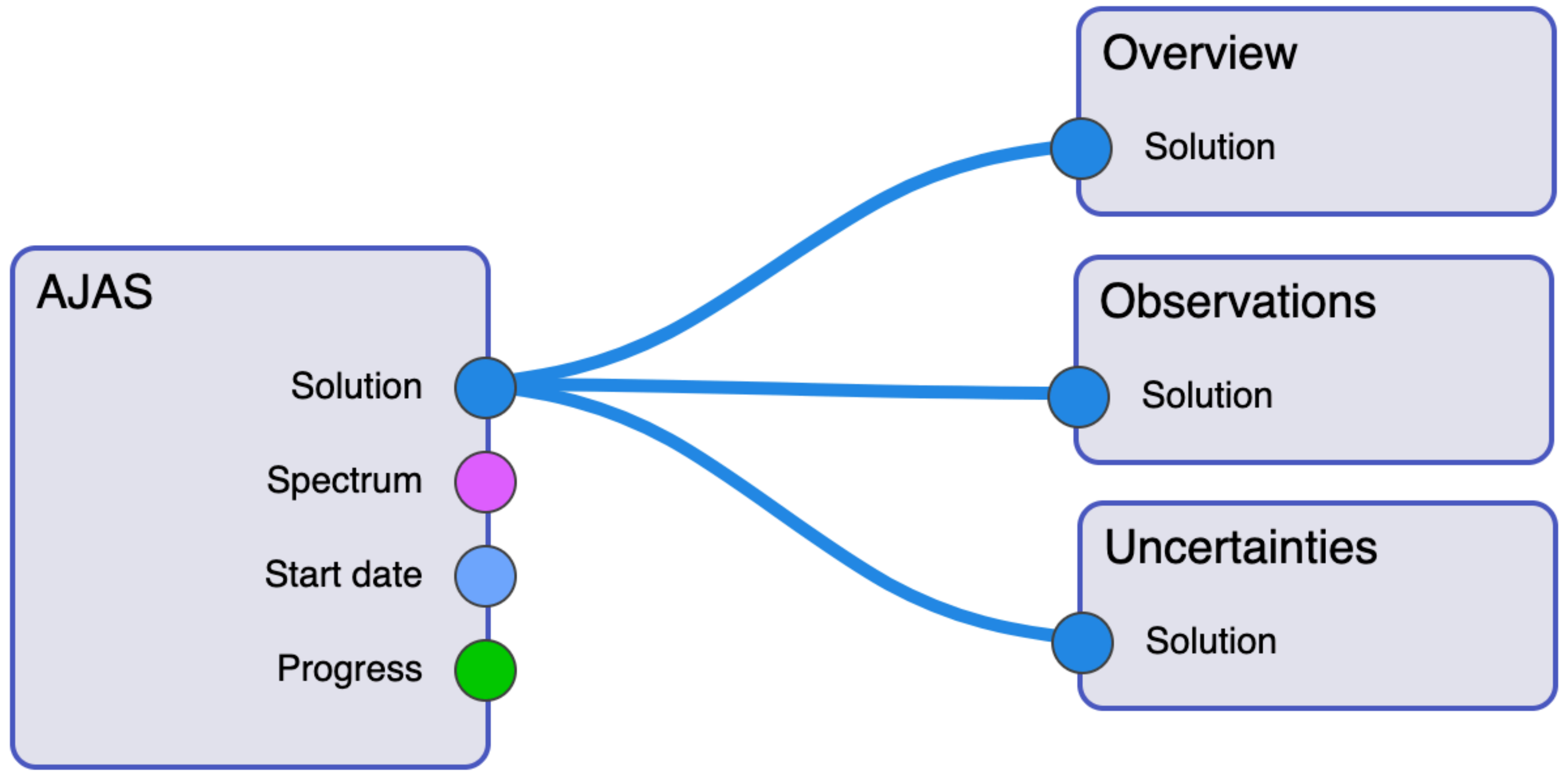


Shneiderman's mantra
of Visual Analytics:
"*Overview first,*
zoom and filter,
then details-on-demand"

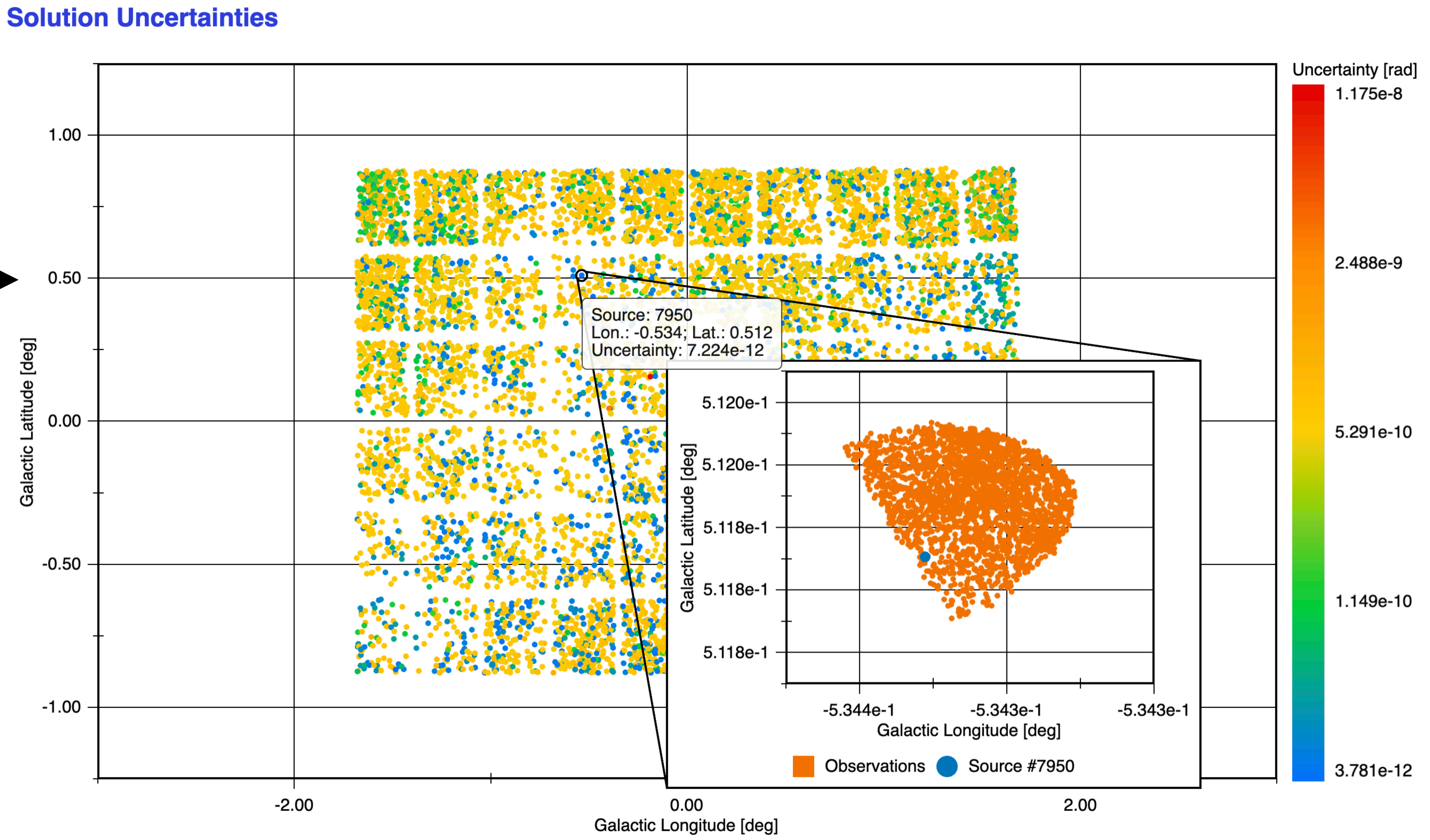
(Shneiderman, 1996)

Solution Analysis & Uncertainty Assessment

Dataflow editor generated by SciVi



Interactive visualization in SciVi



Shneiderman's mantra
of Visual Analytics:
"*Overview first,*
zoom and filter,
then details-on-demand"

(Shneiderman, 1996)

Conclusions and Future Work

Achieved:

1. Astrometry bridged with the In-Situ Visual Analytics
2. For AJAS:
    a. Tractability loop organised using SciVi
    b. High-performance data access library (RACCOONS) developed
3. For SciVi: ontology-driven API proposed and developed

Future Work:

1. Develop more operators for solution analysis in SciVi
2. Further optimise data access routines in RACCOONS

ZENTRUM FÜR ASTRONOMIE

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

Thank you for attention!

This work is financially supported
by the German Aerospace Agency
(Deutsches Zentrum für Luft- und Raumfahrt e.V., DLR)
through grant 50OD2201

Konstantin Ryabinin,
konstantin.riabinin@uni-heidelberg.de

Wolfgang Löffler,
loeffler@ari.uni-heidelberg.de,

Olga Erokhina,
olga.erokhina@uni-heidelberg.de

Gerasimos Sarras,
gerasimos.sarras@uni-heidelberg.de

Michael Biermann,
biermann@ari.uni-heidelberg.de

Singapore – 2025