UNMANNED SYSTEMS LAB

TEXAS A&M UNIVERSITY
J. Mike Walker '66 Department of Mechanical Engineering

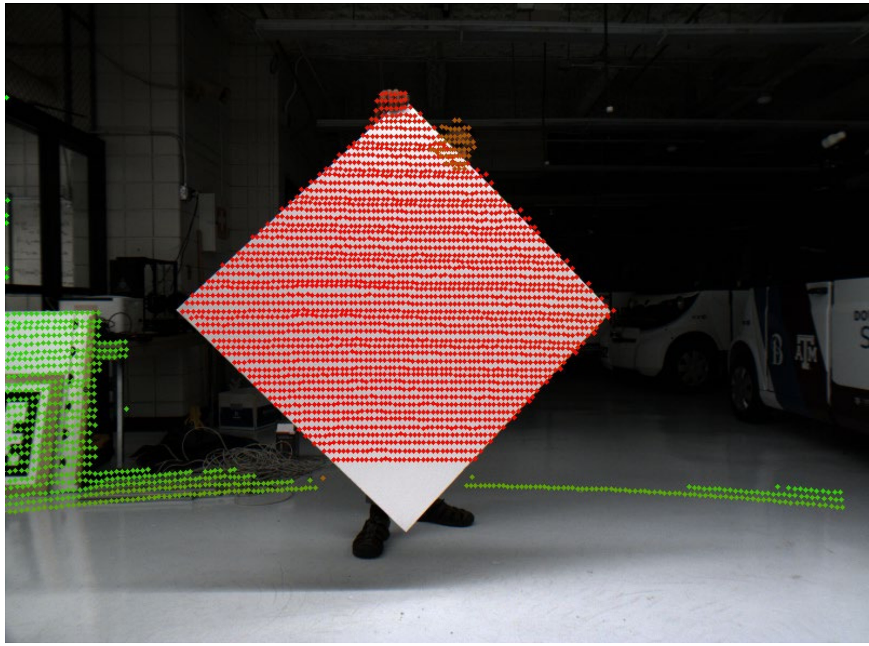
An Integrated Modeling and Testing Architecture for ROS Nodes

Jacob Hartzer

ROSCon 2023

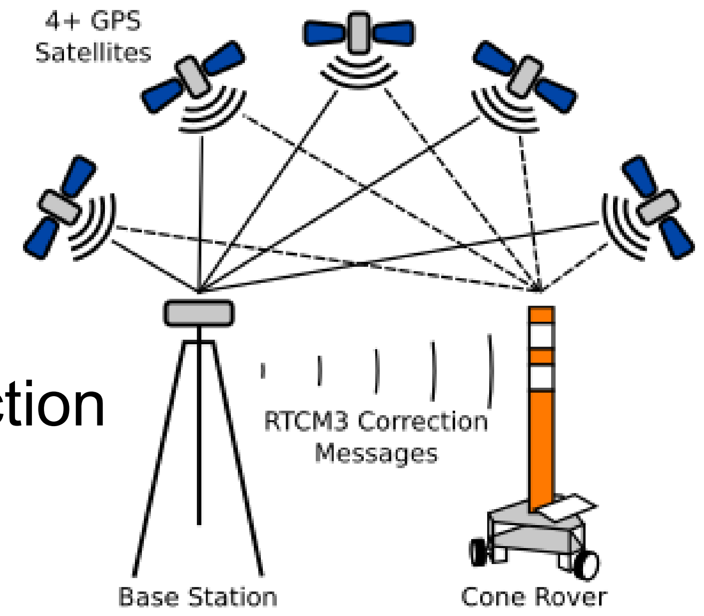
Unmanned Systems Lab

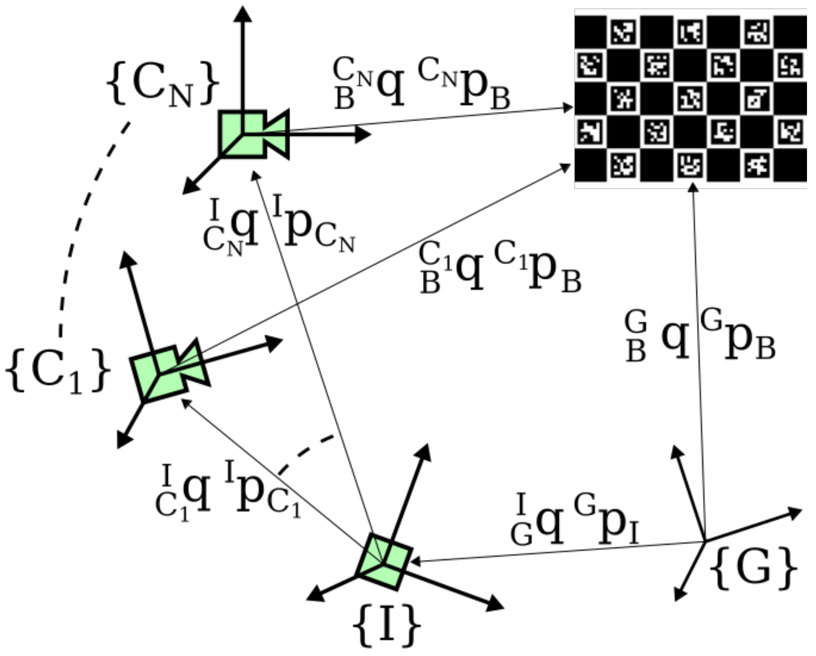
- On/Off Road Autonomy
- LIDAR and RADAR Odometry and Perception
- Multi-Sensor Fusion and Calibration

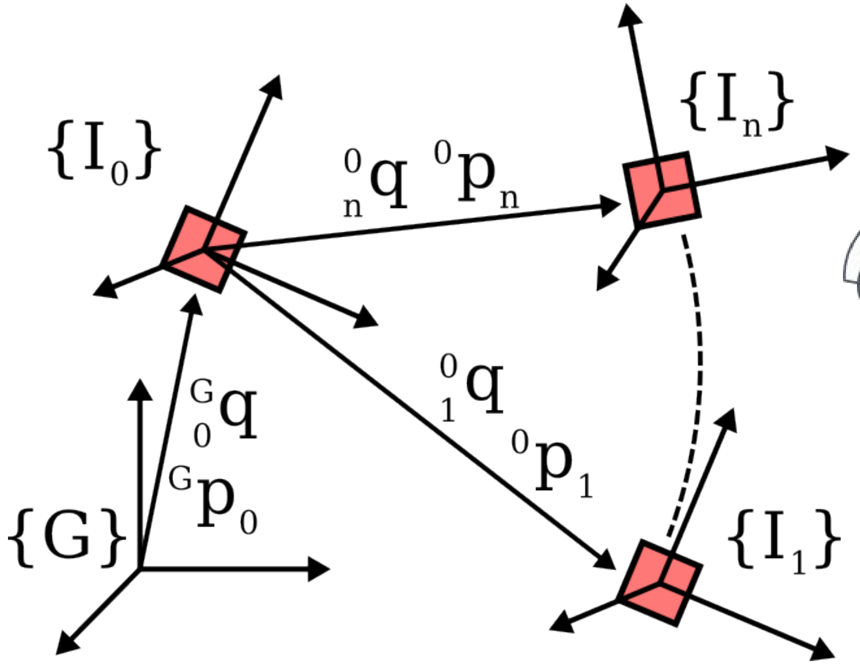


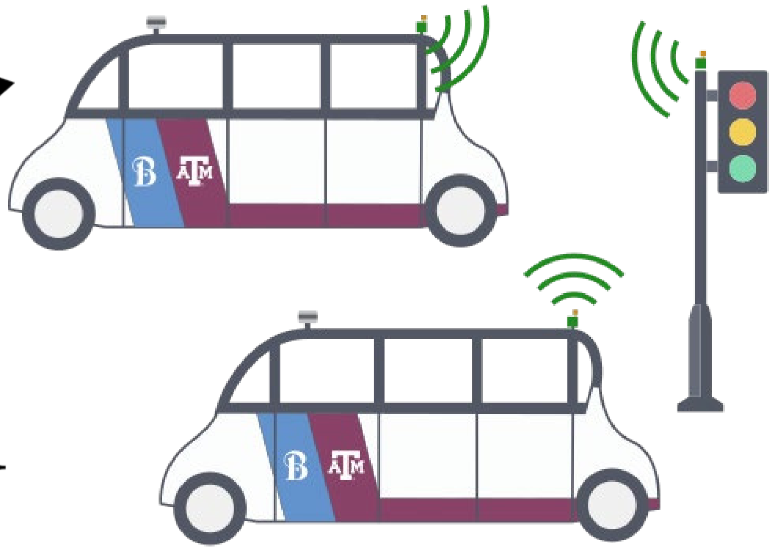
Our Work - Calibration

- Utilize a wide variety of sensors and environments
- RTK GPS in highway applications
- Ultra-Wideband for cooperative localization and detection
- Multi-IMU, Multi-Camera fusion





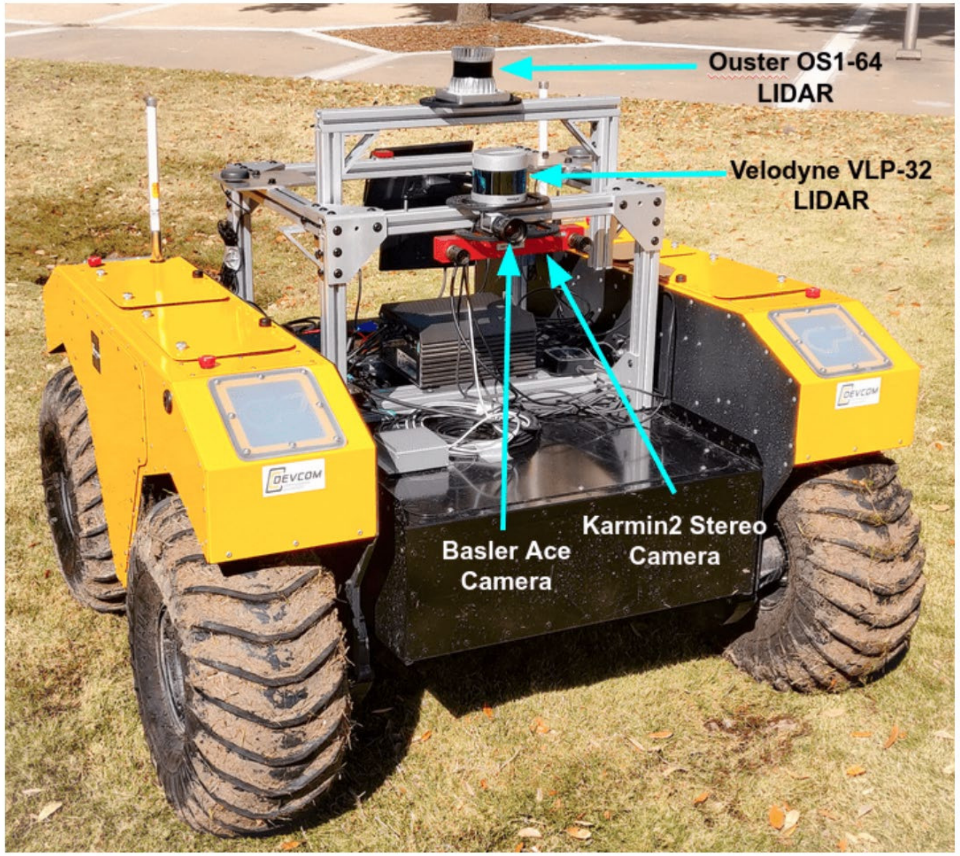




The Challenge of Calibration

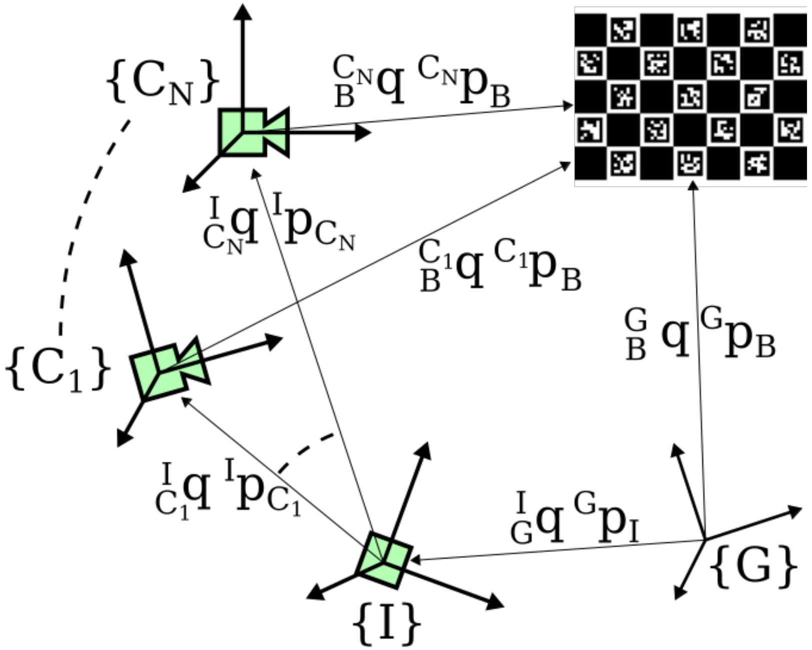
- Sensors do not work well when uncalibrated
- Ideally would like to calibrate sensors *online*
- Re-calibrating on the fly is more robust
- Proving stability and consistency is <u>hard</u>
- Sensor flexibility makes it even harder

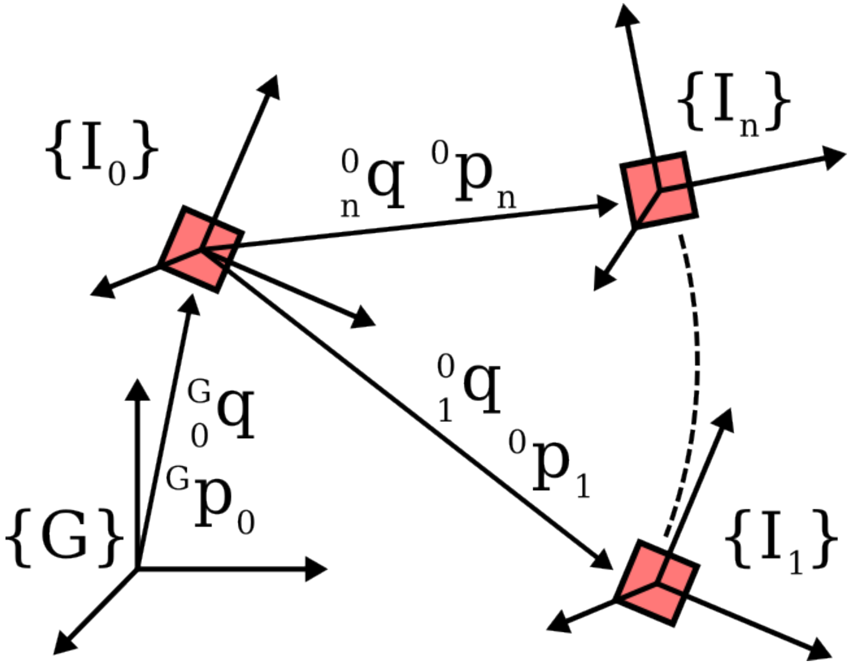
Must test many, many datasets and environments



EKF-CAL Package

- EKF-CAL is flexible MSCKF-based sensor calibration package
- Inputs are YAML based and compatible with ROS 2 parameter declarations
- Inherently multi-sensor (IMU, Camera, and GPS soon)
- Developed with integrated testing and Monte Carlo simulation in mind
- Open Source!





Typical Development

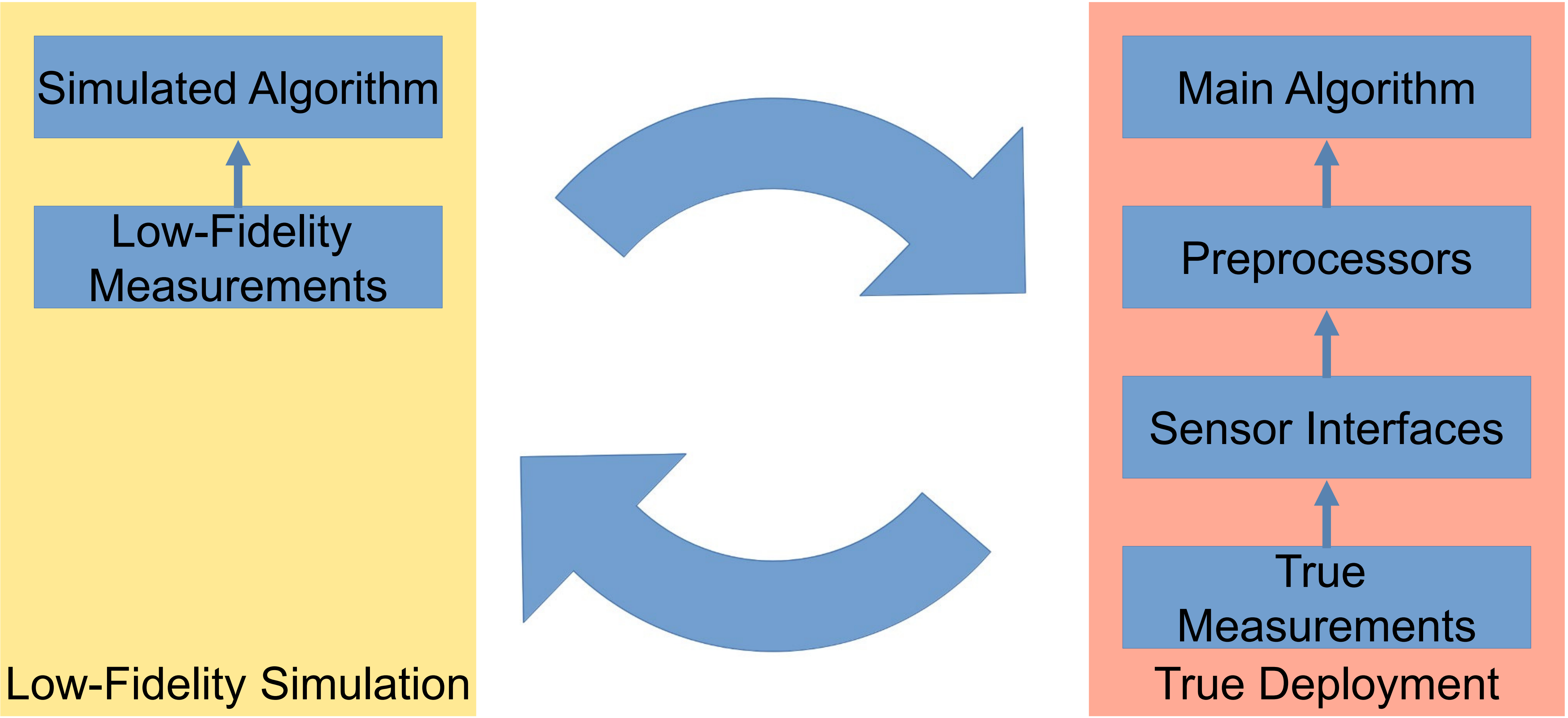
Need performance / Real-Time

Develop Algorithm in scripted language (MatLab, Julia, etc.)

Deploy code in compiled language (C++, Rust, etc.)

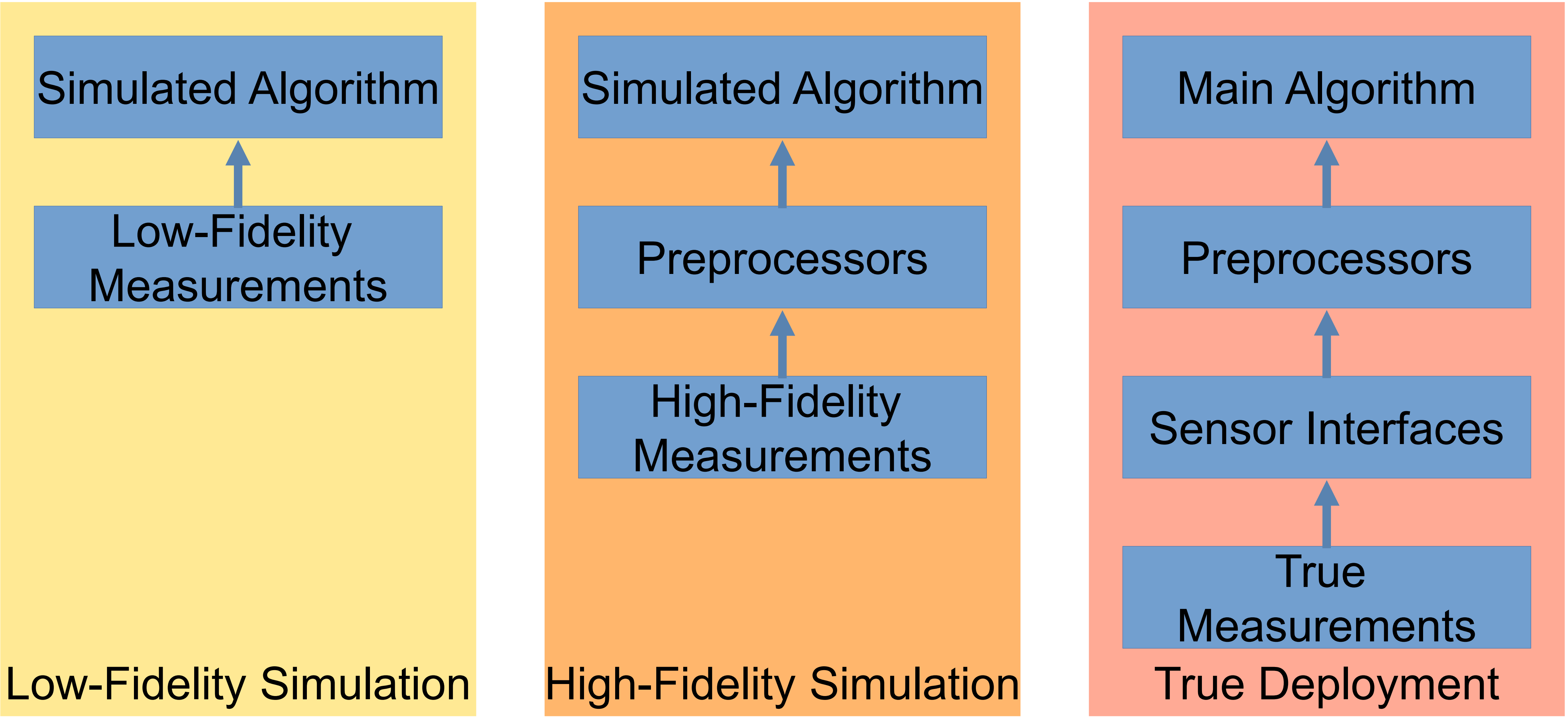
Find a bug / invalid assumptions

Typical Development

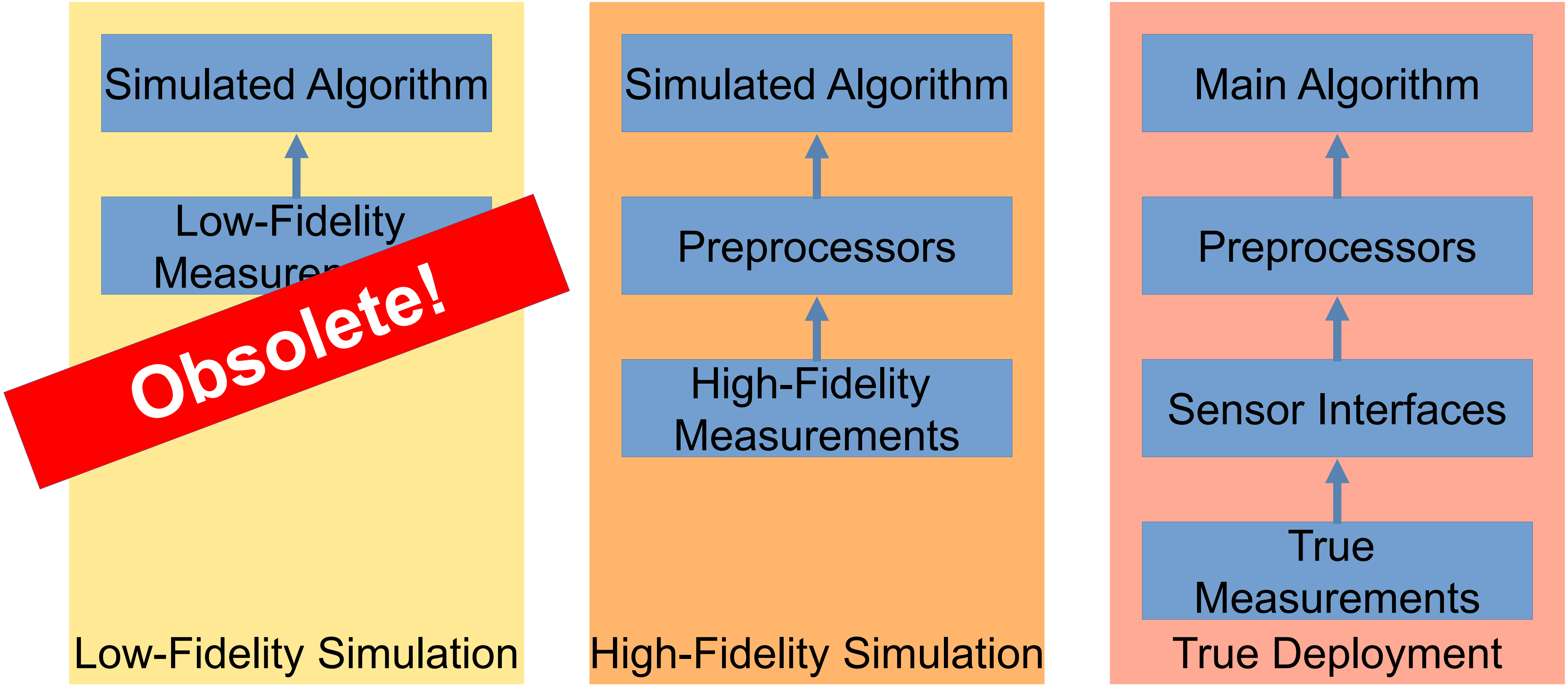


Typical Development

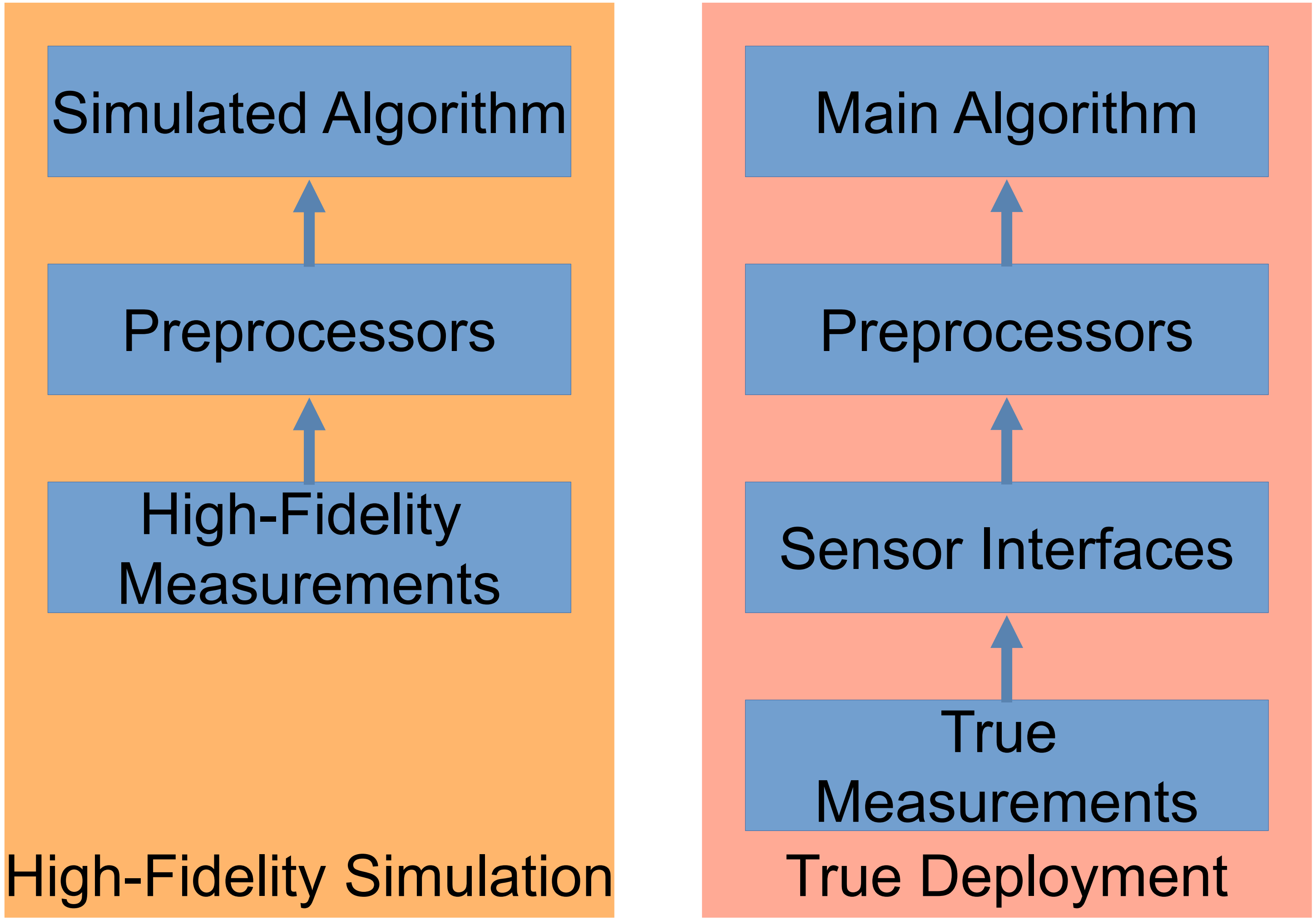
Typical Development



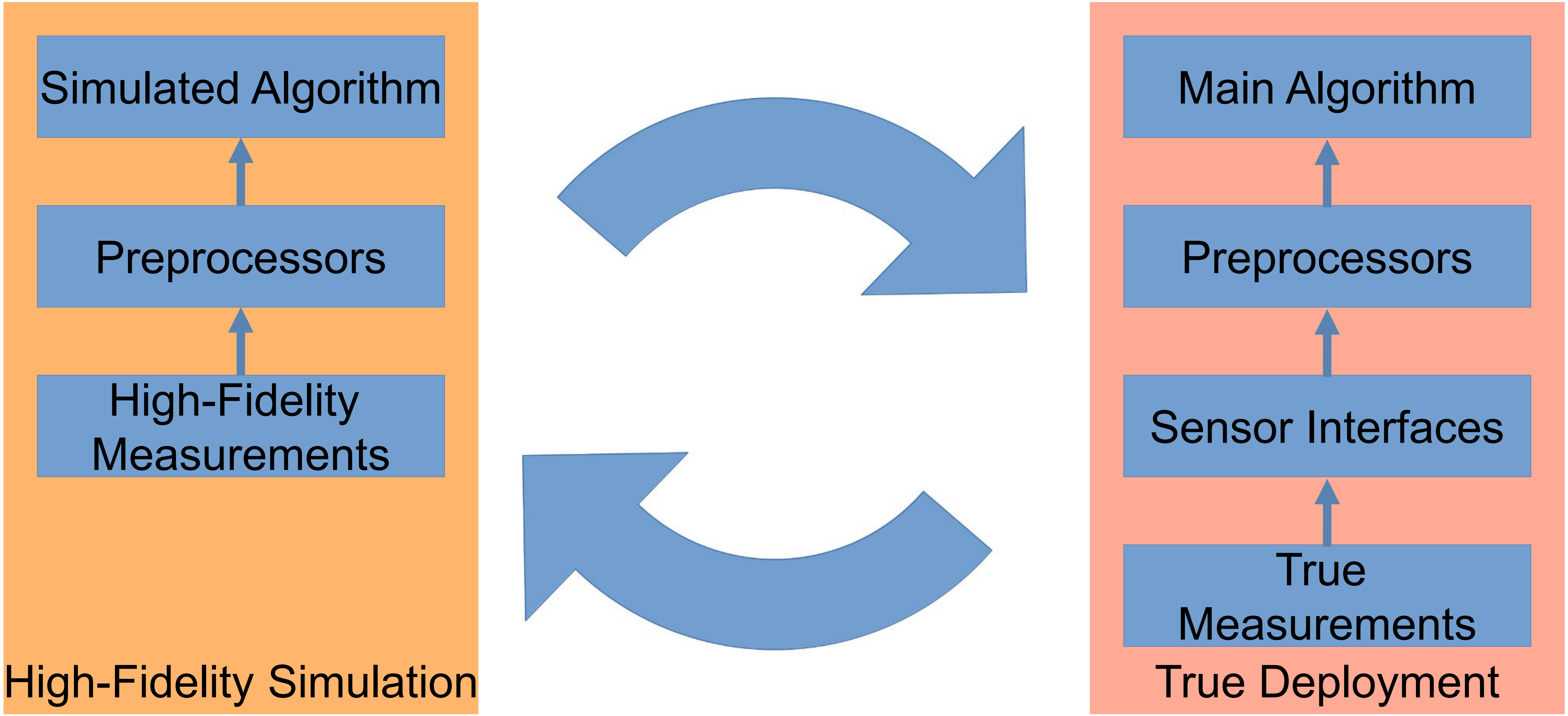
Typical Development



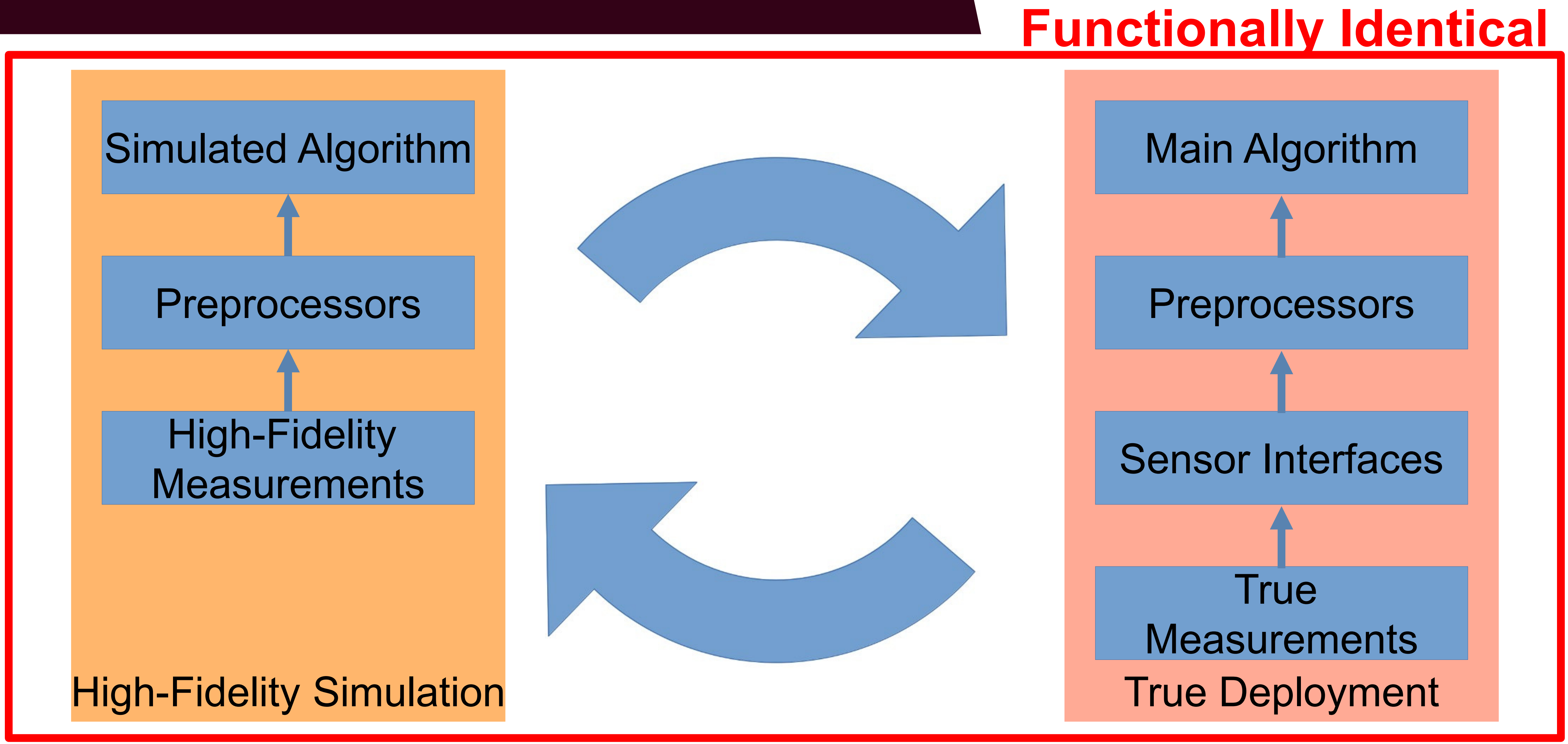
Typical Development



Typical Development



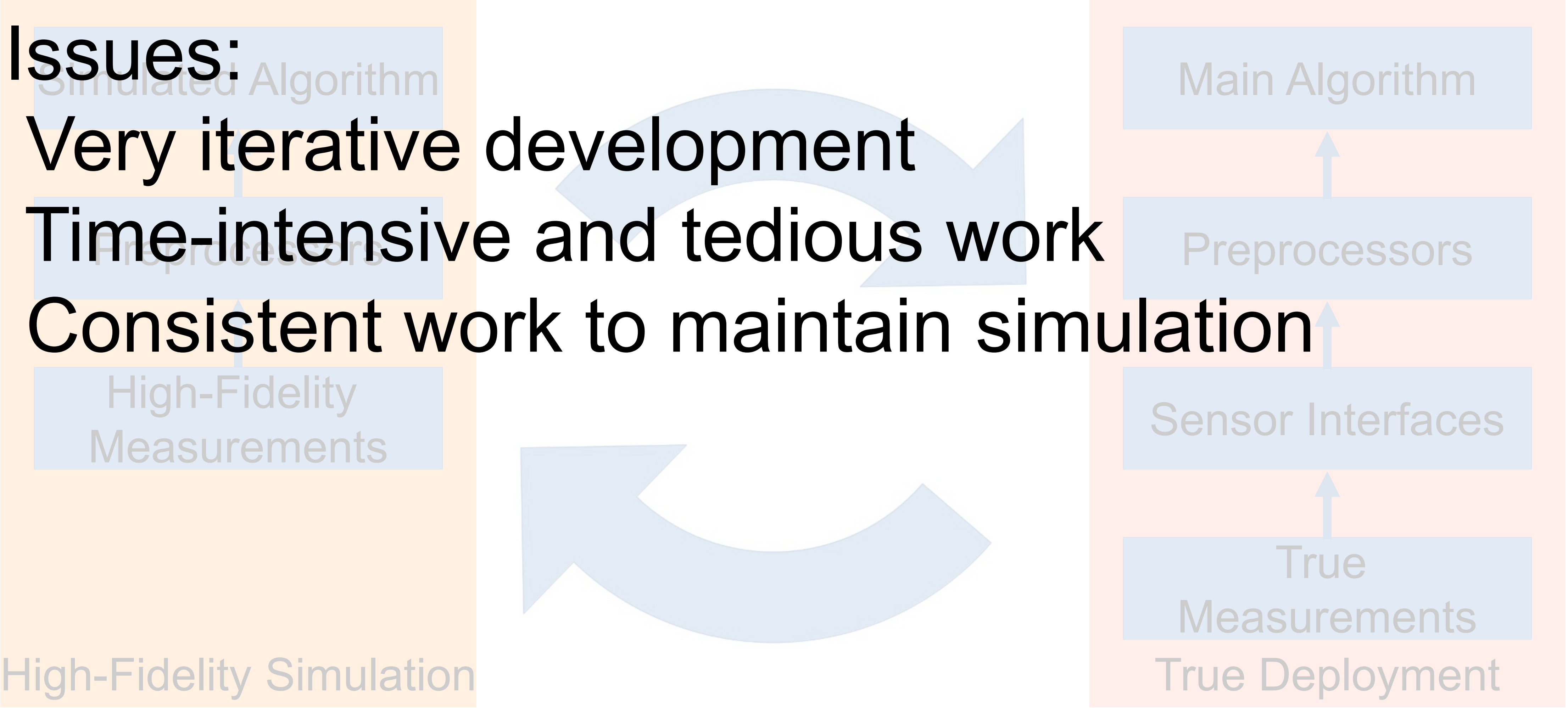
Typical Development



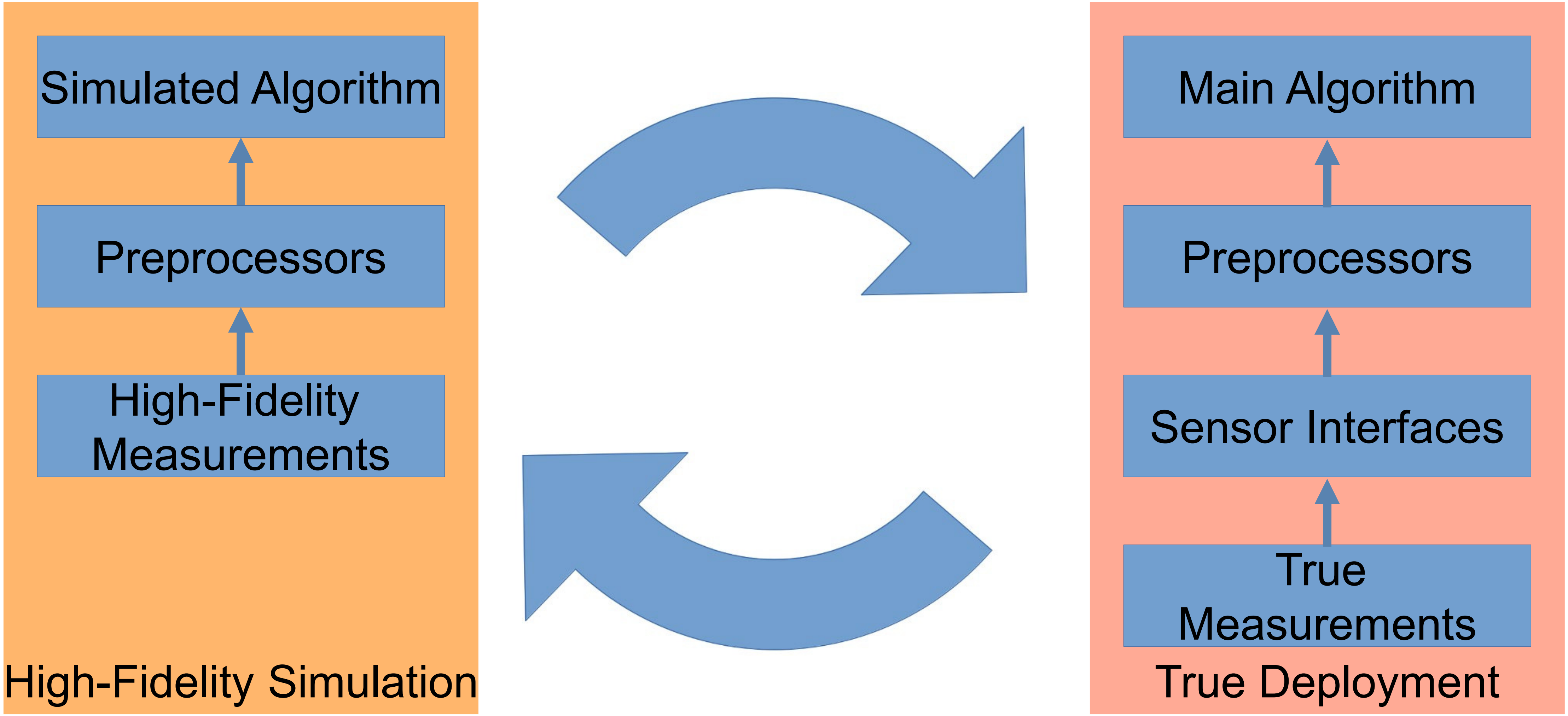
Typical Development

Functionally Identical

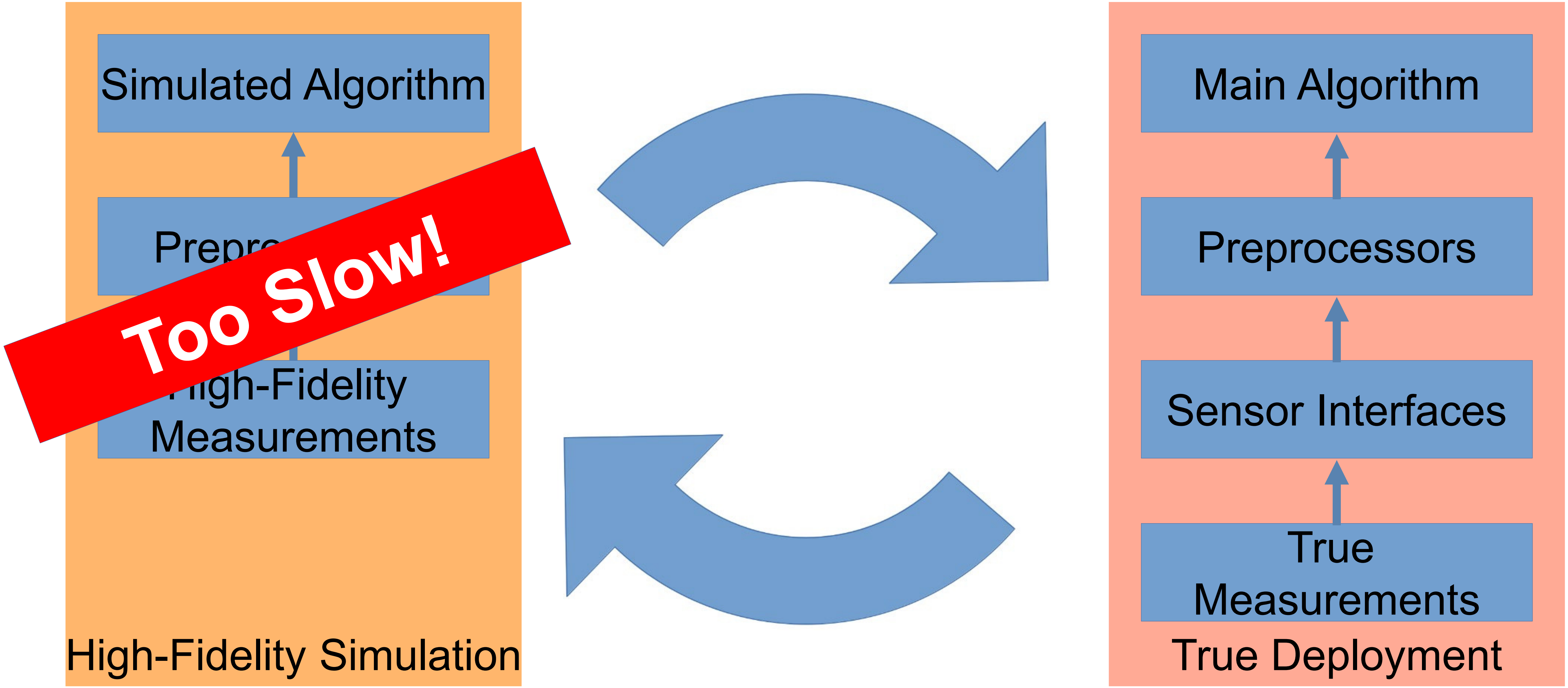
- Issues:
- Very iterative development
- Time-intensive and tedious work
- Consistent work to maintain simulation



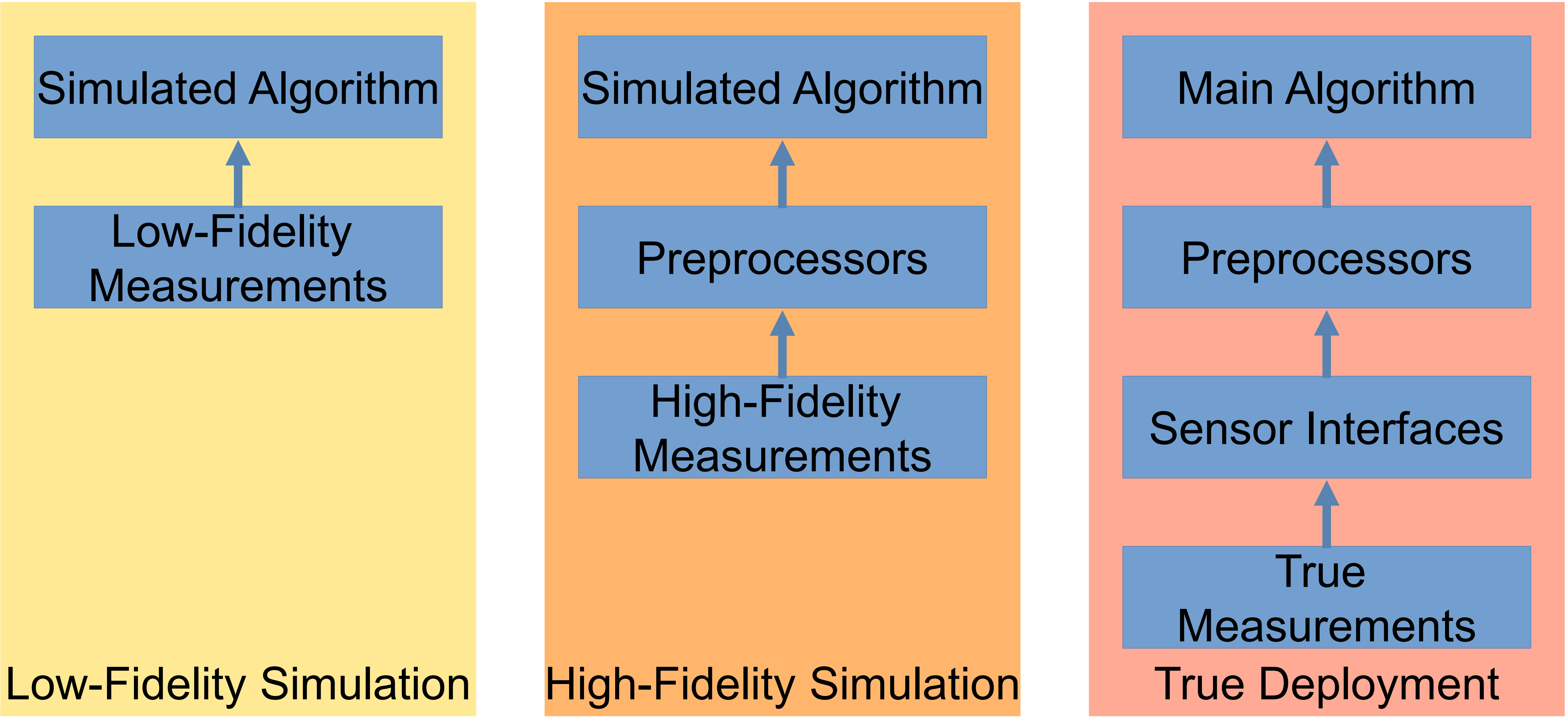
Typical Development



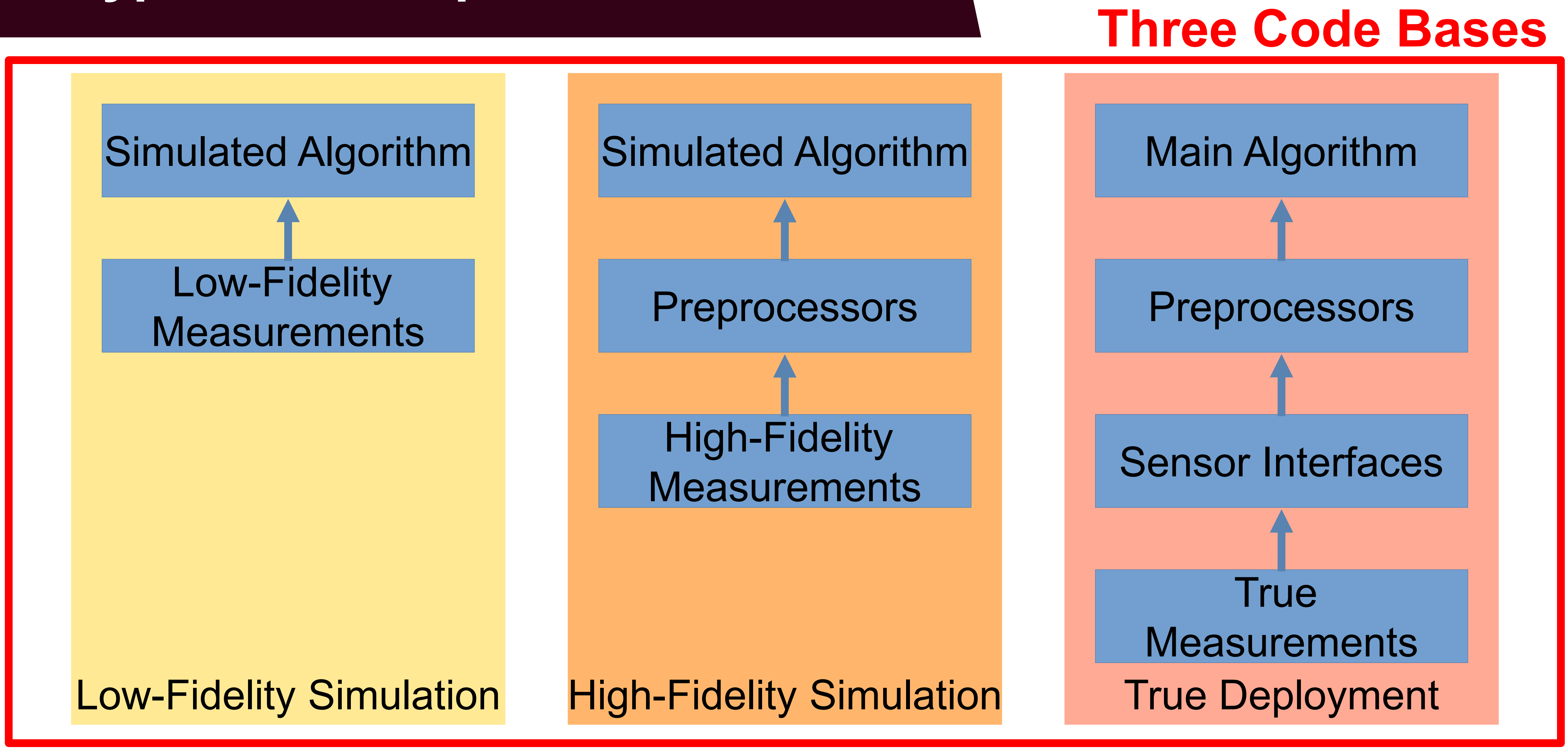
Typical Development



Typical Development



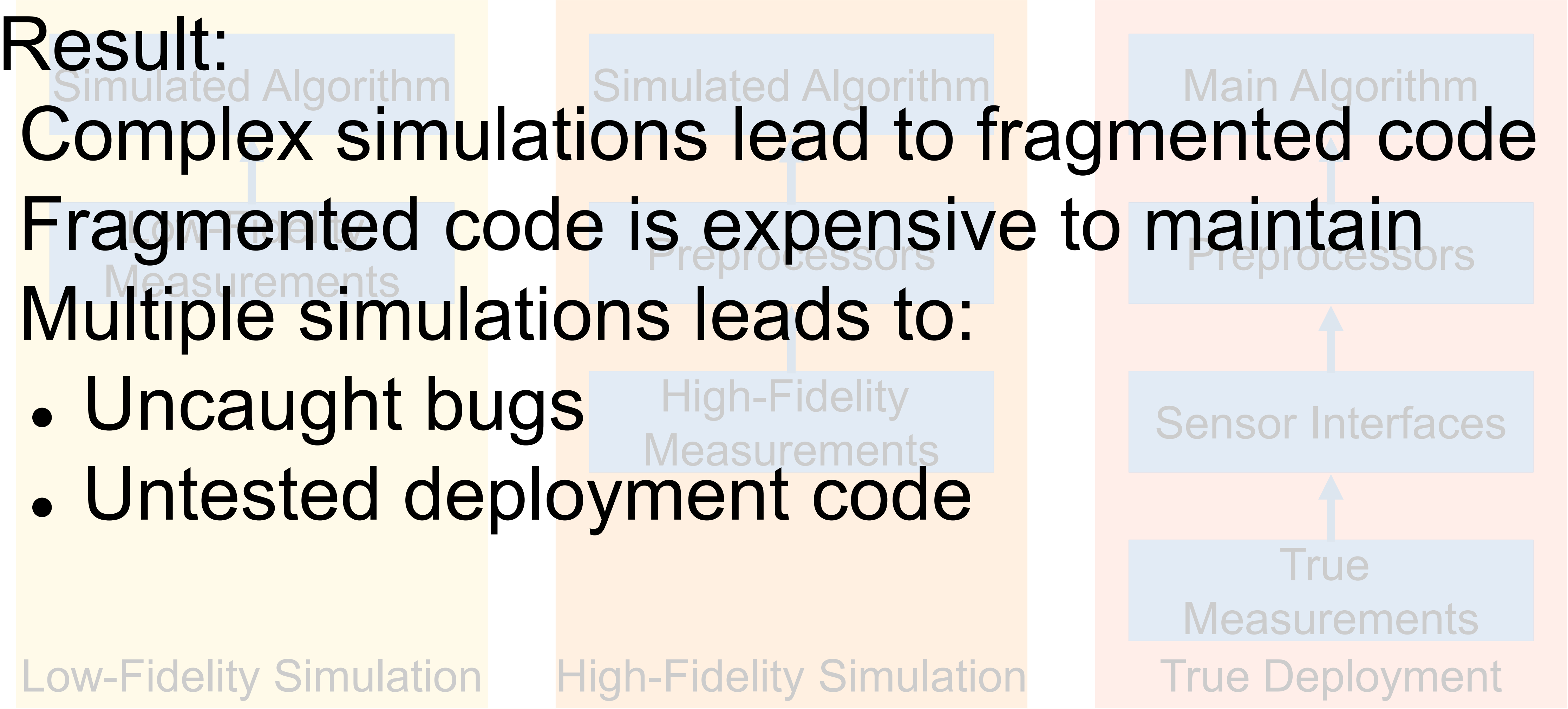
Typical Development



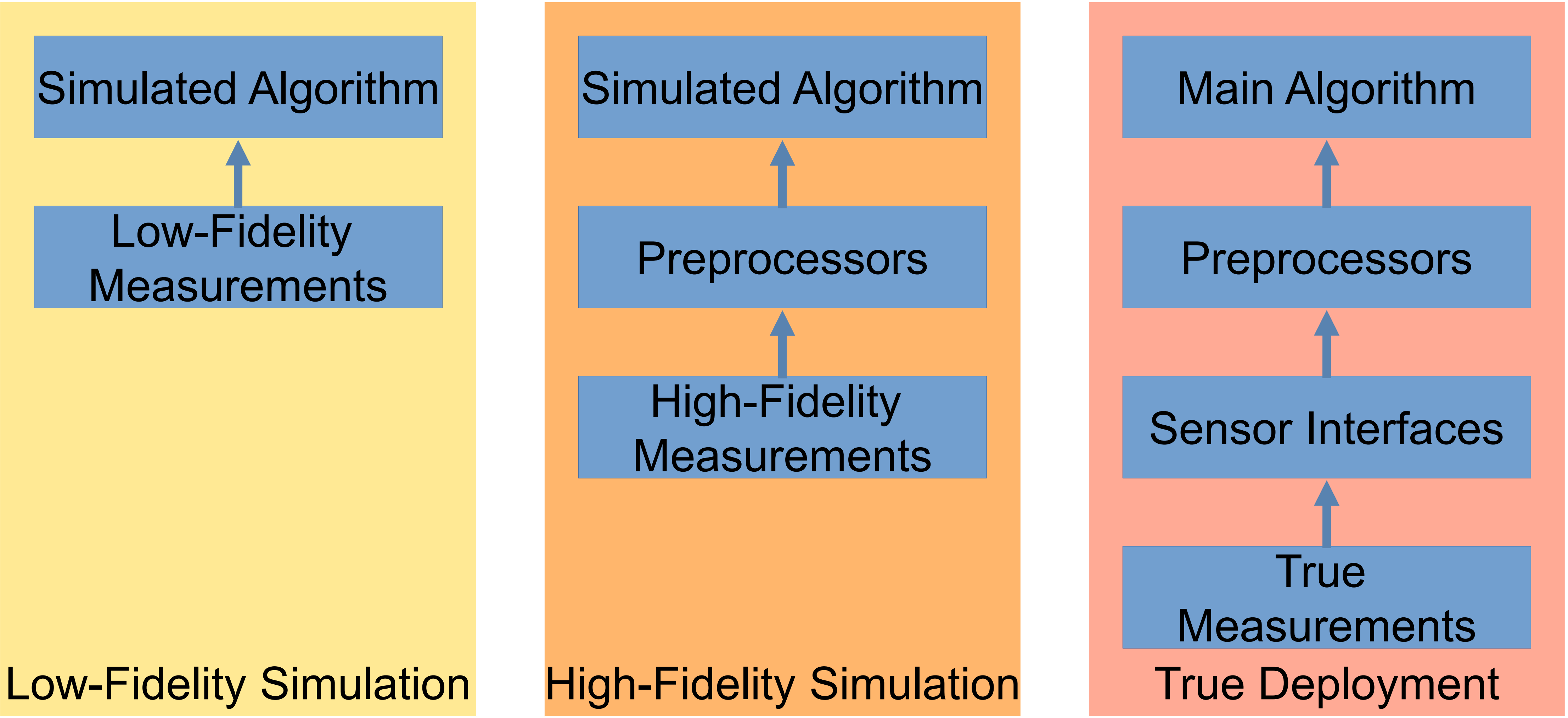
Typical Development

Three Code Bases

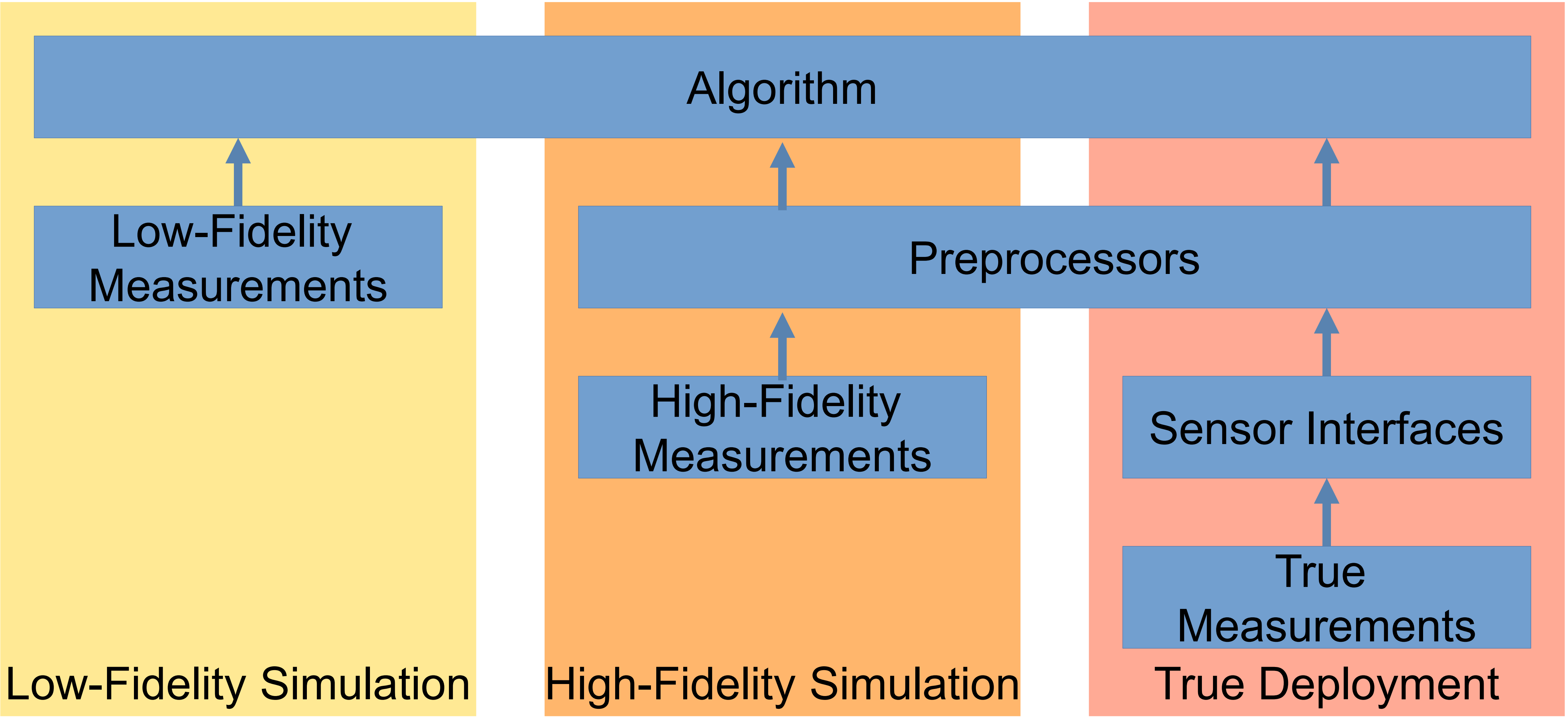
- Result:
- Complex simulations lead to fragmented code
- Fragmented code is expensive to maintain
- Multiple simulations leads to:
  - Uncaught bugs
  - Untested deployment code



Typical Development

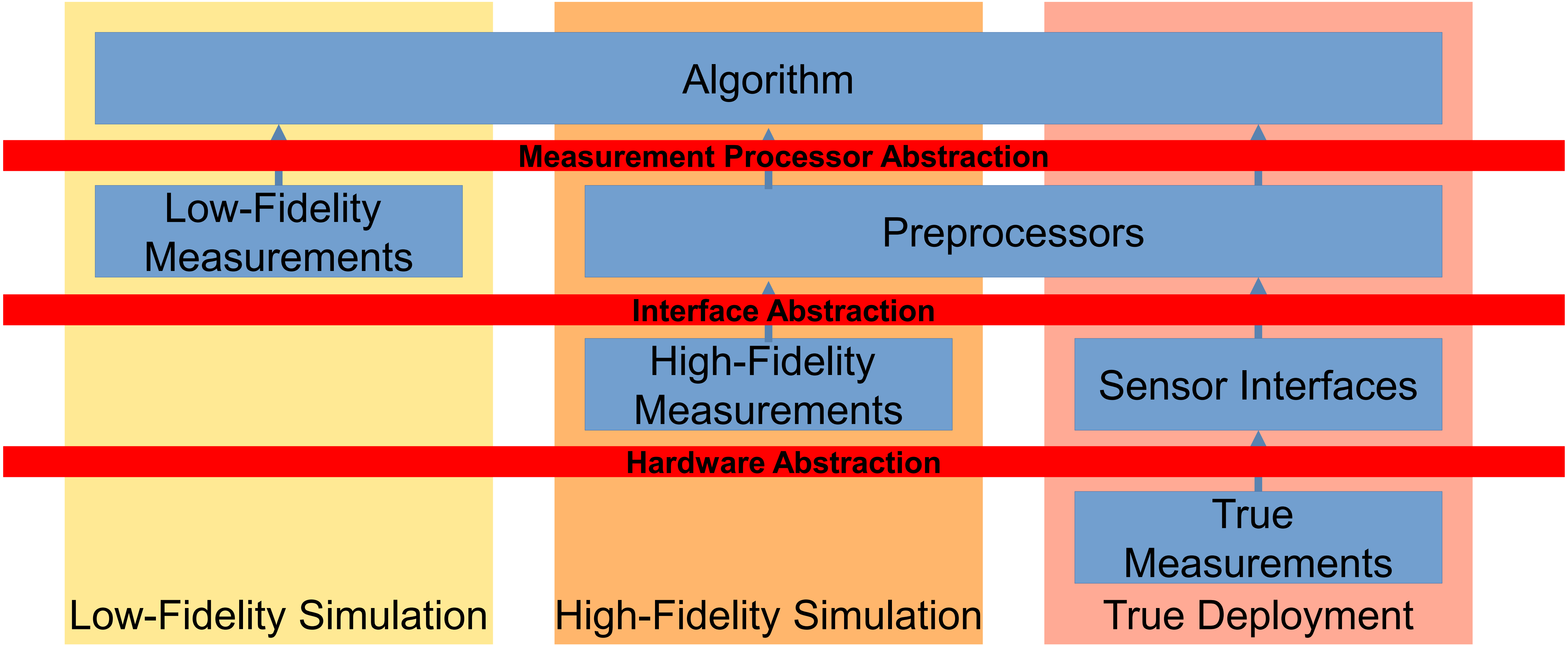


Proposed Solution

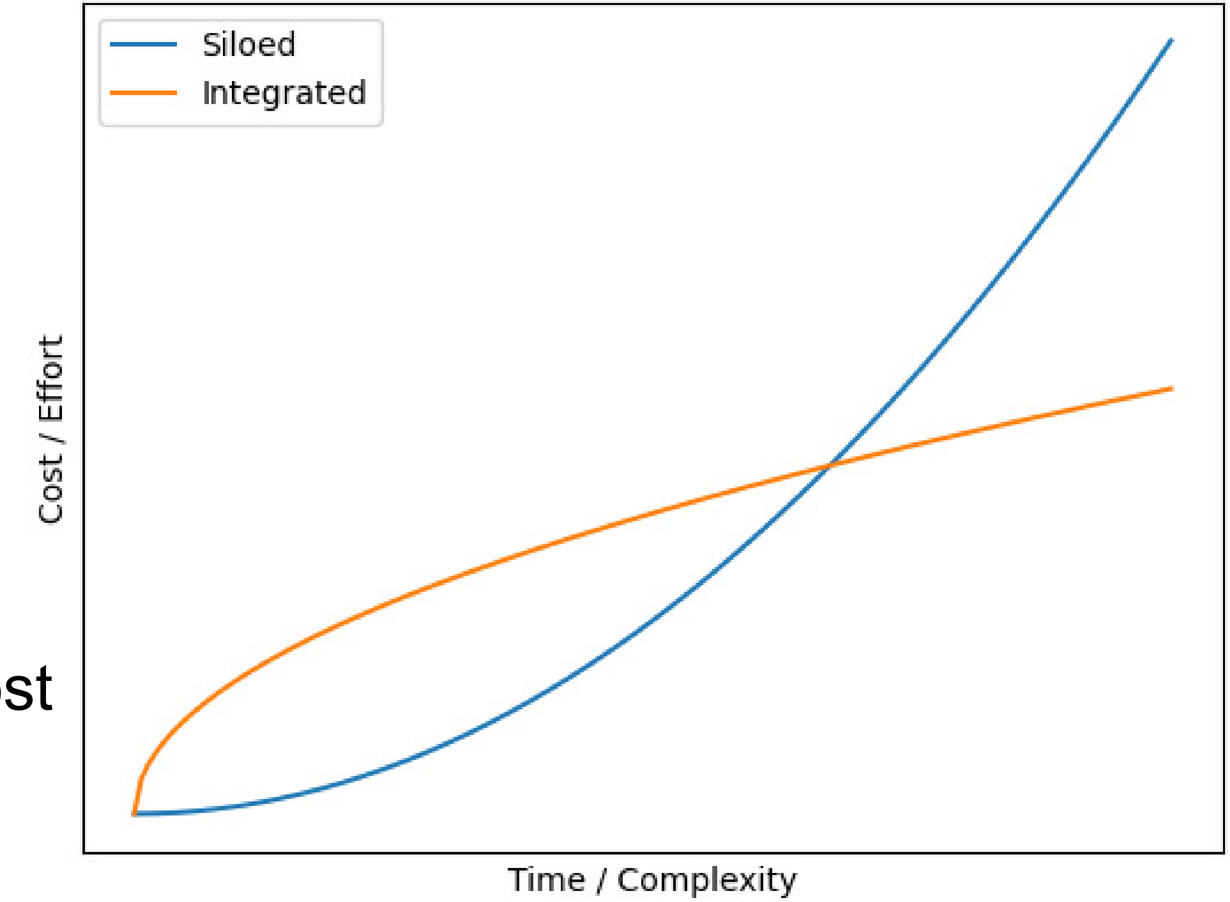


Proposed Solution

Abstraction Layers!



Cost of Integrated Coding

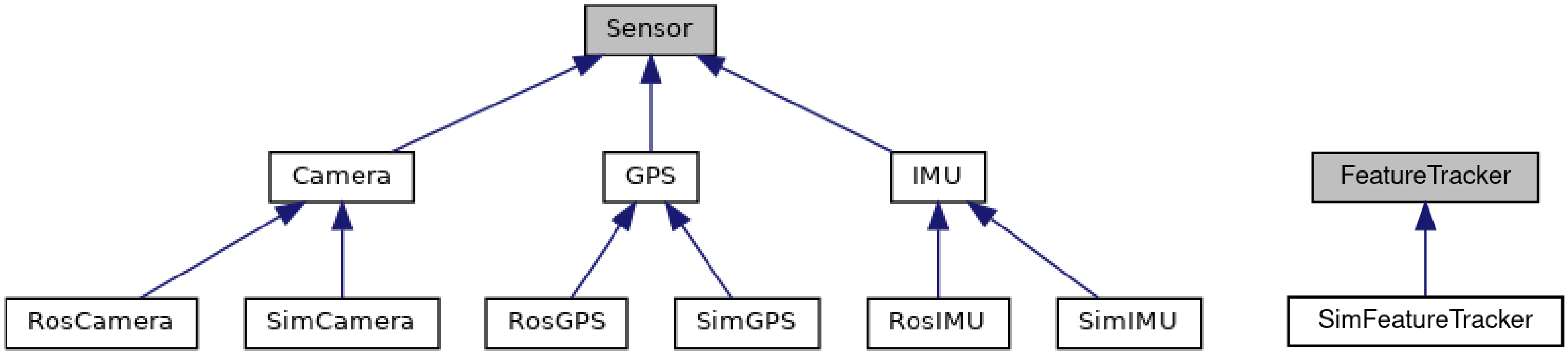


Higher initial cost

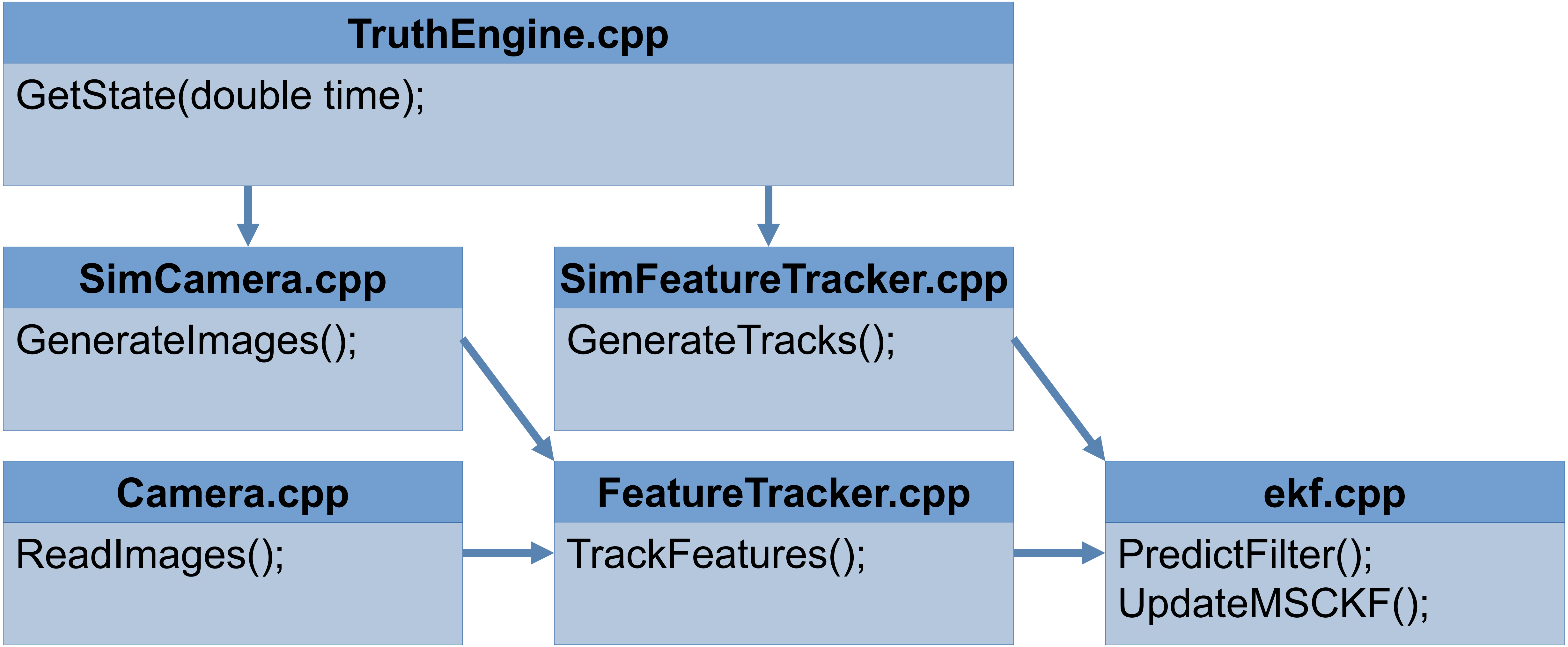
Pays off in long-run

Abstraction - Example

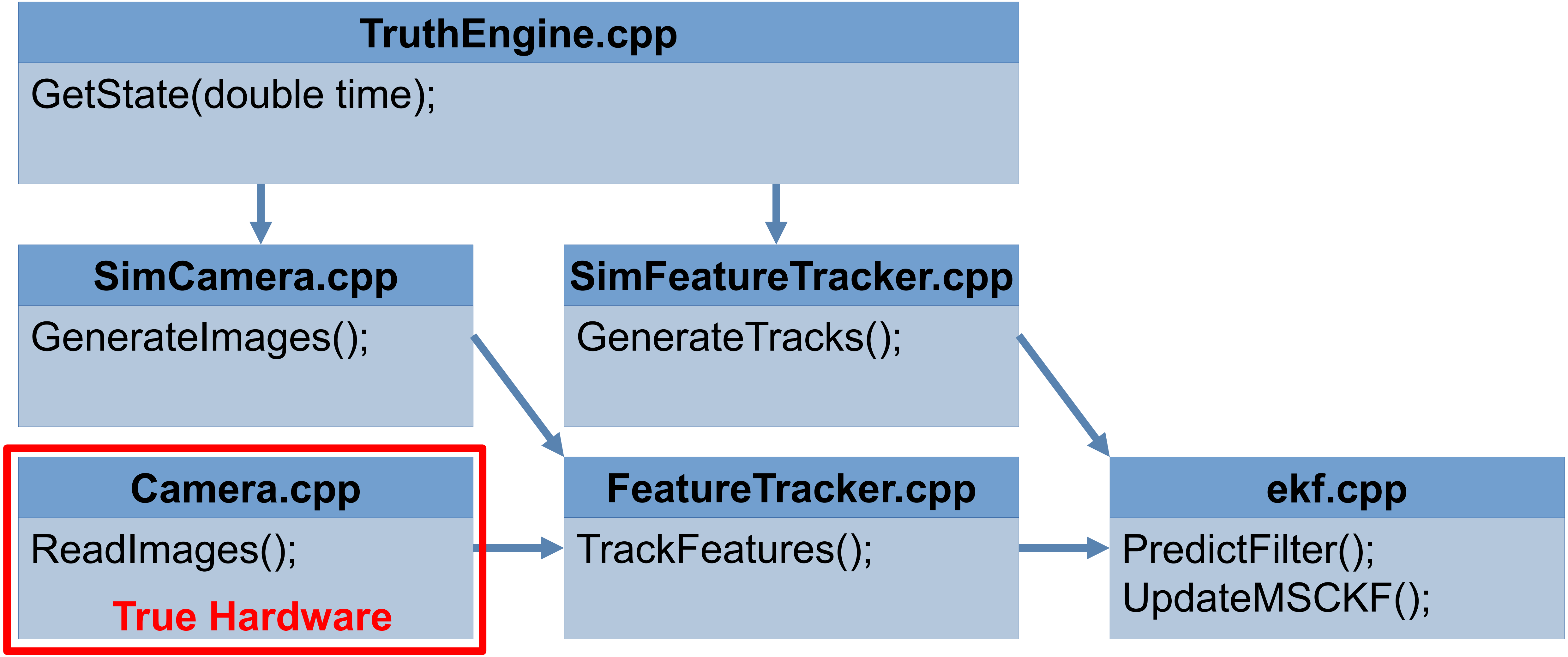
- Sensors call updates to filter / algorithm
  - Utilize real or simulated sensor messages
- Feature tracker utilizes camera measurements
  - True deployment utilizes true camera measurements
  - High-Fidelity simulation provides ray-traced images
  - Low-Fidelity simulation provides “pre-tracked features”



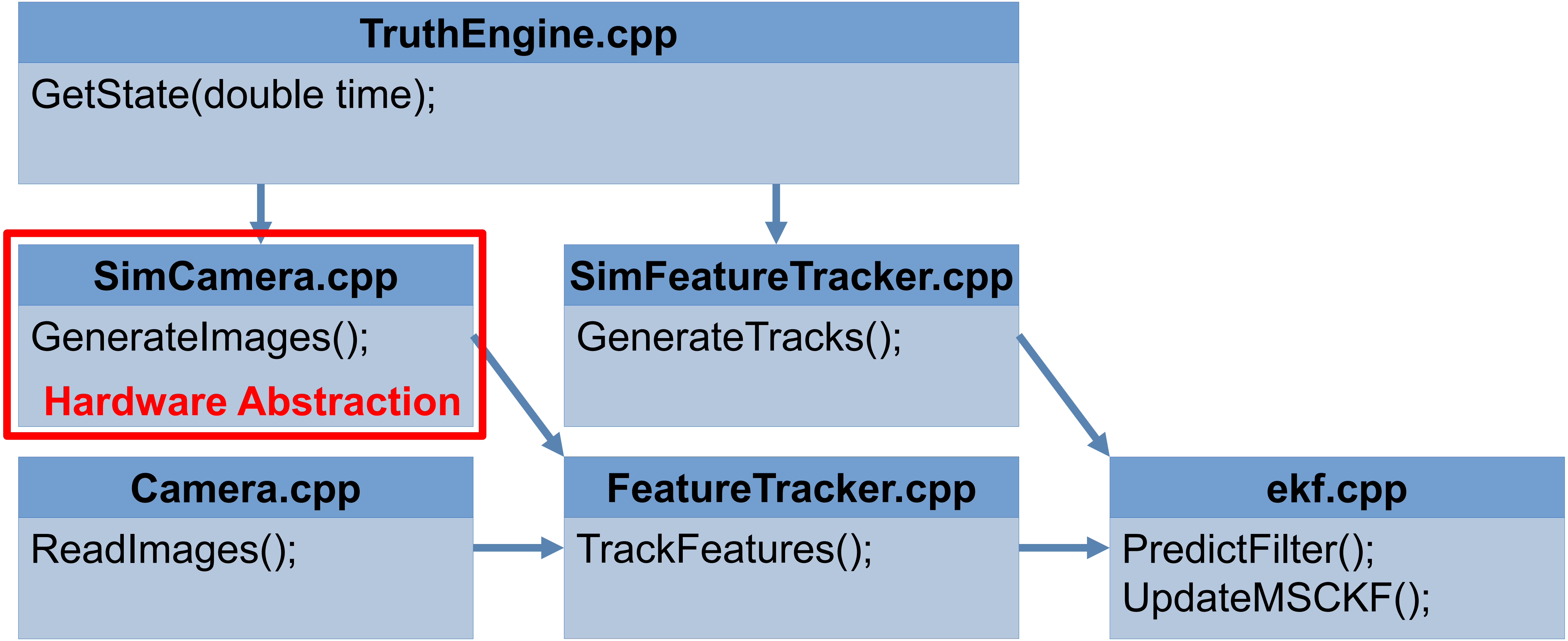
Abstraction - Example



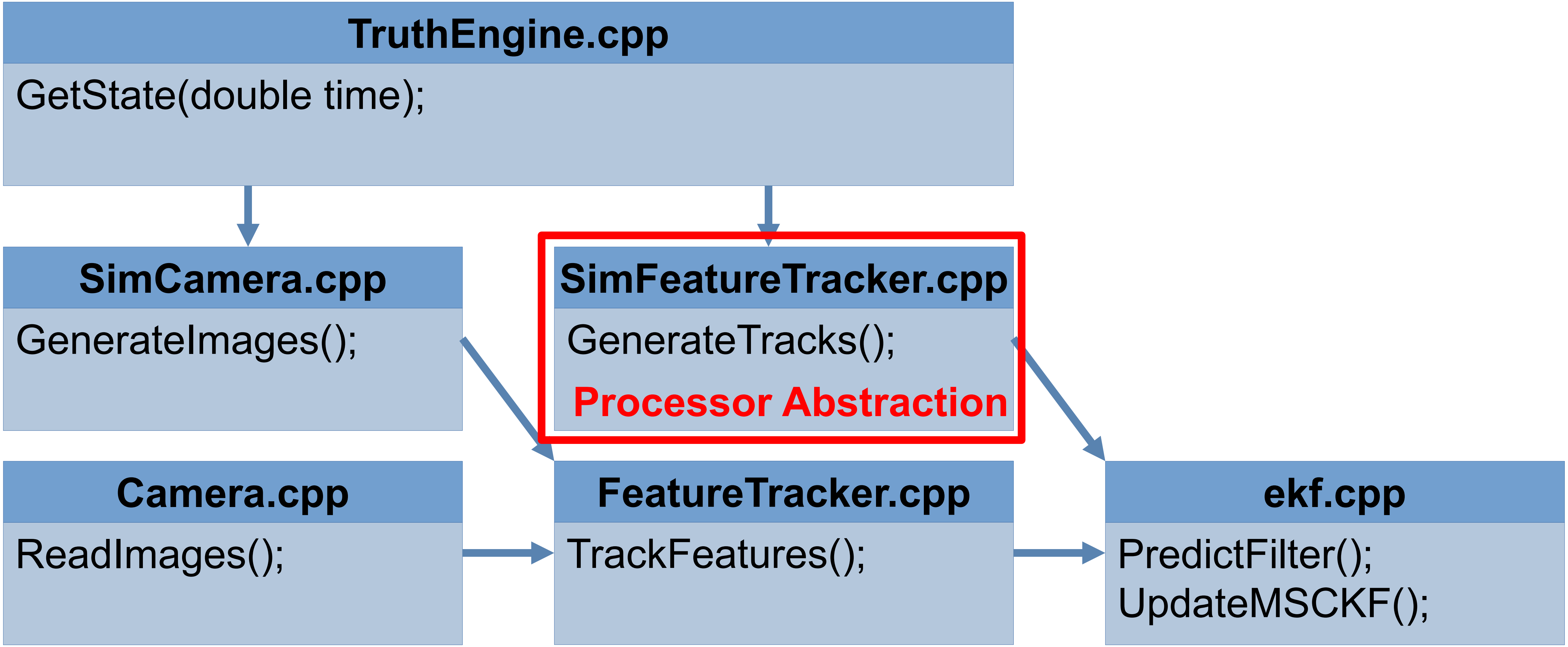
Abstraction - Example



Abstraction - Example

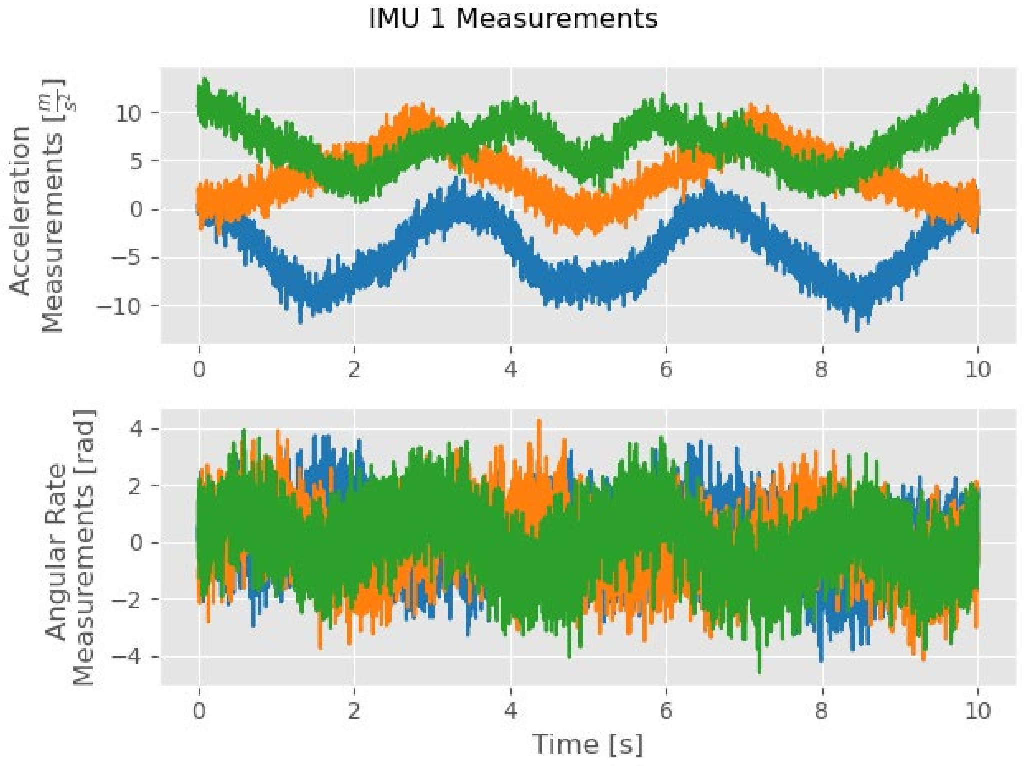


Abstraction - Example



Abstraction Models: IMU

$$\begin{bmatrix}\tilde{a}_x\\ \tilde{a}_y\\ \tilde{a}_z\end{bmatrix} = \begin{bmatrix}S_{ax} & 0 & 0\\ 0 & S_{ay} & 0\\ 0 & 0 & S_{az}\end{bmatrix}\begin{bmatrix}1 & \alpha_{a1} & \alpha_{a2}\\ \alpha_{a3} & 1 & \alpha_{a4}\\ \alpha_{a5} & \alpha_{a6} & 1\end{bmatrix}\left(\begin{bmatrix}a_x\\ a_y\\ a_z\end{bmatrix} + \begin{bmatrix}b_{ax}\\ b_{ay}\\ b_{az}\end{bmatrix}\right) + \begin{bmatrix}n_{ax}\\ n_{ay}\\ n_{az}\end{bmatrix}$$



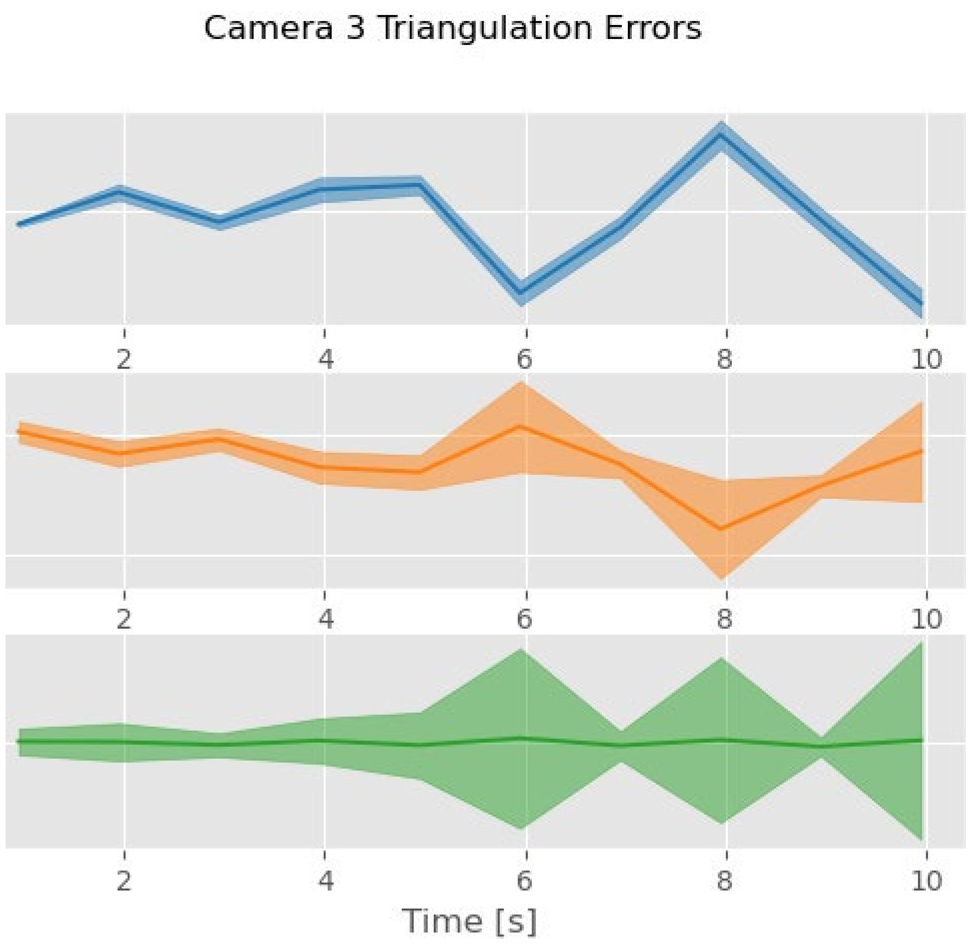
Abstraction Models: Camera

$$x_n = \frac{{}^C\boldsymbol{p}_x f_x + c_x}{w\,{}^C\boldsymbol{p}_z} - 1$$

$$y_n = \frac{{}^C\boldsymbol{p}_y f_y + c_y}{h\,{}^C\boldsymbol{p}_z} - 1$$

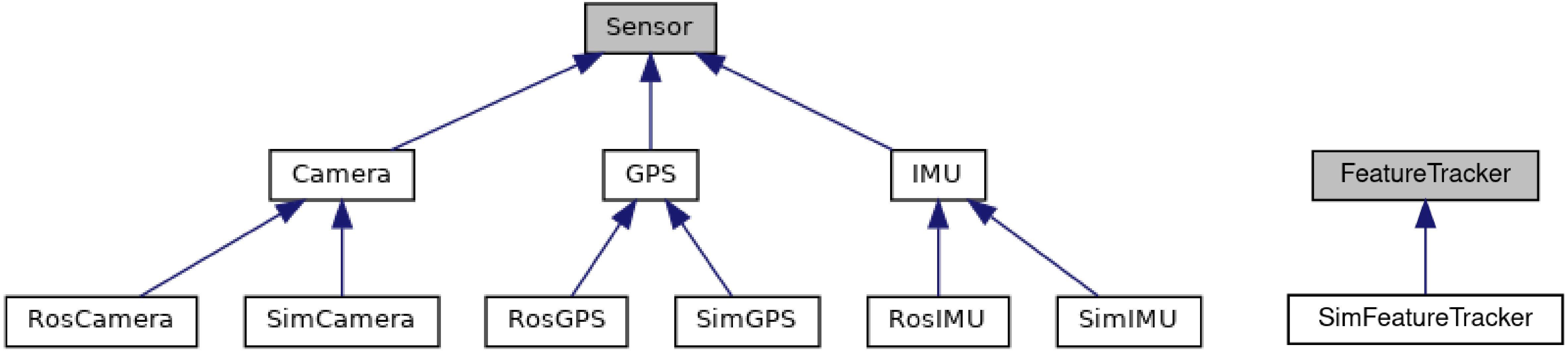
$$r^2 = x_n^2 + y_n^2$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} \cdot \underbrace{\begin{bmatrix} 1 + k_1 r^2 + k_2 r^4 + k_3 r^6 \\ 1 + k_1 r^2 + k_2 r^4 + k_3 r^6 \end{bmatrix}}_{\text{Radial}} + \underbrace{\begin{bmatrix} 2p_1 x_n y_n + p_2 \cdot (r^2 + 2x_n^2) \\ p_1 \cdot (r^2 + 2y_n^2) + 2p_2 x_n y_n \end{bmatrix}}_{\text{Tangential}}$$



Abstraction - Benefits

- Improves accuracy of simulation
- Catches more bugs earlier
- Reduces rework (no code divergence)
- More beneficial unit testing
- Robust Monte Carlo testing



Unit Testing

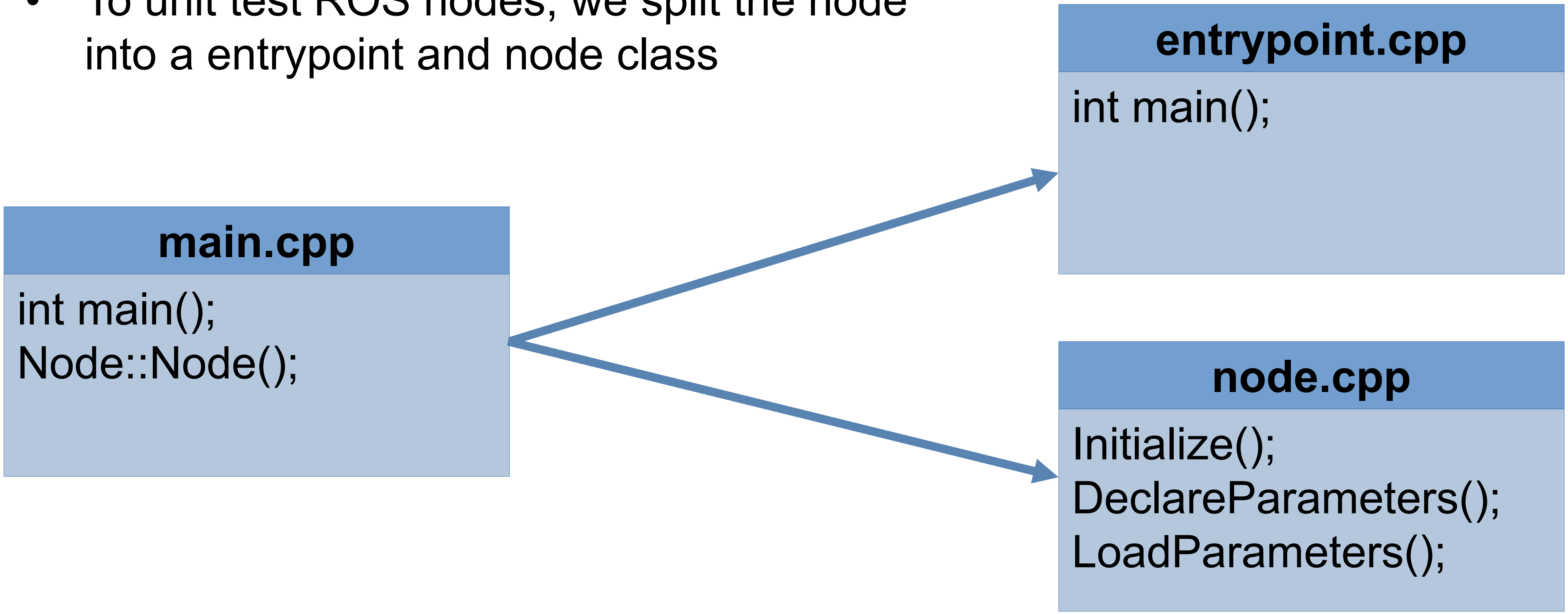
It's Easy!

- This architecture allows test-driven development
- Any tweaks or examples can become tests
- Tests ensure code accuracy and functionality
- Can be automated per commit / merge request

GoogleTest + CMake + Colcon + lcov

Unit Testing Nodes

- Ideally, we should test as much as possible
- To unit test ROS nodes, we split the node into a entrypoint and node class



Unit Testing Nodes

entrypoint.cpp

```
int main();
```

node.cpp

```
Initialize();
DeclareParameters();
LoadParameters();
```

test.cpp

```
TEST_F(ExampleNode, ExampleNode_test)
{
  ExampleNode node;

  node.Initialize();
  node.DeclareParameters();

  node.set_parameter(rclcpp::Parameter("param1"));
  node.set_parameter(rclcpp::Parameter("param2"));

  node.LoadParameters();
}
```

Unit Testing Nodes

entrypoint.cpp

```
int main();
```

node.cpp

```
Initialize();
DeclareParameters();
LoadParameters();
```

test.cpp

```
TEST_F(ExampleNode, ExampleNode_test)
{
  ExampleNode node;

  node.Initialize();
  node.DeclareParameters();

  node.set_parameter(rclcpp::Parameter(“param1”));
  node.set_parameter(rclcpp::Parameter(“param2”));

  node.LoadParameters();
}
```

Unit Testing Nodes

entrypoint.cpp

```
int main();
```

node.cpp

```
Initialize();
DeclareParameters();
LoadParameters();
```

test.cpp

```
TEST_F(ExampleNode, ExampleNode_test)
{
  ExampleNode node;

  node.Initialize();
  node.DeclareParameters();

  node.set_parameter(rclcpp::Parameter(“param1”));
  node.set_parameter(rclcpp::Parameter(“param2”));

  node.LoadParameters();
}
```

Unit Testing Nodes

entrypoint.cpp

```
int main();
```

node.cpp

```
Initialize();
DeclareParameters();
LoadParameters();
```

test.cpp

```
TEST_F(ExampleNode, ExampleNode_test)
{
  ExampleNode node;

  node.Initialize();
  node.DeclareParameters();

  node.set_parameter(rclcpp::Parameter(“param1”));
  node.set_parameter(rclcpp::Parameter(“param2”));

  node.LoadParameters();
}
```

Unit Testing Nodes

entrypoint.cpp

```
int main();
```

node.cpp

```
Initialize();
DeclareParameters();
LoadParameters();
```

test.cpp

```
TEST_F(ExampleNode, ExampleNode_test)
{
  ExampleNode node;

  node.Initialize();
  node.DeclareParameters();

  node.set_parameter(rclcpp::Parameter(“param1”));
  node.set_parameter(rclcpp::Parameter(“param2”));

  node.LoadParameters();
}
```

Unit Testing Nodes

entrypoint.cpp

```
int main();
```

node.cpp

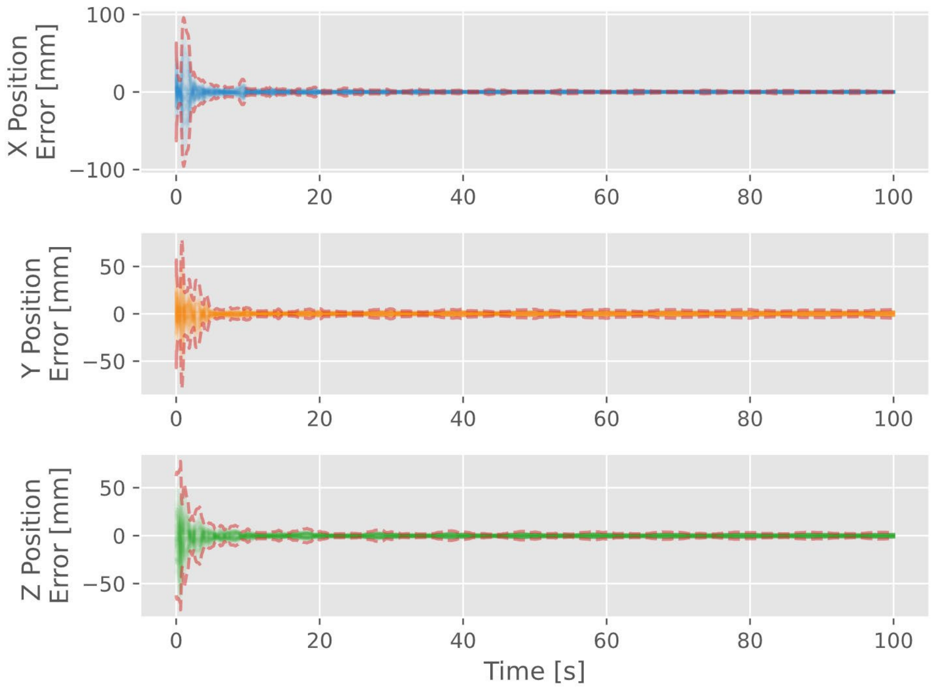
```
Initialize();
DeclareParameters();
LoadParameters();
```

test.cpp

```
TEST_F(ExampleNode, ExampleNode_test)
{
  ExampleNode node;

  node.Initialize();
  node.DeclareParameters();

  node.set_parameter(rclcpp::Parameter("param1"));
  node.set_parameter(rclcpp::Parameter("param2"));

  node.LoadParameters();
}
```

Monte Carlo Testing

- With fast enough simulations, we can run thousands of example datasets
- Random initialization and measurement errors are inserted
- Utilizing abstractions increases confidence of filter stability



Key Takeaways

- Stop developing simulations separate from deployments
- Utilize existing models for integrated simulations
- Abstract layers as low as possible
- Utilize multiple layers of abstraction for various fidelity / execution speed
- Try out EKF-CAL! We love feedback and collaboration!

Presentation References

1. J. Hartzer and S. Saripalli, " Online Multi Camera-IMU Calibration", IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), 2022. **IEEE**, **arXiv**

2. P. Jiang and S. Saripalli, "LiDARNet: A Boundary-Aware Domain Adaptation Model for Point Cloud Semantic Segmentation," 2021 IEEE International Conference on Robotics and Automation (ICRA), Xi'an, China, 2021, pp. 2457-2464, **IEEE**

3. Experimental Evaluation of 3D-LIDAR Camera Extrinsic Calibration, S. Mishra, P. Osteen, G. Pandey and S. Saripalli, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS, 2020), **arXiv**

4. Extrinsic Calibration of a 3D-LIDAR and a Camera, S. Mishra, G. Pandey and S. Saripalli, IEEE Intelligent Vehicles Symposium (IV, 2020) **arXiv**

5. Chustz, G., & Saripalli, S. (2021). ROOAD: RELLIS Off-road Odometry Analysis Dataset. **arXIv**

Questions?

EKF-CAL Repository

Unmanned Systems Lab

TEXAS A&M UNIVERSITY

J. Mike Walker ’66 Department of Mechanical Engineering