DFC++ FRAMEWORK

A novel approach for flexible signal processing on embedded systems

Fraunhofer
IIS

Dominik Soller, dominik.soller@iis.fraunhofer.de

AGENDA

1. Introduction
2. Framework Structure
3. Data Transport Mechanisms
4. Platform Support
5. Conclusion

AGENDA

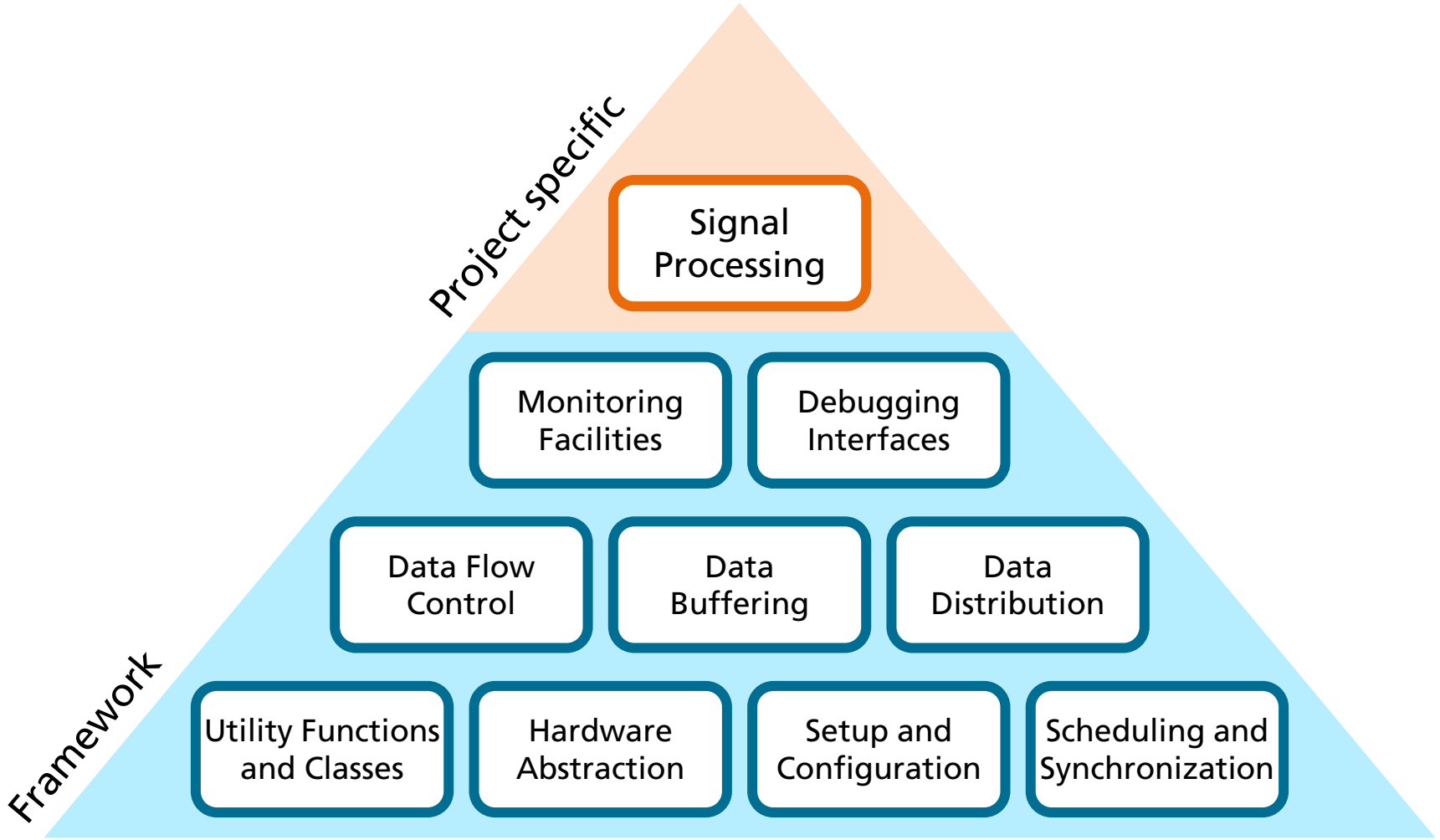
1. Introduction
2. Framework Structure
3. Data Transport Mechanisms
4. Platform Support
5. Conclusion

SDR Frameworks

- Implementation of basic facilities and mechanisms
- Definition of common interfaces and structures

✓ More efficient SDR development

Basic Facilities and Mechanisms

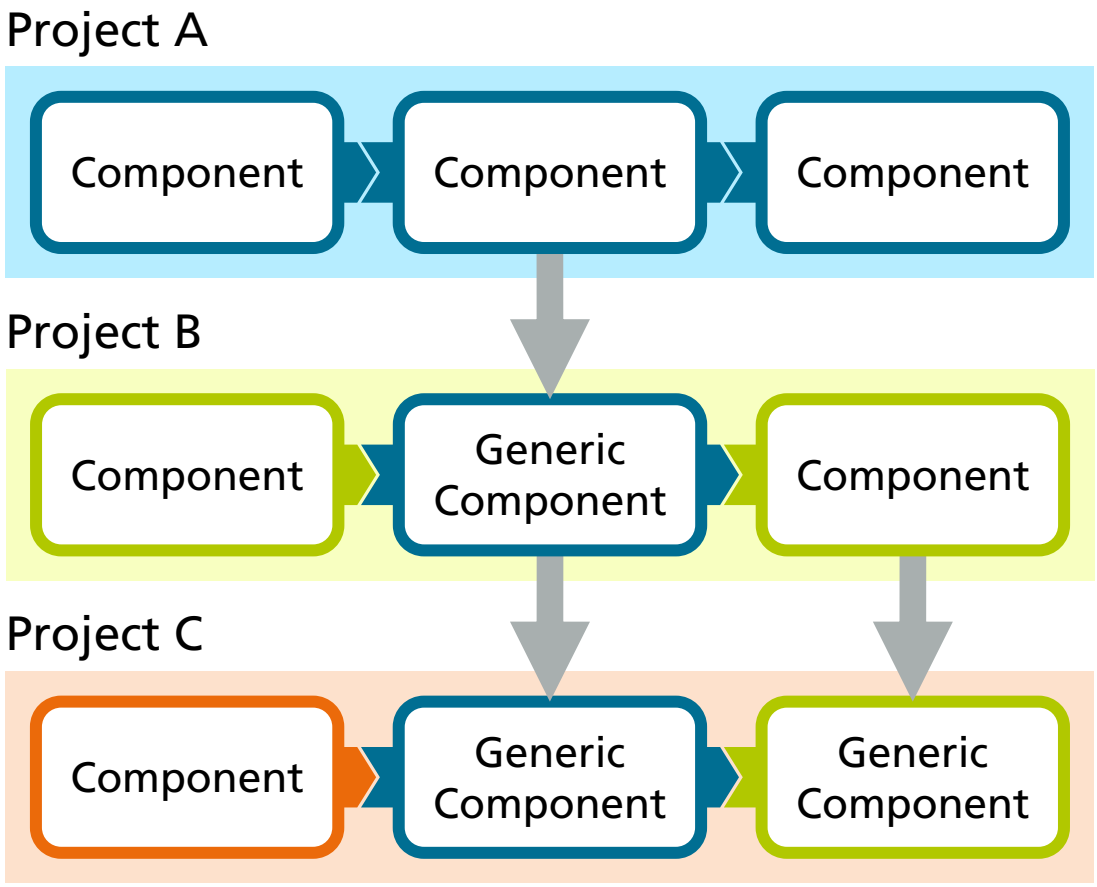


Basic Facilities and Mechanisms

- Allows focusing on development of signal processing algorithms
- Reduces effort allocated in auxiliary and peripheral functionality
- Provides a familiar environment for designers across projects

✓ Acceleration of development processes

Common Interfaces and Structures

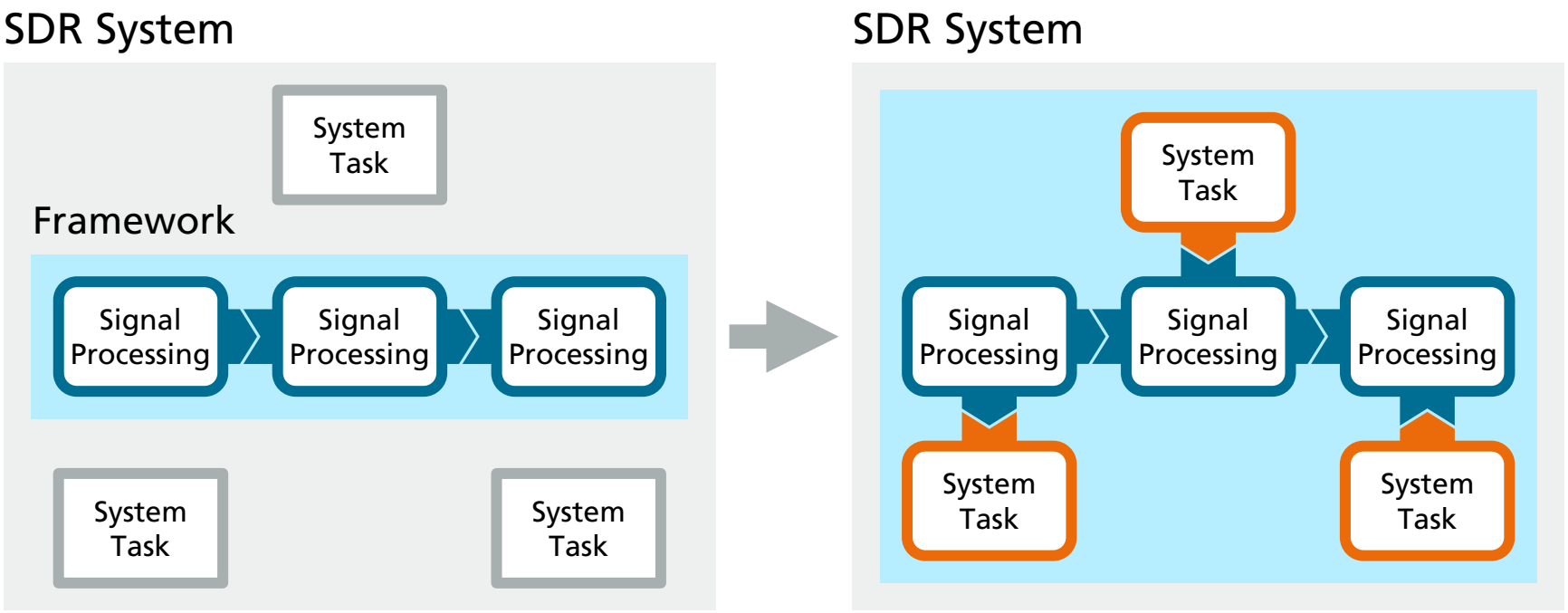


Common Interfaces and Structures

- Improved reusability of components across different projects
- Growing library of generic components aids future projects
- Configuration interfaces simplify the design of generic components

✓ Exploitation of cross-project synergies

Flexibility and Portability



Flexibility and Portability

- Integrate more functionality within the framework
- Simplify maintenance and live debugging
- Cover different platforms with a unified approach

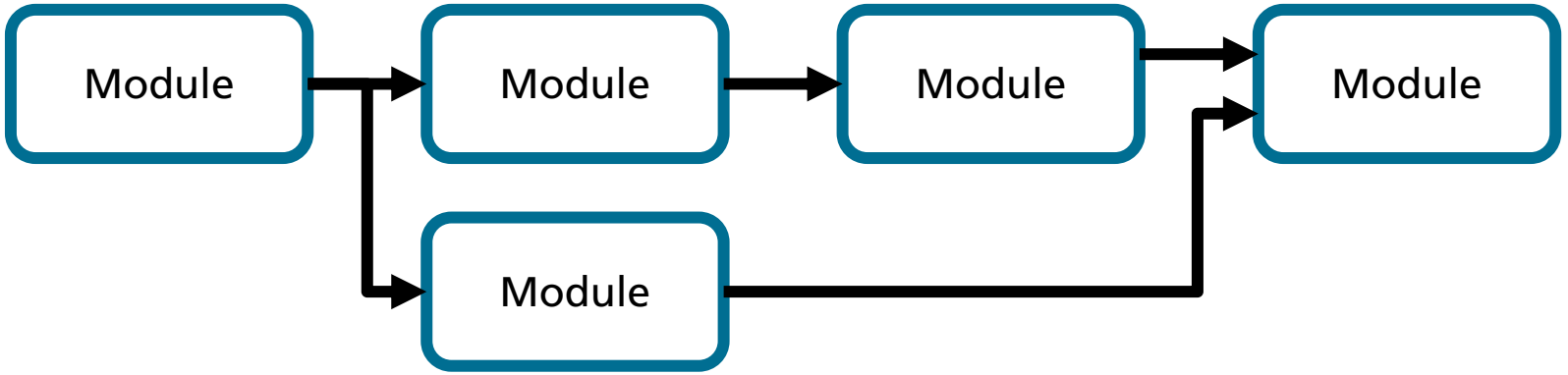
✓ Leveraging of framework advantages

AGENDA

1. Introduction
2. Framework Structure
3. Data Transport Mechanisms
4. Platform Support
5. Conclusion

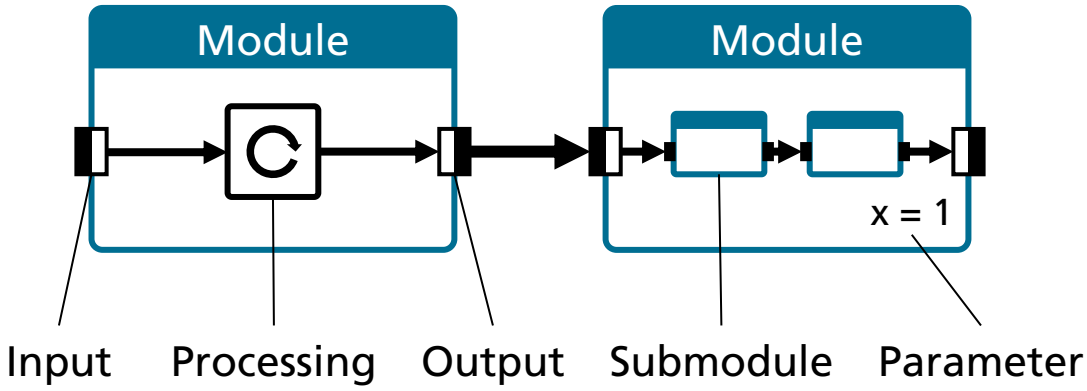
Framework Structure

- Components are organized in autonomous modules
- Modules have standardized interfaces
- Processing chains are formed by connecting multiple modules



Modules

- Every processing component is implemented as a module
- Inputs and outputs provide data interfaces
- Parameters provide configuration interfaces
- Other modules can be added as submodules

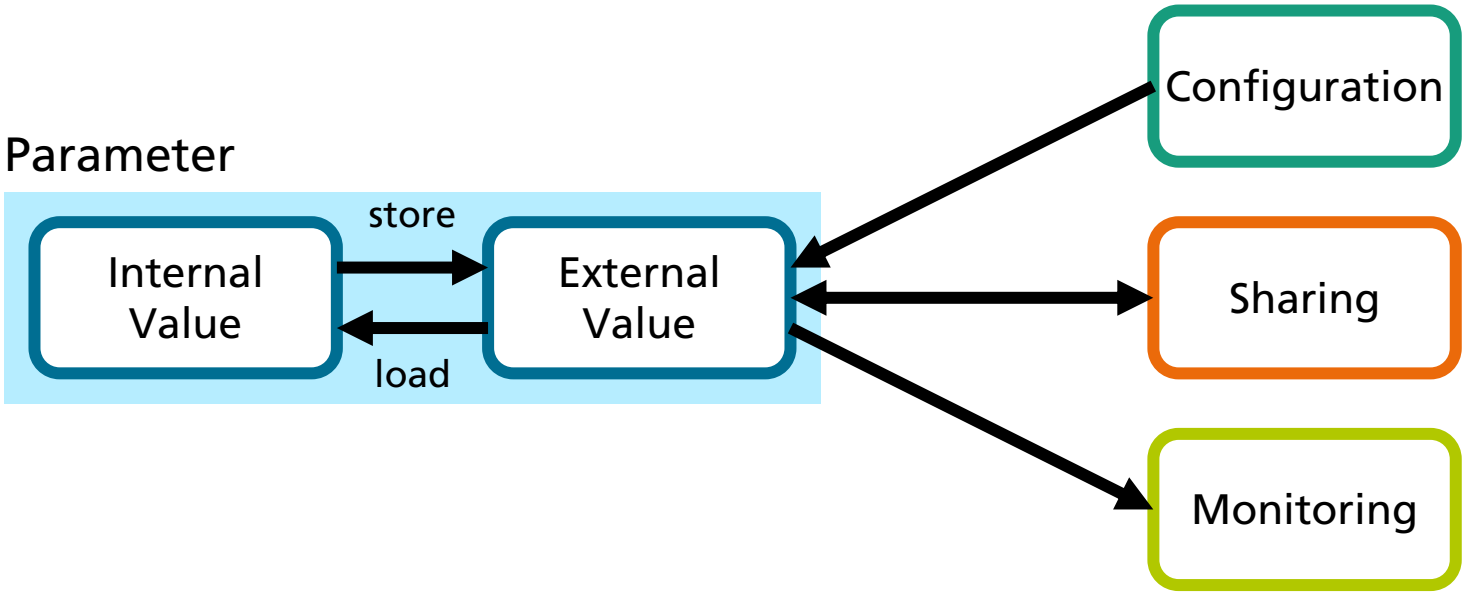


Inputs and Outputs

- Manage data exchange between modules
- Multiple inputs can be connected to one output
- Allow dynamic reconnecting at runtime
- Provide data buffering and flow control
- Reverse channel to signal request status

Parameters

- Standardized interface for configuration and monitoring
- Can be connected to shared parameters
- Separation of internal and external value ensures consistency

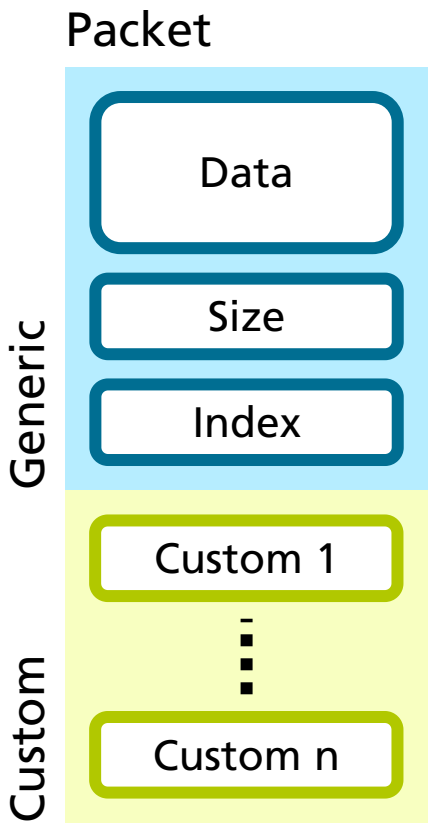


AGENDA

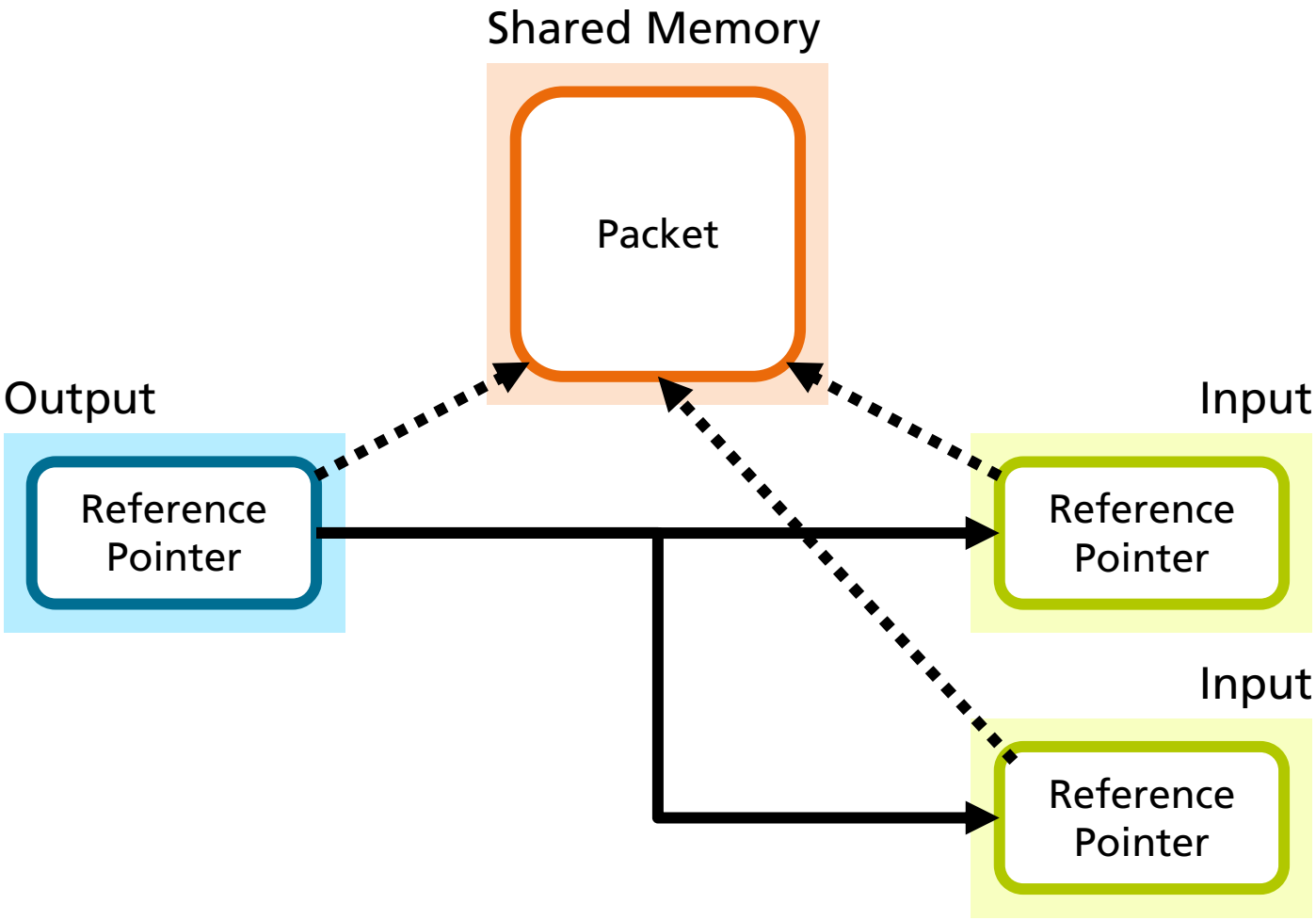
1. Introduction
2. Framework Structure
3. Data Transport Mechanisms
4. Platform Support
5. Conclusion

Data Organization

- Data organized in packets of variable size and type
- Generic data packets for simple raw data transfer
- Custom packet types can add arbitrary meta information
- Packets are handled via reference pointers

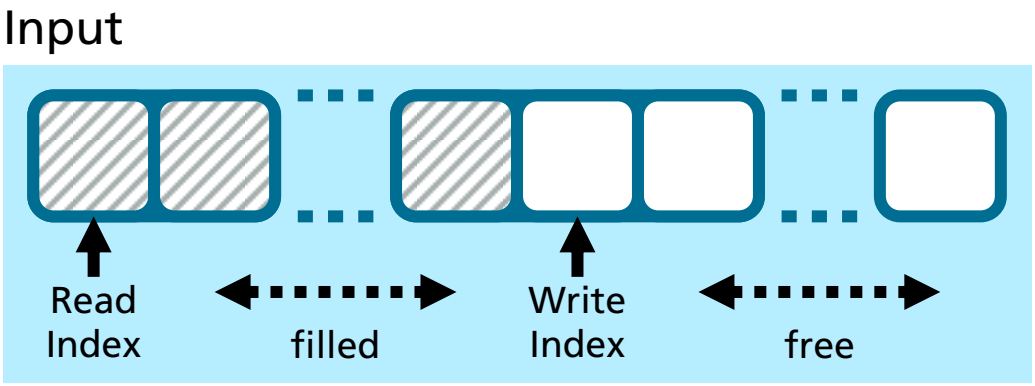


Data Propagation



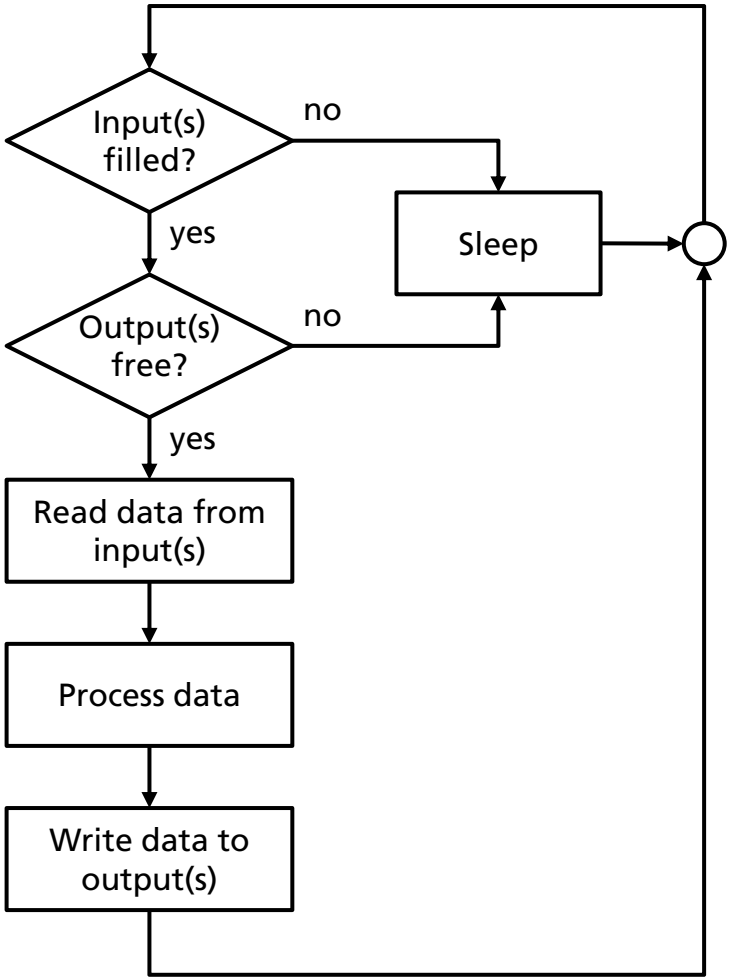
Packet Queue

- Inputs have a Packet queue to store reference pointers
- Ring buffer allows independent write and read access
- Lock free operation via atomic fill level synchronization
- Adjustable queue length to optimize for latency or performance



Flow Control

- Flow control via two sided queue query procedure
- Handles source, sink or throughput limited paths
- Load balancing through dynamic suspension

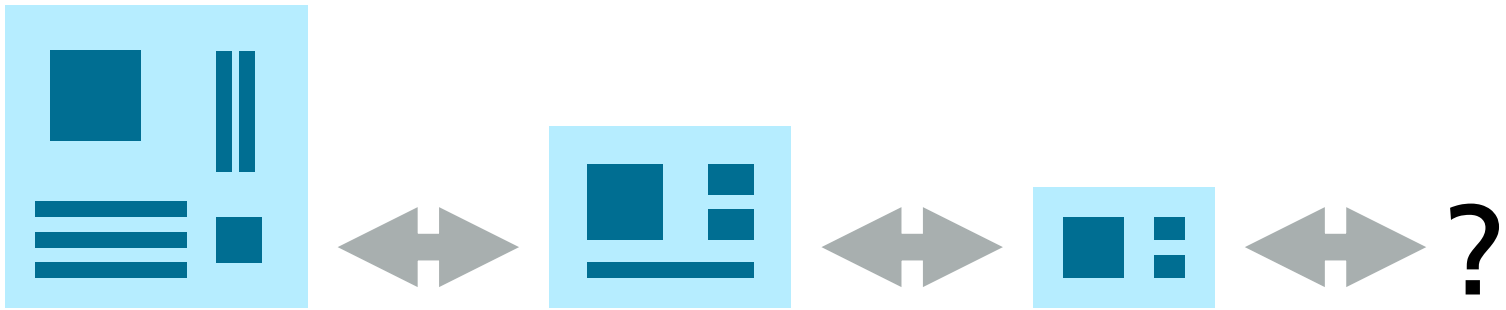


AGENDA

1. Introduction
2. Framework Structure
3. Data Transport Mechanisms
4. Platform Support
5. Conclusion

Portability

- Framework entirely written in C++
- No C++11, Boost or other large libraries required
- Dependencies isolated on internal abstraction Layer
- Low level utilities available to user modules



AGENDA

1. Introduction
2. Framework Structure
3. Data Transport Mechanisms
4. Platform Support
5. Conclusion

Conclusion

- ✓ Flexibility
  - Fully runtime dynamic routing of data paths
  - Adaptive flow control for variable data rates
- ✓ Performance
  - Entire framework implemented in native C++
  - Reference pointer based data distribution
- ✓ Portability
  - No dependencies on large external libraries
  - Isolated platform dependent interface layer

QUESTIONS?

Thank you for listening!