A Data Centric Approach for Modular Assurance

Workshop on Real-time, Embedded and Enterprise-Scale Time-Critical Systems
23 March 2011

The Real-Time Middleware Experts

Gabriela F. Ciocarlie
Heidi Schubert
Rose Wahlin

Agenda

- Introduction
  - Mixed criticality systems
  - The challenge
- Data Centric Architecture
  - Modularity
  - Separation Kernels
  - Data Distribution Service (DDS)
- Example
- Recommendations

Mixed Criticality Systems

- Any system that has multiple assurance requirements
  - Safety, at different assurance levels
  - Security, at different assurance levels
- Example: Unmanned Air Vehicle
  - Flight control is safety critical
  - Payload management is mission critical
- Ideally a system is built from components each with their own assurance requirements

The Challenge

- Design a modular plug-and-play architecture to reduce cost and reuse components
- Components must interact
  - The behavior of one component can affect another
  - It can be advantageous to have components at different criticality levels exchange data
  - Once a component interacts with another, then the whole system must be certified, not the individual components

The Solution

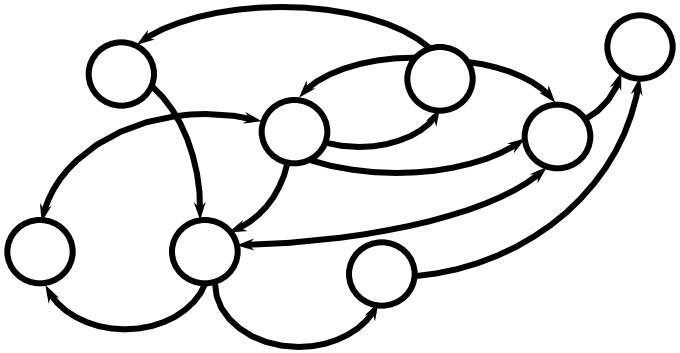
- Move from a component-interaction model to a **data-centric model**
- The data-centric model defines the data types and attributes in the system
- A component complies with the data model in terms of data it sends and receives
- This **decouples** the applications

Agenda

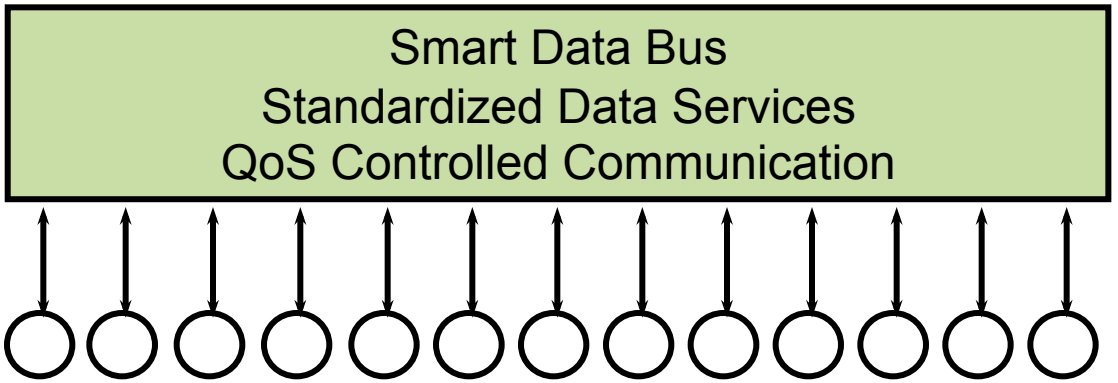
- Introduction
  - Mixed criticality systems
  - The challenge
- Data Centric Architecture
  - Modularity
  - Separation Kernels
  - Data Distribution Service (DDS)
- Example
- Recommendations

The Modular Approach

Monolithic Approach
- Certify whole system
- Connection oriented
- Tightly coupled
- Hard to evolve



Modular approach
- Certify components
- Data oriented
- Loosely coupled
- Evolvable



The Data Contract

- First, all data in the system is defined
- Next, data characteristics are defined
  - For example “airspeed” is flagged as flight critical
- Then components define data delivery attributes
  - A flight critical component specifies data rate that flight critical data must be delivered
- This creates a **“data contract”**

Data Centric Approach for Layered Assurance

- Data contract includes
  - Data type
  - Name
  - Quality of Service
- Sender/Receiver of the data is anonymous
- Validation
  - Component validation – does it conform to the data model
  - System validation – is there a producer at correct assurance level for each required data

Realization in a Layered Assurance System

- Separation Kernels
  - Guarantees isolation of components
  - Controls data flow

- Object Management Group (OMG) Data Distribution Service
  - Used to implement the data model and distribute data

Separation Kernels

- Base of the solution for mixed-criticality systems certification
- Isolation and Control
  - Each guest operating system (OS) **runs in its own partition**
  - Each guest OS is **isolated** over both **time and space**
  - Information flows are **tightly controlled**
  - Components can be pre-certified and composed quickly into new configurations
- **Caveat**
  - Does not address interdependency between components or interactions between components on separate computers

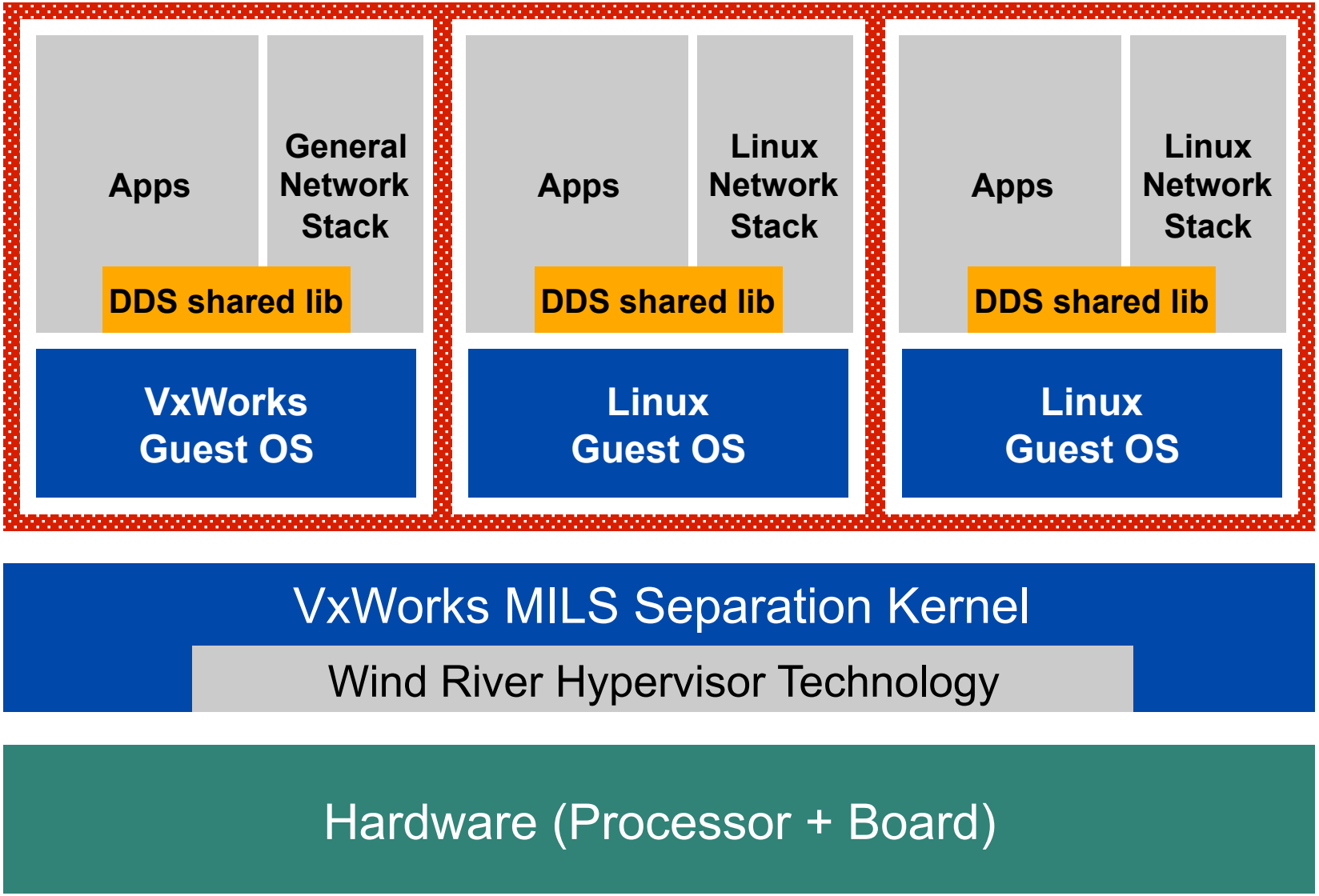
Data Distribution Service (DDS)

- Data-centric publish-subscribe middleware for real-time communication
  - Strong data typing
  - Quality-of-Service (QoS) parameters
    - *e.g.,* deadlines for message delivery, bandwidth control, reliability model control, failover and backup specification, data filtering etc.
- DDS QoS parameters characterize:
  - the **data contracts** between participants
  - the **properties of** the overall **data model**
  - **real-time communication and delivery requirements** on a per-data-stream basis

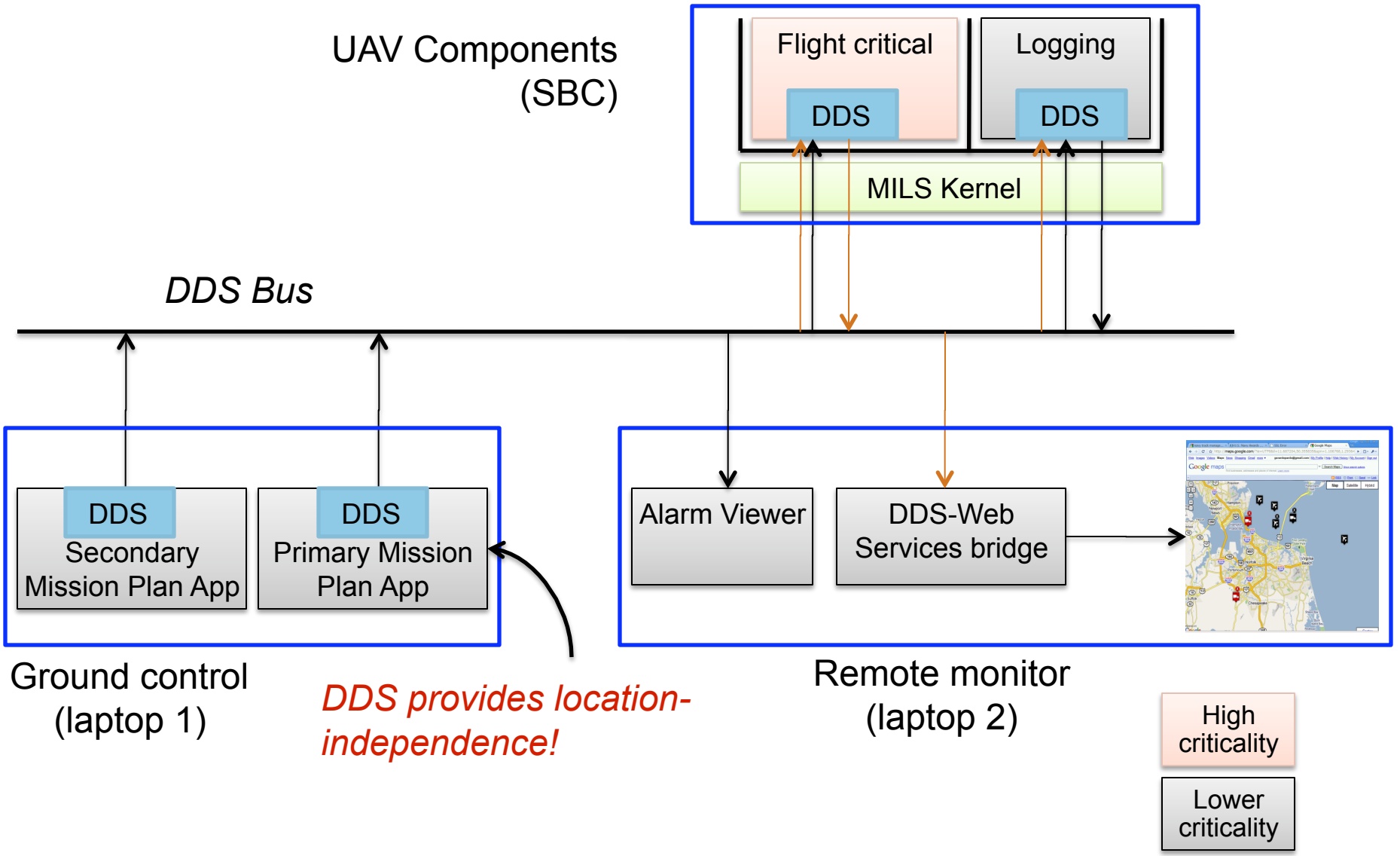
Agenda

- Introduction
  - Mixed criticality systems
  - The challenge
- Data Centric Architecture
  - Modularity
  - Separation kernels
  - Data Distribution Service (DDS)
- Example
- Recommendations

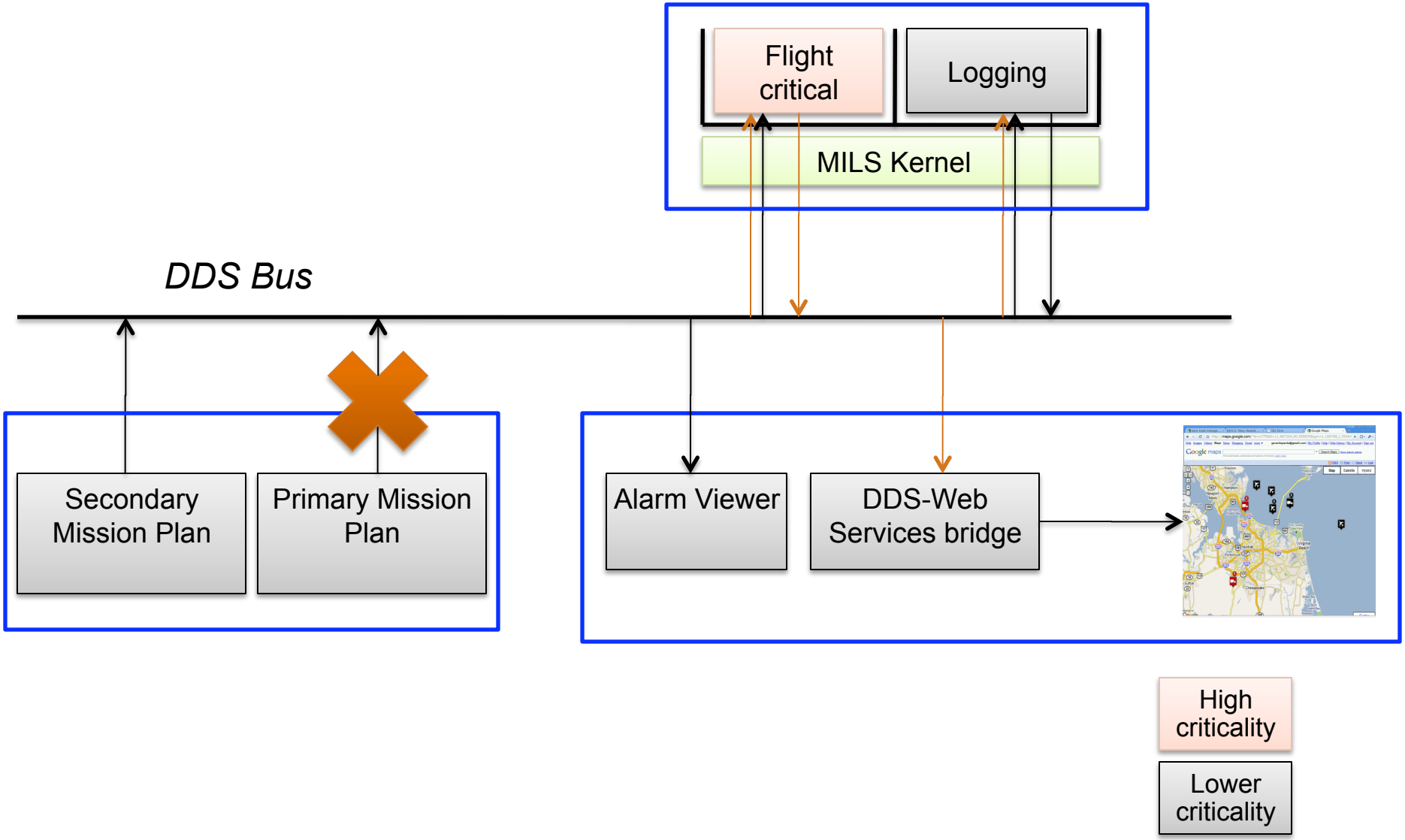
Example: Wind River VxWorks MILS and RTI Data Distribution Service



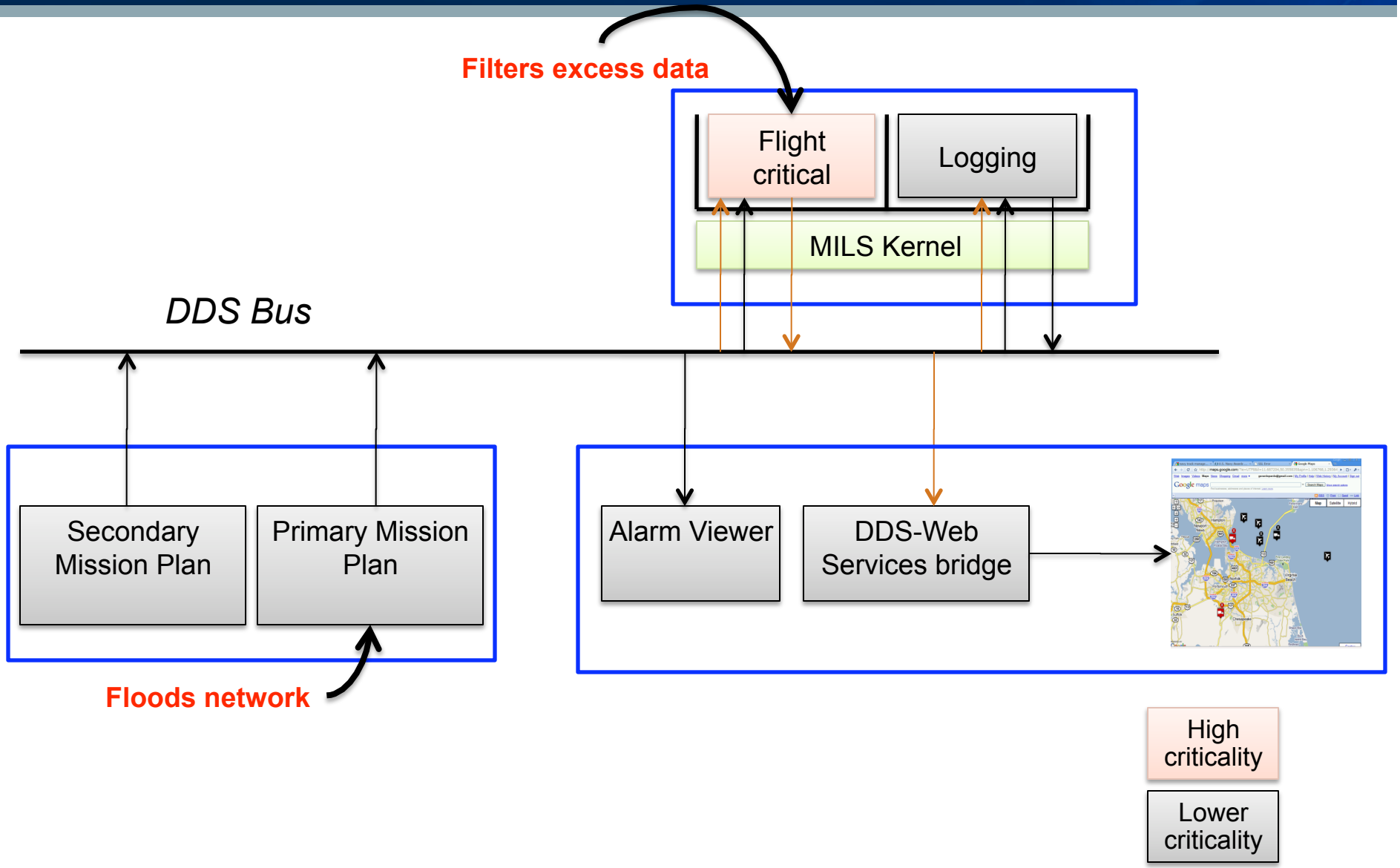
Example Demo Overview



Scenario 1: Failover of Lower Criticality



Scenario 2: Lower Criticality Floods the Network



Agenda

- Introduction
  - Mixed criticality systems
  - The challenge
- Data Centric Architecture
  - Modularity
  - Separation kernels
  - Data Distribution Service (DDS)
- Example
- Recommendations

Tenets for Developing Safety-Critical Software

- Reduce code size
- Consider testability in design
- Enable verification
  - Avoid recursion
  - Set limits – for example limits on iterations
- Deterministic in time
- Deterministic in memory
  - No dynamic memory allocation after startup

Challenges DDS for in Safety-Critical Systems

- DDS is designed to be dynamic
  - Entities discovered at run-time
  - Number of nodes and endpoints can change
  - DDS adaptable to changes in the environment, for example increasing a sample queue

- DDS is feature rich
  - Implementations can have many lines of code, making certification costly
  - Many features either not suitable or not applicable to safety-critical systems

DDS Discovery

- The process by which domain participants find out about each other's entities
  - Each participant maintains database on other participants in the domain and their entities
- Happens automatically behind the scenes
  - "anonymous publish-subscribe"
- **Dynamic** discovery
  - Participants must refresh their presence in the domain or will be aged out of database
  - QoS changes are propagated to remote participants

DDS Discovery in Safety-Critical Systems

- Do not want
  - An a priori unknown number of participants connecting
  - An a priori unknown number of remote Data Writer/Data Readers
- Do want
  - To know if remote participants are up
  - A simple protocol
- Solution
  - Stage 1: the same, dynamic participant discovery
  - Stage 2: static loading of endpoints

Memory Model

- In DDS, queue sizes can change
  - Discovery queues grow when more nodes join the system
  - Data queue sizes grow to accommodate more data
- In a safety-critical system, memory must be deterministic
- Solution
  - Set all resource limits before creating entities
  - Memory is only allocated during _create calls
  - There is no memory growth policy

DDS Feature Set

- Support the same entites
  - Domain Participant, Publisher, Subscriber, Data Reader, Data Writer, Topic
- Need core DDS APIs
  - Create entities
  - Write/Read
  - Listener for data available
  - Get QoS and Entities
- RTPS wire protocol compatibility

QoS supported

- QoS needed for safety-critical systems
  - Best-effort communication
  - Reliable communication
  - History queue
  - Reader and Writer Deadline
  - Manual assertion of liveliness by topic
  - Time-based filter – Filter only on the reader
  - Ownership
  - Ownership strength

Conclusions

- Mixed-criticality systems certification can go a long way
- We can leverage:
  - Isolation and control capabilities through separation kernels
  - Modularity through a data-centric architecture
- It is possible to build mixed criticality systems that provide:
  - Modularity
  - Evolvability
  - Fault tolerance

Acknowledgments

- **Wind River** – provided their MILS platform as well as valuable feedback
- **United States Air Force** - contract FA8650-10-M-3025
  - The content of this work is the responsibility of the authors and should not be taken to represent the views or practices of the U.S. Government or its agencies.

Thank You