# Service Oriented Architecture in Embedded Multi-Processor System

David Richmond – L-3 Communications

# What is Needed in the Architecture

- L-3 does Data Communication Systems
  - Data Path:
    - High Rate signal processing happens in hardware and FPGAs
  - Command/Status Path:
    - Inter Process Communications, historically outside of data path

- Requirements of Architecture
  - Scalability
  - Extensibility
  - High Availability
  - Loose Coupling

# Requirements of Architecture

- Scalability
  - Need support for various configurations with varying number of instantiations of components with minimal complexity impact
- Extensibility
  - Need to enhance capabilities with minimal software updates
    - Continuously creating new hardware modules
    - Need to cleanly combine existing hardware modules in different configurations

# Requirements of Architecture

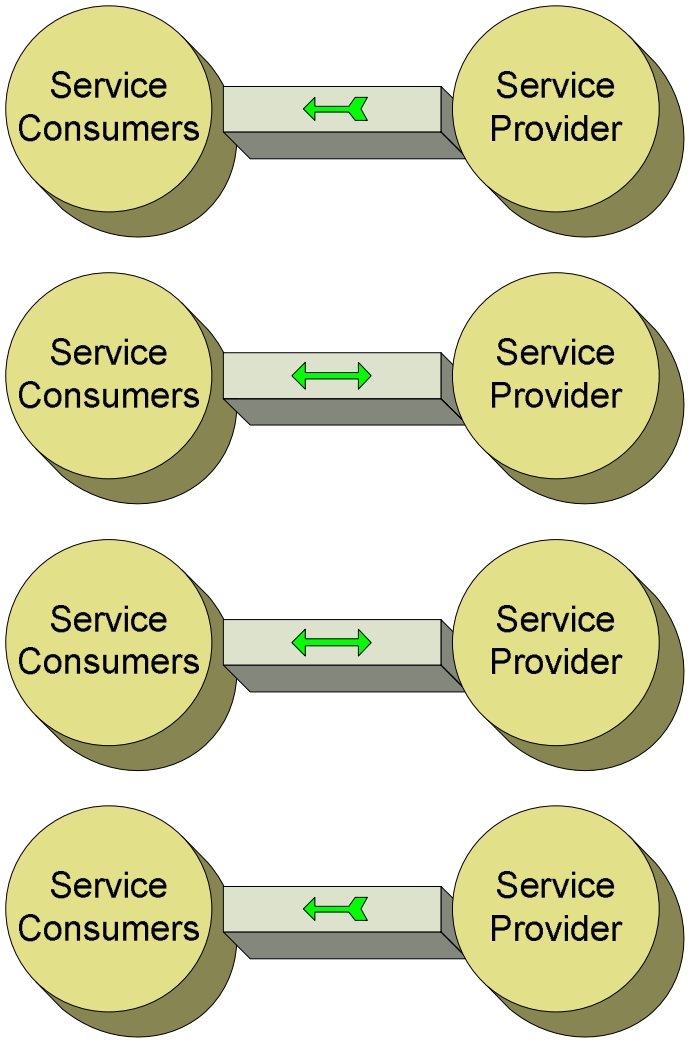
- High Availability
  - Need deterministic system state
  - Need to have the system tolerate temporary component outages on sub-assemblies without degrading functionality outside of those components
  - System must degrade predictably
    - No reduced system responsiveness if a component goes down
- Loose Coupling
  - System must allow a component to be updated or replaced with minimal impact to existing components

# Answer – Message Oriented SOA

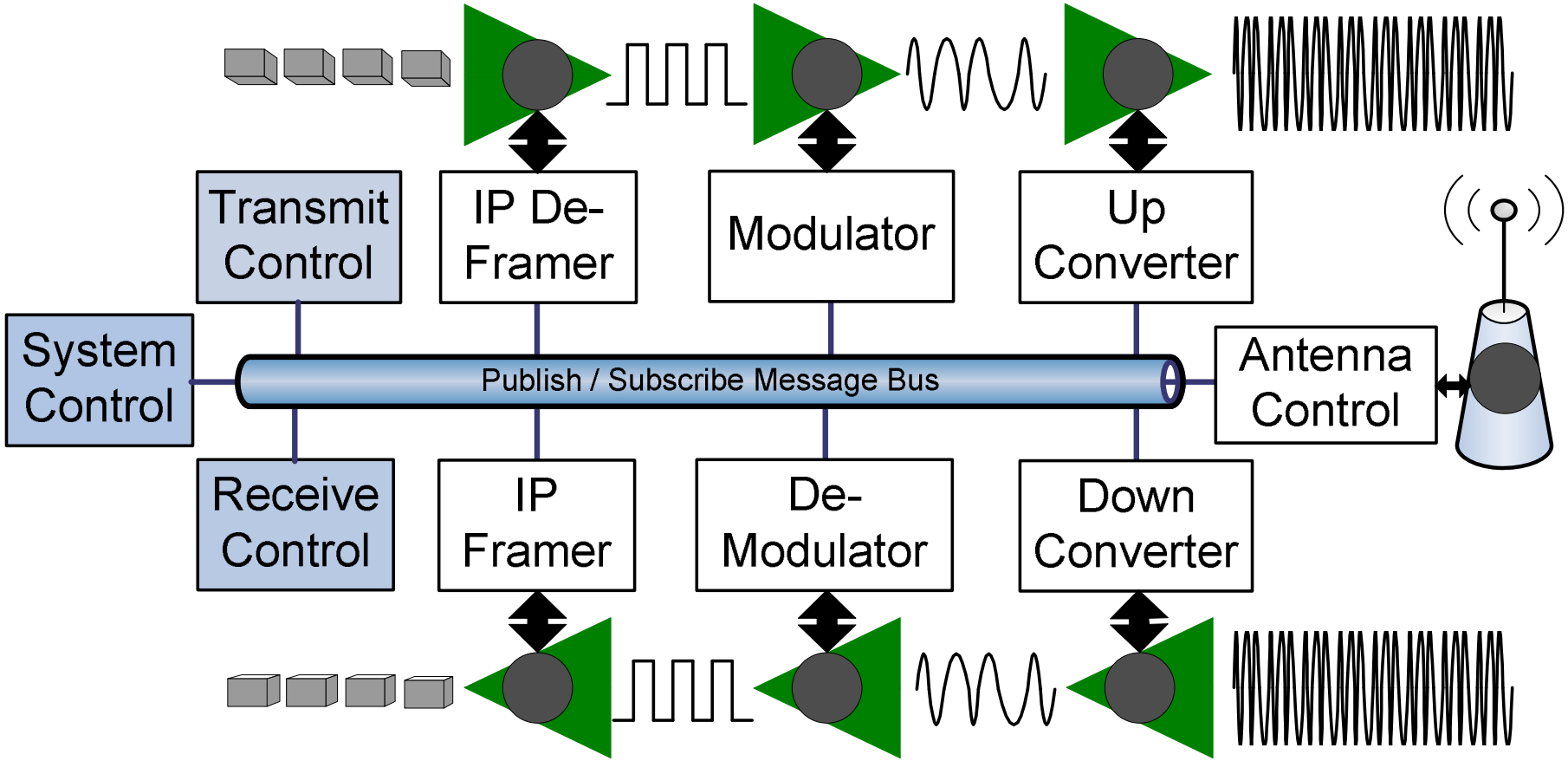
- Event Driven SOA
- Use a Publish / Subscribe, Message Oriented Middleware (MOM) for all Inter-process communications
- Translates cleanly to event driven state machines behind the service contract
- SOA contracts remain stateless
  - Contract remains the same – Supporting messages
    - Sub-types of messages provide the extensibility
- *Service Consumers* are typically a GUI or a controlling software component

# Classes of Messages

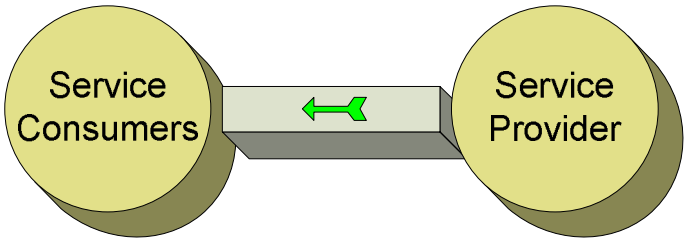
- Publish Capabilities
  - Status Only
- Publish Configuration
  - Command and Status
- Publish Operations
  - Command and Status
- Publish Metrics
  - Status Only



# Example Software Components

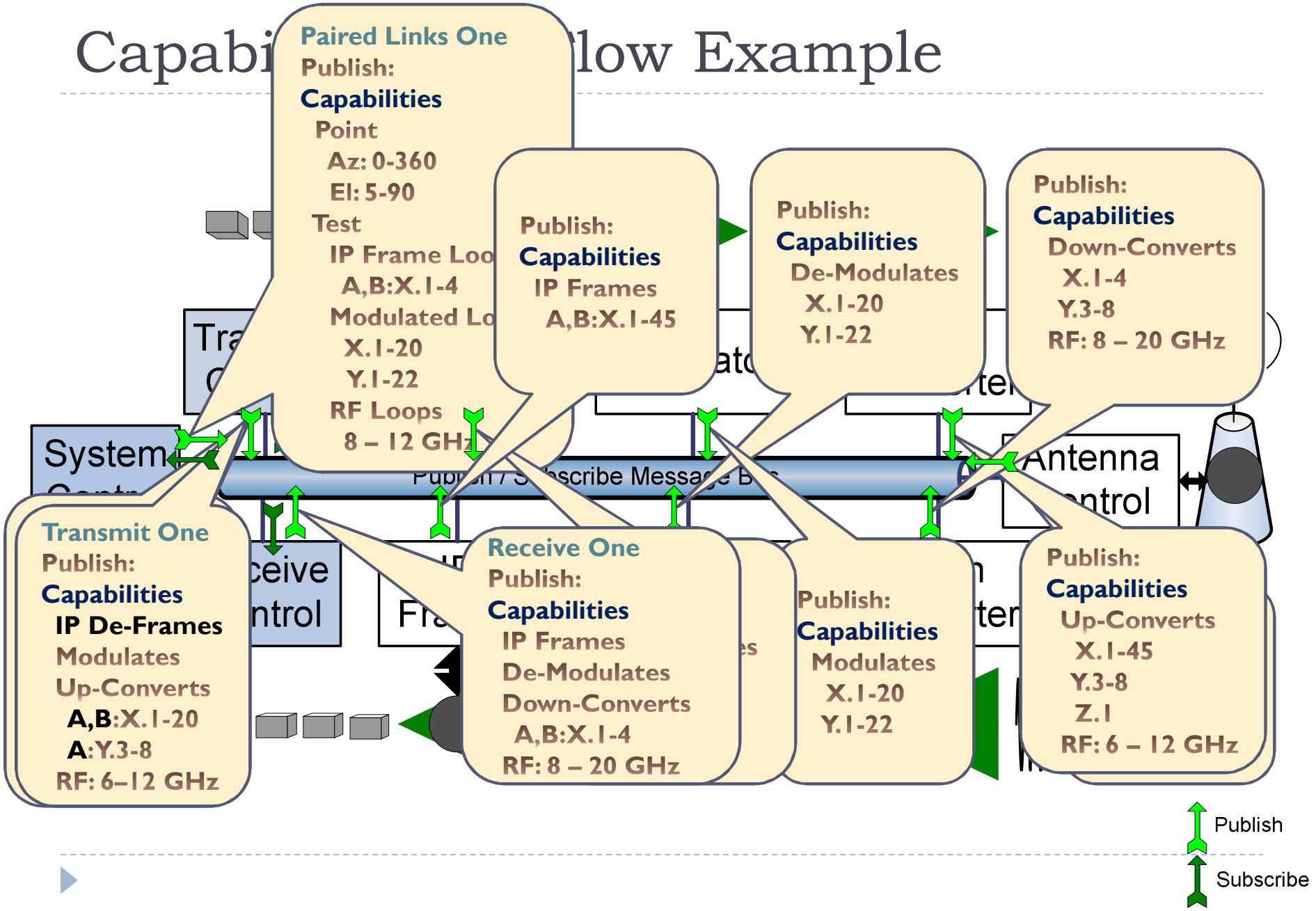


# Publish Capabilities



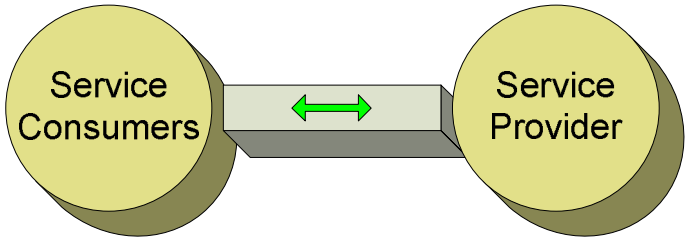

- Capabilities is published from the *Service Providers*
- *Service Consumers* subscribe to capability messages
- *Controller Component* or *Controllers* are *Internal Service Consumers* which are also *Service Providers* to abstract internal details
- *Controller Components* subscribe to capability messages
- *Controller Components* publish combined capabilities
  - Knows when a message class does applies to a type of component

# Capabi low Example

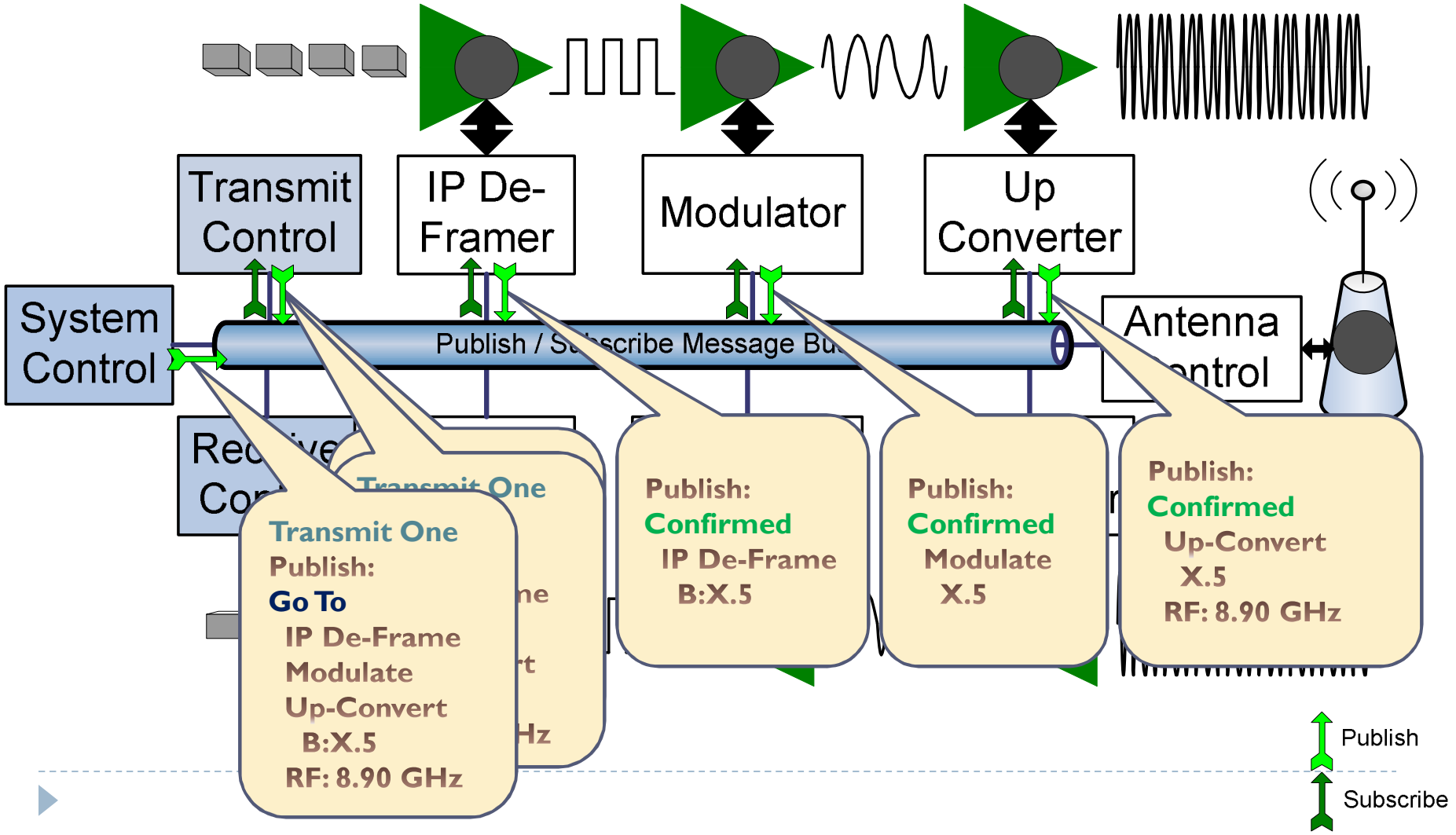


# Publish Configurations

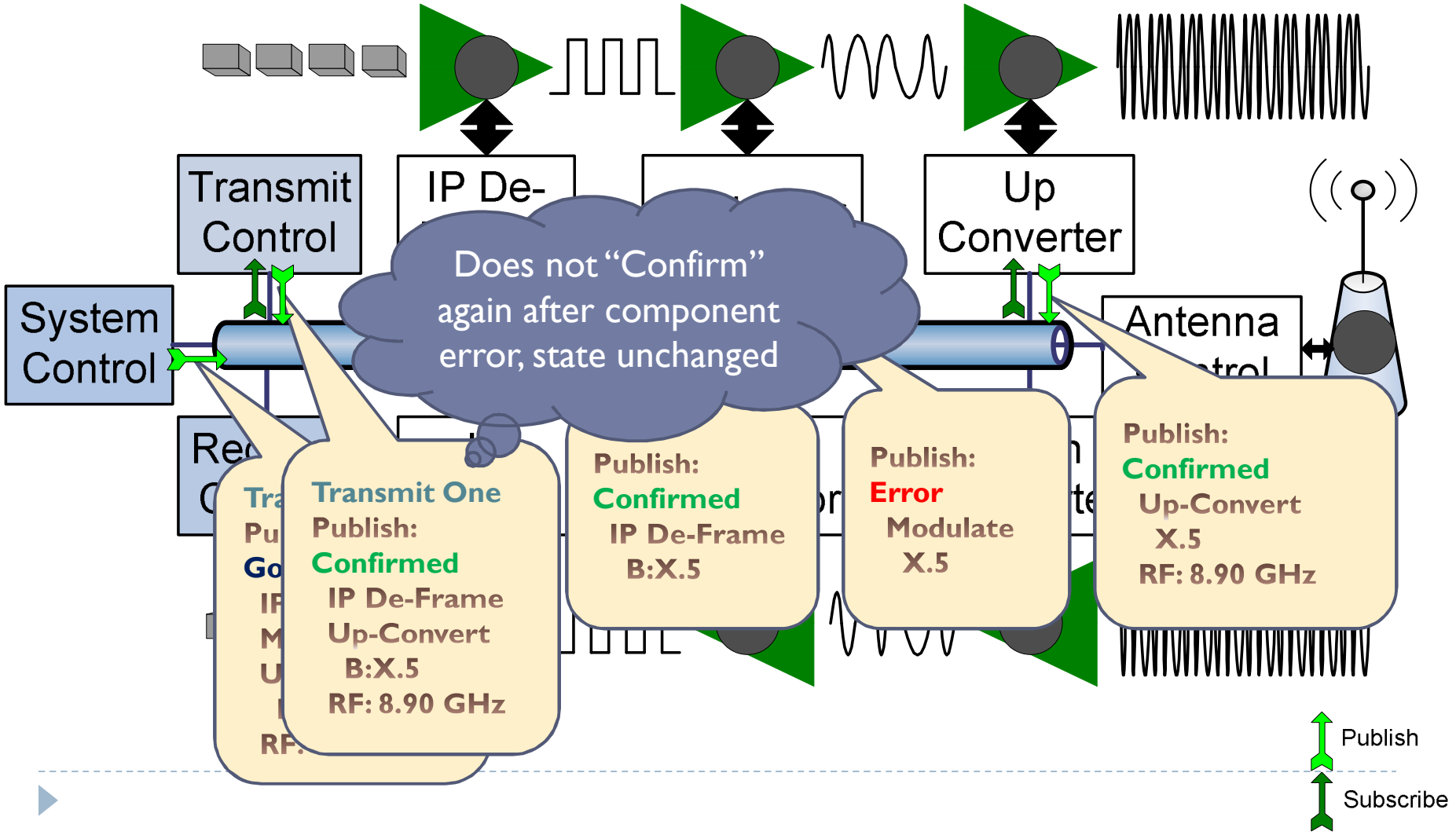
- Configuration is a published command message from the controlling *Service Consumers*
- The *Controller Components* publish commands which abstract the internal system configuration from the external *Service Consumers*
- The *Service Provider* publishes the configuration state



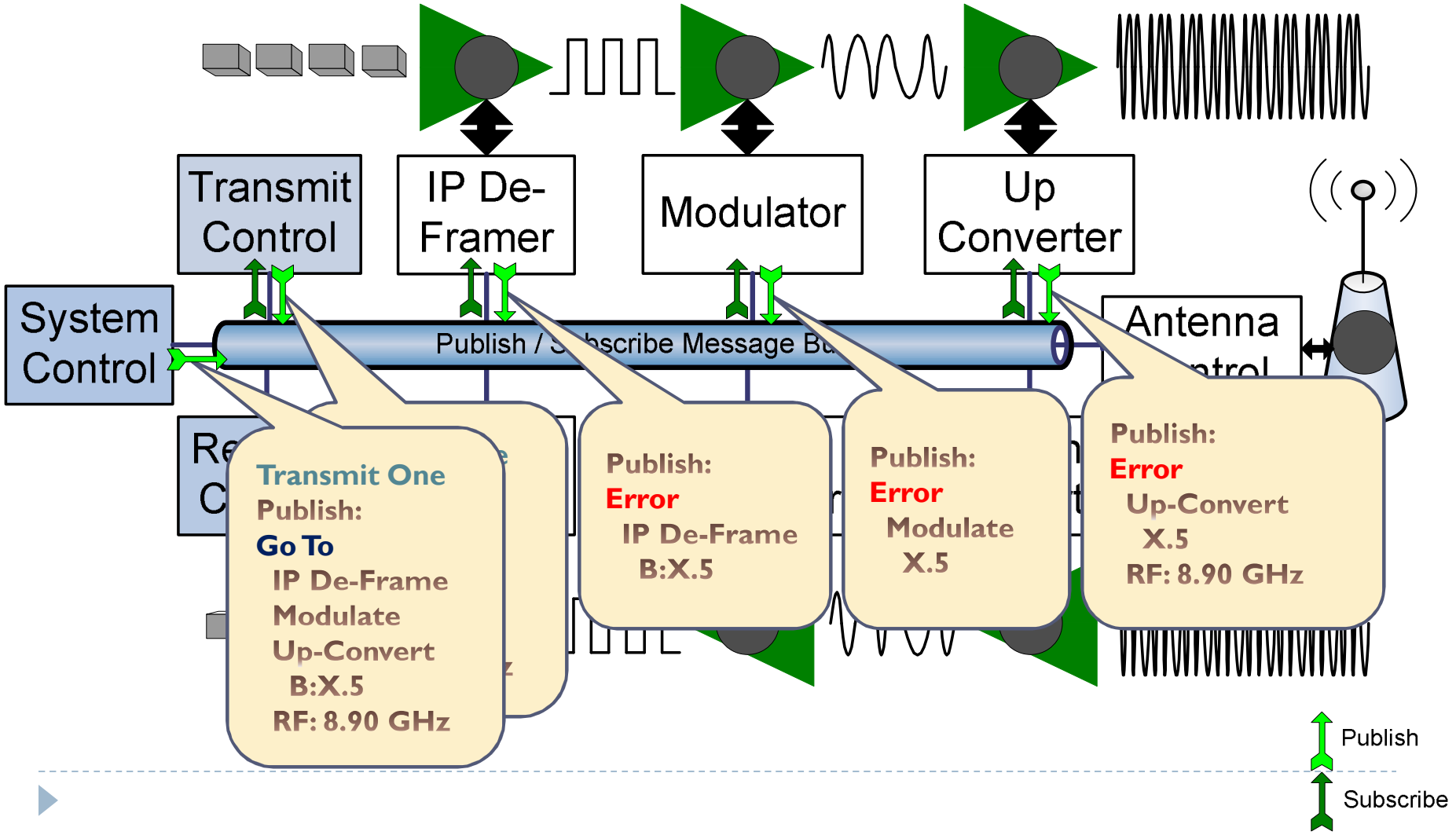

# Configuration Down Flow Example



# Configuration Partial Success Example

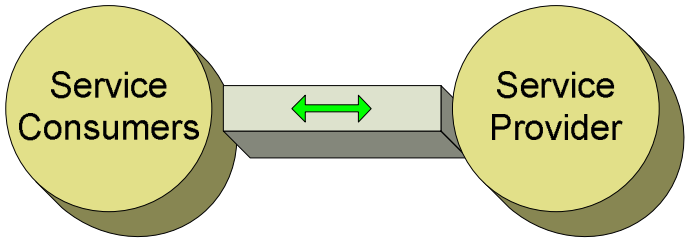


# Configuration Failure Example

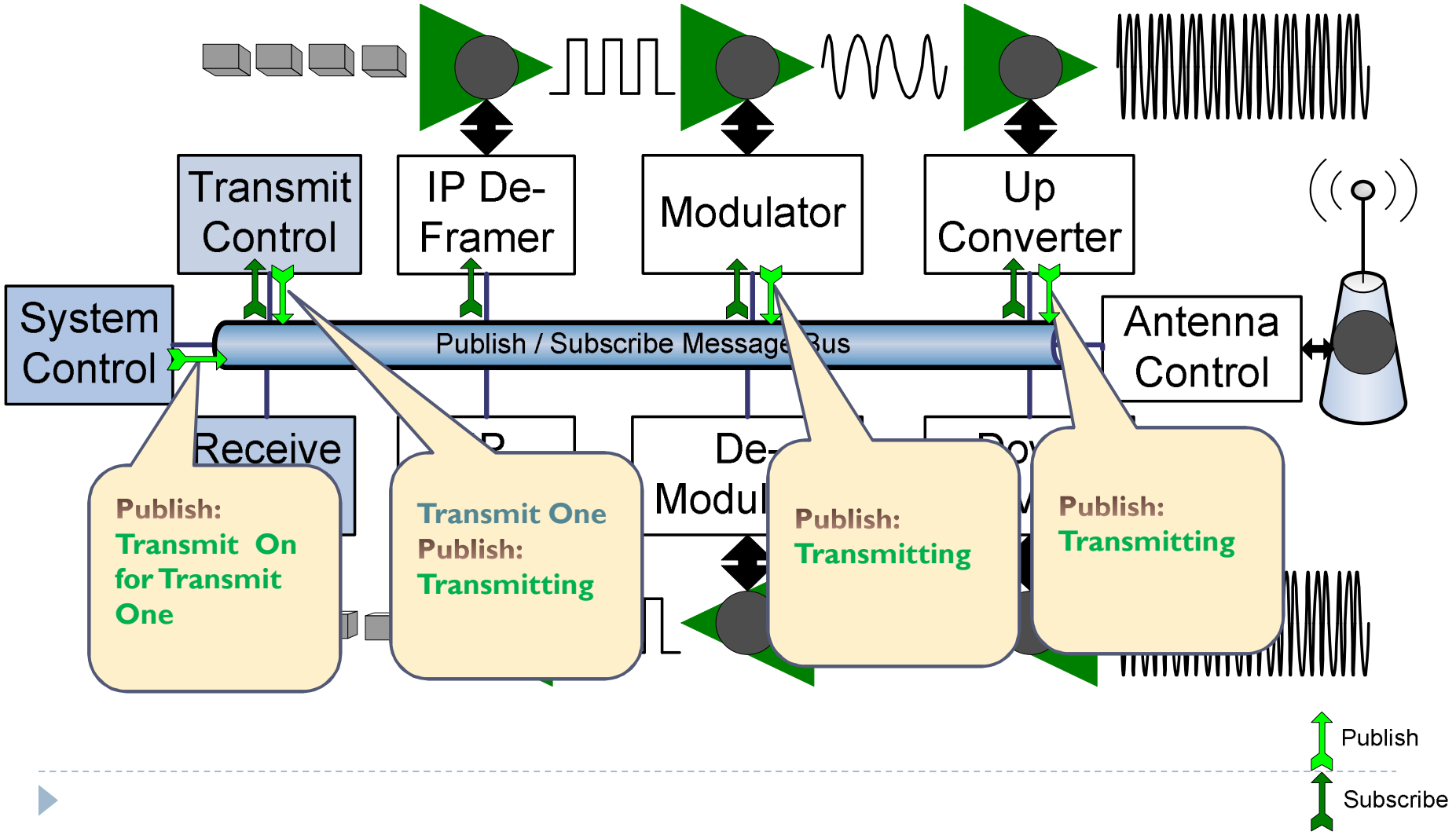


# Publish Operations

- Operations are published command messages from the *Service Consumers*
- *Controller Components* may decompose Operation messages into published commands for controlled component
  - Providing abstraction for the external *Service Consumers*
- *Service Provider* publishes operational state

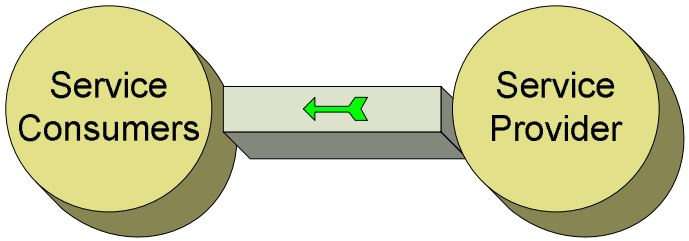



# Operation Command Down Flow

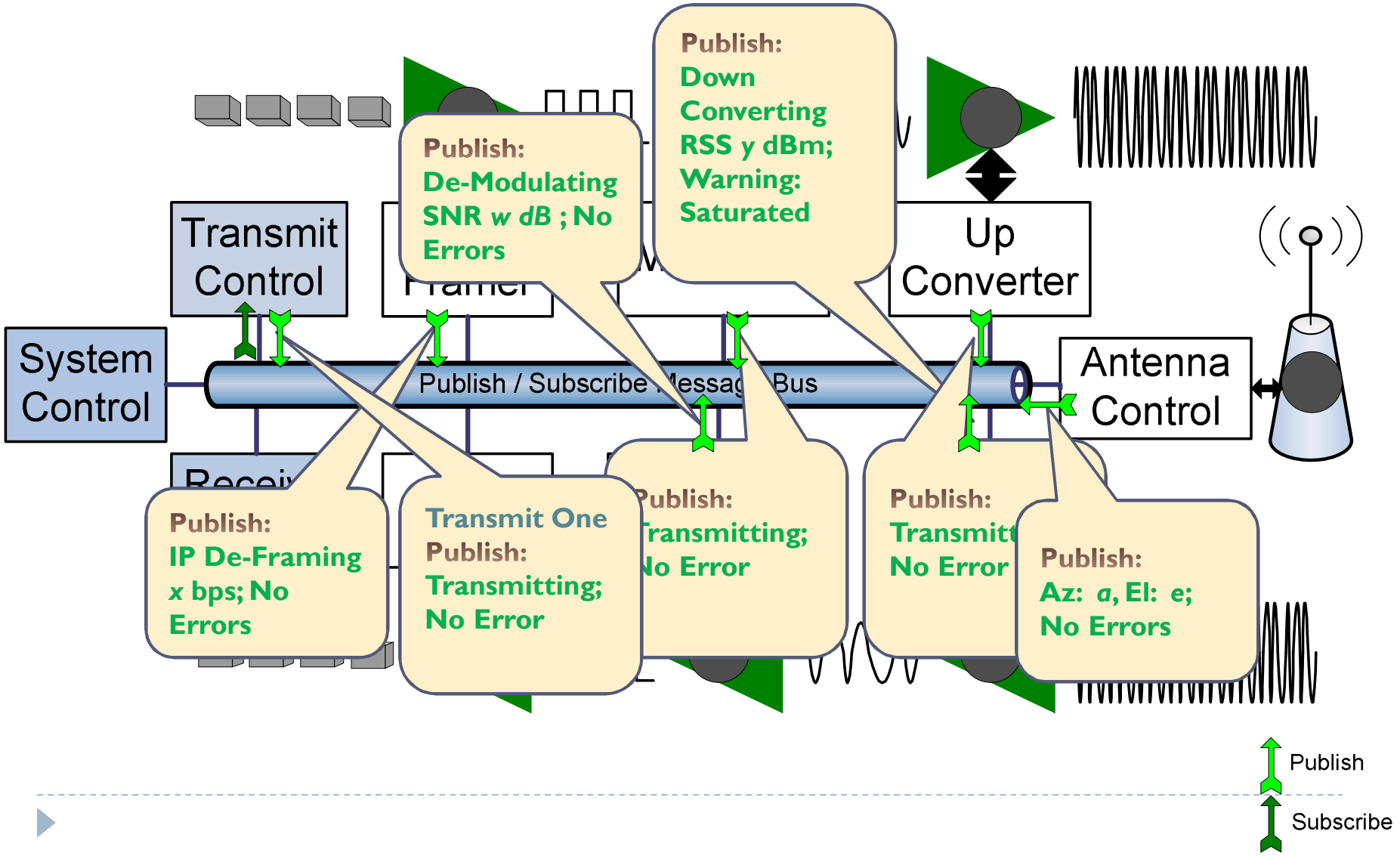


# Publish Metrics

- *Service Providers* (including *Controller Components*) publish all System Metrics
- *Service Consumers* subscribes to the metrics in which it is interested
  - Any interested software component subscribe to the needed metrics messages



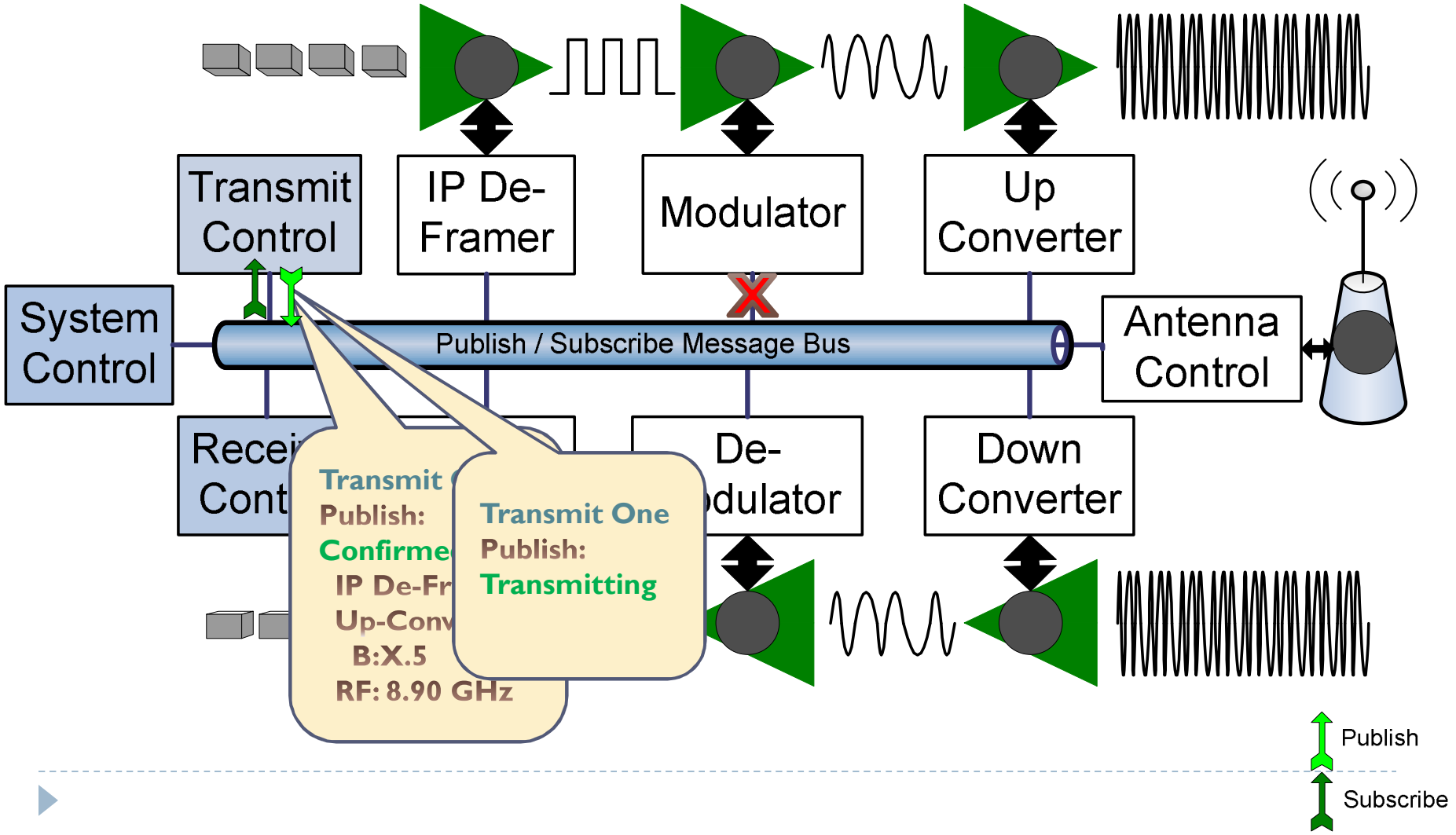

# Metrics Flow



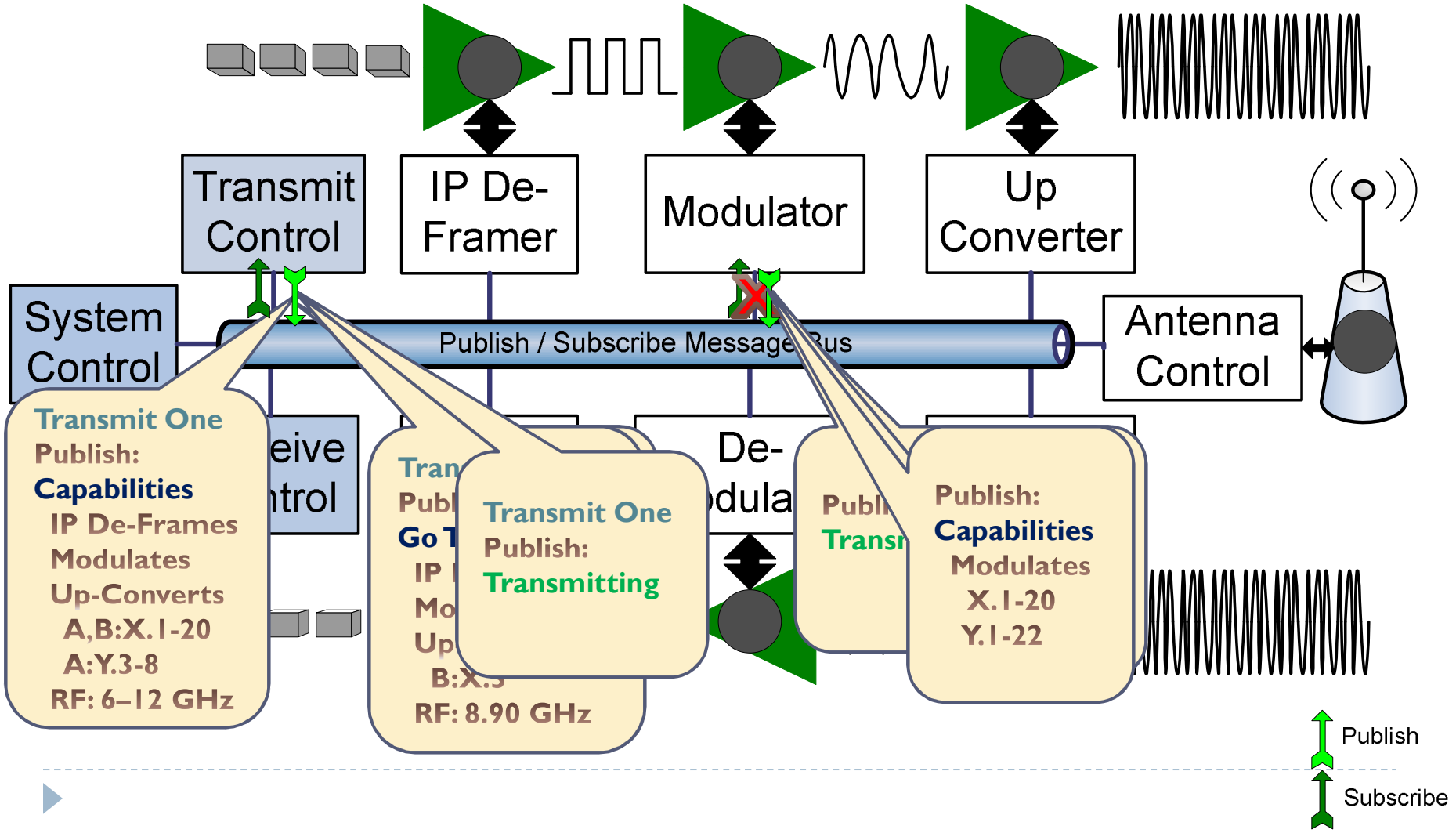
# High Availability

- Need deterministic system state
- Need to have the system tolerate temporary component outages on sub-assemblies without degrading functionality outside of those components
- System must degrade predictably
  - No reduced system responsiveness if a component goes down
- Essential for un-manned, embedded systems

# Component Communication Outage



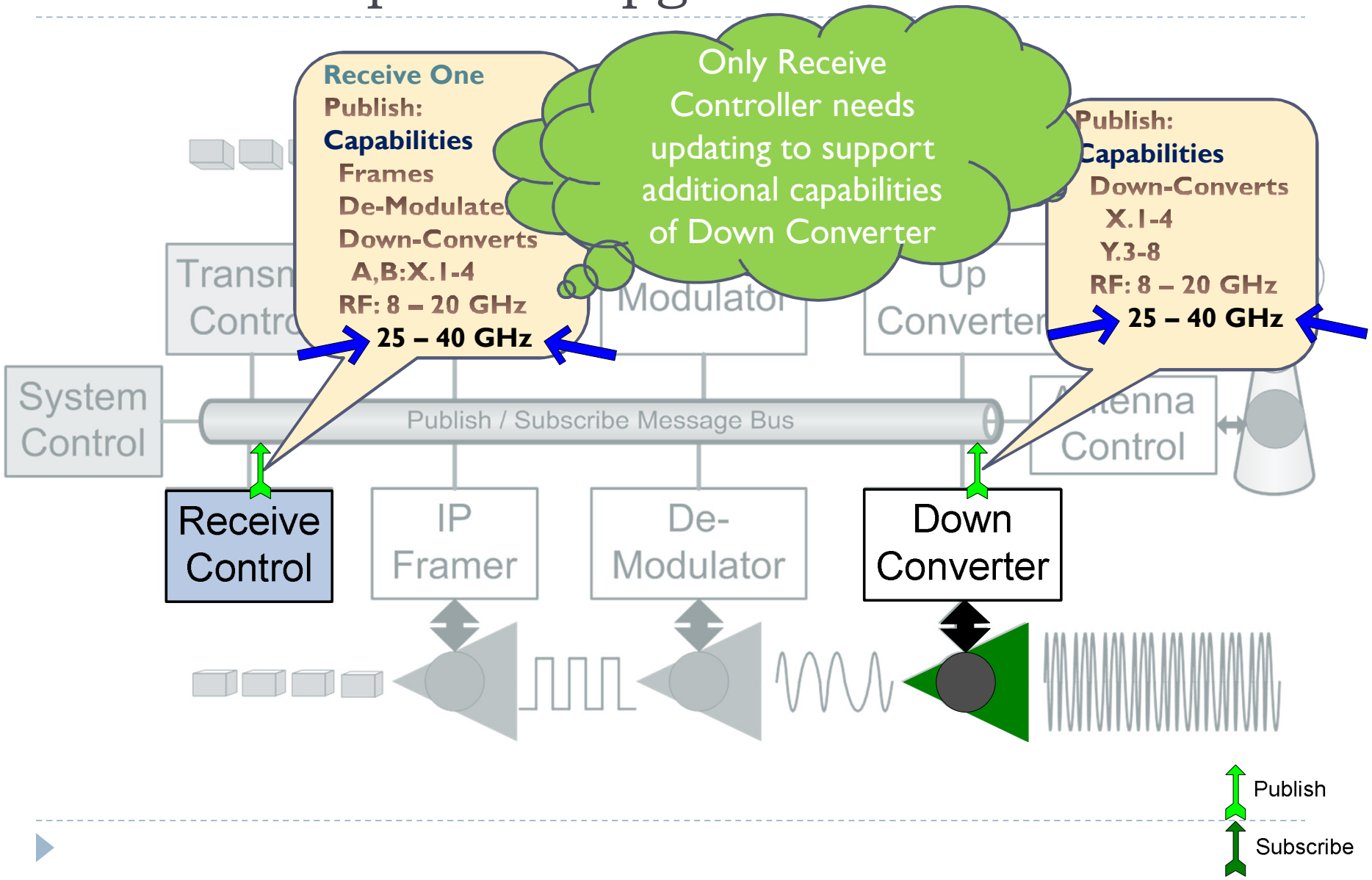
# Component Communication Recovery



# Extensibility / Loose Coupling

- Need to enhance capabilities with minimal software updates
  - Continuously creating new hardware modules
  - Need to cleanly combine existing hardware modules in different configurations
- System must allow a component to be updated or replaced with minimal impact to existing components

# Extensibility/Loose Coupling – Component Upgrade

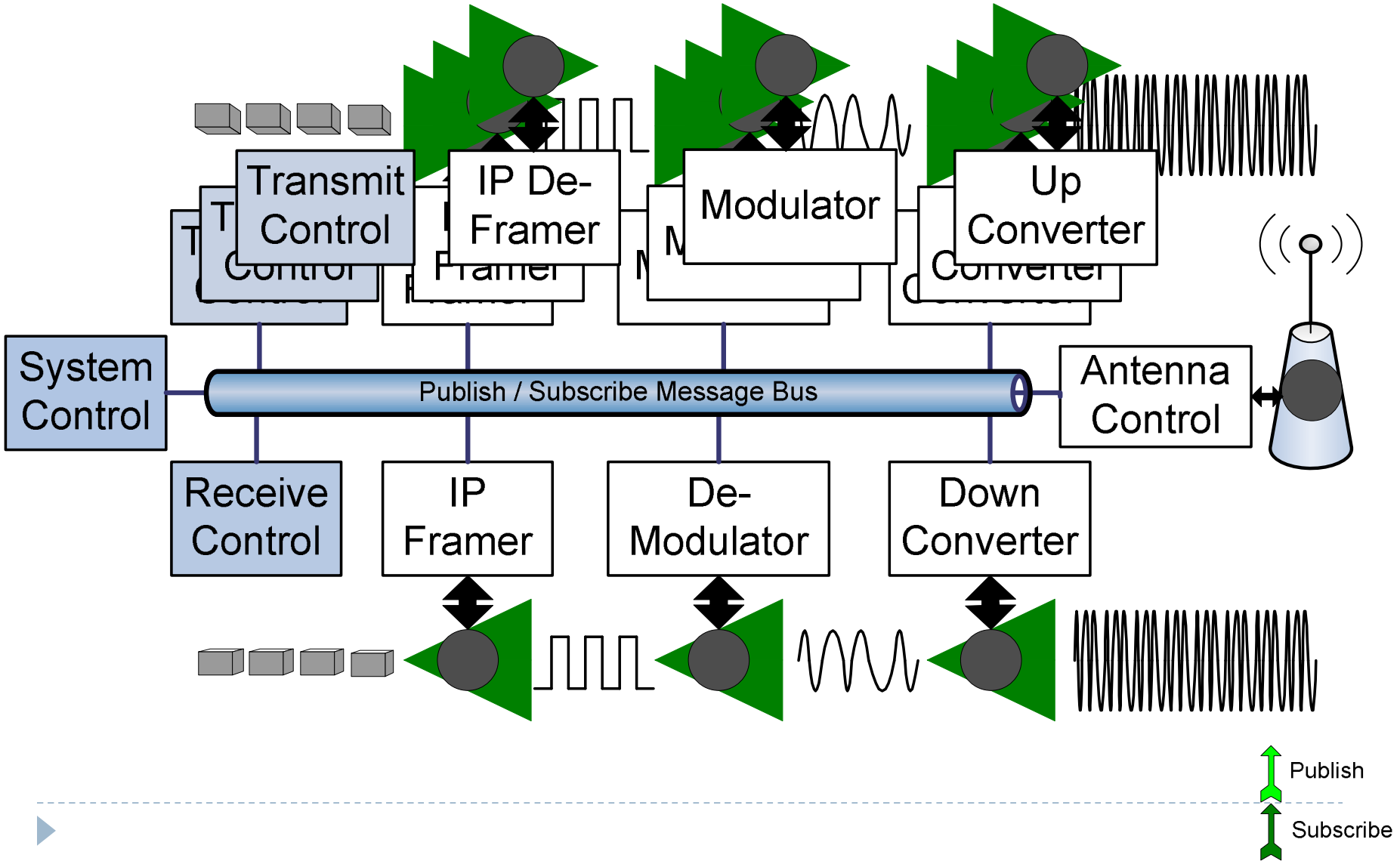


# Scalability

- Need support for various configurations with varying number of instantiations of components with minimal complexity impact

- Publish / Subscribe approach avoids $N^2$ growth of connections

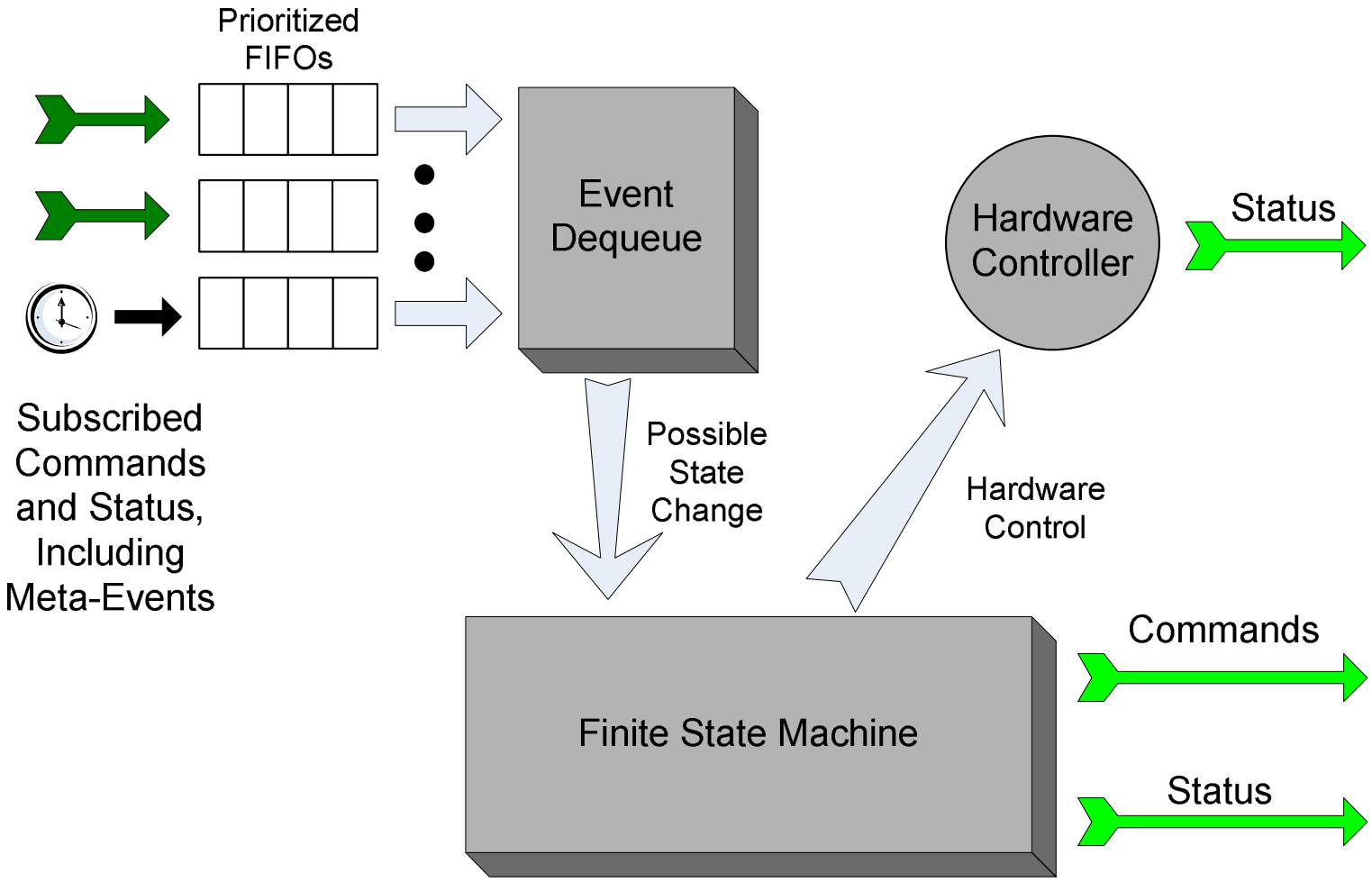
# Scalability Example



# Publish / Subscribe to Event Driven Architecture

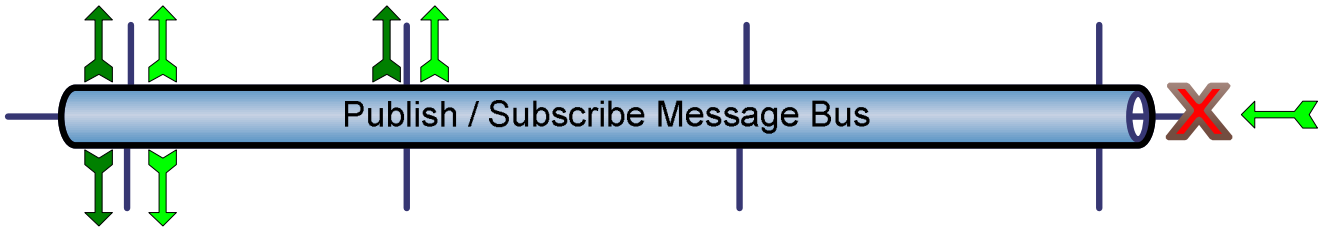
- Published Messages become Events
- Metadata relevant to the subscribed messages become events
  - Example: Subscribed message has not been heard within an expected time

# Event Driven Architecture



# Data Distribution Service (DDS) as MOM

- Open Standard
  - Published by OMG, Ratified (v1.2) January 2007
- Designed for Messaging
  - Better solution than RPCs manipulated into a message service
- Manages Publish/Subscribe efficiently
  - Only transmits messages if a subscriber exists
  - Implementations smart about switching between unicast, multicast, and subnet broadcast based on number and topology of subscribers




# DDS

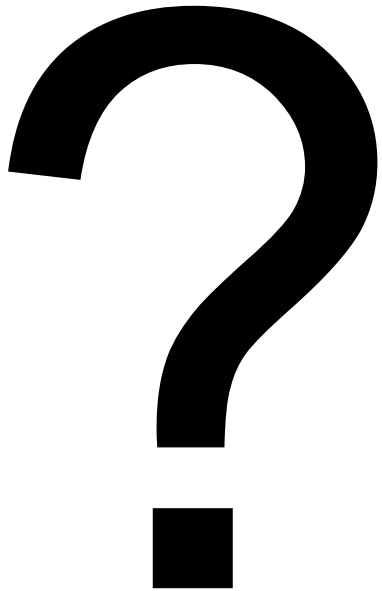
- Messages defined using IDL or XML
  - Open standard for definition format
  - Messages strongly typed, formats negotiated out-of-bounds
- Divides message space into "Domains"
  - Domains typically implemented on separate IP ports
- Supports "timeliness" of data
  - Retains published messages within a defined time window
- Supports subscribers joining after published data
  - Late subscribers get retained published data

# Requirements Satisfied

- Scalable
  - Adding components is a linear growth
  - Adding additional components only affects Controller Components
- Extensibility
  - Adding additional message support to some components does not force interface update on all components
    - Example: Adding a new modulation does not require any other component to update to keep working with existing modulations

# Requirements Satisfied

- High Availability
  - The dynamic published capabilities allows for components to "come and go" with the services adapting
  - State is always known in the system
- Loose Coupling
  - No waiting for RPC to return
  - New messages can be added without any impact code handling existing messages
  - Messages conform to system abstraction



# Acronyms

| Acronym | Definition | Acronym | Definition |
|---|---|---|---|
| Az | Azimuth (of antenna) | IP | Internet Protocol |
| bps | bits per second | MOM | Message Oriented Middleware |
| dB | decibels | OMG | Object Management Group |
| dBm | decibels referenced to one milliwatt | RF | Radio Frequency |
| DDS | Data Distribution Service | RPC | Remote Procedure Call |
| El | Elevation (of antenna) | RSS | Received Signal Strength |
| FPGA | Field Programmable Gate Array | SNR | Signal to Noise Ratio |
| GHz | Giga Hertz | SOA | Service Oriented Architecture |
| IDL | Interface Definition Language | XML | Extensible Markup Language |