Instrumentation and Monitoring of Real-Time Distributed Systems

May 18, 2011
SSTC 2011

The Real-Time Middleware Experts

Heidi Schubert
heidi@rti.com

Real-Time Distributed Systems

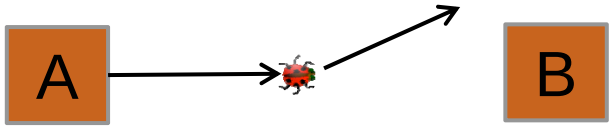
Why is debugging hard?

- Big, complex things
  - Difficult to reproduce the system in the lab
  - System grows and changes over time
  - Need to debug operational system
  - Contains computers with vastly different capabilities

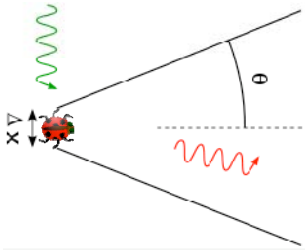
- Cyber-physical systems
  - Physical systems can behave in unexpected ways
  - Safety and security important – don't want computers breaking the physical system
  - Can't slow down time to debug

Pernicious Bugs in Distributed Systems

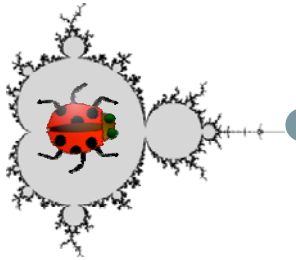
- Distributed bug



  - Each node works as expected, but not together



- Heisenbug
  - Disappears when you look for it
- Scaling bug
  - Shows up as system get bigger

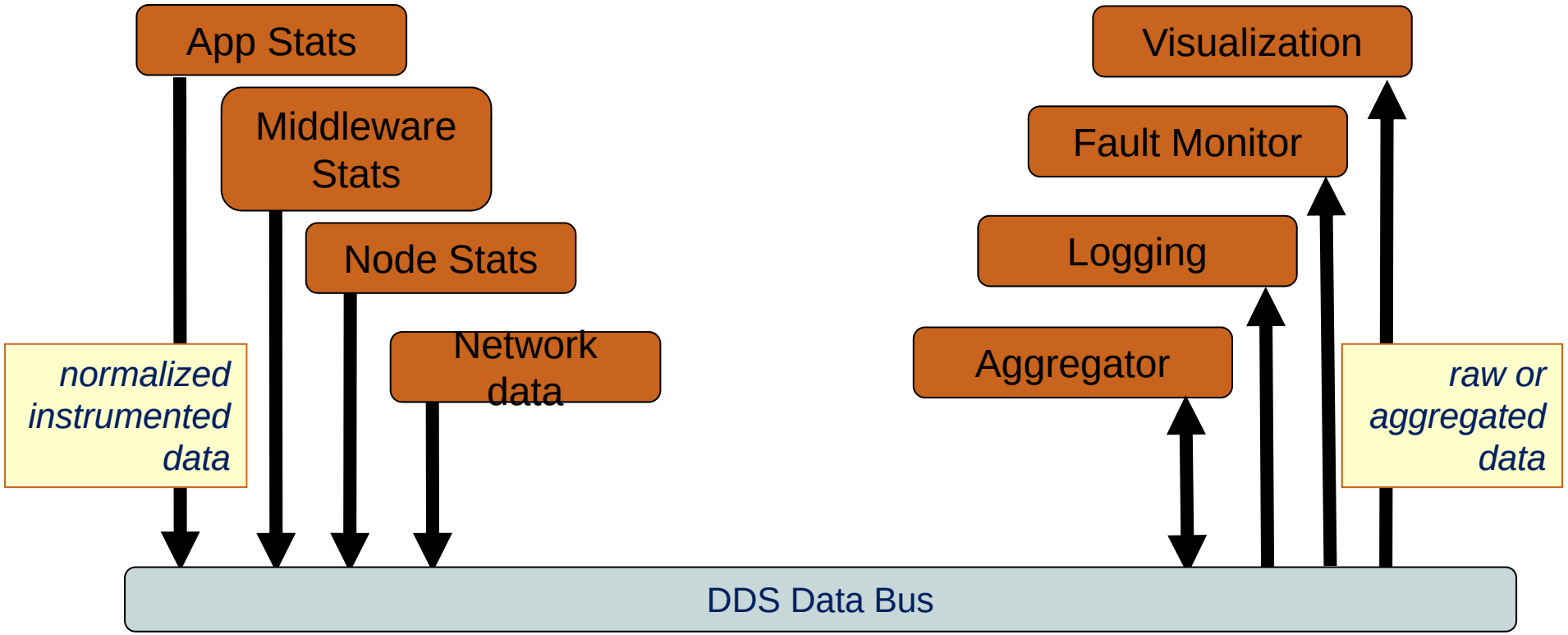


- Mandelbug
  - Cause is complex and unknown

Need

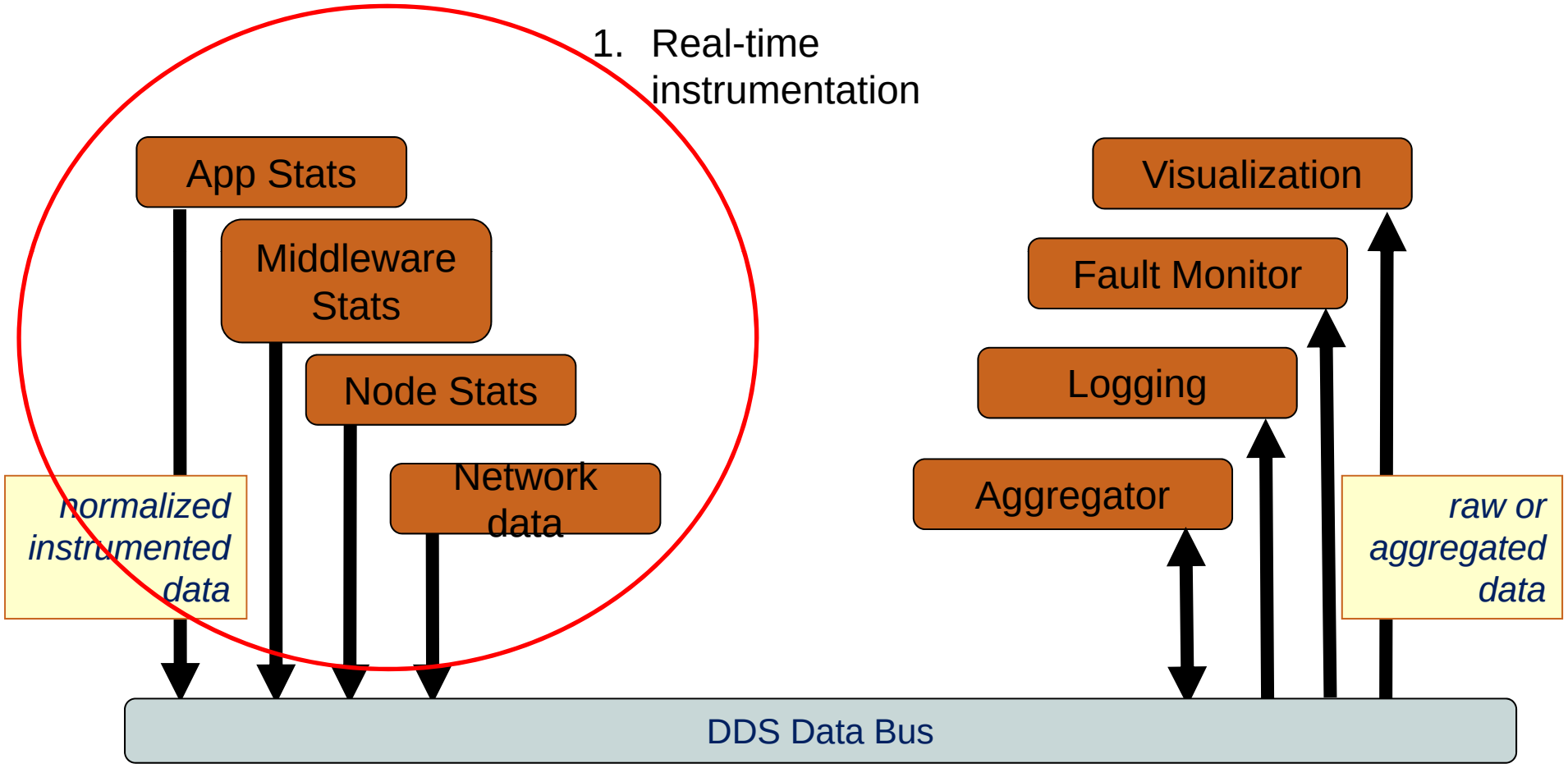
- Visibility into the live operation
  - Node level instrumentation
  - Minimally intrusive monitoring
  - Continuous instrumentation
- Understanding at the system level
  - Collect data from multiple nodes
  - Analysis of the data to detect and debug

Open, Extensible Framework



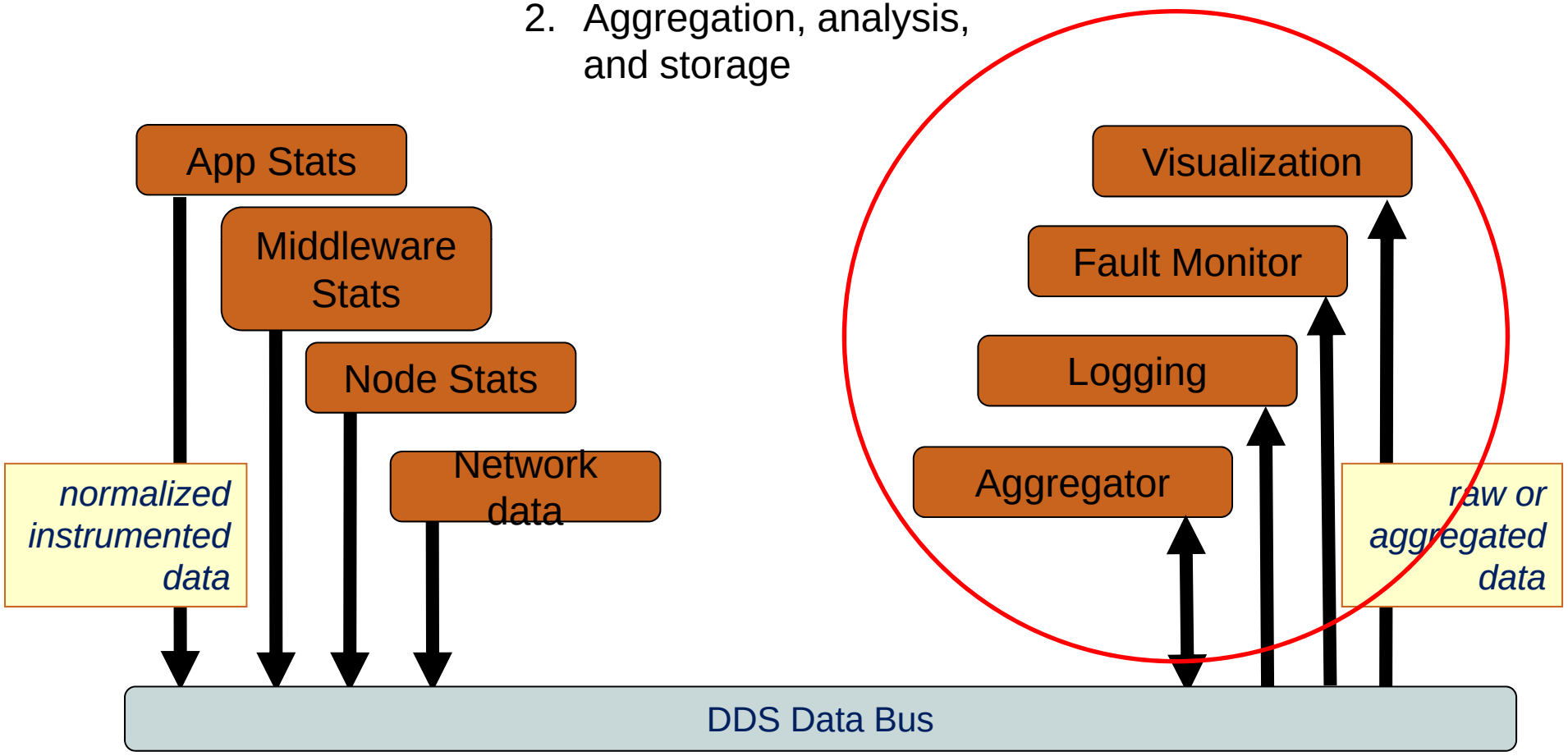
3 key pieces to the solution

Open, Extensible Framework



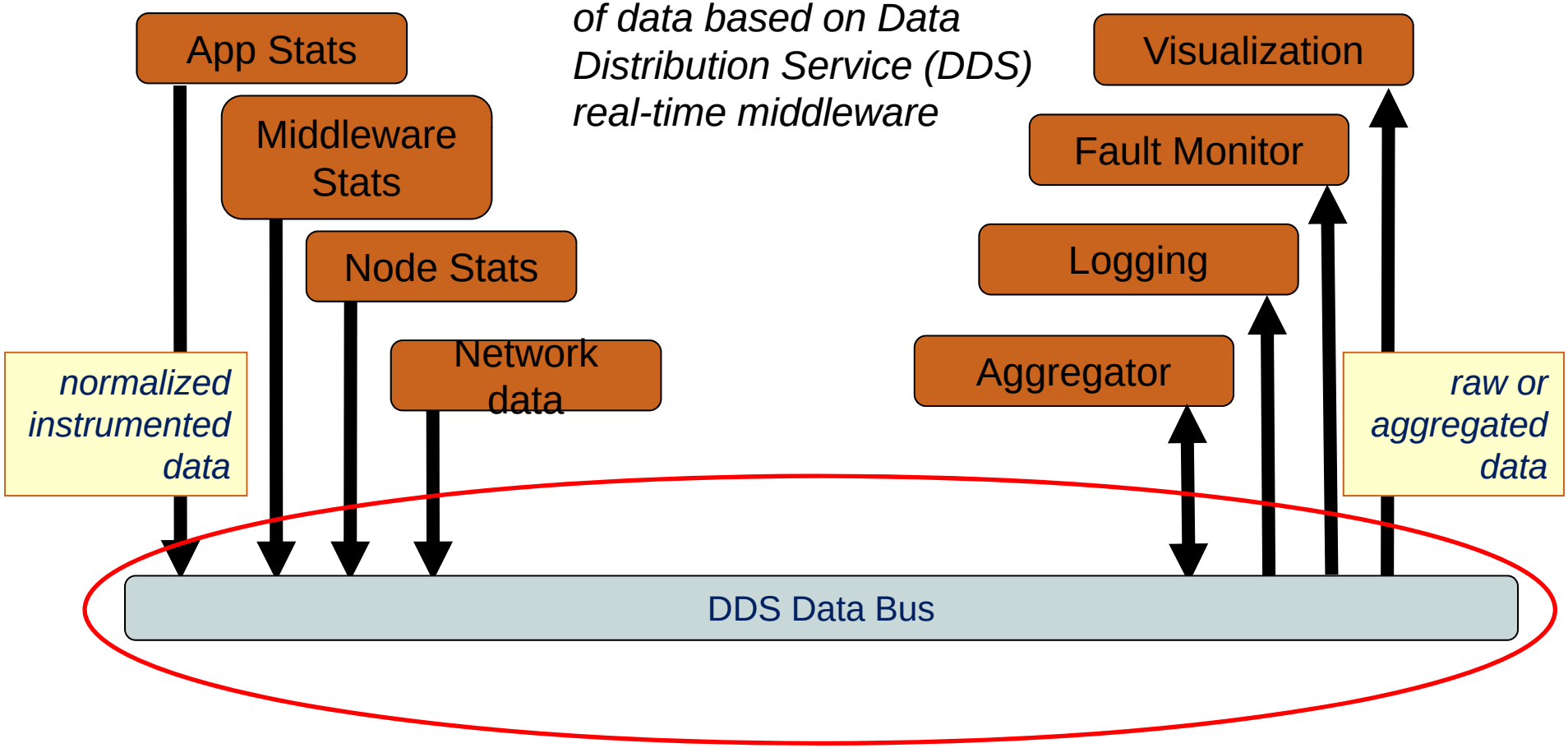
Open, Extensible Framework

2. Aggregation, analysis, and storage



Open, Extensible Framework

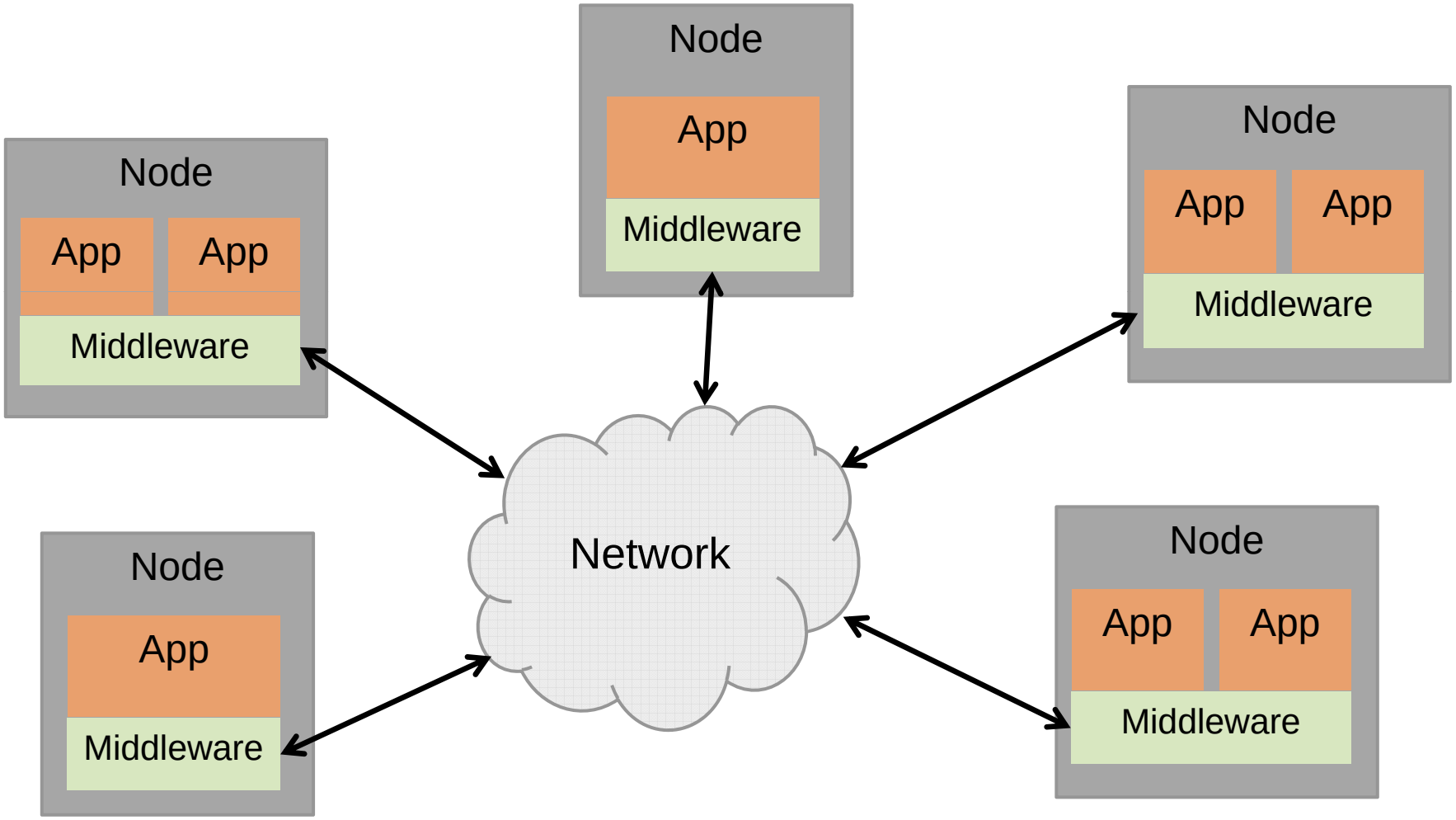
3. Data Centric Design
   Collection and distribution of data based on Data Distribution Service (DDS) real-time middleware



Instrumentation

- Ideal solution
  - Capture everything a computer does from code execution to packets sent and received
  - Do that with zero overhead added to the processor
- Practical solution
  - Select key information to instrument
  - Minimize overhead – especially in real-time threads
  - Continuous instrumentation
  - Build an extensible framework

Distributed System Components



Instrumentation

- Want to instrument all components
  - Application
  - Middleware
  - Processor
  - Network
- Need to minimize intrusion
  - For middleware, processor, and network data use lower priority process to collect data
  - For real-time applications, minimize instrumentation overhead in real-time thread.
  - Run instrumentation continuously as part of the system – eliminates the heisenbug problem

Node Instrumentation

- The processor itself can give a good indication of a problem
- Often first indicator of a scaling bug problem
- Key attributes to monitor
  - CPU usage
  - Context switches and interrupts
  - Memory
  - Network usage

Application Instrumentation

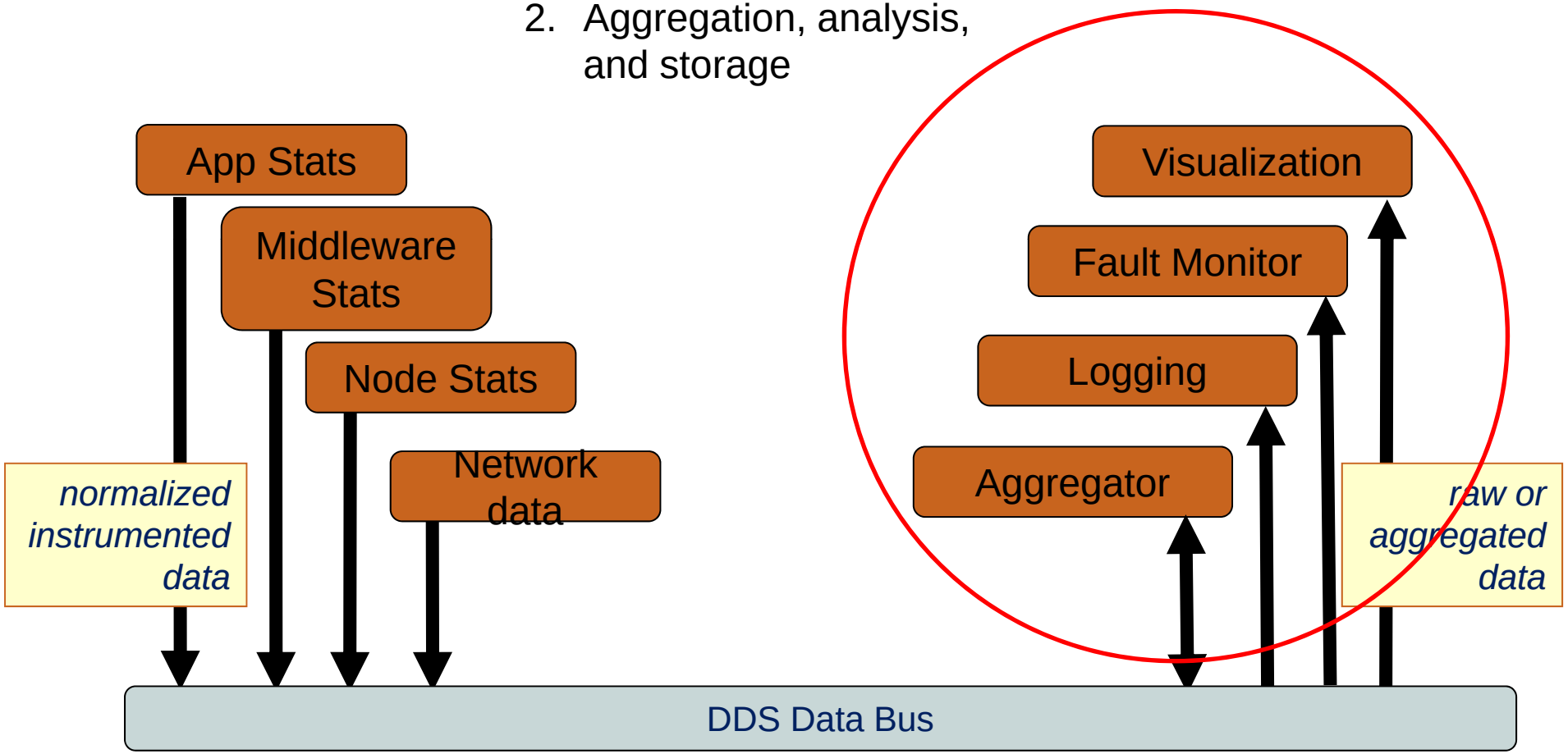
- Need to balance between real-time requirements and amount of data collected
- Select key variables to instrument
- Have a low overhead instrumentation in the real-time process
  - Simple copy into a buffer somewhere else
  - Have a lower priority thread do processing
  - Processing may include aggregation or checks on the process itself to minimize the amount of data that distributed

Middleware Instrumentation

- Middleware often contains a wealth of useful information
- Useful to find distributed bugs
- Statistics that can be gathered
  - Throughput – samples sent and received
  - Reliability protocol behavior
  - Discovery/connection statistics
  - Problems such as
    - samples dropped
    - failures to meet deadlines
    - Type or QoS mismatches

Part 2 of Framework

2. Aggregation, analysis, and storage



Using the Data to Monitor and Debug

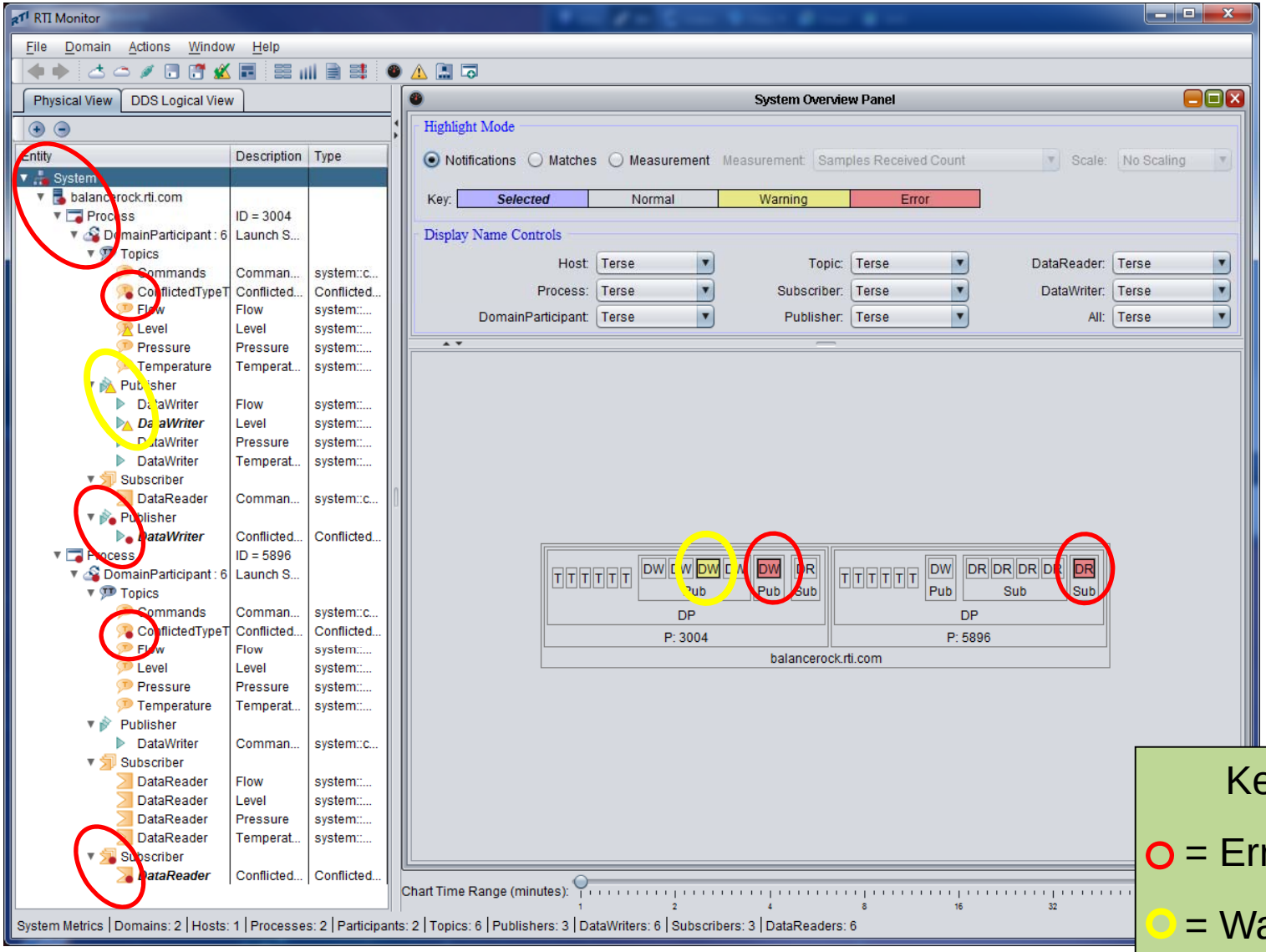
- Logging
  - Store the data for post analysis if something has gone wrong
  - Useful to find mandelbugs that are hard to recreate
  - Can replay recorded data to debug or test
- Aggregation and analysis
  - Aggregate data to give a system view
  - Automate analysis to look for anomalies or errors
- Visualization
  - Present high-level, system view information to the user
  - Enable drill-down to more detail
  - Combine views of different instrumentation data

Visualization Example

- Visualization Goals
  - Catch problems early
  - Debug issues
  - Tune the system for optimal performance

- Example screen shots from RTI Monitoring Service

System Overview of Notifications



Key

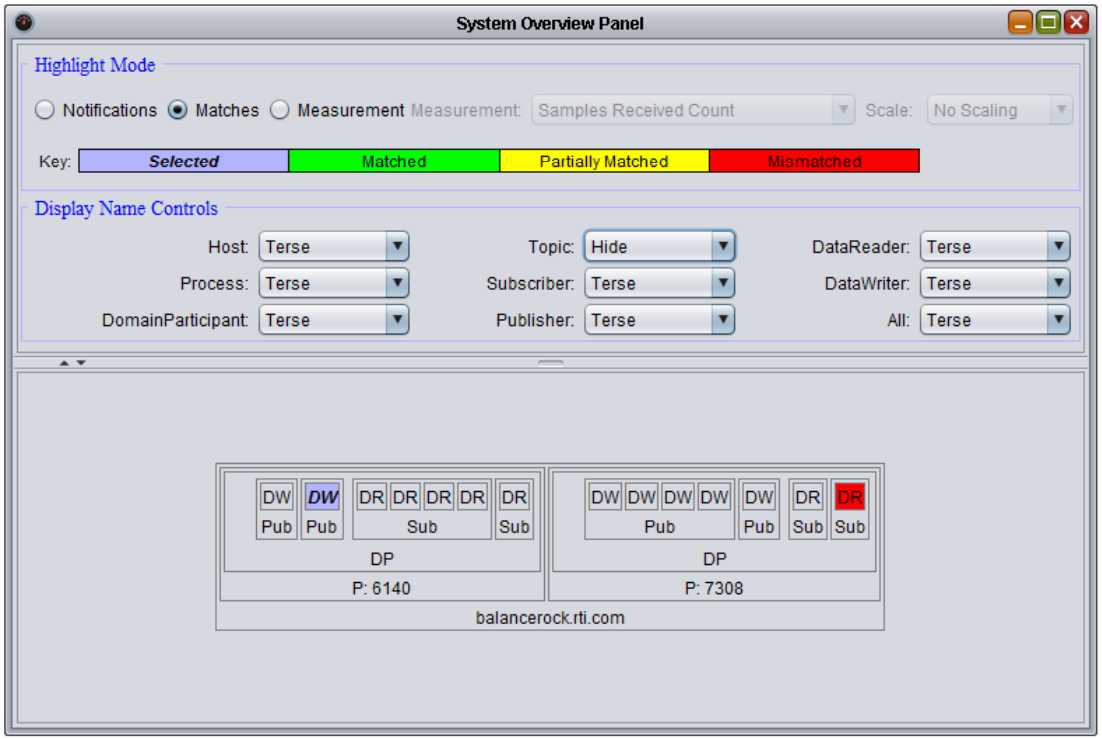
= Error

= Warning

Notifications Panel

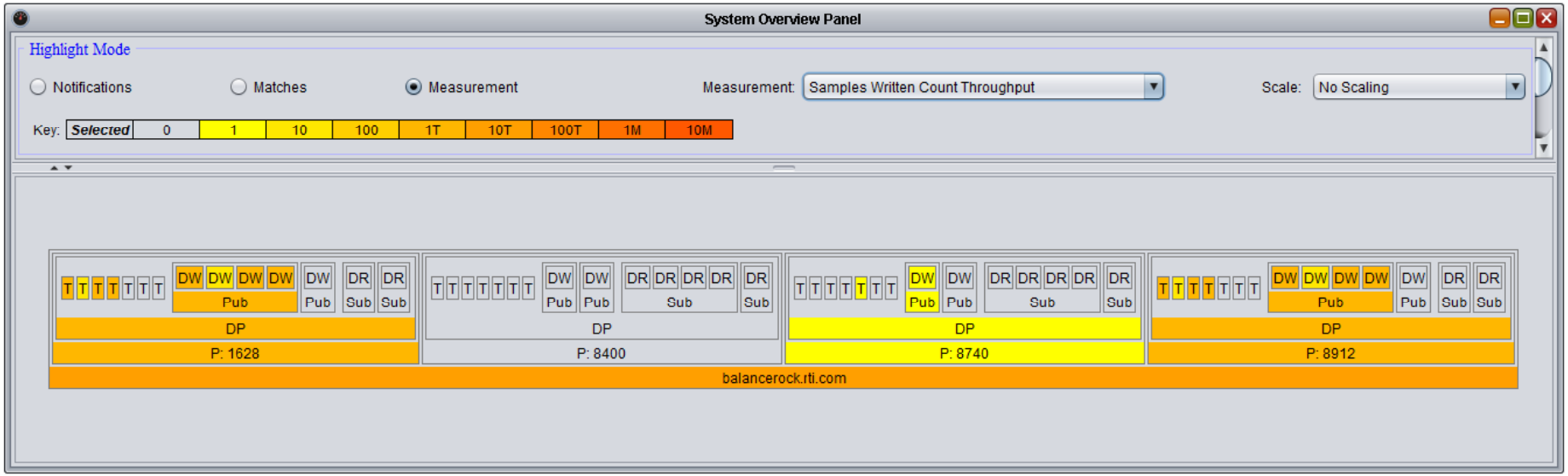
- Select an object from the overview to get more details
- This panel shows the historical notifications for a single object.
- In this case, the data consumer has been getting warnings about missed deadlines.

Discovery (Connection Status)



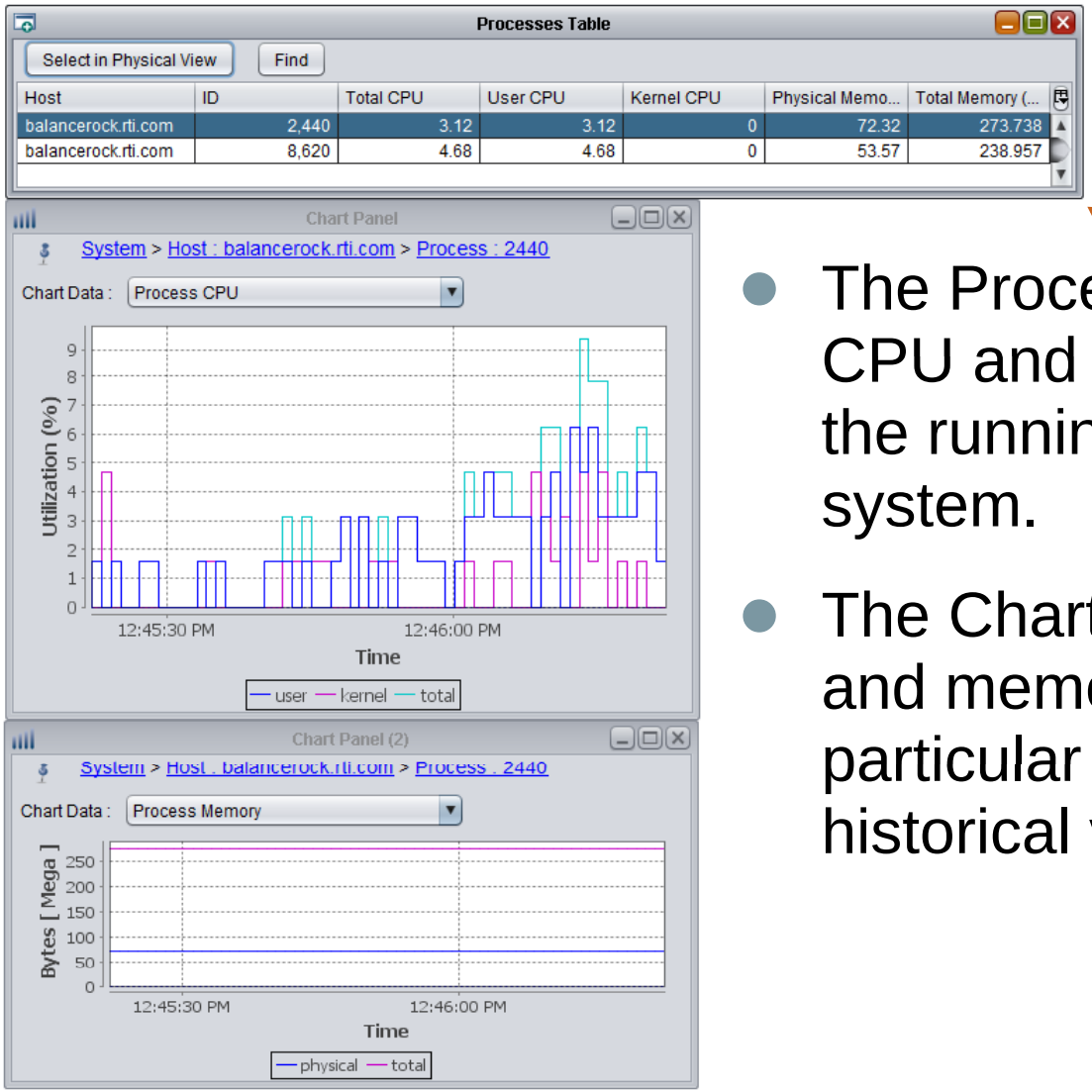
- The System Overview panel displays discovery matches, partial matches, and mismatches.
- Color coded for easy identification of a problem

System Overview Panel: Performance Metrics



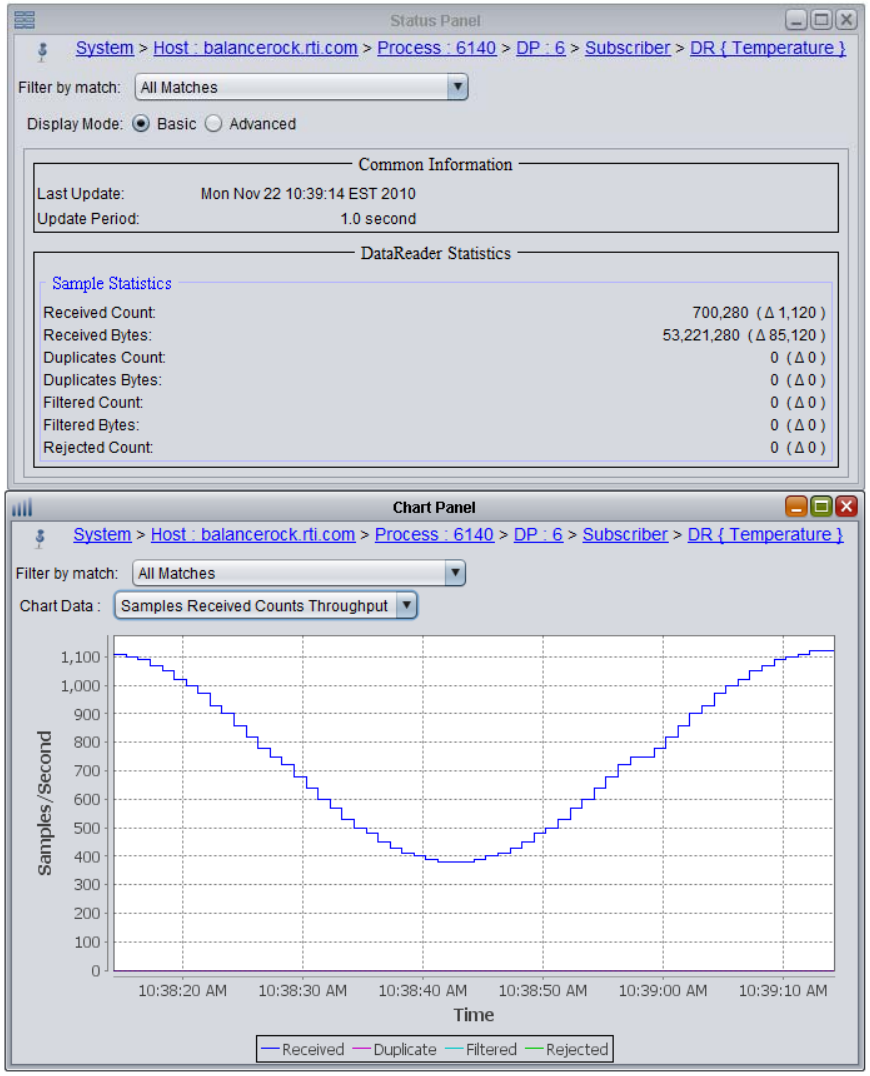
- This System Overview Panel shows throughput (units of samples/counts).
- This view is useful to find ‘hot spots’ in a system for a given metric.
- Metrics include throughput, reliability metadata, and CPU.

Process Information



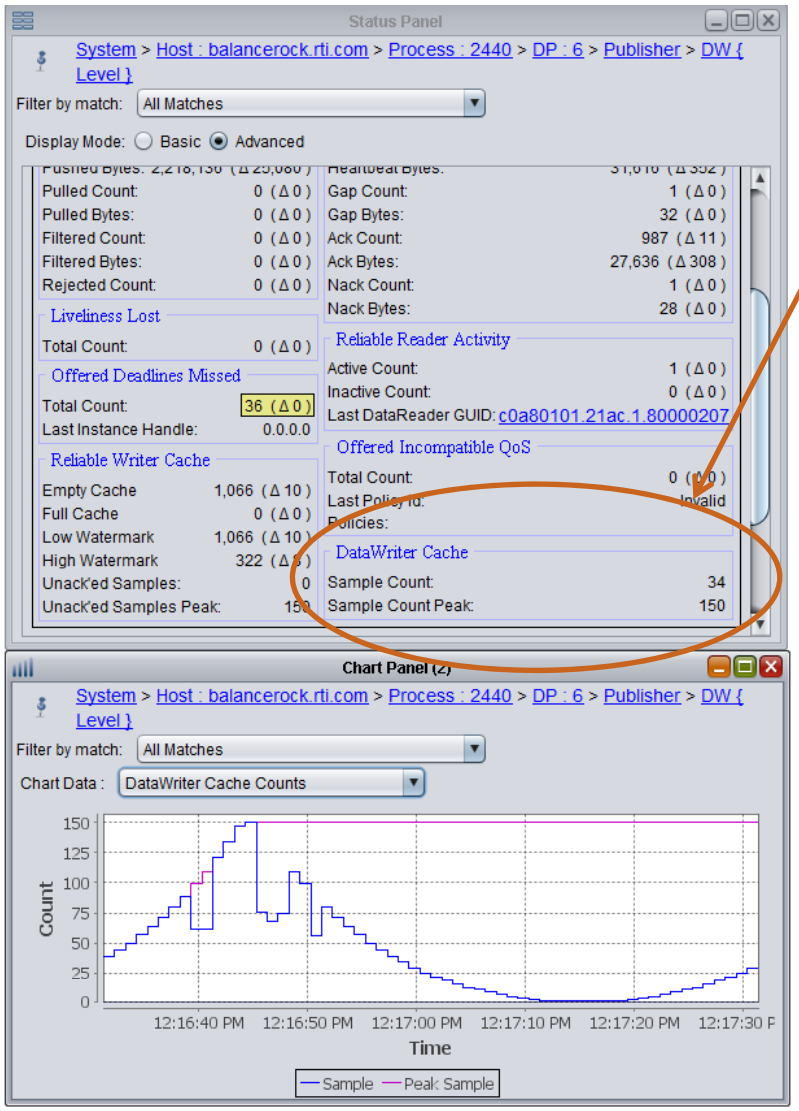
- The Process Table displays the CPU and memory usage by all of the running processes in the system.
- The Chart Panel displays the CPU and memory consumption by a particular process including historical values.

Throughput per Application



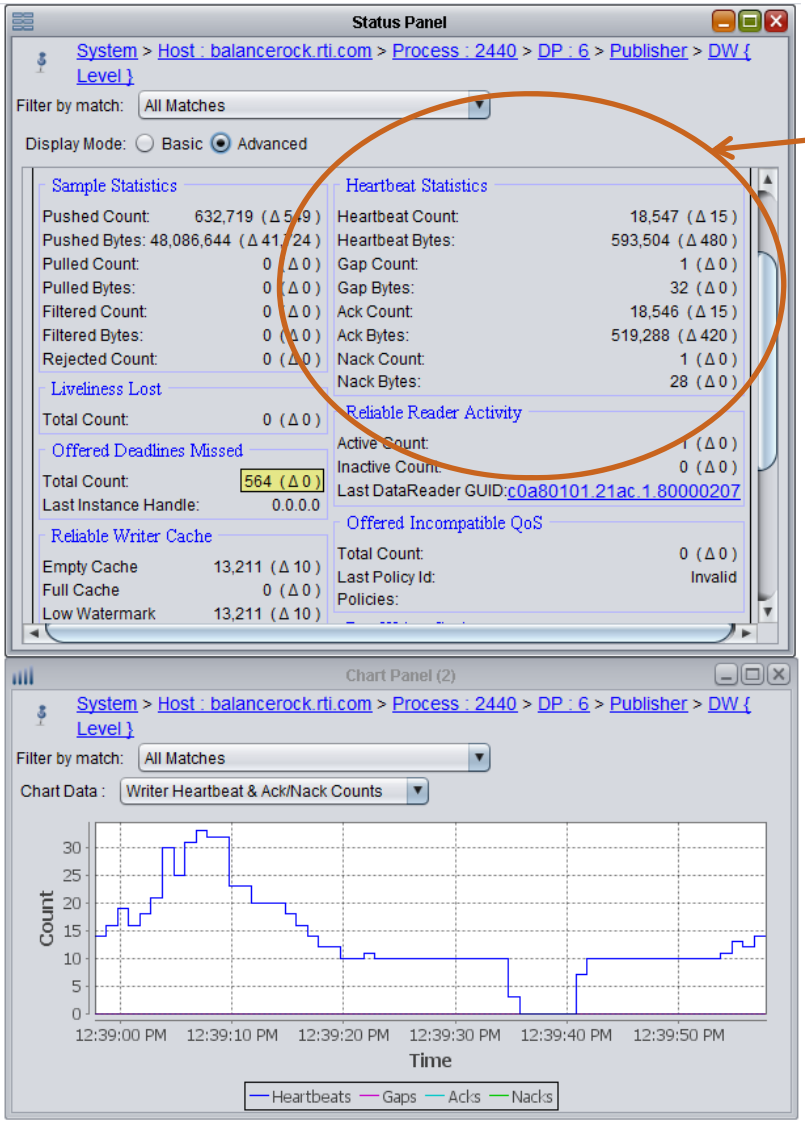
- The Status Panel has a textual display of the throughput of an application in both sample counts as well as bytes.
- The Chart Panel can chart the throughput over time. Throughput can be charted as sample counts or bytes.

Reliability Cache



- The Status Panel shows the DataWriter & DataReader cache current counts.
- The cache counts can also be displayed in a chart showing the historical data for this value.
- This DataWriter's cache grew significantly during this period.

Reliability Metatraffic (Heartbeats, Acks, Nacks, Gaps)

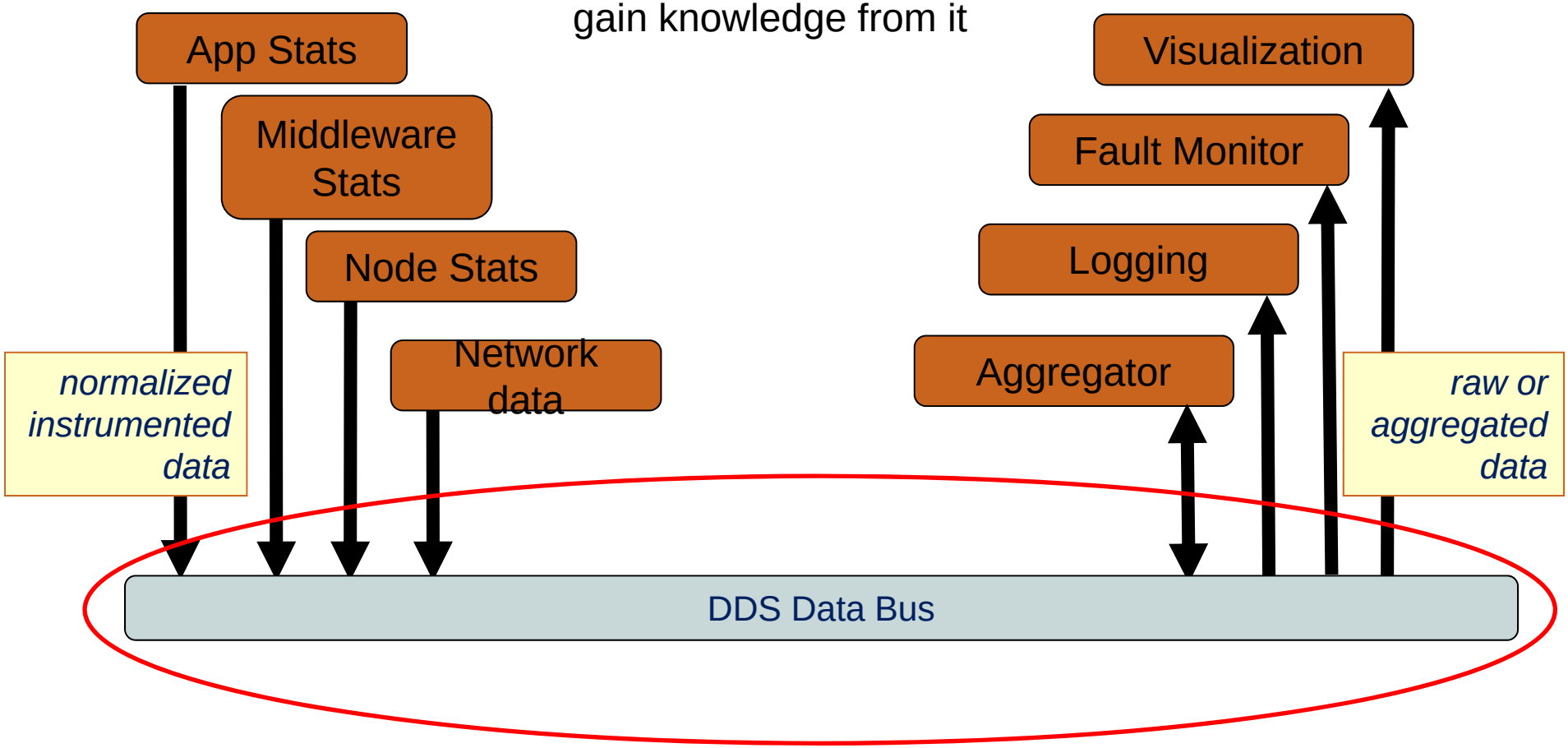


- The Status Panel shows reliability metatraffic (heartbeats, acks, nacks, and gaps) for both DataWriters (shown) and DataReaders.

- The reliability metatraffic can also be displayed in a chart to gain historical context.

Data, Data Everywhere …

3. Need to collect, normalize, and distribute the data to gain knowledge from it



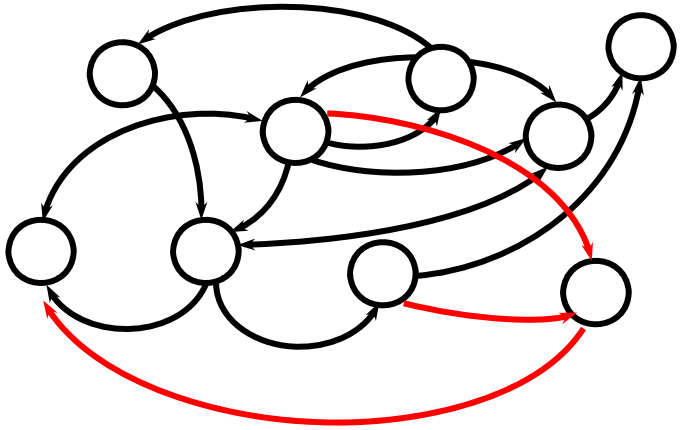
Collect, Normalize, Distribute

- Key to creating an extensible framework is data communication
- Needs
  - Collect data from a variety of different sources
  - Deliver in real-time to one or more analysis tools
  - Minimize impact of instrumentation data on network capacity
  - Enable new applications, tools, and data

Solution: Data-Centric Framework

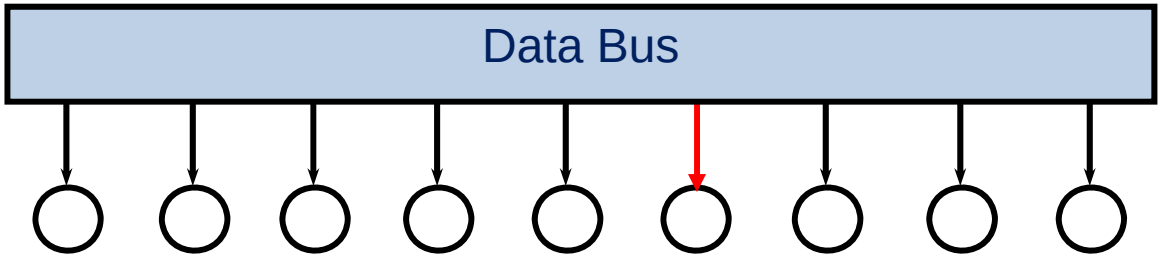
Connection-oriented

- Hard-wired
- Multi-hop
- Brittle
- Hard to evolve



Data-centric

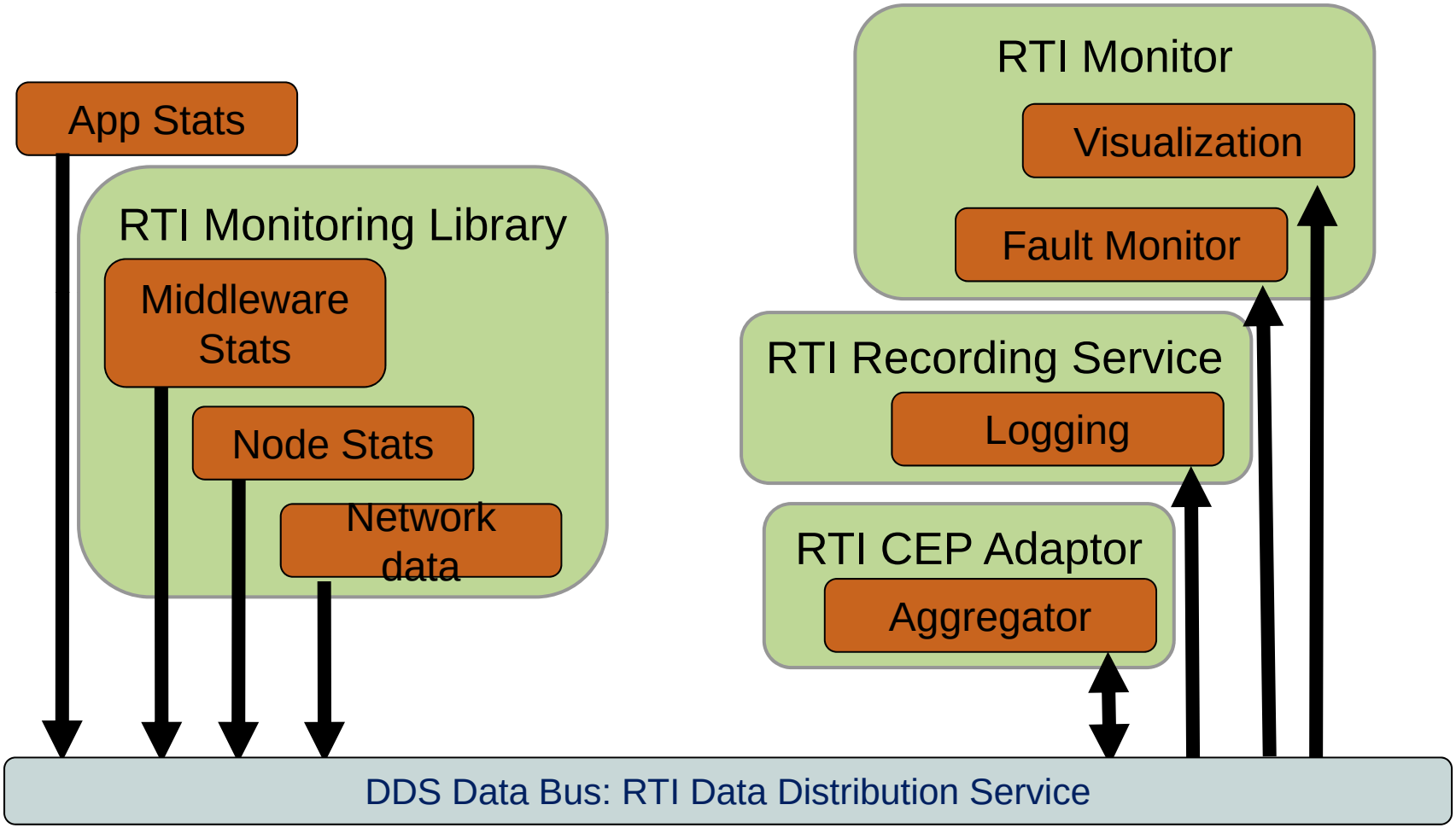
- Loosely coupled
- Peer-to-peer
- Scalable
- Evolvable



Data-centric framework

- Define data model for instrumented data ahead of time
- Creates plug-and-play architecture for different instrumentation components
- Real-time middleware enables distribution of instrumentation data within minimum system impact
- DDS middleware
  - Sends data description only once, minimizing packet size of data samples
  - Uses peer-to-peer delivery, so no additional service required
  - Can use multicast for efficient network use

Real-Time Innovations Technology



Acronyms

- CEP – Complex Event Processing
- DDS – Data Distribution Service
- QoS – Quality of Service
- RTI – Real-Time Innovations, Inc.