
Agilent Technologies

# Goal

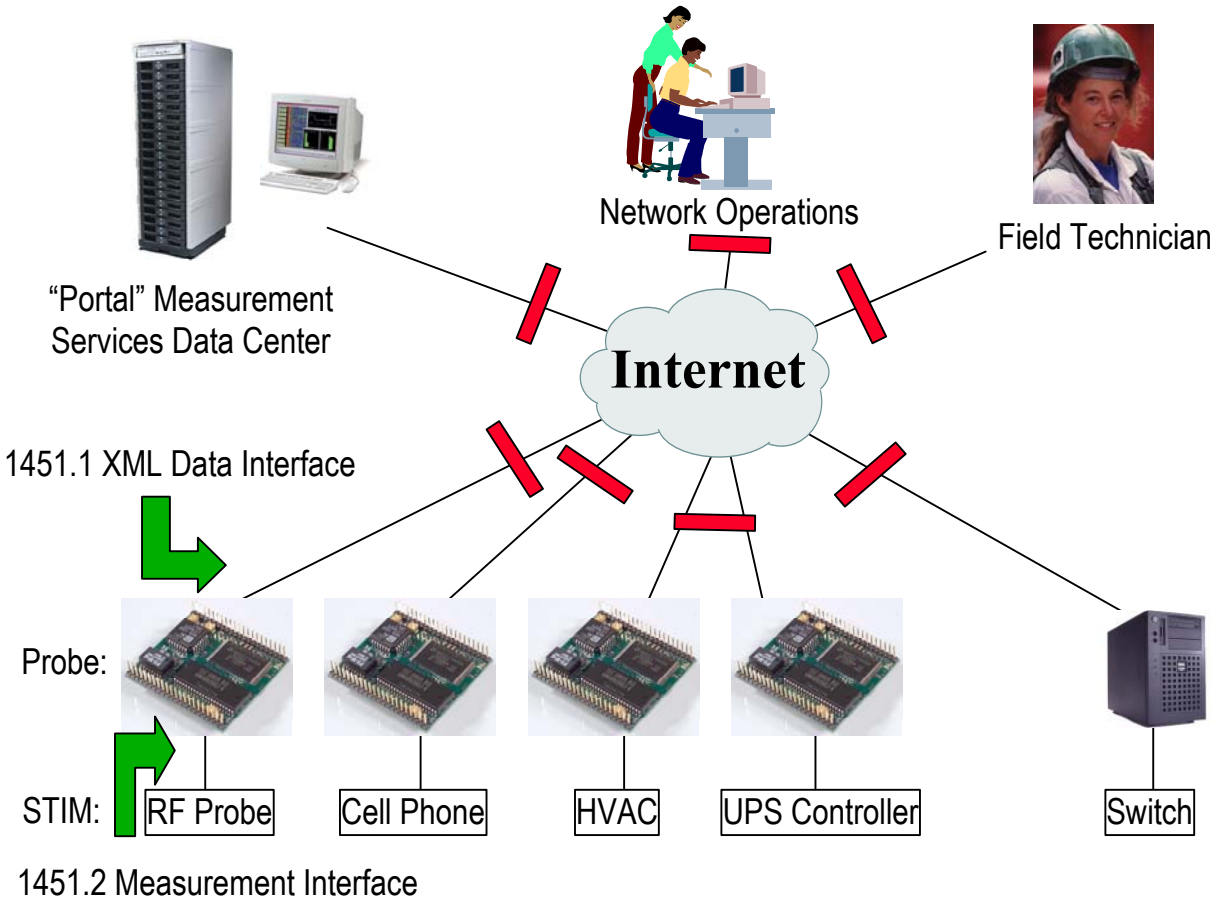
- Understanding issues in distributed measurement and control.
- Familiarity with JDDAC
  - Various API (Application Programming Interfaces)
  - Key Components.
  - Data Model.
- Ability to write JDDAC components
- Ability to assemble and configure components to build a JDDAC probe.

# Agenda

- Concepts
  - Background information on measurement and control
  - Introduction to IEEE 1451
- JDDAC Introduction
  - API
  - Components
  - Using JDDAC
- Hands-on Lab with JDDAC
  - Interface to sensors
  - Create processing block
  - Assemble into application




# Distributed Measurement and Control







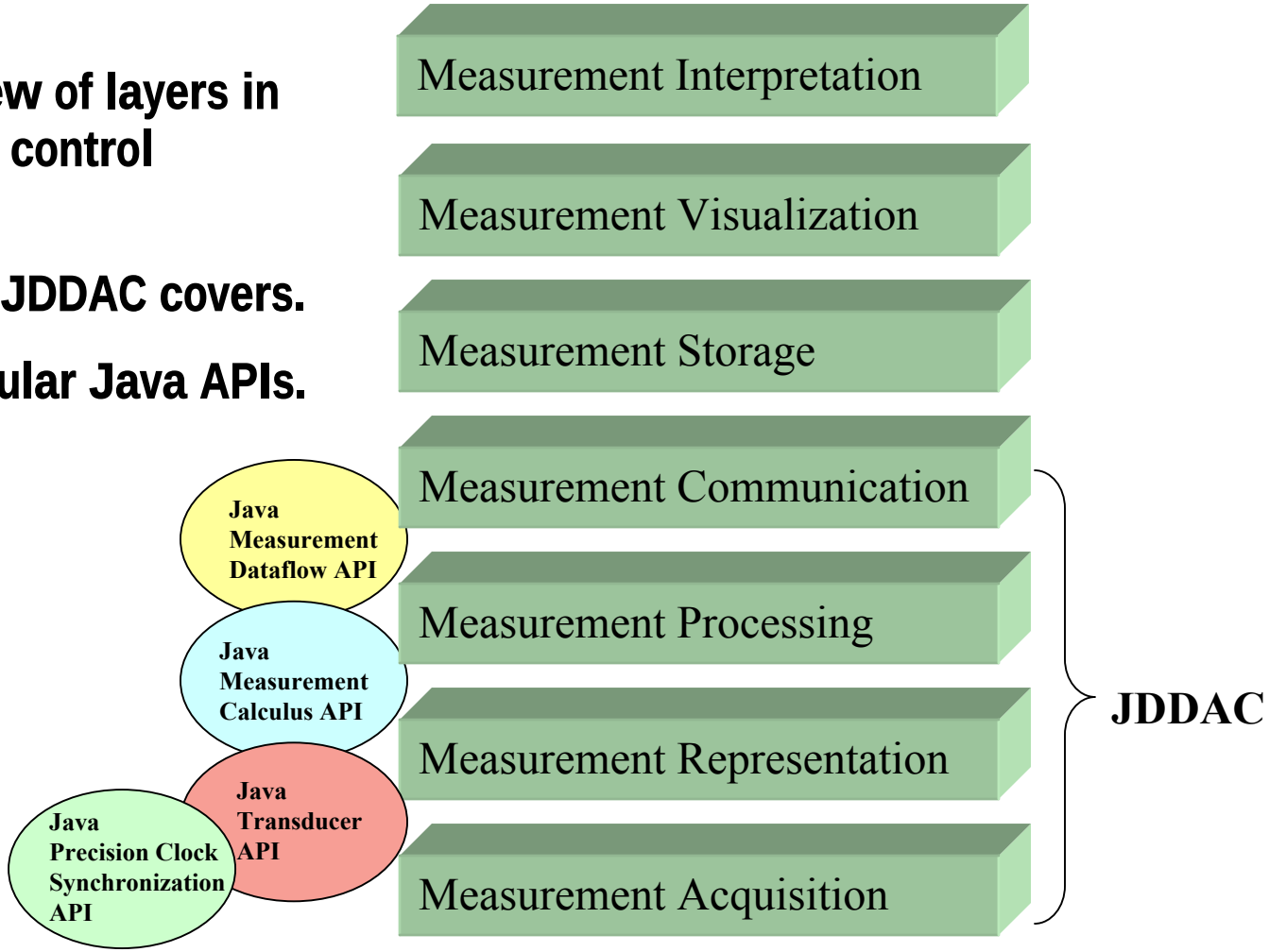

Agilent Technologies

# Distributed Measurement and Control

- JDDAC – Java Distributed Data Acquisition and Control
- How is measurement/control different than ‘regular’ programming
  - Embedded platform – less capable, fewer resources.
  - Real world moves according to its own schedule
  - Actuation issues
  - Synchronization issues
  - Data-centric paradigm.
  - Representation of real world – uncertainties in measurement and representation
  - Units do matter – need more than just values.

# JDDAC in a DMC environment.

- **Data-centric view of layers in measurement & control systems.**
- **Indicates areas JDDAC covers.**
- **Supplement regular Java APIs.**








Agilent Technologies

# Agenda

- Concepts
  - Background information on measurement and control
  - Introduction to IEEE 1451
- JDDAC Introduction
  - API
  - Components
  - Using JDDAC
- Hands-on Lab with JDDAC
  - Interface to sensors
  - Create processing block
  - Assemble into application



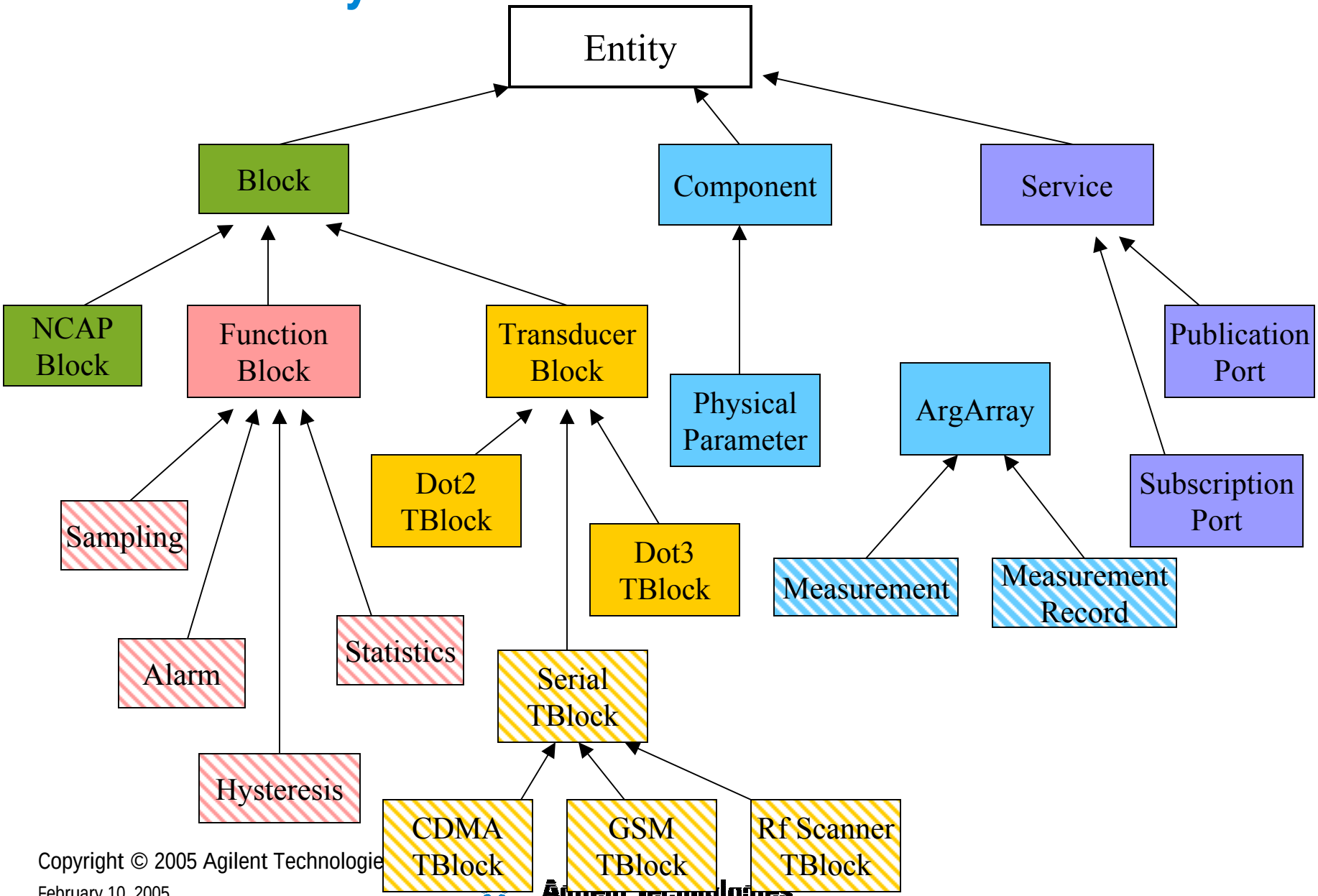

# IEEE 1451

- Family of standards for connecting smart transducers to networks.
- Enable “Plug and Play” Transducers and Distributed Measurement and Control Applications.
- Sponsored by NIST – Automation in Manufacturing Initiative
- Began in 1994
- IEEE 1451.2 Standard 1997 – Transducer Interface and electronic datasheet
- IEEE 1451.1 Standard 1999 – Programming Model & Communications.
- IEEE 1451.4 Standard 2003 – Analog Transducers & TEDS
- Ongoing activity: IEEE 1451.0
- Related: IEEE 1588 Standard 2002 – Time Synchronization

# IEEE 1451.1 Key Concepts

- Programming model for Distributed Measurement Applications.
- Object Oriented / Modular / Reconfigurable Architecture
  - Defines a Dataflow processing model. Data flows from block to block. Execution is asynchronous (for now)
  - Block Model – application logic resides in blocks.
- Network Independence
- Communication Model
- Measurement Data Model
- Configuration and Deployment Issues

# Class Hierarchy

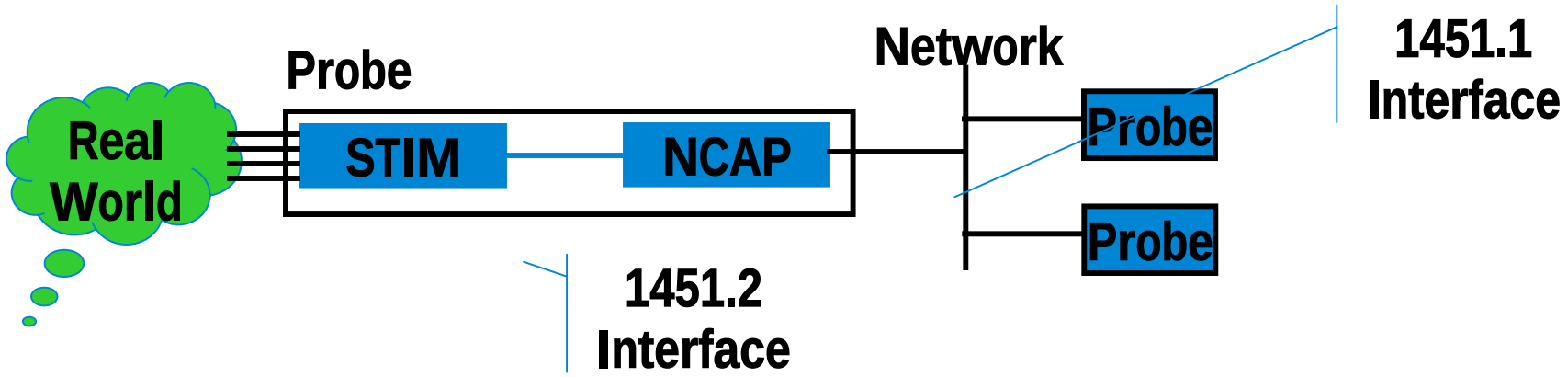


# IEEE 1451.2 Standard

## A Smart Transducer Interface for Sensors and Actuators - Transducer to Microprocessor Communications Protocols and Transducer Electronic Data Sheet (TEDS) Formats

Approved 1997, undergoing 5-year review

- How to interface a transducer to a communications network? Ethernet, CAN, LonWorks, Fieldbus, ...
- STIM - Smart Transducer Interface Module
- NCAP – Network Capable Applications Processor





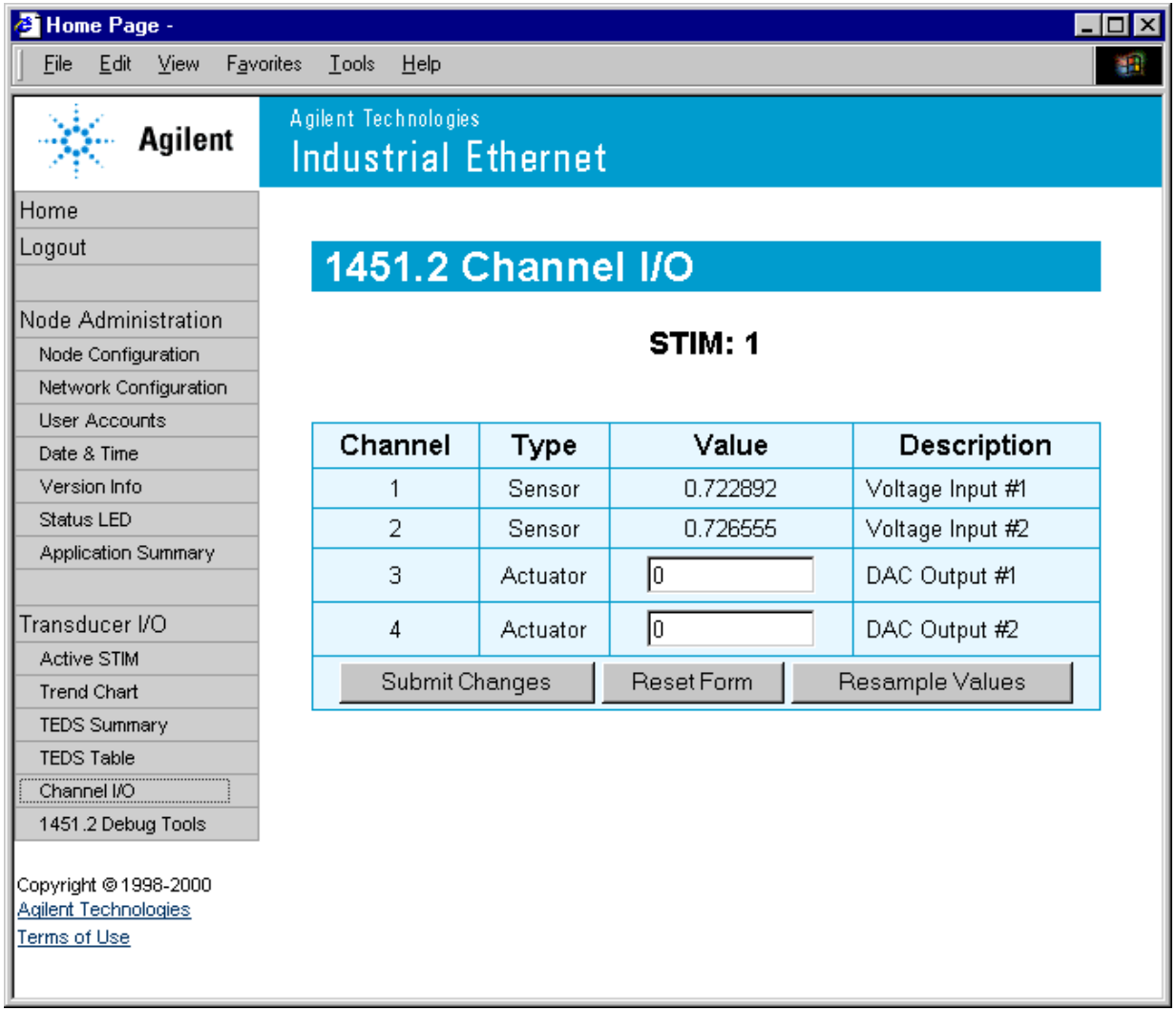



Agilent Technologies

# IEEE 1451.2
# Transducer Electronic Data Sheet

- Enables “plug and play” transducers
- TEDS resides on the STIM
- Information Provided
  - Number and configuration of transducer channels (sensors, actuators, buffered, event, ...)
  - Data Model (float, double, integer, unsigned byte, ...)
  - Correction information to/from NCAP and STIM representations
  - SI Units for each channel
  - Transducer accuracy, limits, measurement timing

# IEEE 1451.2 Measurement Abstraction

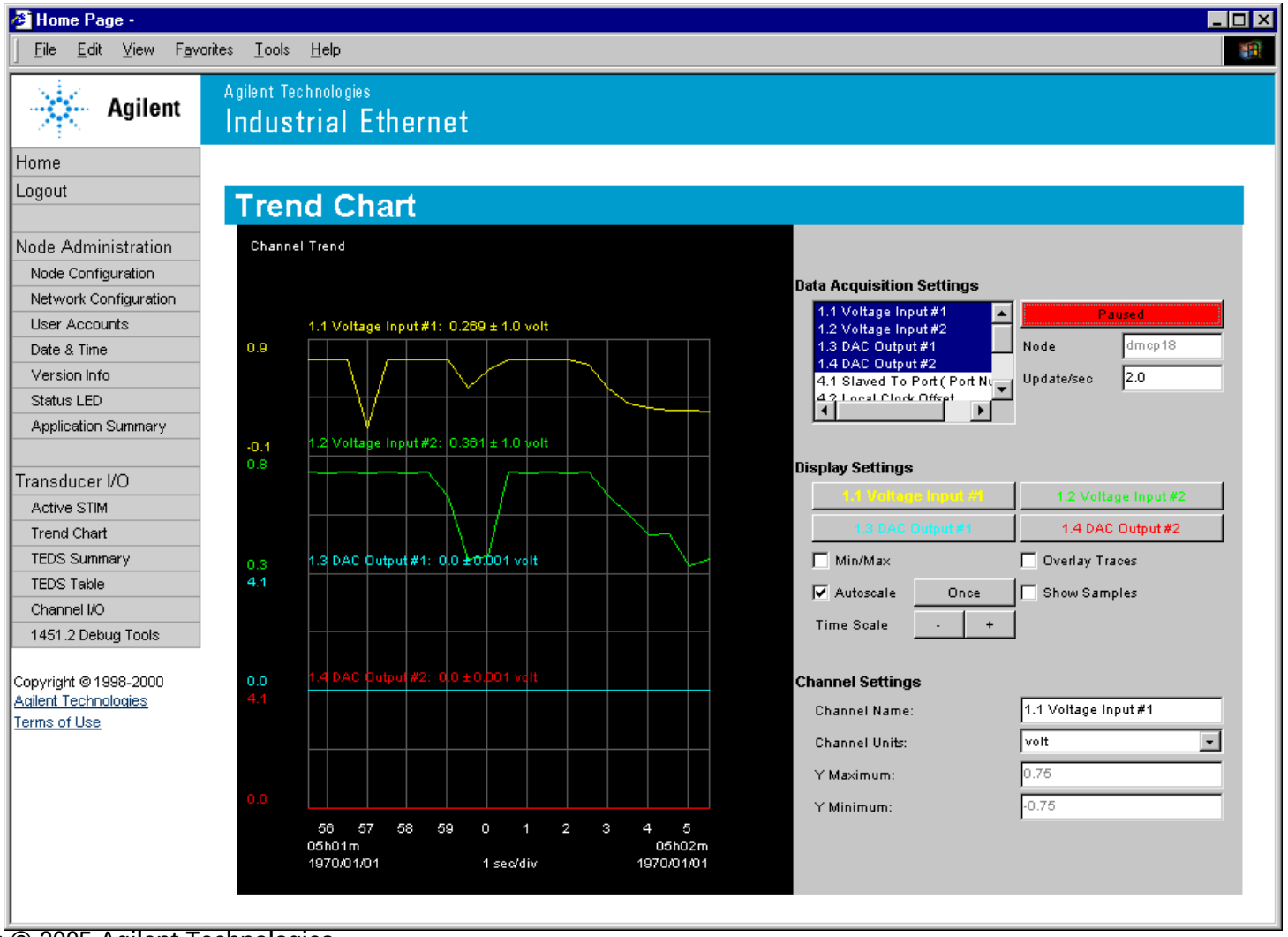







Agilent Technologies

# Enabling Generic Measurement Applications that use Measurement Metadata








Agilent Technologies

# Agenda

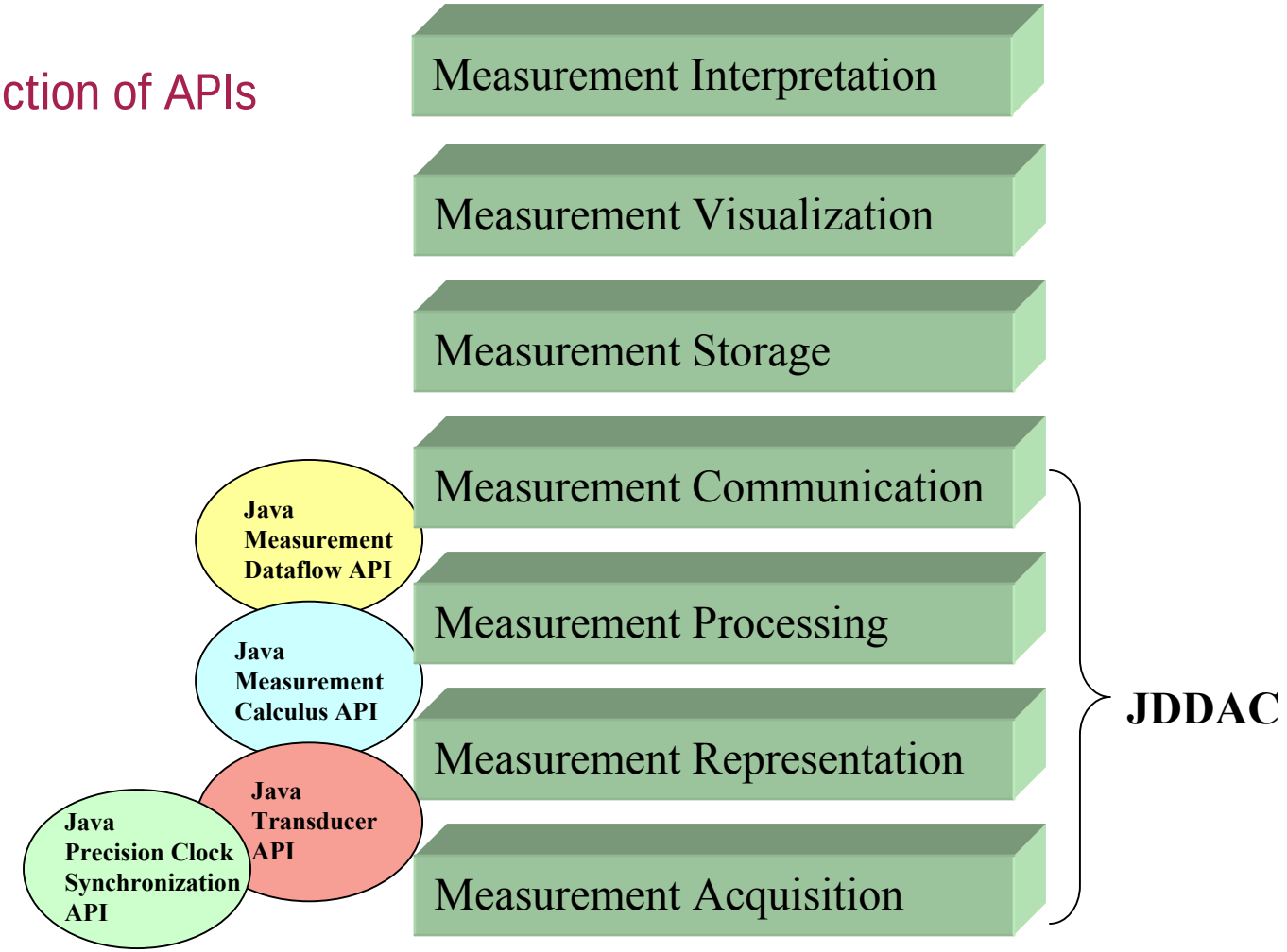
- Concepts
  - Background information on measurement and control
  - Introduction to IEEE 1451
- JDDAC Introduction
  - API
  - Components
  - Using JDDAC
- Hands-on Lab with JDDAC
  - Interface to sensors
  - Create processing block
  - Assemble into application




# Introduction to JDDAC

## Composed of a collection of APIs

- JMDI
- JMCI
- JTI
- JPSCI








Agilent Technologies

# Java Measurement Dataflow Interface (JMDI)

- ‘Mapping’ of IEEE 1451.1 into Java.
- Defines
  - Block architecture.
  - Communication mechanism.
  - Lifecycle.
  - Naming.
  - Data Model.

# Java Transducer Interface (JTI)

- Mapping' of IEEE 1451.2 into Java."
- Defines interface to transducers
- Defines Transducer Electronic Data Sheets (TEDS)

# Java Measurement Calculus Interface (JMCI)
# Java Precision Clock Synchronization Interface (JPCSI)

- JMCI
  - Derived in part from work on JSR-108.
  - Provides support for arithmetic and logical operations between measurements.
- JPCSI
  - Based on IEEE 1588
  - Interfaces to manage synchronized clocks.
  - Interfaces to execute time triggered actions.

# Platforms

- J2EE
  - Uses Struts, Struts Tiles, JDBC.
  - Measurement server.
- J2SE
  - Vanilla Java –
  - Network probes, devices attached to PC
- J2ME
  - MIDP, IMP.
  - Cell phone, PDA
- Differences between platforms
  - API differences
  - Floats (CLDC 1.0 and earlier)

# Agenda

- Concepts
  - Background information on measurement and control
  - Introduction to IEEE 1451
- JDDAC Introduction
  - API
  - Components
  - Using JDDAC
- Hands-on Lab with JDDAC
  - Interface to sensors
  - Create processing block
  - Assemble into application




# Function Blocks

- All processing logic in Function Block objects.
- Application consists of instances of Function Blocks connected via Communication Ports.
- Specialized versions of Function Blocks to interface to transducers (T-Blocks) and act as NCAP manager (NCAP Blocks).

```
public abstract class FunctionBlock extends Entity
                                    implements SubscriberCallback
```

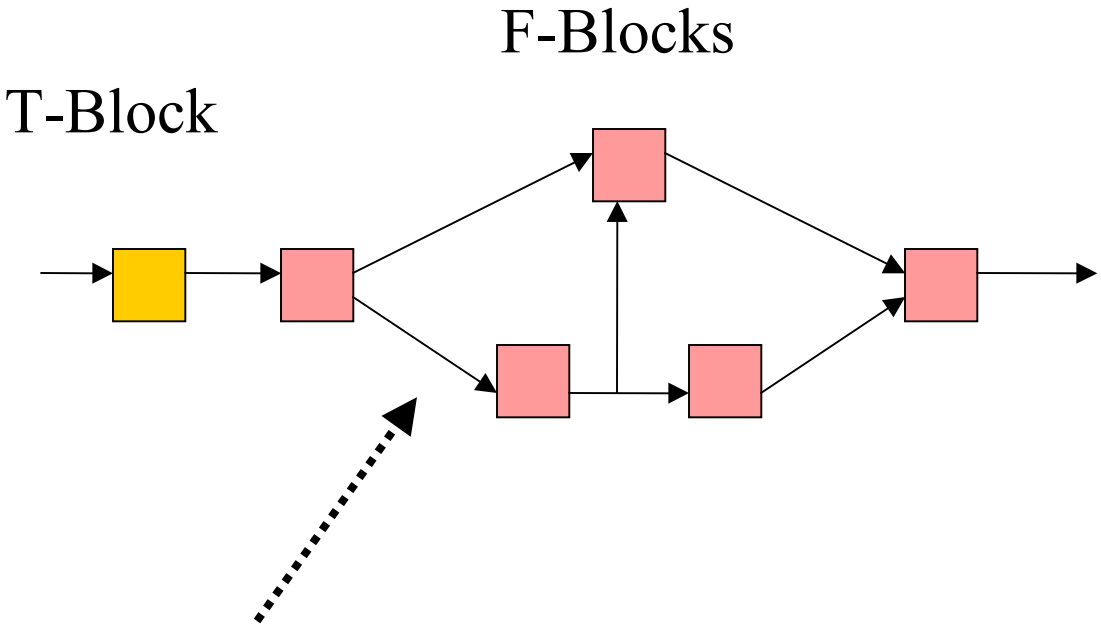



# Communications

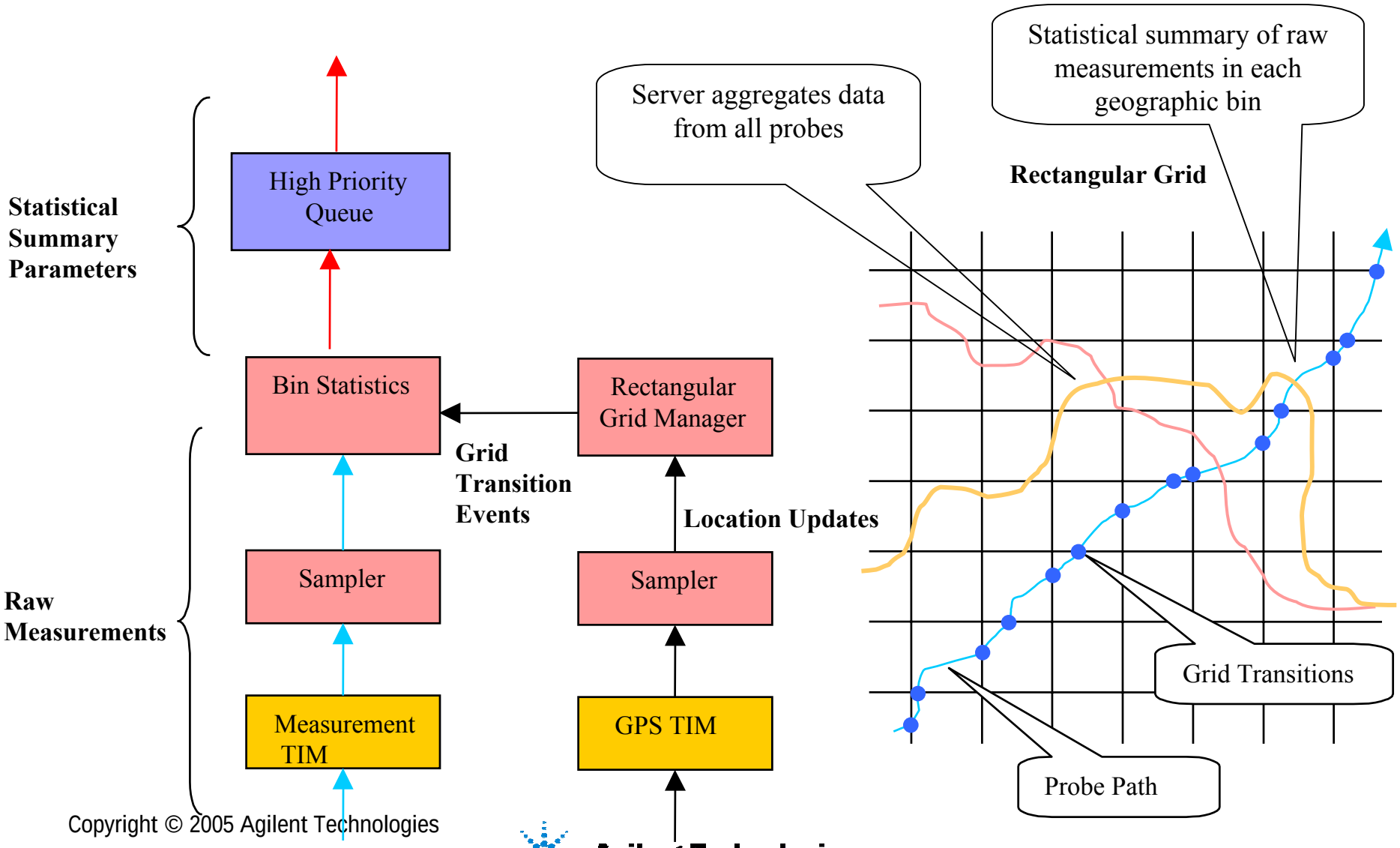
- Occurs via Port objects.
- Client / Server style
  - Always One-to-One.
  - Bi-directional information flow.
  - Optional one-way.
  - Asynchronous.
- Publish / Subscribe style
  - One-to-Many, may be One-to-One or Many-to-One.
  - Unidirectional information flow.
  - Non-acknowledged.
- Fully Symmetric Communication
  - NCAPs and higher-level server can be Client, Server, Publisher, and Subscriber simultaneously.

# Anatomy of JDDAC Application

- A JDDAC application contains multiple instances of different kinds of F-Blocks wired together to form a measurement processing chain.



# Statistical Binning Application Example

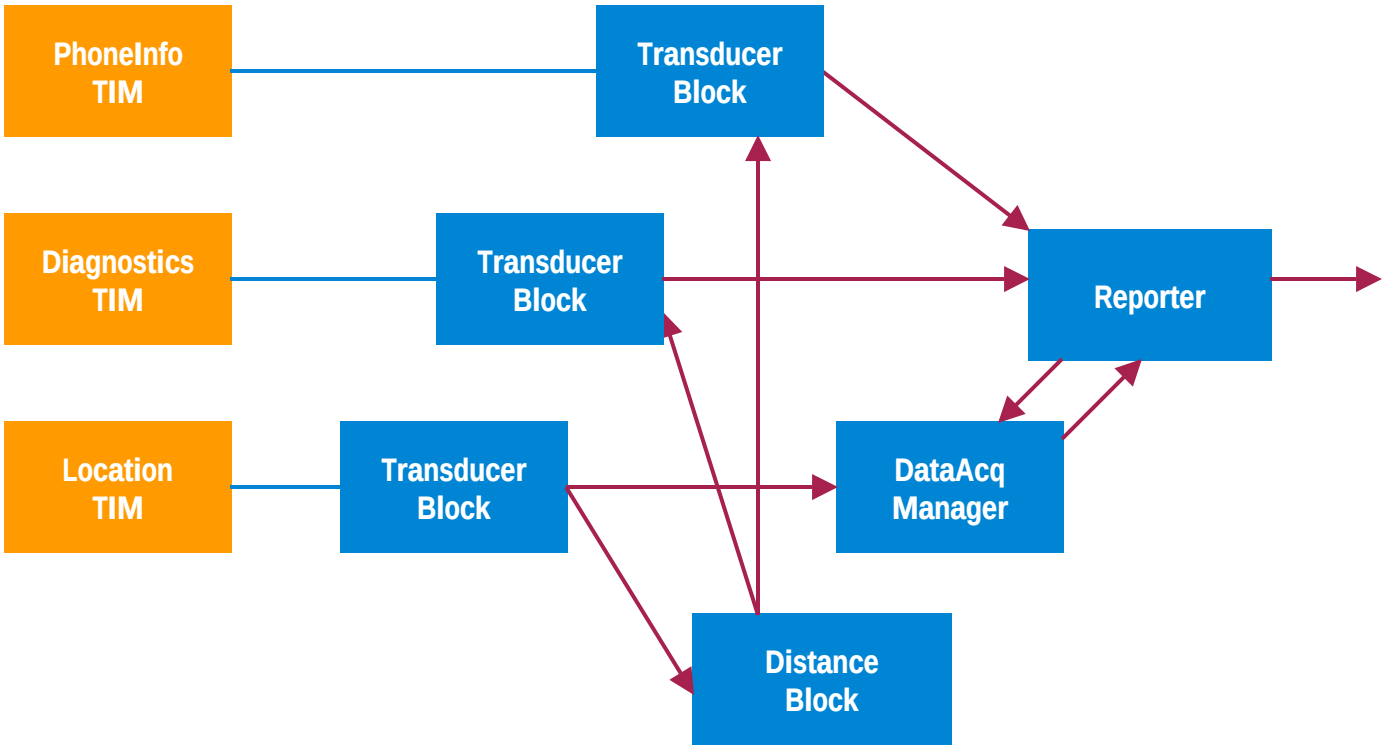







Agilent Technologies

# Probe Based Example

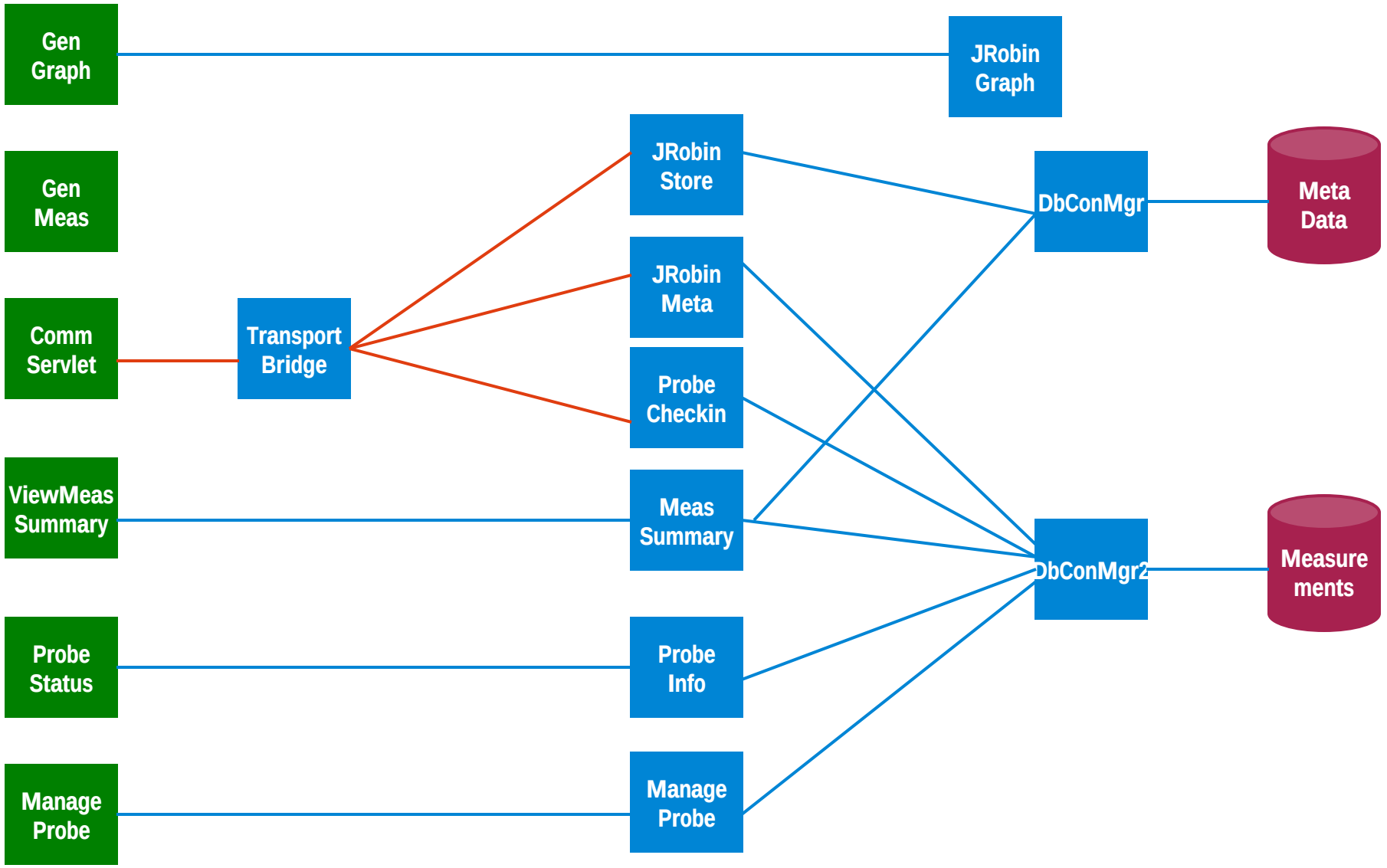







Agilent Technologies

# Server Example








Agilent Technologies

# JDDAC Common Data Model

- Weakly typed system – easier to accommodate changes in dynamic system.
- Most checks performed at run-time.
- Derived from IEEE 1451.1 Argument class.
- Almost all data items are Arguments – Integer, String, Unit, TimeRepresentation.
- ArgumentArray holds Arguments.
  - Name/value pairs.
- Mapped ArgumentArray to ArgArray and java.util.Vector classes.

# JDDAC Schema for Measurements

- Records – contains multiple measurements
  - Measurement – contains data and metadata
    - Arg – Name/value pairs
      - T = type
      - L = length
      - N = Name

```
<argArray l="2" t="record" n="Value">
    <argArray l="4" t="measurement" n="totalMemory">
      <arg n="mid">OCGQtIZC2RGN4/dmN+Z8eA:1</arg>
      <arg t="long" n="value">2031616</arg>
      <arg t="timeRepresentation“ n="timestamp">1106342455.640000</arg>
      <arg t="id" n="id">

  jddac://00308*62188*48809*57693:Testola@jddac.labs.agilent.com/i/Memory Usage
  Stim?ch=totalMemory

      </arg>
    </argArray>

</argArray>
```

# Configuring Application

- Application driven by configuration file.
- XML based
  - Defines instances of Blocks
  - Provides attribute values for the Blocks.
  - Specify how Blocks are wired together via publications and subscriptions.
- Consists of two tags – ArgArray and Arg.
  - Can add new types without needing to update schema.
  - Use attributes to denote different instances of Arg.

# Example configuration file

```
<argArray>
   <arg n="className">net.java.jddac.jmdi.fblock.EchoBlock</arg>
   <arg n="instanceName">EchoBlock</arg>
   <argArray n="Block.subPorts">
      <argArray n="incomingData">
         <arg n="qual">storeMeasurement</arg>
      </argArray>
   </argArray>
   <argArray n="Block.pubPorts">
    <argArray n="result">
            <arg n="topic">storeMeasurement</arg>
    </argArray>
 </argArray>
</argArray>
```

# Writing JDDAC Components
# Using ArgArray class

- Generic container of name/value pairs.
- Commonly used by JDDAC components to communicate with each other.
- Contains accessor methods.
  - Type safe
  - Defaults
- Lock contents
- Merge two ArgArray
- Serialize to String (XML).

# Writing JDDAC Components
# Function Blocks

- Inherit from net.java.jddac.FunctionBlock
- Write methods containing application logic.
- Add configure() method to support configuration info from configuration file.
- Add perform() method to support generic invocations.
- Add notify() method to support publications.
- Available helper methods:
  - publish(int, ArgArray) – publishes an ArgArray on the specified port.

# Example Function Block

```
import net.java.jddac.jmdi.block.*;
public class SimpleBlock extends FunctionBlock
{
      public SimpleBlock ();
      public boolean configure(ArgArray configuration)
                                          throws Exception;
       public void notify(short publication_id,
                                 String publicationTopic,
                                 ArgArray payload);
       protected void publish(int portNum, ArgArray publication_contents)
                                          throws Exception;
}
```

# configure() method

```
public boolean configure(ArgArray configuration) throws Exception
{
        String name = (String) configuration.get(“name");
        SetName(name);                  // or some other internal method

        boolean rtn = super.configure(configuration);
        return rtn;
}
```

# perform() method

```
public ArgArray perform(
        String server_operation_id,
        ArgArray server_input_arguments)
        throws OpException
  {
        ArgArray server_output_arguments = new ArgArray();
        Integer opNum = (Integer)opLookup.get(server_operation_id);

        try {
                switch (opNum.intValue()) {
                        case OpIdSubscribeIndex : {
```

# notify() method

```
public void notify(short subscription_id,
                   String publicationTopic,
                   ArgArray payload) throws OpException
  {
        System.out.println("Echo received " + payload);
  }
```

# Writing JDDAC Components
# TIM (Transducer Interface Module)

- Responsible for interfacing with transducer
- Inherits from class net.java.jddac.jmdi.transducers.TIM.
- Specify TEDS (Transducer Electronic Data Sheet)..
- Fill in the readChannel() method.
    - Channel 0 reserved for ‘all channels’.

# Example TIM

```
public class MemoryUsage extends BasicTIM {
    private final Object teds[][] = {
    {
                MeasAttr.NAME,"MemoryUsage",
                MeasAttr.DESCRIPTION,"JVM Memory Monitor",
                MeasAttr.VERSION, "1.0",
                MeasAttr.MANUFACTURER, "jddac",
                MeasAttr.METADATA_ID,"BEzaX7tD2RGZ0+ymeNxcTQ:1"
    }};
    public MemoryUsage(){
        setChannelTeds(teds);
    }
    protected Measurement readChannel(int chanNum);
}
```

# Agenda

- Concepts
  - Background information on measurement and control
  - Introduction to IEEE 1451
- JDDAC Introduction
  - API
  - Components
  - Using JDDAC
- Hands-on Lab with JDDAC
  - Interface to sensors
  - Create processing block
  - Assemble into application




# Using JDDAC Installation

- Distribution provided as Zip file from java.net.
- Alternatively, fetch individual files via CVS.
- All needed files and libraries except Java SDK included in distribution.
  - Make sure J2SE 5.0 and Jakarta Ant is installed.
- Unzip into target directory

# Using JDDAC with Eclipse

- Unpack distribution into directory
- Create a new Java Application Eclipse project
- Set the workspace to the root directory of JDDAC distribution.
- To build, select ‘run ant’ on appropriate build.xml
- To run, select ‘Run as’ from Tools menu.

# JDDAC Directory Structure
## Top Level

- apps - Contains applications built from the JDDAC APIs. There is a subdirectory for each application.
- core - Contains core components used by all JDDAC APIs.
- docs - Contains documentation and javadocs.
- jmci - Contains the source code for Java Measurement Calculus Interface (JMCI)
- jmdi - Contains the source code for Java Measurement Dataflow Interface (JMDI)
- jti - Contains the source code for Java Transducer Interface (JTI)
- lib - Contains the compiled JDDAC libraries.
- lib-ext - Contains the third party library packages used by JDDAC.
- tools - Contains miscellaneous tools used in building JDDAC.

# JDDAC Directory Structure
# Within each top level source directory...

- common - code components that runs on all Java platforms.
- j2ee - code components that runs only on J2EE
- j2me - code components that runs only on J2ME
- j2se - code components that runs only on J2SE

  Contents

  - build.xml – ant build script
  - src – contains source code
  - bin – Class files
  - stage – staging area

# Using JDDAC
# Package Namespace

- All Java source files are in the namespace **net.java.jddac.*.**
- There are some classes which are used by all JDDAC APIs, including classes that represent the common data structures and classes that provide basic utilities such as logging. These classes are found in **net.java.jddac.common.***
- The classes which make up the Java Measurement Dataflow Interface (JMDI) are in **net.java.jddac.jmdi.*.**
- The classes which make up the Java Transducer Interface (JTI) are in **net.java.jddac.jti.*.**
- The classes which make up the Java Measurement Calculus Interface (JMCI) are in **net.java.jddac.jmci.*.**
- All three of these packges have subpackages below them.

# Building Application

- Use Jakarta Ant
  - ‘Make’ for Java.
- Automatically invokes Java compiler and other tools.
- Ant build file in each directory.
  - Can build in overall directory.
  - Can also build specific application
- Procedure
  1. Build the core components
  2. Place JAR files in lib directory
  3. compile the application source files
  4. Package together into JAR file with library JAR files.

# Tests

- Use JUnit as a regression testing framework.
- Integrated into most Java IDE's.
- Usually tests individual classes.
- See http://www.junit.org for additional details.
- JUnit directory located with each source code directory.

# Agenda

- Concepts
  - Background information on measurement and control
  - Introduction to IEEE 1451
- JDDAC Introduction
  - API
  - Components
  - Using JDDAC
- Hands-on Lab with JDDAC
  - Interface to sensors
  - Create processing block
  - Assemble into application



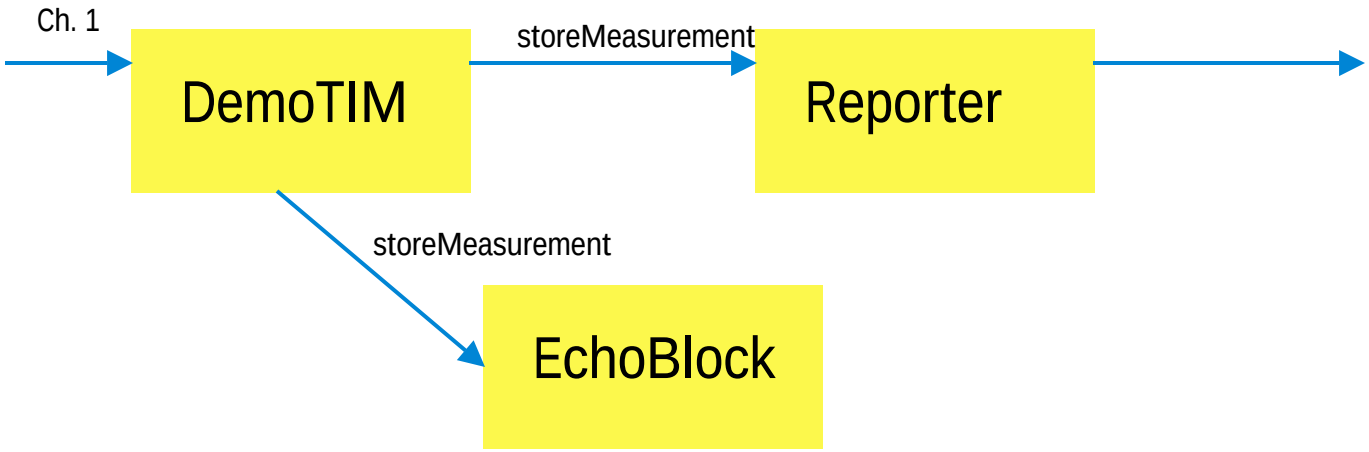

# Hands-on JDDAC Lab
# Overview

- Build a console application (J2SE based probe).
- Add a new data source to measurement system.
- Learn how TIMs and Blocks are written.
- Add TIMs and processing block (Function Block) to JDDAC application.
- Go thru a complete build cycle.
- Use JDDAC server.

# Part 1
# Build a working probe

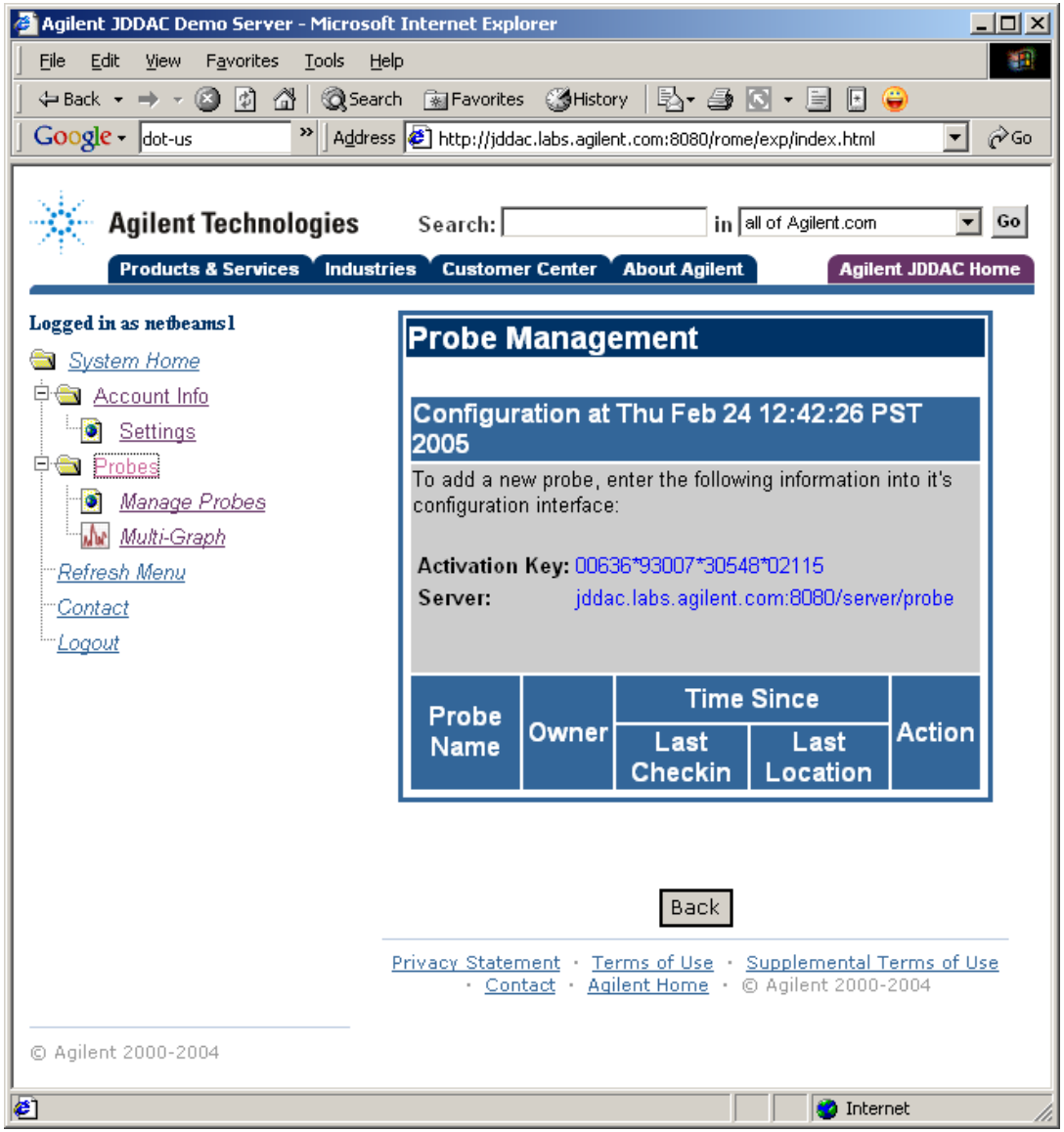
. Look in apps/netbeams.

- Source code in /src subdirectory.
  - TIM in net.java.jddac.netbeams.transducer.DemoTIM.
  - Application main() in net.java.jddac.jmdi.NetBEAMS.
  - Remaining files part of distribution.
- Configuration file in startup.xml.



# Part 1
# Build a working probe

- User accounts have already been created
  - netbeams1...6
- Log into server.
  - http://jddac.labs.agilent.com
- Click on Probes and note license key.



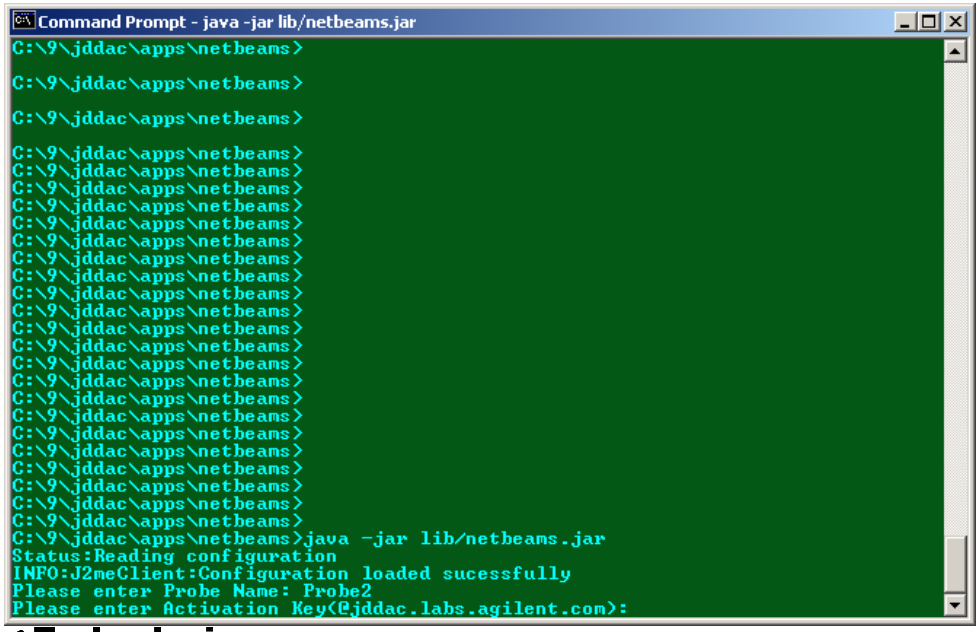





Agilent Technologies

# Part 1
# Build a working probe

- To build probe source code:
  - cd to root of JDDAC directory.
  - Invoke *ant*
  - Be sure and rebuild when there are changes to source files or configuration file.
- To run probe source code
  - *java –jar lib/netbeams.jar*
  - Enter the license key, probe name, and server name (use default).
- Log into server, select probe, select measurement, see graph.

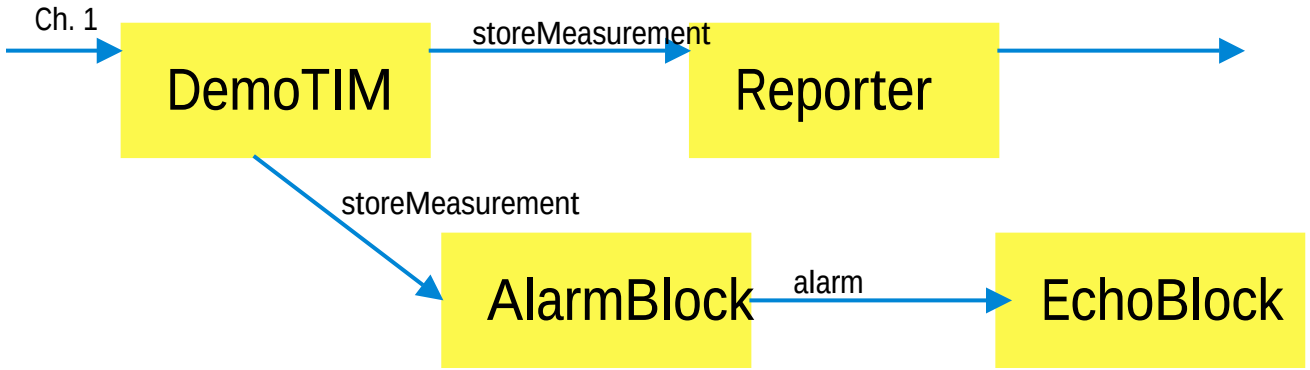


# Part 2
# Adding a new channel to TIM

- We will add two new channels to TIM, and watch the new measurements appear on the server.
  - Source code: net.java.jddac.netbeams.transducer.DemoTIM in *apps/netbeams*
- Change the algorithm in readChannel().
- Add two new channels to report the amount of free memory and total memory in the virtual machine.
  - Update TEDS array.
  - Update readChannel() method.
  - *long freeMem = Runtime.getRuntime().freeMemory();*
  - *long totalMem = Runtime.getRuntime().totalMemory();*
- Recompile, run, observe new measurements on server.

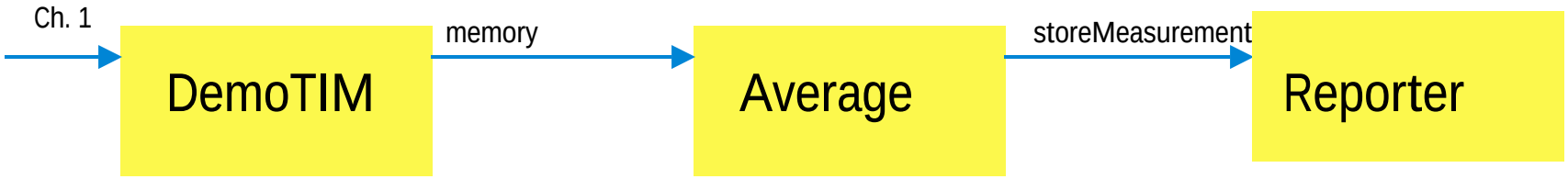
# Part 3
# Adding a new processing block

- Examine startup2.xml.
  - Note the addition of Alarm block in system
- Compile using startup2.xml.
  - Rename startup2.xml to startup.xml (*mv startup2.xml startup.xml*)
  - *ant*
- Run code
  - *java –jar lib/netbeams.jar*



# Part 4
# Writing a new processing block.

- Use MyFunctionBlock as skeleton, write a block to calculate the average memory readings and publish it to reporter.
  - Receive publication.
  - Extract measurement values.
  - Calculate average.
  - Publish calculated average.
- Edit startup.xml to include Average Block.






Agilent Technologies