

# Advanced FieldServer Functions

Presented by:  
Gordon MacLachlan (Mac)



# Presentation Highlights

## **Data Organization**

- Data Move Section
- Reasons for moving data
- Types of data moves available
- Data Move Parameters

## **Data Manipulation**

- Special moves
- Understanding the influence of Data Array types.
- Bit packing
- Scaling
- Logic

## **Testing tools**

- Checking your configuration

## **Special Applications**

- Hot Standby
- Active Server applications
- Passive Client applications
- Port Expansion – Mapping modbus to modbus
- Reporting Node Status

# Data Organization

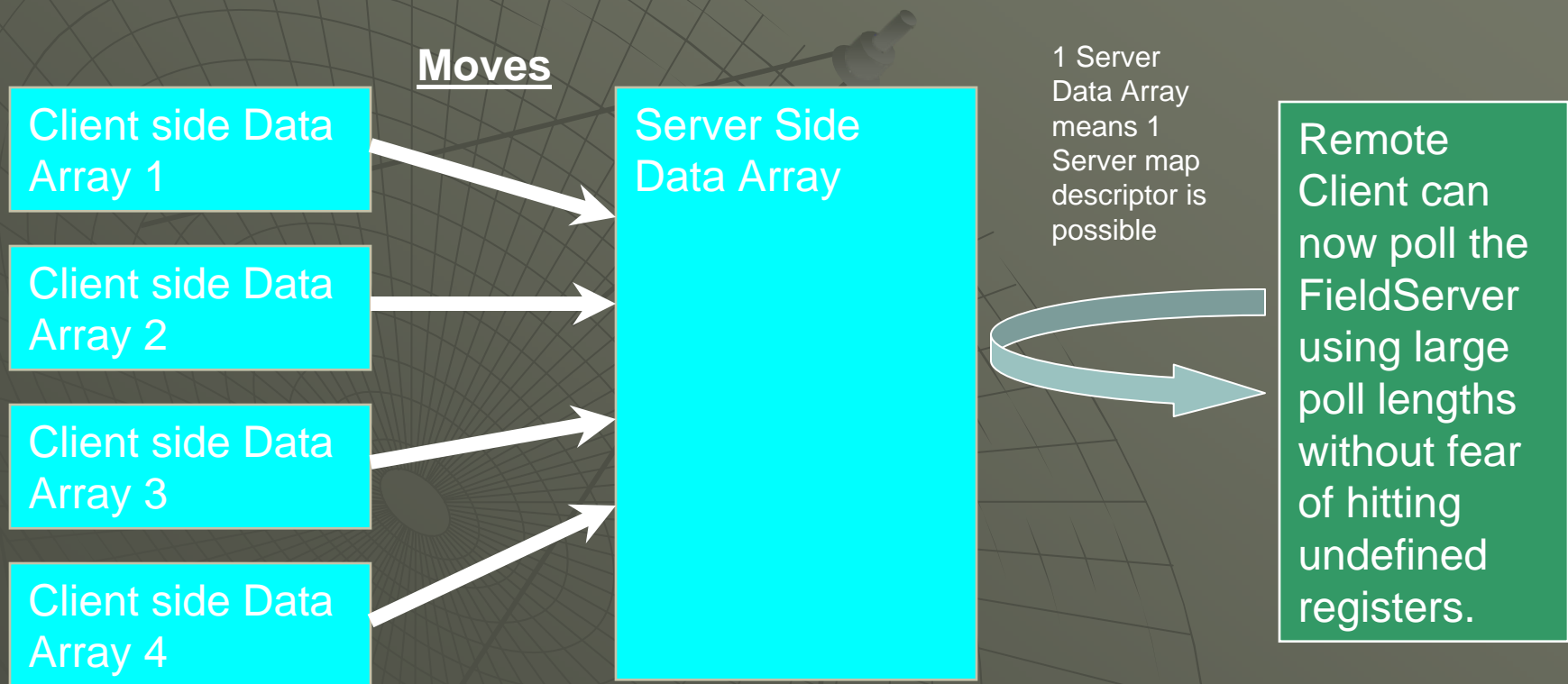
## Data Moves

```
//-----  
//  
// Data Arrays  
//  
// Declare Data Arrays for the data to be moved. In the real world example,  
// the Data Arrays may already be declared.  
//  
  
Data_Arrays  
Data_Array_Name , Data_Format , Data_Array_Length  
Source_DA , Float , 200  
Target_DA , Float , 200  
  
//-----  
//  
// Set up the moves to move the data.  
//  
  
Moves  
Source_Data_Array , Source_Offset , Target_Data_Array , Target_Offset , Length , Function  
Source_DA , 0 , Target_DA , 40 , 20 , Move_Only
```

# Data Organization

## Reasons for moving data

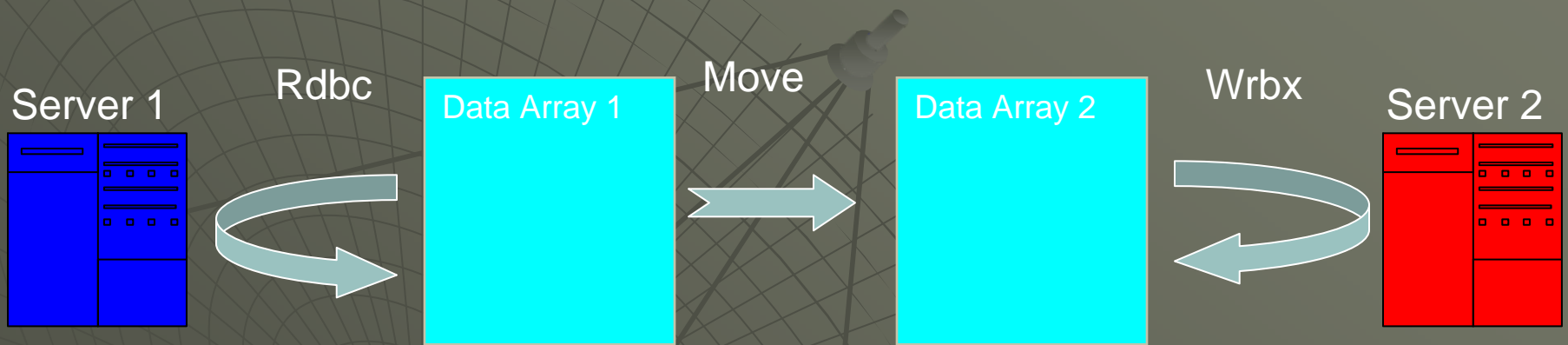
- ◆ Reason 1 – Grouping points for clean server mapping



# Data Organization

## Reasons for moving data

- ◆ Reason 2 – Separating Responsible map descriptors



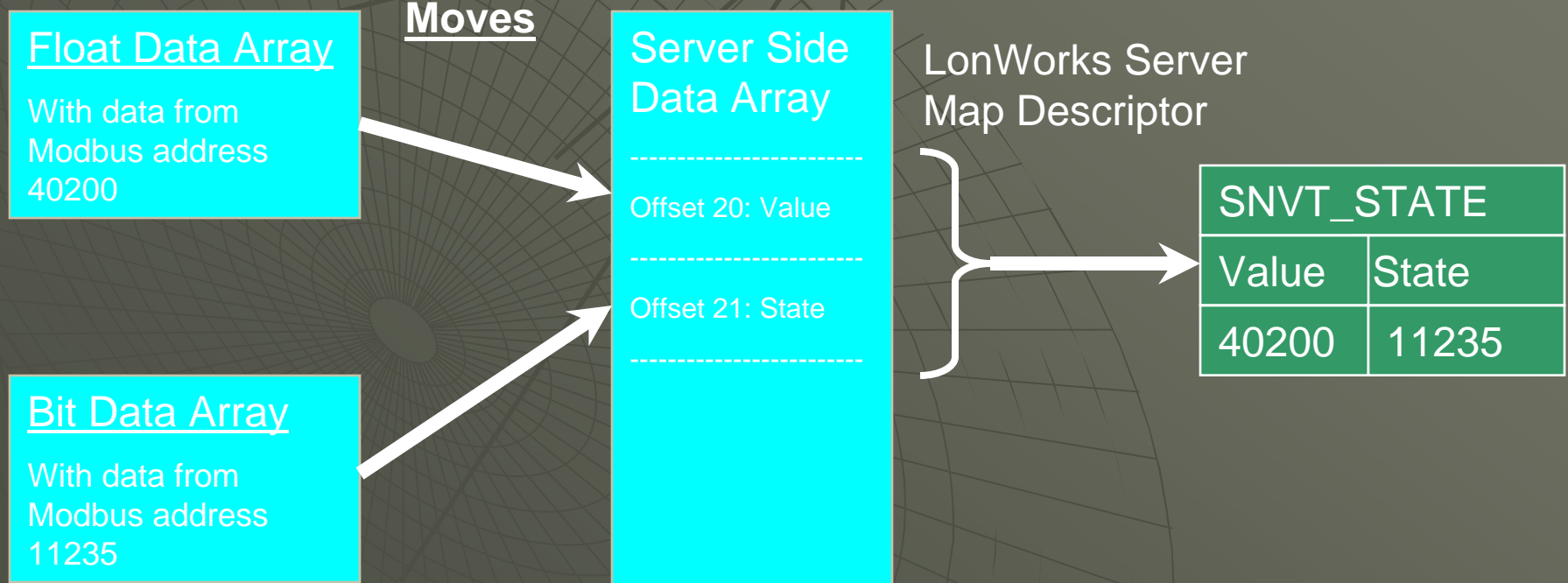
- Responsible Map Descriptors are active map descriptors that control the Communications. Two responsible map descriptors cannot share the same Data array offset due to monitoring functions present in the kernel.
- Responsible map descriptors are recognized by having the function rdbc or wrbx

# Data Organization

## Reasons for moving data

- ◆ Reason 3 – Compiling Complex Objects

Example: Creating a LonWorks SNVT\_Switch variable from 2 Modbus registers

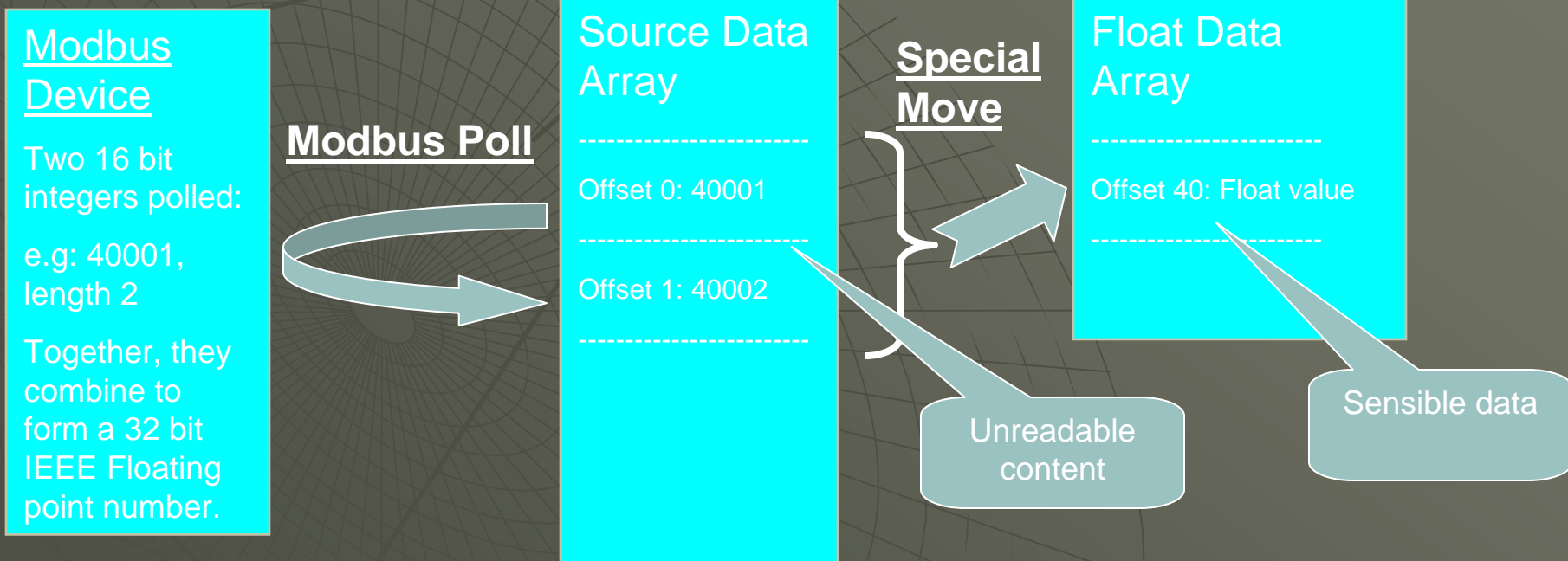


# Data Manipulation

## Special Moves

### ◆ Modbus Floating Point Moves

Moves  
Source\_Data\_Array , Source\_Offset , Target\_Data\_Array , Target\_Offset , Length , Function  
Source\_DA , 0 , Target\_DA , 40 , 10 , 2. i 16-1. float-sw



• Same principle applies for byte and 32 bit integer data

# Data Manipulation

## Typecasting (FieldServer does this by default)

- ◆ Each protocol variable type is allocated a default Data type
- ◆ Moves between dissimilar data types result in type-casting
- ◆ To avoid type-casting, use matching data array types or special functions such as Floating point moves or Packed Bit Data Arrays

### Example variable types

Bacnet AI  
Default Type = Float

Modbus 3xxxx  
Default Type = Uint16

Metasys DI  
Default Type = Bit

### Poll

Change  
value to a  
floating  
point  
value

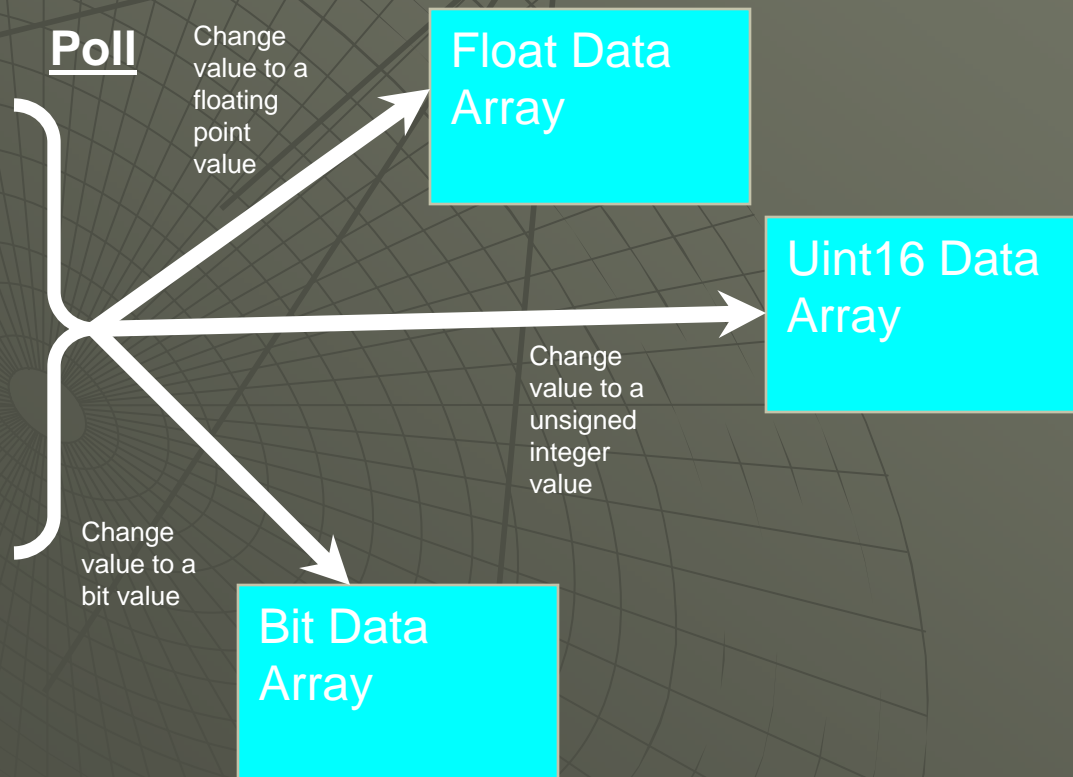
Float Data  
Array

Uint16 Data  
Array

Change  
value to a  
unsigned  
integer  
value

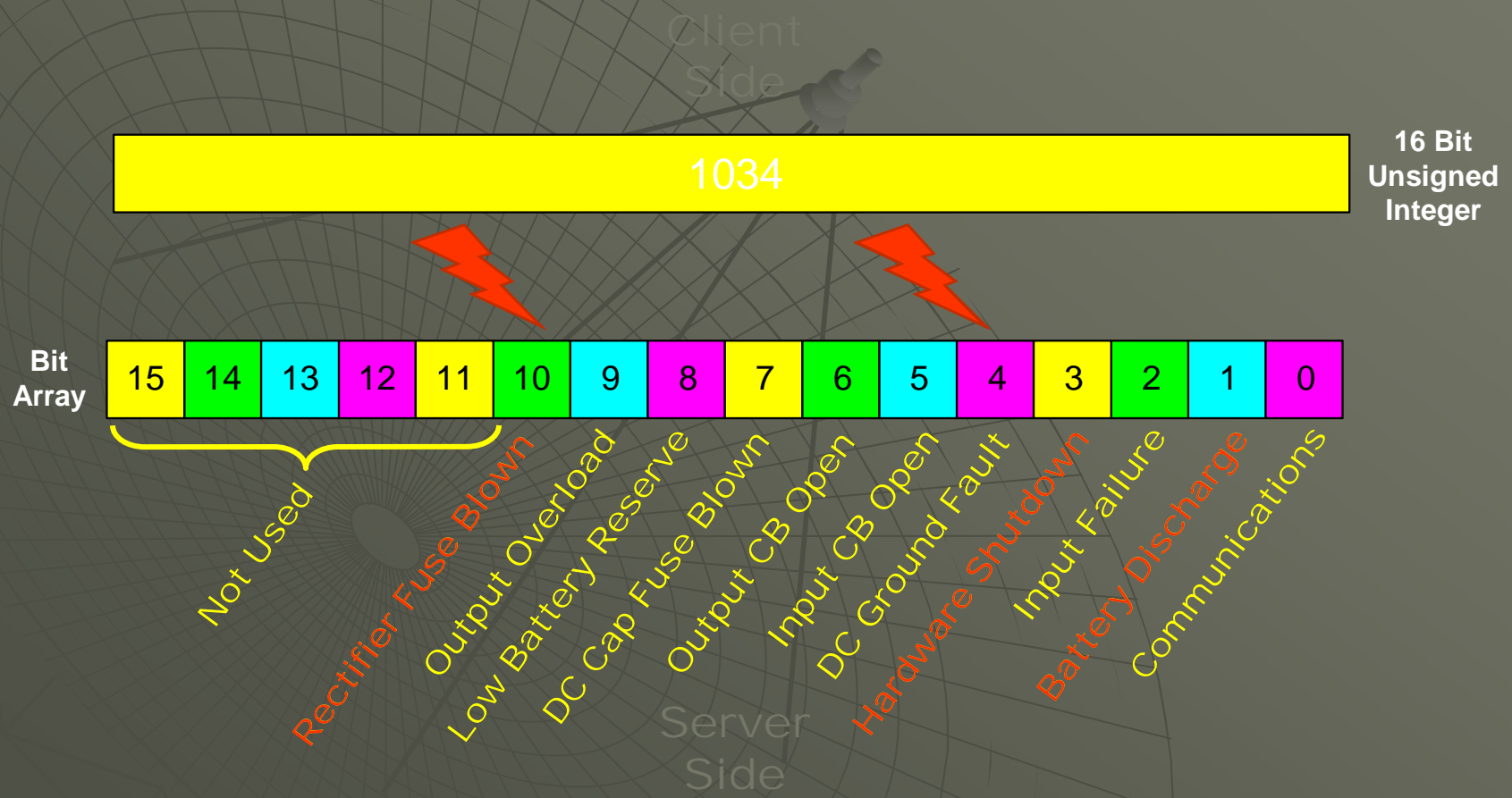
Change  
value to a  
bit value

Bit Data  
Array



# Data Manipulation

Bit Packing  
Concept:



# Data Manipulation

## Bit Packing

### Mechanics:

```
//-----  
//  
// Data Arrays  
// Packed_Bit is enabled merely by declaring the data array to be  
// of type: Packed_Bit  
Data_Arrays  
Data_Array_Name , Data_Format , Data_Array_Length  
Packer , Packed_Bit , 200
```

### Packed Bit Data Array

Offset 0 30001	Offset 0 DI 1 ... DI16
Offset 1 30002	Offset 16 DI17 ... DI32

Poll



No typecasting occurs. Data Array is treated as type Uint16 to Match variable type

Poll



No typecasting occurs. Data Array is treated as type Bit to match variable type

Modbus 3xxx  
Default Type = Uint16

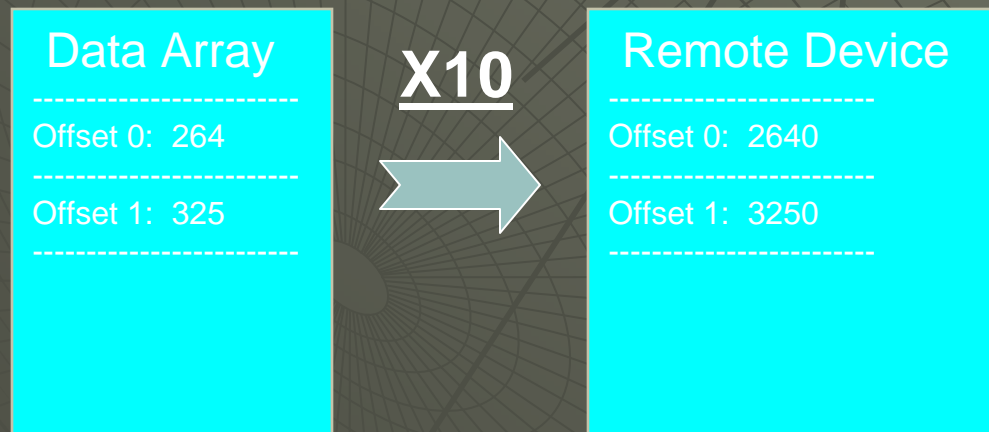
Metasys DI  
Default Type = Bit

# Data Manipulation

## Scaling

Map\_Descriptors

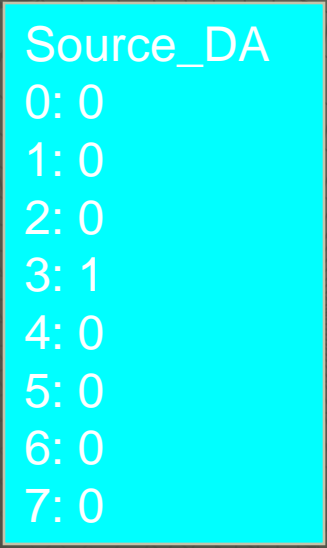
Node_Name	Address	Length	Data_Array_Low_Scale	Data_Array_High_Scale	Node_Low_Scale	Node_High_Scale
MBP_Srv_1	30001	200	0	10	0	100



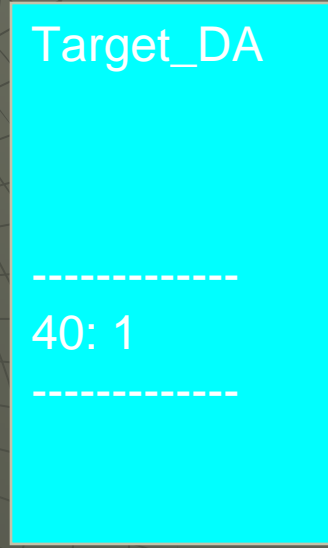
# Data Manipulation

## Logic

```
// Config declaration:  
Moves  
Source_Data_Array , Source_Offset , Target_Data_Array , Target_Offset , Length , Function  
Source_DA , 0 , Target_DA , 40 , 8 , OR
```



**OR**



# Testing tools

## Checking your configuration - Hints and tips:

### Step1- Fix all Syntax errors:

1. Ruinet error screen is the first place to look for Syntax Errors
2. Follow advice on configuration errors, and fix
3. Check Troubleshooting manual for further advice

### Step2 – Test data flow:

1. If connected to a device and communicating, check that data comes into data arrays where it is expected.
2. If not connected to device, do a manual check and follow a few points through the configuration to verify they are served to the expected memory locations.
3. Moves can be tested by plugging values into the source data array and verifying correct destination.

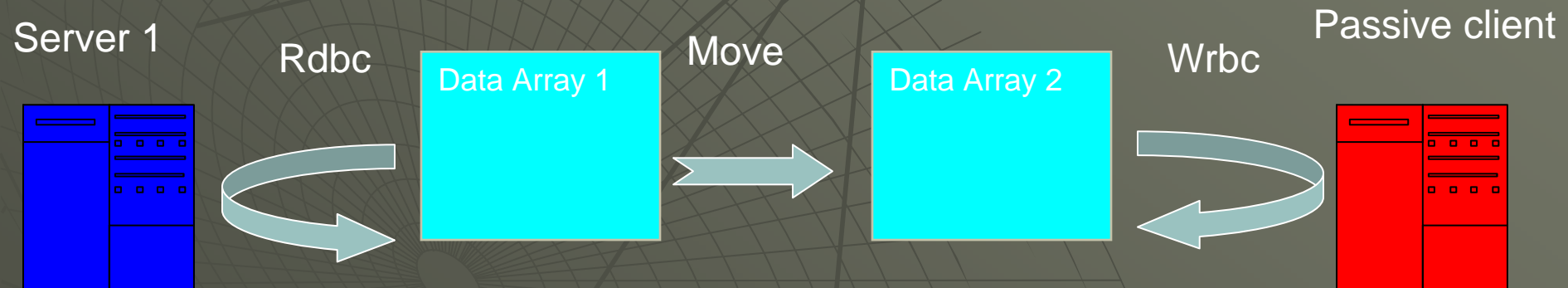
### Step 3 – look for common errors:

1. Make sure points being polled exist in remote device
2. Check remote device documentation for clues that special moves may be needed
3. Check that all nodes needed have been declared
4. Check that points are being mapped to right nodes
5. Check scaling is the right way around

# Special Applications

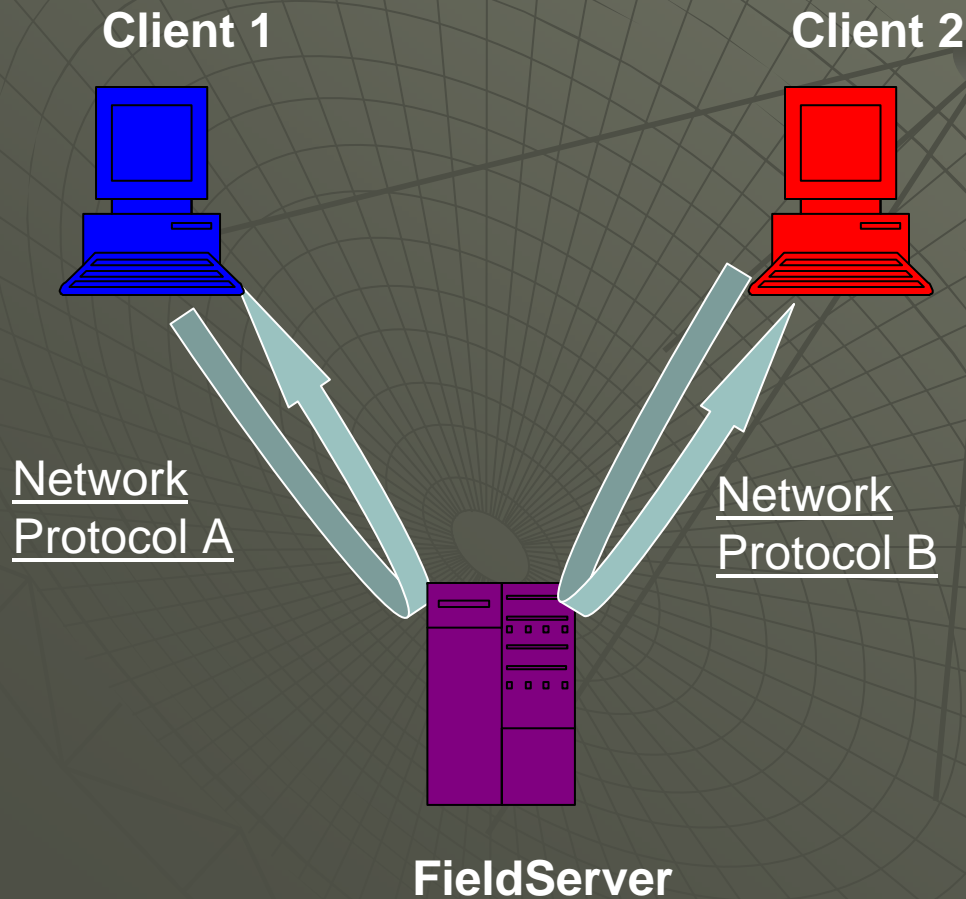
## Active Server Applications

- ◆ Normally found where data needs to be written to a passive device.
- ◆ Wrbc is normally the best function for writing status data
- ◆ Moves may be necessary to separate responsible map descriptors



# Special Applications

## Passive Server only Applications



- This “Data Sharing” configuration is useful in applications where clients on different networks need to share stored data
- The same setup can provide a passive server/passive client functionality too
- Map descriptor function used for both protocols A and B is “passive”
- Fieldserver is non-intrusive into both networks, and responds to queries and commands only.

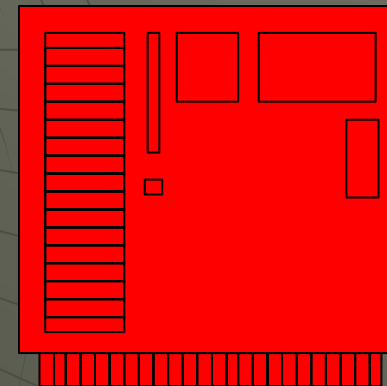
# Special Applications

## Passive Client Applications

- FieldServer function in the client side map descriptor is passive
- FieldServer mapping can typically include more points than needed without causing problems
- FieldServer typically cannot deduce the connection status, since it does not know when to expect data and consequently cannot conclude when communications has stopped.



Remote device  
writes  
alarms/status  
to FieldServer



# Special Applications

## Port Expansion

Full Port expansion configuration:

### Connections

Port	Baud	Parity	Data_Bits	Stop_Bits	Protocol	Handshaking	Poll_Delay
R1	9600	None	8	1	Modbus_RTU	None	0.100s
R2	9600	None	8	1	Modbus_RTU	None	0.100s
P7	9600	None	8	1	Modbus_RTU	None	0.100s
P8	9600	None	8	1	Modbus_RTU	None	0.100s
P1	9600	None	8	1	Modbus_RTU	None	-
P2	9600	None	8	1	Modbus_RTU	None	-

### Nodes

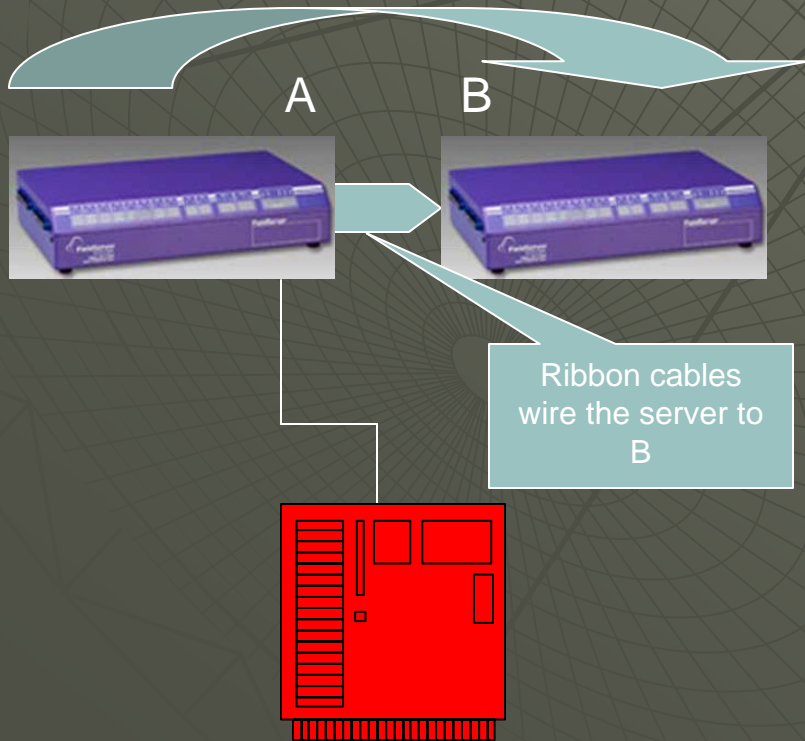
Node_Name	Node_ID	Protocol	Port
PLC_07	7	Modbus_RTU	P7
PLC_08	8	Modbus_RTU	P8
PLC_11	11	Modbus_RTU	R1
PLC_12	12	Modbus_RTU	R1
PLC_13	13	Modbus_RTU	R1
PLC_21	21	Modbus_RTU	R2
PLC_22	22	Modbus_RTU	R2
PLC_23	23	Modbus_RTU	R2

# Special Applications

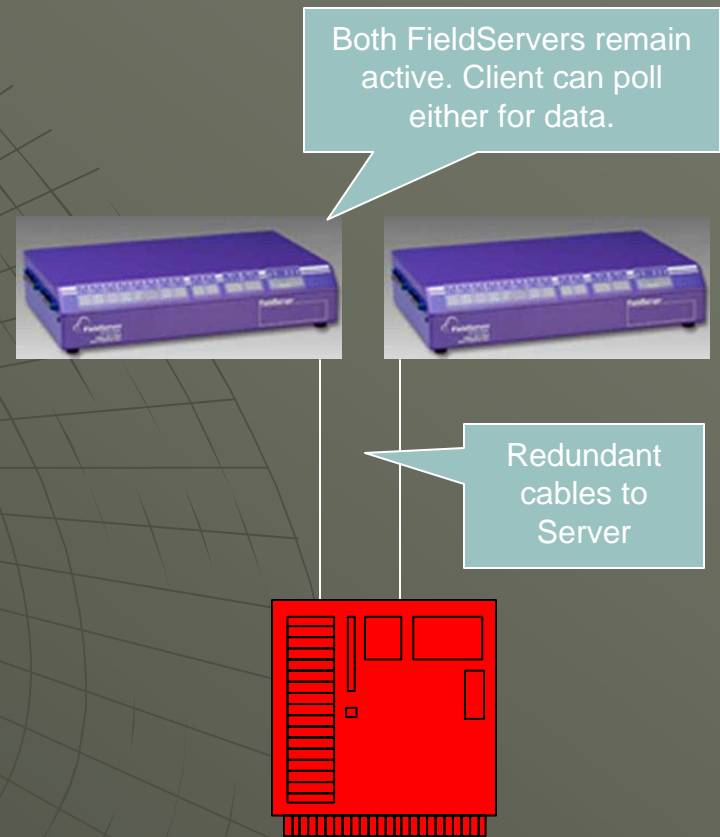
## Hot Standby

### Mode 1 (Hot standby)

Switch over to B only when A fails



### Mode 2 (Dual Redundant)



# Special Applications

## Node Status

```
//=====
//
// Notes : All that is needed to enable node status is the node status data array
//         declaration. Note, though, that Node ID's in the config need to be
//         unique, otherwise the FieldServer will incorrectly report duplicate Node ID's
//
//
//=====

//=====,
//
// Data Arrays
//
Data_Arrays
Data_Array_Name , Data_Format , Data_Array_Length , Data_Array_Function
DA_Comm_Status , Bit , 256 , Node_Status
```

# Resources



[www.fieldserver.com](http://www.fieldserver.com)

- FieldServer Configuration manual
- FieldServer Troubleshooting manual

# Questions?

Email Mac at:

[mac@fieldserver.com](mailto:mac@fieldserver.com)



# THANK YOU!

.....for taking the time to attend  
this presentation.