# Train Communication Network

# IEC 61375 - 2

# Real Time Protocols

## Process Variables

# Real-Time Protocols

1. General Principles

2. Variables
    1. Principle of cyclic Process Data broadcast
    2. Traffic Stores principle and implementation
    3. Process Variables and Datasets
    4. Software structure
    5. Application Layer Interface for Process Variables
    6. Networking

3. Messages
    1. Principle of Message Data communication
    2. Link Layer Interface
    3. Networking and Routing
    4. Transport protocol
    5. Software structure
    6. Application Interface

# TCN Data Traffic

Two traffics share the same bus:

## Process Data

short and urgent data items carrying the trains's state

... motor current, axle speed, operator's commands,...

*Since variables are refreshed periodically, there is no need for a retransmission protocol in case of transmission error.*

Periodic Transmission

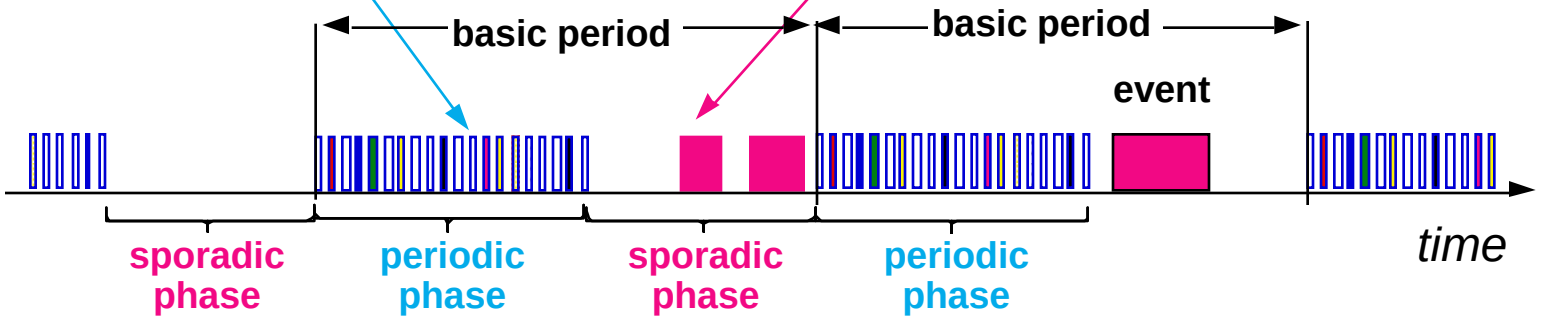
## Message Data

infrequent, sometimes lengthy messages reporting events, for:

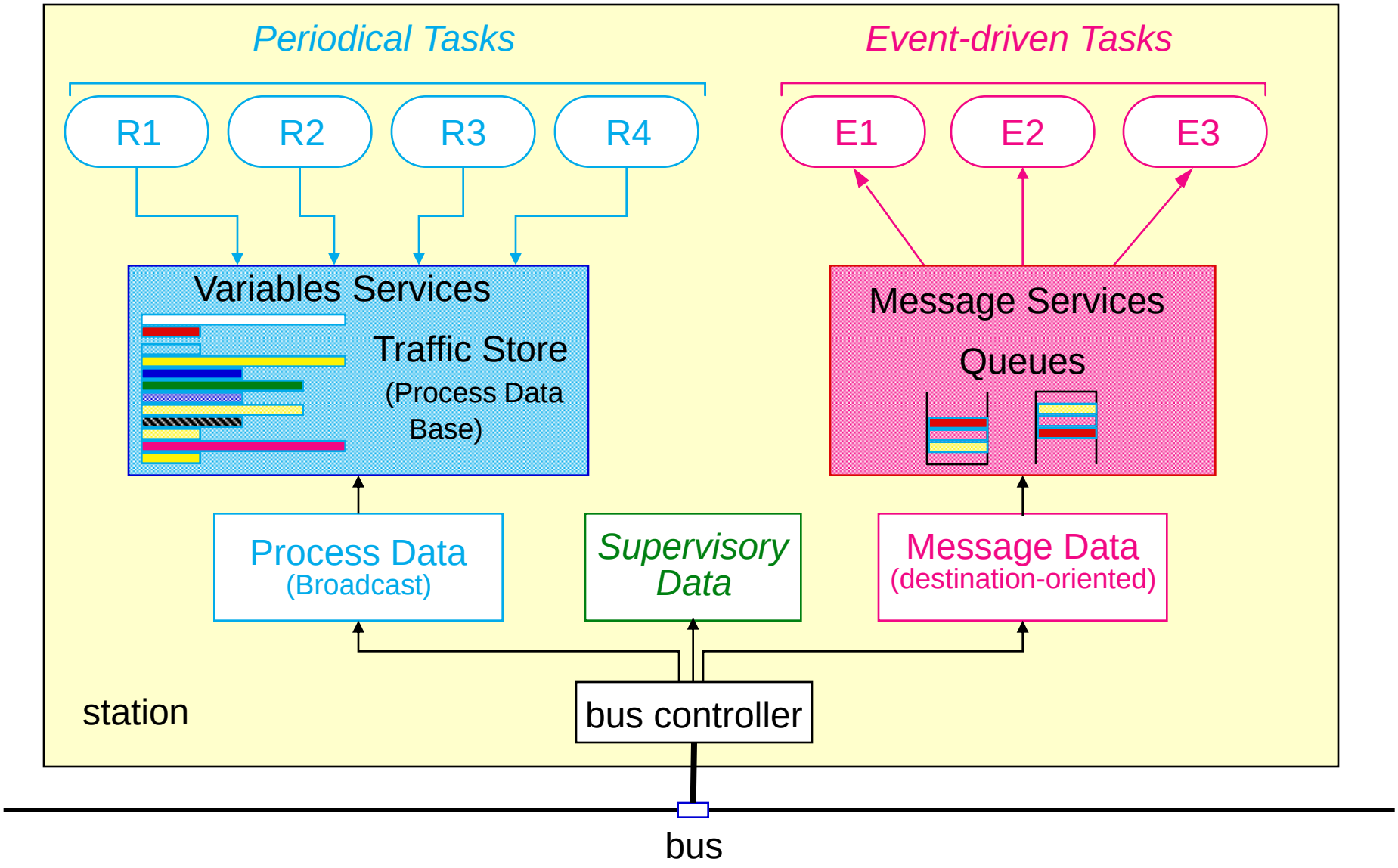
- Users: diagnostics, status
- System: initialisation, down-loading, ...

*Since messages represent state changes, they may not get lost a protocol recovers transmission errors.*

Sporadic Transmission



# Application Sight Of Communication



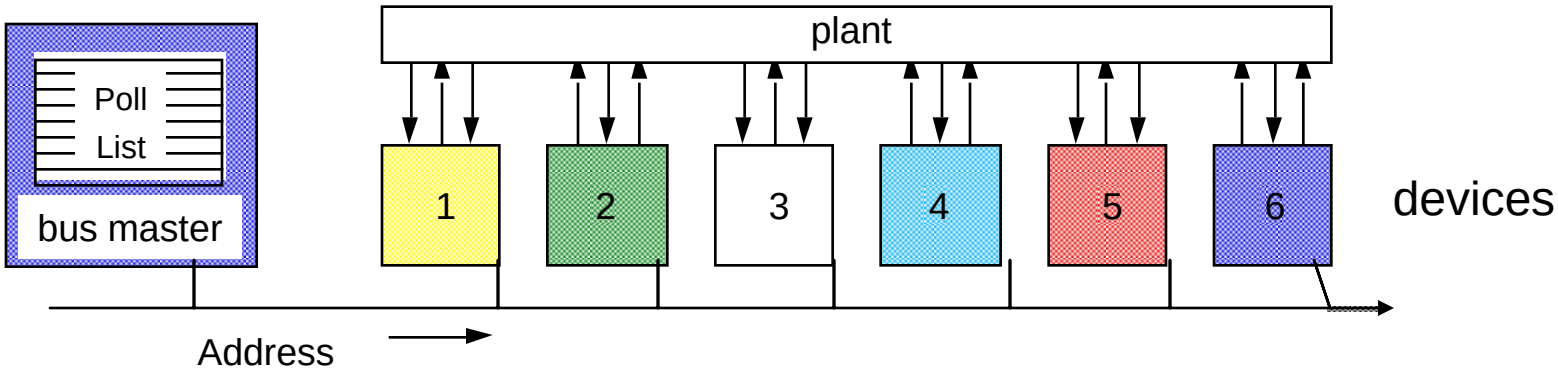
# Principle of Cyclic Source-Addressed Broadcast

1. General Principles

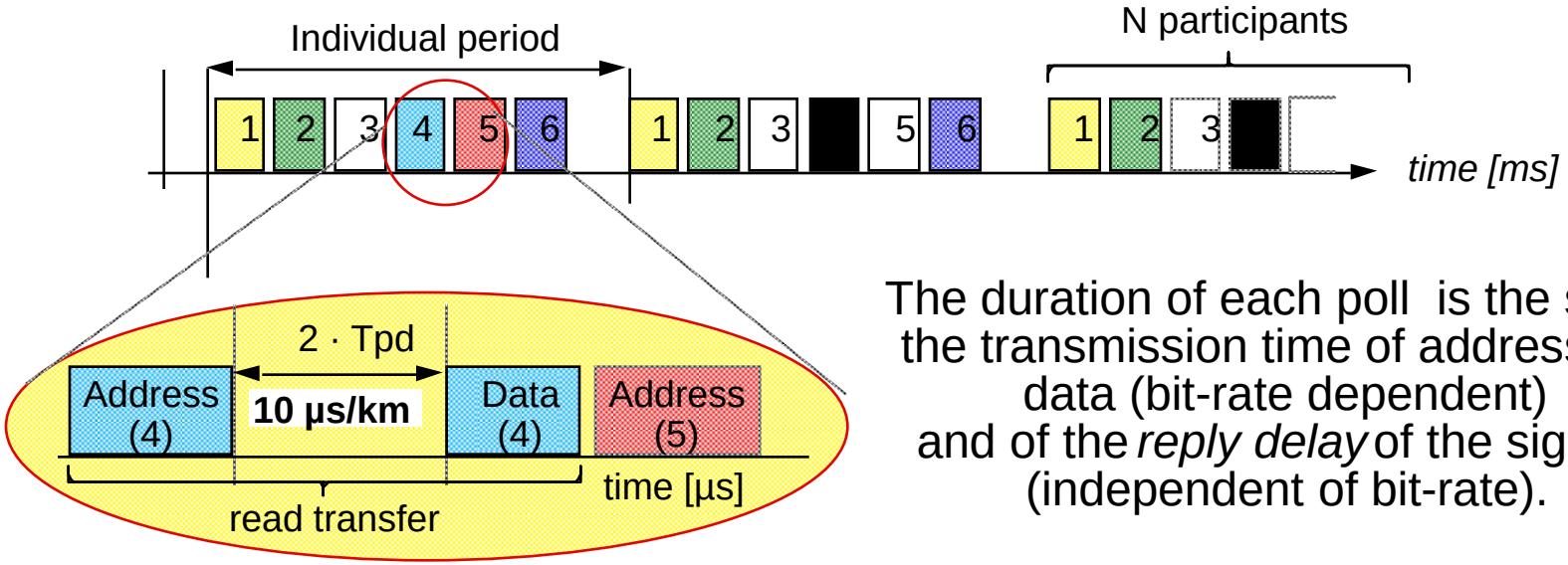
2. Variables
    1. Principle of cyclic source-addressed broadcast
    2. Traffic Stores principle and implementation
    3. Process Variables and Datasets
    4. Software structure
    5. Application Layer Interface for Process Variables
    6. Networking

3. Messages
    1. Principle of Message Data communication
    2. Link Layer Interface
    3. Networking and Routing
    4. Transport protocol
    5. Software structure
    6. Application Interface

# Cyclic Data Transfer




The master polls addresses in a fixed sequence, according to its Poll List.




The duration of each poll is the sum of the transmission time of address and data (bit-rate dependent) and of the *reply delay* of the signals (independent of bit-rate).

# Cyclic Operation

Cyclic operation is used to transmit the state variables of the train.
Periodically transmitted state variables are called Periodic Data.

Periodic Data are transmitted at fixed intervals , whether they changed or not.

Their delivery delay (refresh rate) is deterministic and constant.

The bus is under control of a central master.

There is no need for error recovery since a fresh value will be transmitted in the next cycle.

Only states may be transmitted, not state changes.

The cycle time is limited by the product of the number of data transmitted by the duration of each poll.
(e.g. 100 µs /point, 100 points => 10 ms)

To keep a low poll time, only small data items may be transmitted (<256 bits)

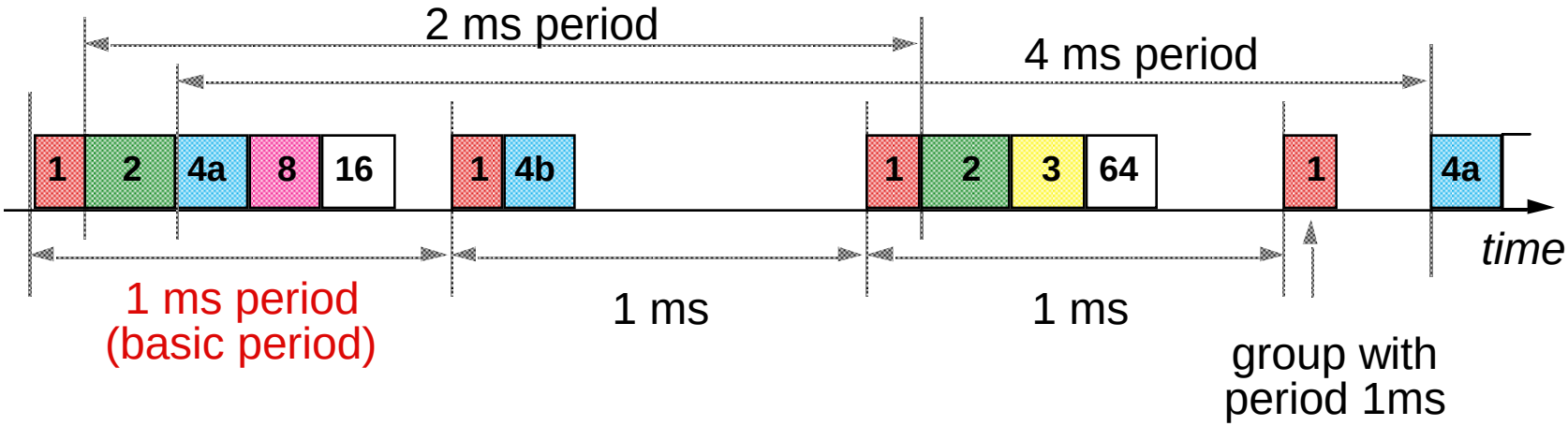
# Optimizing Cyclical Operation

Cyclic operation uses a fixed portion of the bus's time

The poll period  increases with the number of polled items

The response time slows down accordingly

**Solution**:  introduce subcycles for less urgent data:




Cyclic polling need tools to configure the poll cycles.

Poll cycles should not be modified at run-time (source of non-determinism)

# Subscription Principle

A device exports many process data (state variables) with different priorities.

If there were only one poll type per device, a device should be polled at the frequency required by its highest-priority data.

Rather than poll a device, the master polls the process data.

Each device is subscribed as source or as sink for a number of process data. Only one device may be source of a certain process data (collision).

The replicated traffic stores can be considered as "caches" of the plant state.

The application accesses them in the same way as a process database.

The bus becomes an on-line data base.

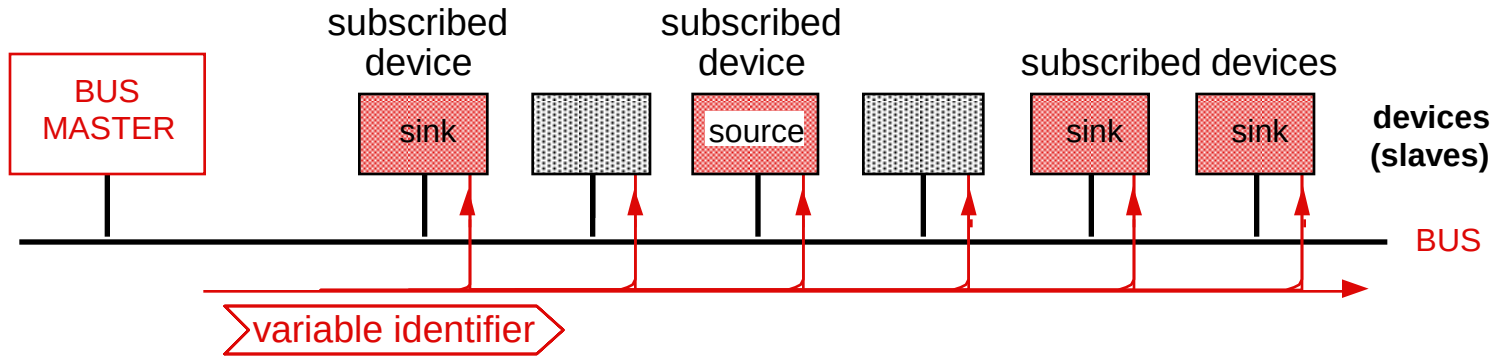
Each station monitors the bus and snoops the variables it is interested in .

Each station needs an associative memory to recognize its variables.

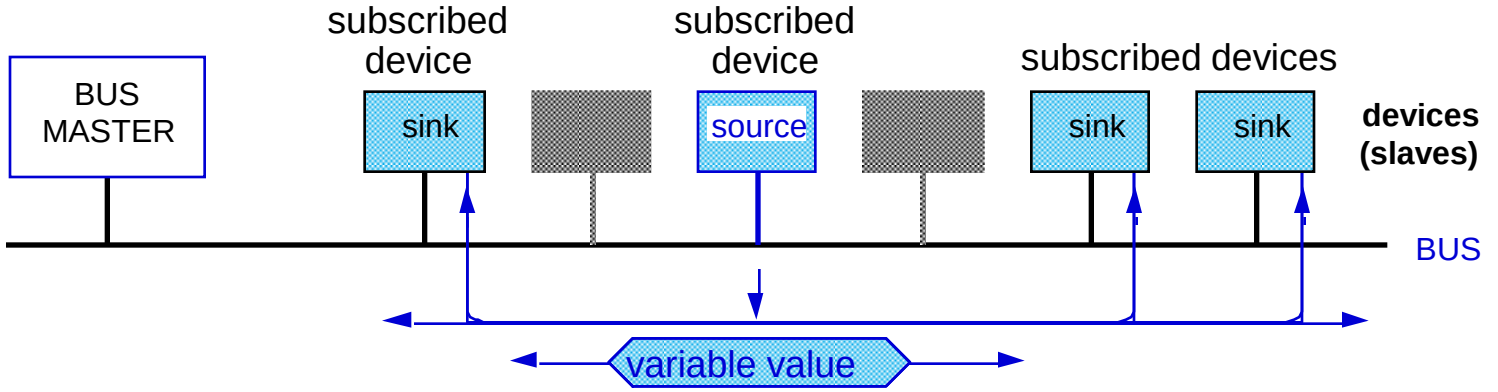
# Process Data Transmission

Process Data are transmitted by *source-addressed broadcast* :

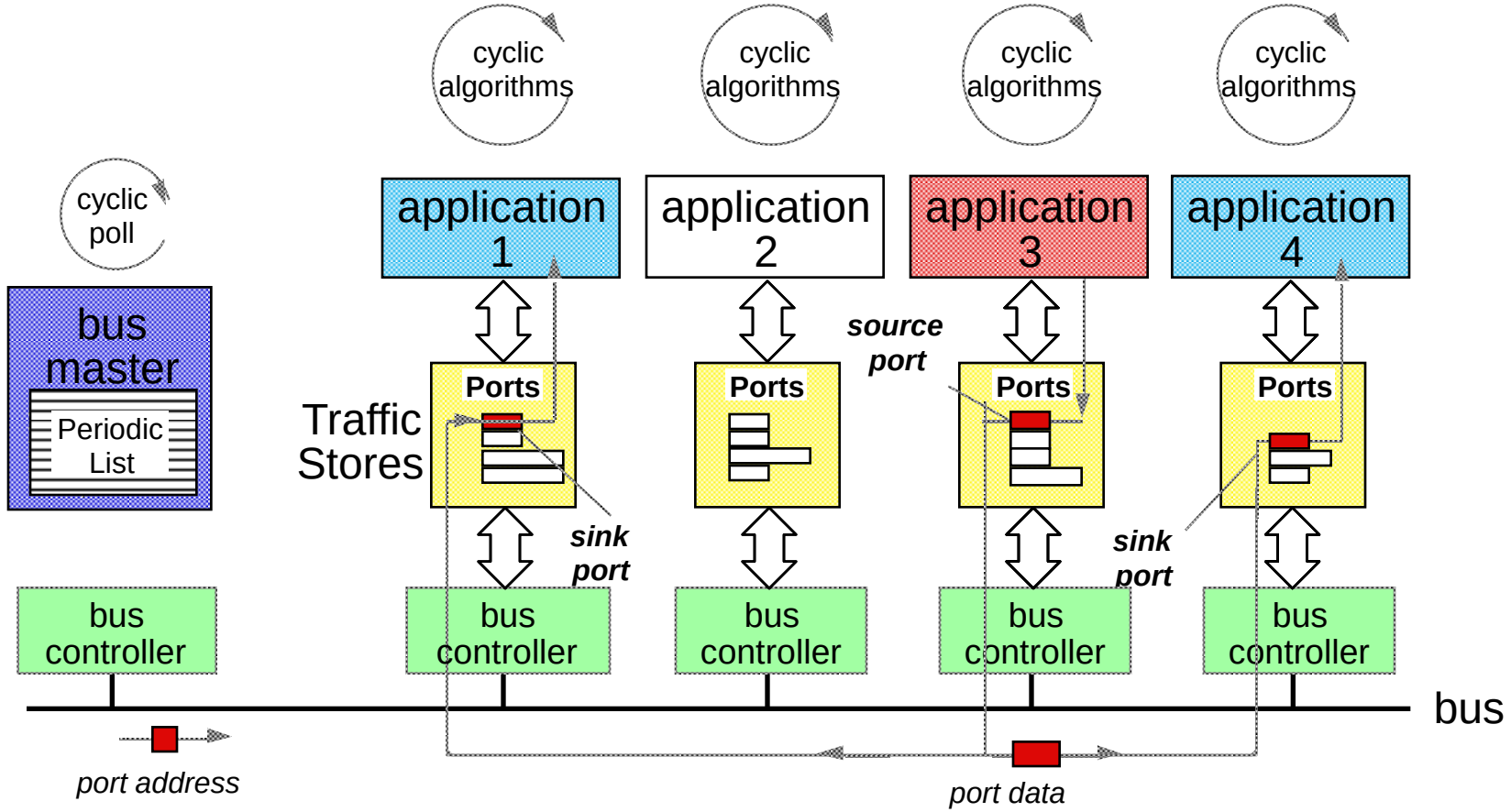
Phase1: The bus master broadcasts the identifier of a variable to be transmitted:




Phase 2: The device which sources that variable responds with a slave frame containing the value, all devices subscribed as sink receive that frame.



# Traffic Stores




The bus traffic and the application cycles are asynchronous to each other.

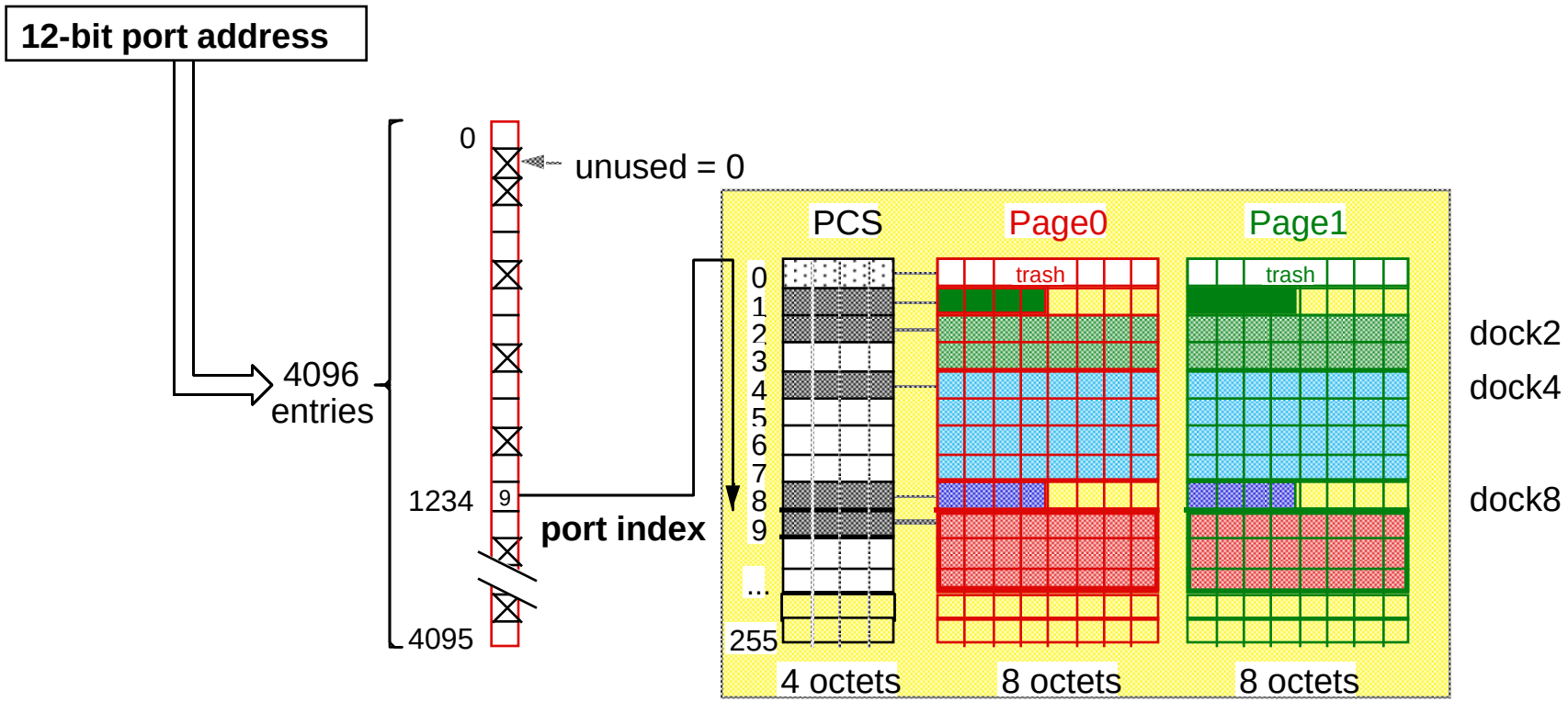
The bus master scans the identifiers at its own pace.

Bus and applications interface through a shared memory, the *traffic store.*

# Traffic Stores

1. General Principles
2. Variables
    1. Principle of cyclic Process Data broadcast
    2. Traffic Stores principle and implementation
    3. Process Variables and Datasets
    4. Software structure
    5. Application Layer Interface for Process Variables
    6. Networking
3. Messages
    1. Principle of Message Data communication
    2. Link Layer Interface
    3. Networking and Routing
    4. Transport protocol
    5. Software structure
    6. Application Interface

# MVB Traffic Store Implementation




| port index table | port control & status | ports odd page | ports even page |
|---|---|---|---|
| 4K ports x 1 octet = 4K octets | 256 PCS x 4 octets = 1K octets | 256 docks x 8 octets = 2K octets | 256 docks x 8 octets = 2K octets |

# MVB Traffic Store, Ports And Docks

Traffic stores are divided into Ports, each Port corresponding to one identifier

A device has normally a small number of Ports (e.g. 256), but a supervisory device may have the full range of 4096 Ports.

A Port may be up to 1024 bit wide (256bits on the MVB)

A Port is divided into Docks of 32 bits for better memory usage.

Its Port Control and Status register indicates if a Port is source or sink.

The Port Index Table forms an associative memory, which indicates for each identifier if it is subscribed to that device, by pointing to the PCS.

The former contents of a Port is overwritten (buffered, not queued).

To provide simultaneous and consistent access by either application or bus, each Port is stored in two independent pages, written alternately.

The PCS performs additional functions like freshness supervision and interrupt generation.

# Freshness Supervision

Process Data are not retransmitted explicitely in case of transmission error, they will be retransmitted anyhow at the next poll.

The application must tolerate an occasional loss of data.

To protect the application from using obsolete data, each Port in the traffic store has a freshness counter.

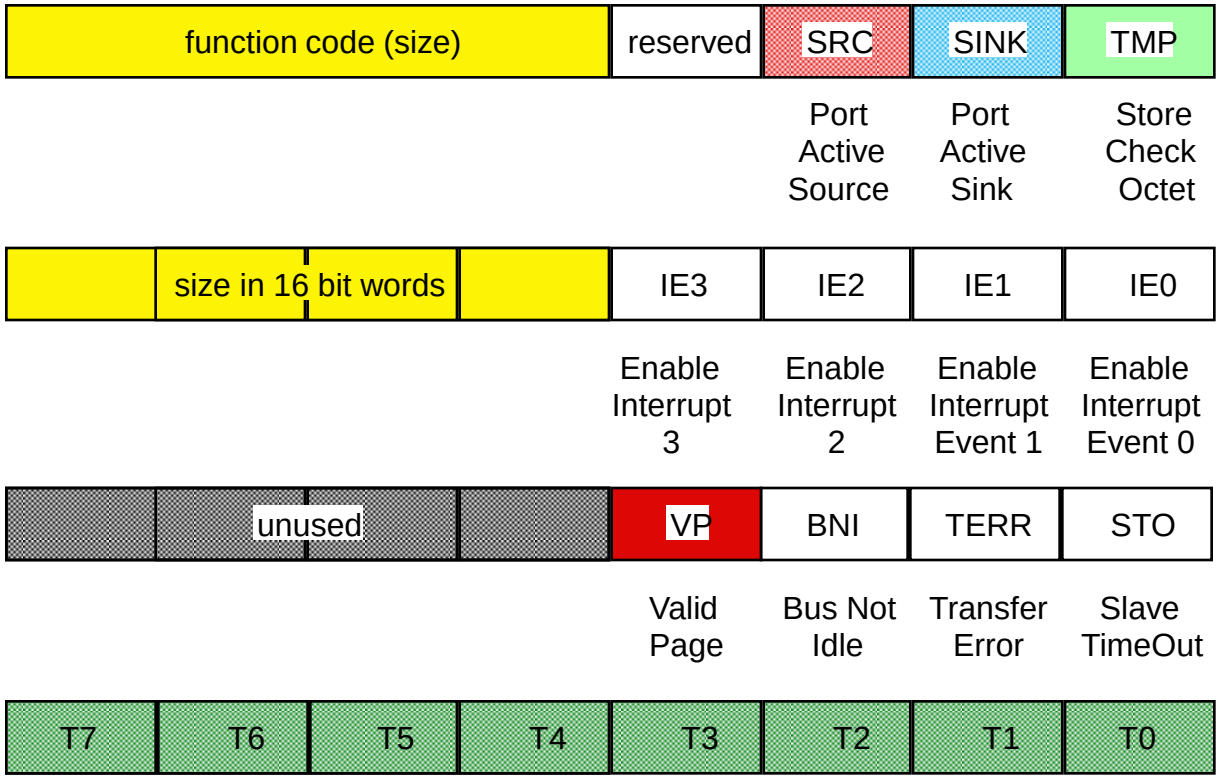
This counter is reset by writing to that Port. It is incremented regularly, either by the application processor (BAP) or by the bus controller (MVBC).

The application should always read the value of the counter before using the Port data and compare it with its tolerance level.

The freshness supervision is evaluated by each reader independently, some readers may be more tolerant than others.

Bus error interrupts in case of severe disturbances are not directed to the application, but to the device management.

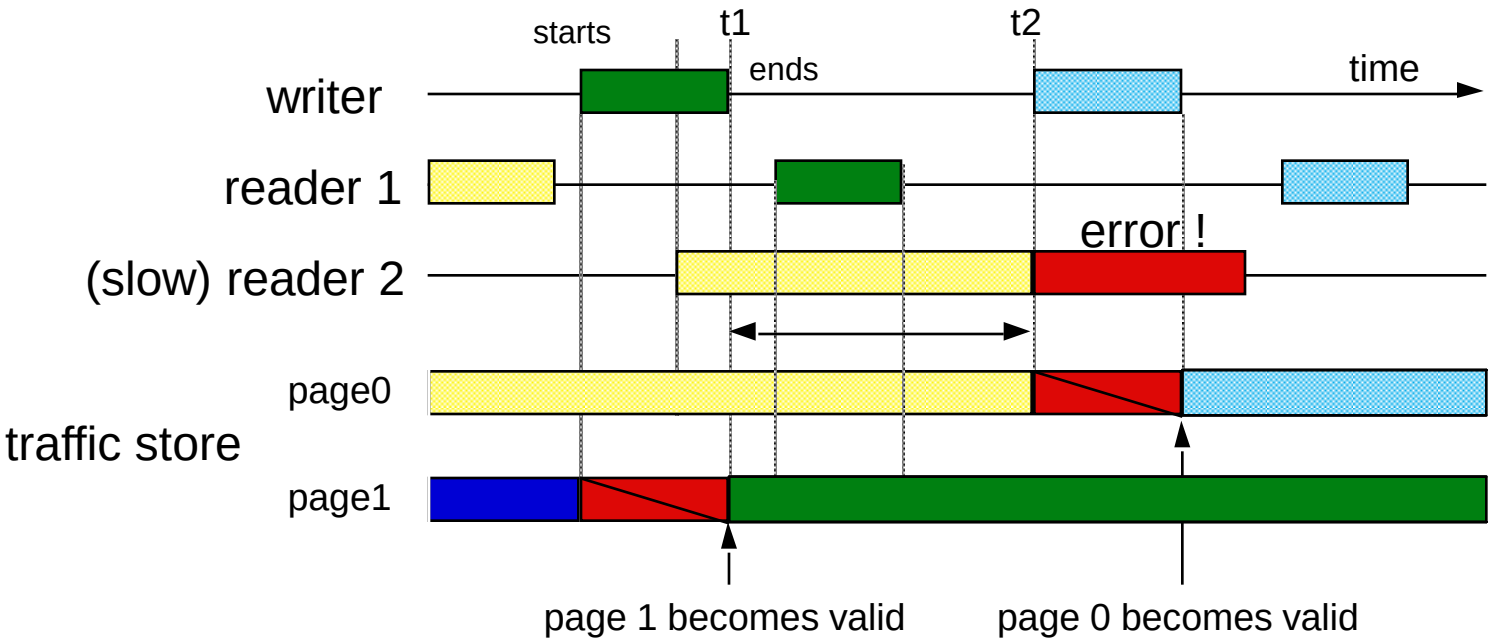
# Implementation of Port Control And Status (BAP controller)




sink time supervision counter

The PCS depends both on the traffic store and on the bus controller

# Traffic Store Access Restriction




- there may be no semaphores to guard access to a traffic store (real-time)
- there may be only one writer for a port, but several readers
- a reader must read the whole port before the writer overwrites it again
- this time is equal to the basic period in the worst case ( 1ms)
- therefore, the processor must read ports with interrupt off.

# Process Variables And Datasets

1. General Principles
2. Variables
    1. Principle of cyclic Process Data broadcast
    2. Traffic Stores principle and implementation
    3. Process Variables and Datasets
    4. Software structure
    5. Application Layer Interface for Process Variables
    6. Networking
3. Messages
    1. Principle of Message Data communication
    2. Link Layer Interface
    3. Networking and Routing
    4. Transport protocol
    5. Software structure
    6. Application Interface

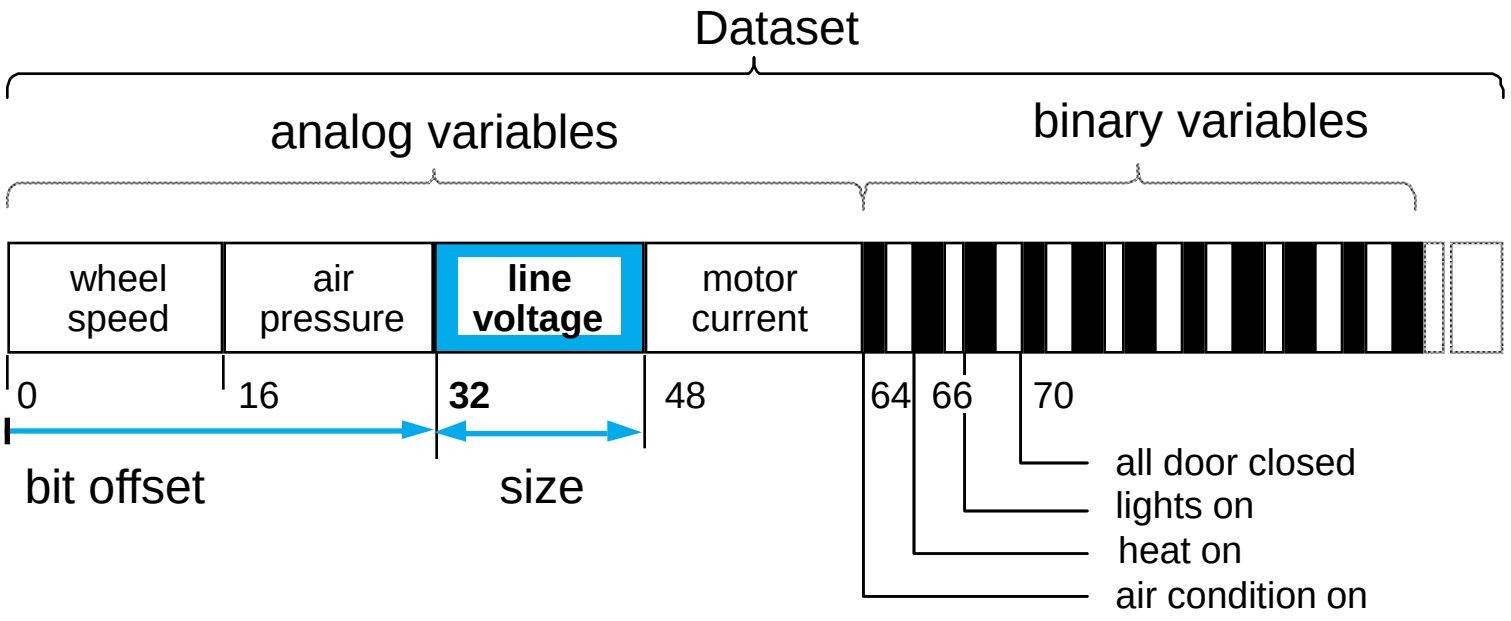
# Datasets

It is economical to transport several variables in the same frame as a dataset.

A dataset is treated as a whole for communication and access.

A variable is identified within a dataset by its offset and its size

Variables may be of different types, types can be mixed.



# Variable Types

Data representation may be different according to the processor
(e.g. Intel's Little-Endians and Motorola's Big-Endians)

However, there may be no ambiguity on the bus

Therefore, each transmitted variable has a defined type associated with it.

This type is agreed beforehand between the applications during configuration

To guarantee consistency, the type of the variable is part of the variable's name

TCN specifies a number of standard network variable types.

The format obeys to the Big-Endian convention (most significant first)

# Predefined Variable Types

BOOLEAN1

0 = FALSE
1 = TRUE

ANTIVALENT2

00 = network error
01 = ok
10= forced
11 = data undefined

BITSETn (array of n boolean)

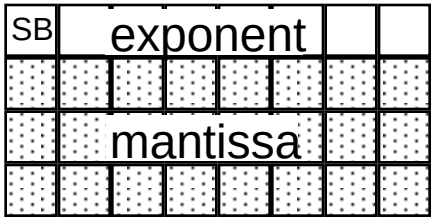
INTEGERn (2's complement)


SB = sign bit

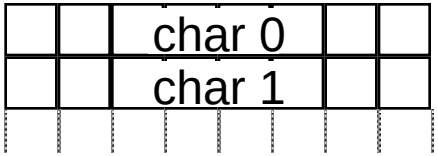
UNSIGNEDn

FRACTIONALn


REAL32 (IEEE)




ARRAY n OF CHARACTER8




TIMEDATE48...

Application data types have been defined in ROSIN

# Configuration Change And Errors

Reconfiguring the whole bus to include new variables is normally not allowed in a running project, since this can affect the time behaviour of all applications.

E.g. in the Train Bus, old coaches cannot be modified to consider new variables.

It also may be that a variable is invalid when produced, e.g. by a defective unit. The producer may not modify the structure of data sets to exclude failed data

Mechanisms are provided to protect against production and transmission errors:

Freshness Supervision at reception

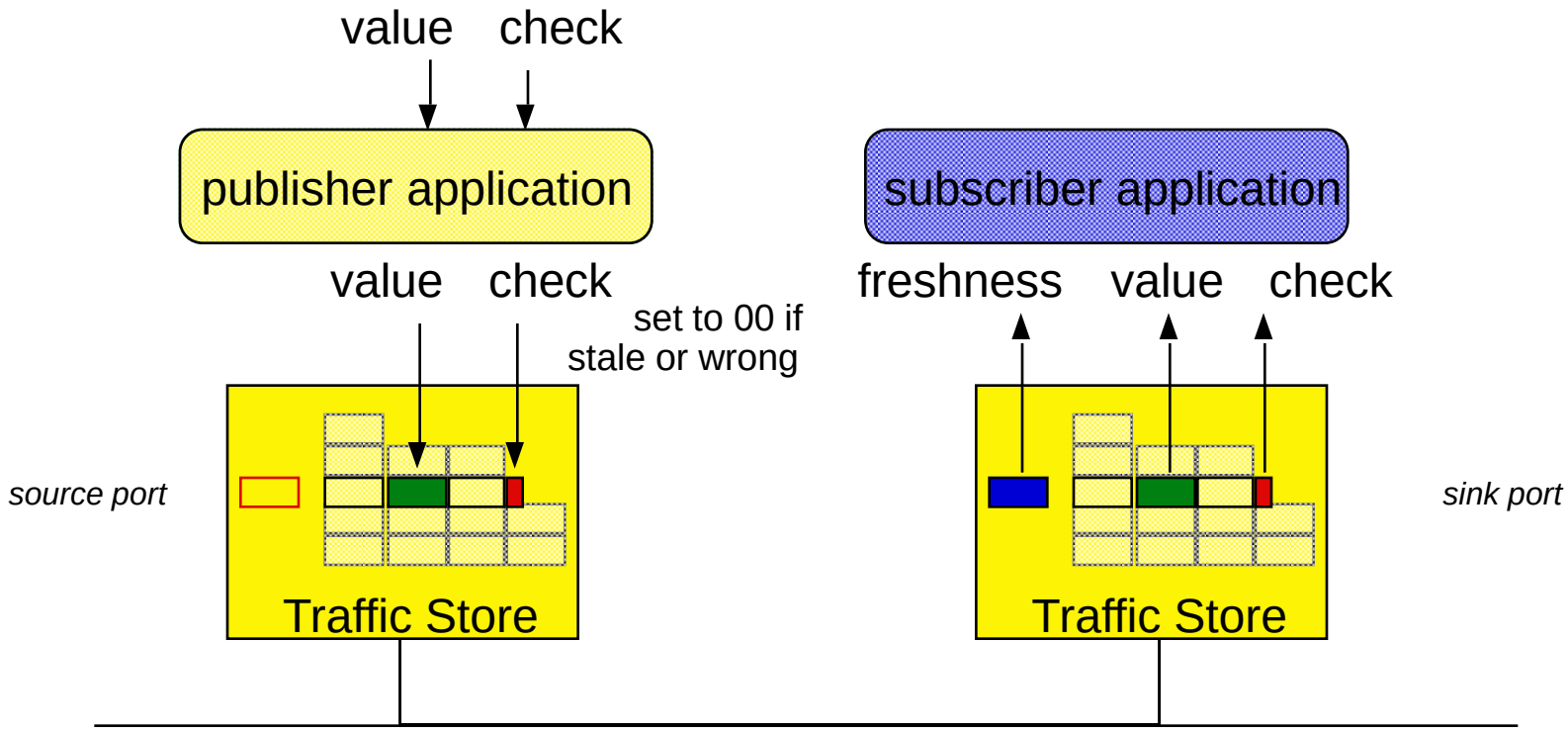
Check Variable: production supervision

Dedicated variables (Lifesigns) let all devices monitor the other's health

# Check Variable

TCN relies on the invalidation of stale data at the source through a check variable:

TCN does not rely on a source freshness supervision

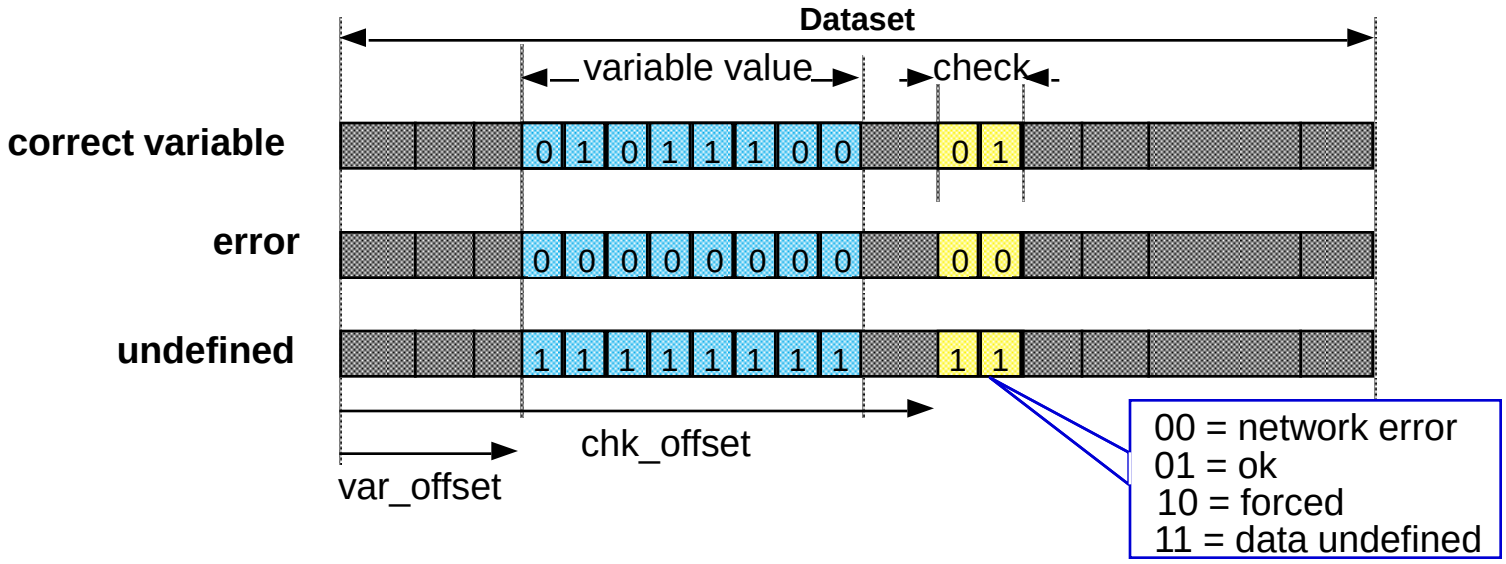


# Variable Extension And Invalidation

To allow later extension, room is left in the datasets for additional variables.
Since the type of these future data is unknown, unused fields are filled with '1".

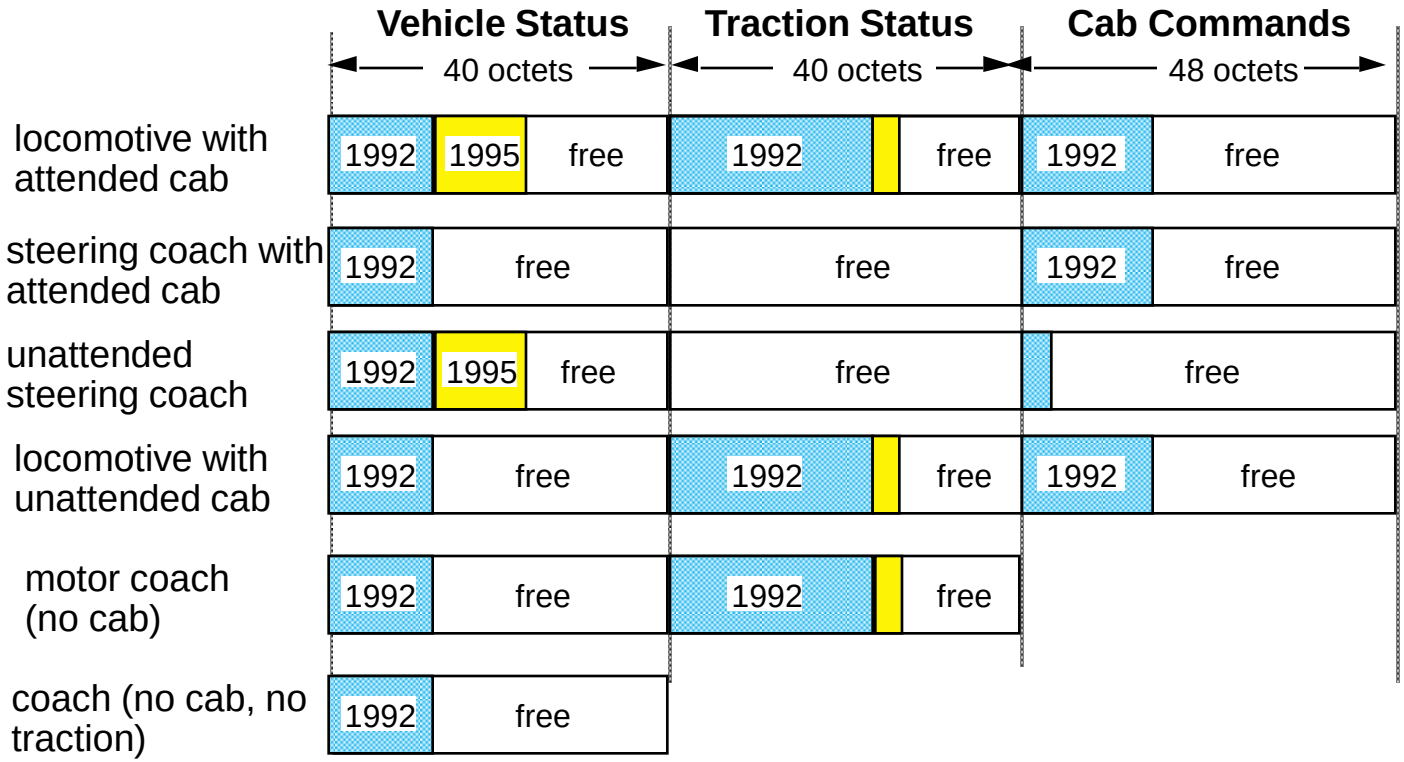
To signal that a variable is invalid, the producer overwrites the variable with "0".

Since both an "all 1" and an "all 0" word can be a meaningful combination, each variable can be supervised by a check variable, of type ANTIVALENT2:




A variable and its check variable are treated indivisibly when reading or writing
The check variable may be located anywhere in the same data set.

# Example: Train Bus Frames Extension

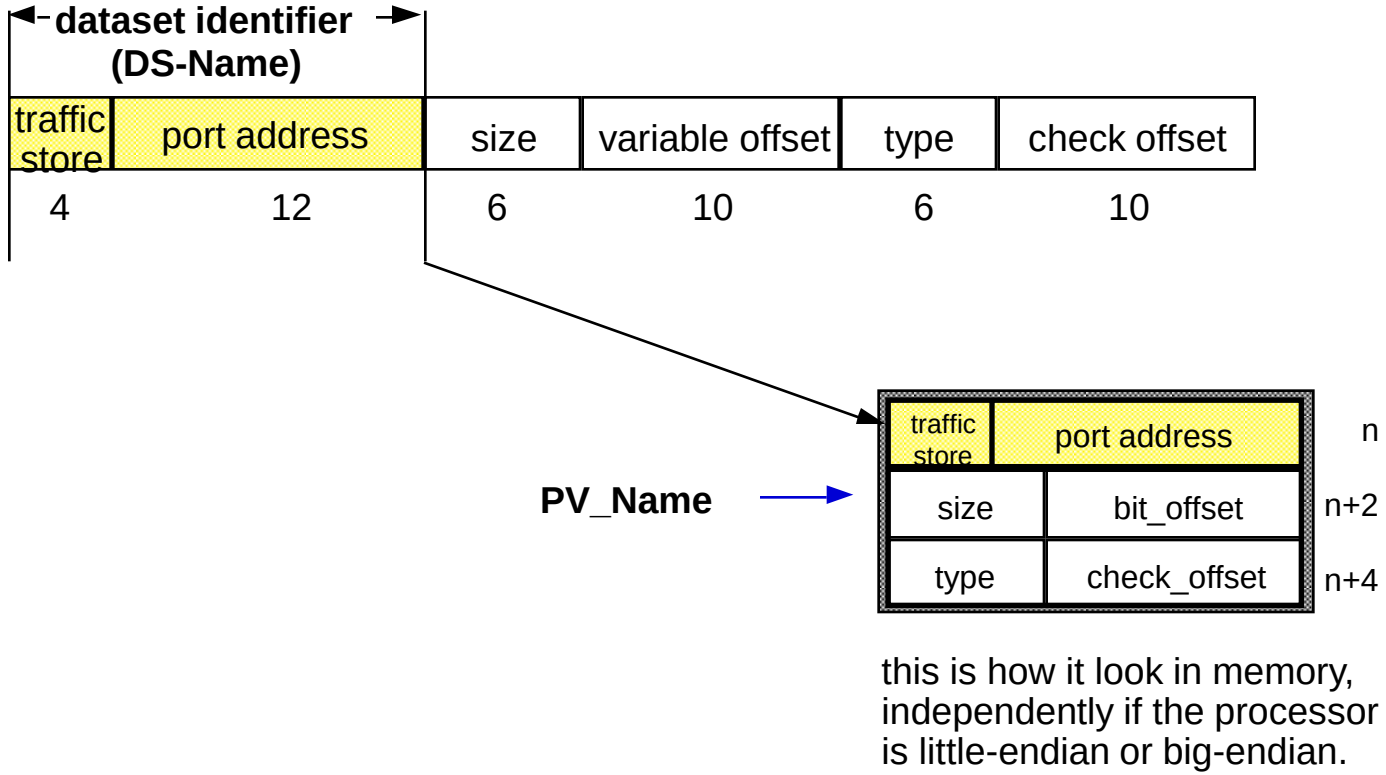



old vehicles understand the new vehicle's 1992 data and ignore 1995 variables.

new vehicles understand the old vehicle's 1992 data and ignore undefined fields.

# PV_Name

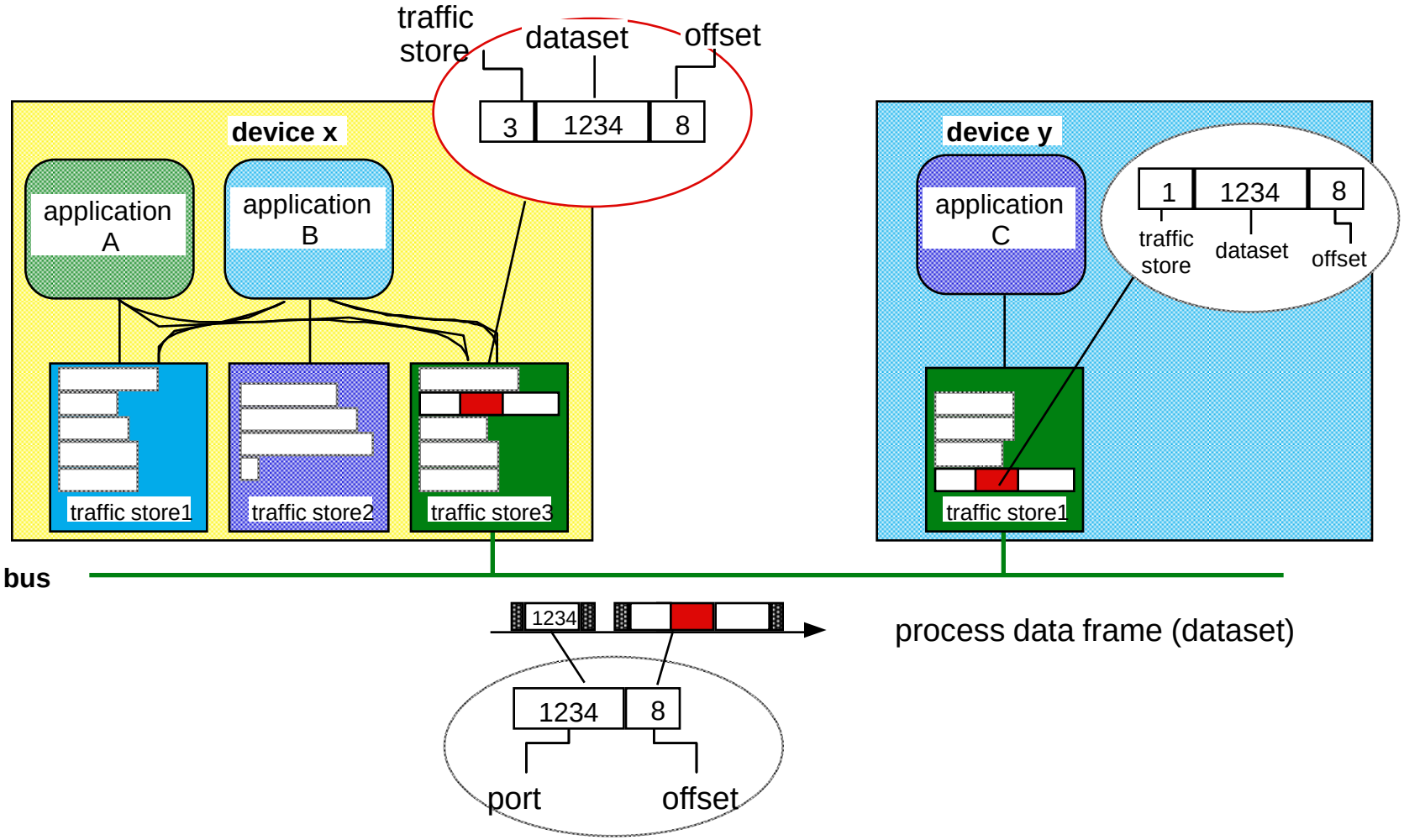
Each Process Variable is identified by a unique PV_Name




To each variable, the application associates a value location and a check location

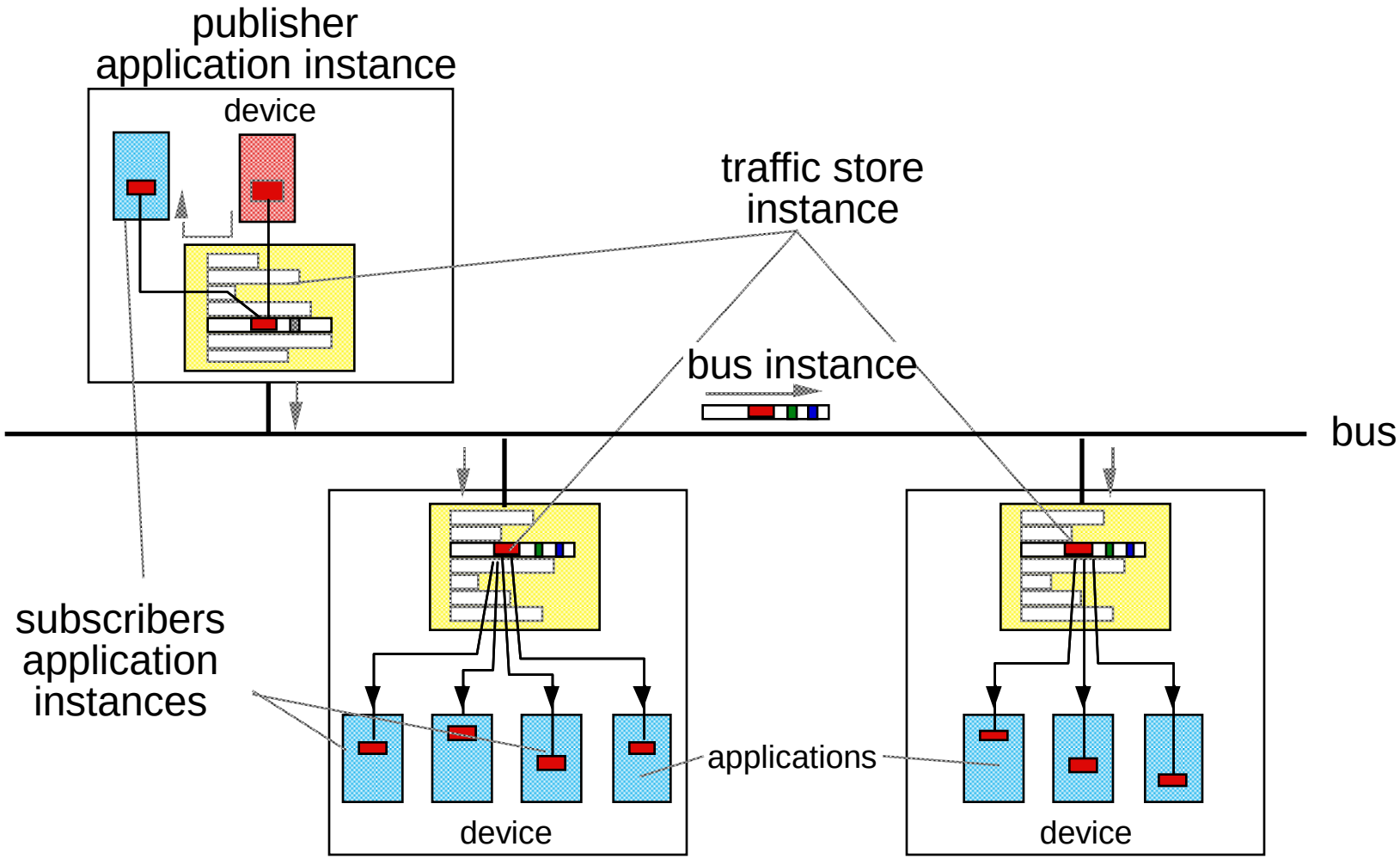
Sink variables have in addition the freshness argument

# Scope Of The PV-name




the port identifier and the offset within the port identifies a variable within a bus or within a traffic store

# Process Variables Instances




There exist a number of (inconsistent) instances of the same Process Variable

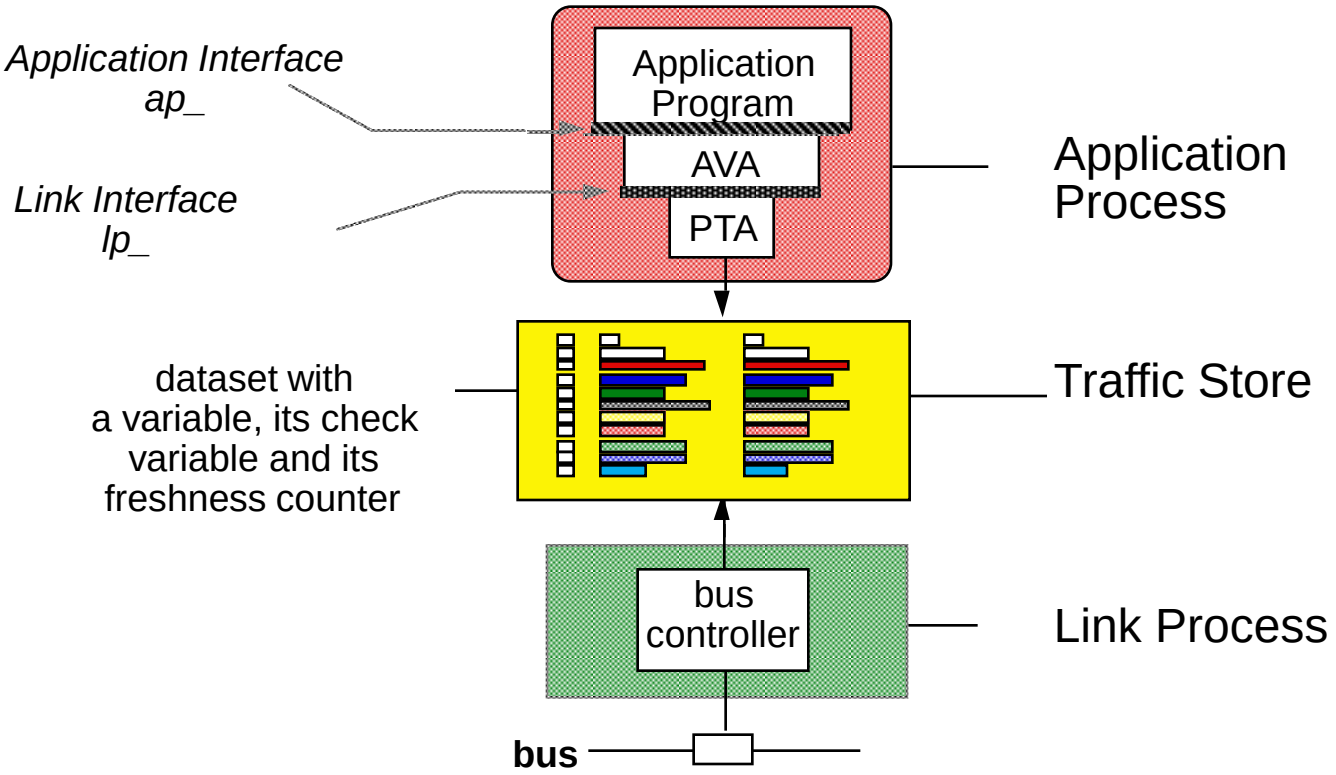
# Software Structure For Process Variables

1. General Principles

2. Variables
    1. Principle of cyclic Process Data broadcast
    2. Traffic Stores principle and implementation
    3. Process Variables and Datasets
    4. Software structure
    5. Application Layer Interface for Process Variables
    6. Networking

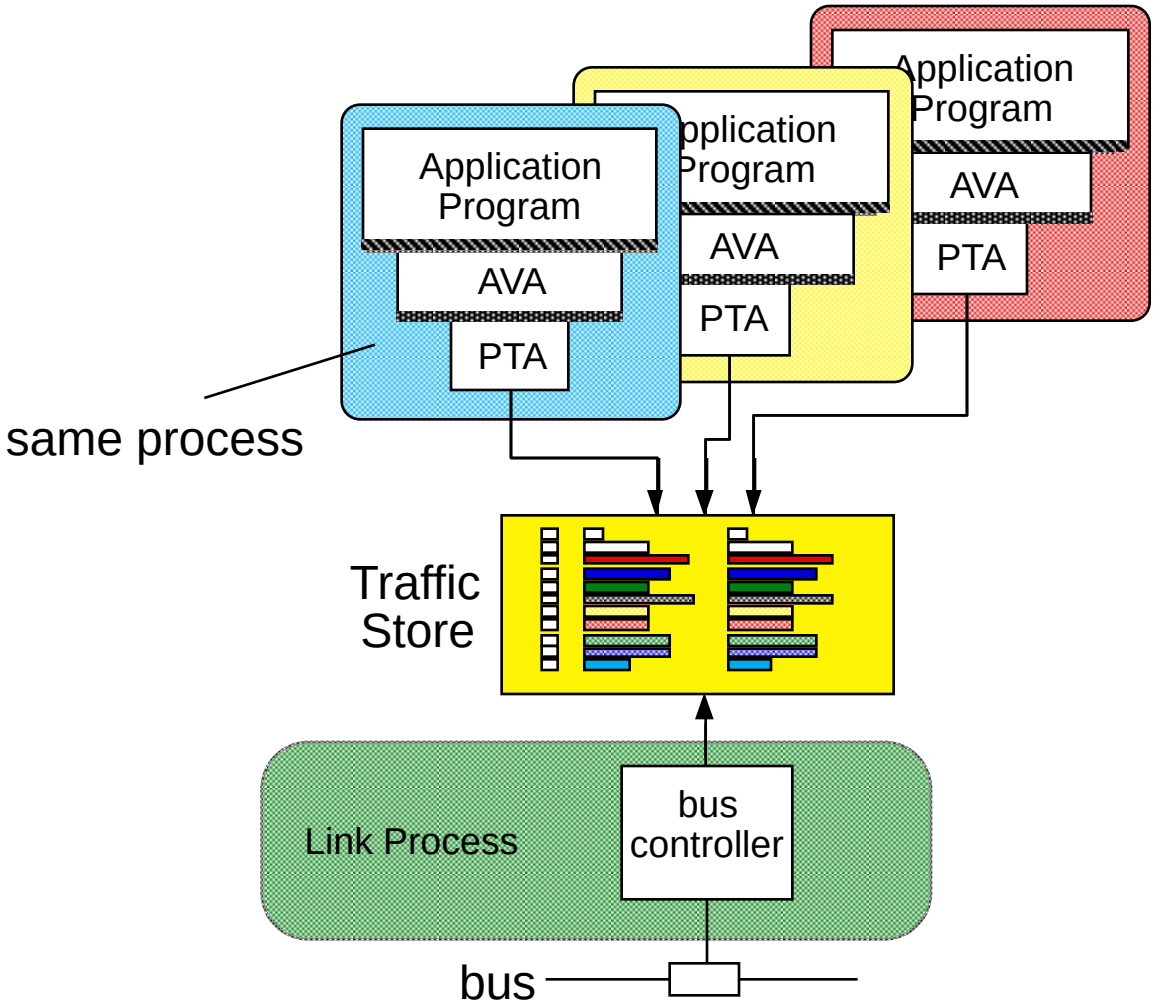
3. Messages
    1. Principle of Message Data Communication
    2. Link Layer Interface
    3. Networking and Routing
    4. Transport protocol
    5. Software structure
    6. Application Interface

# Application Access To Process Variables



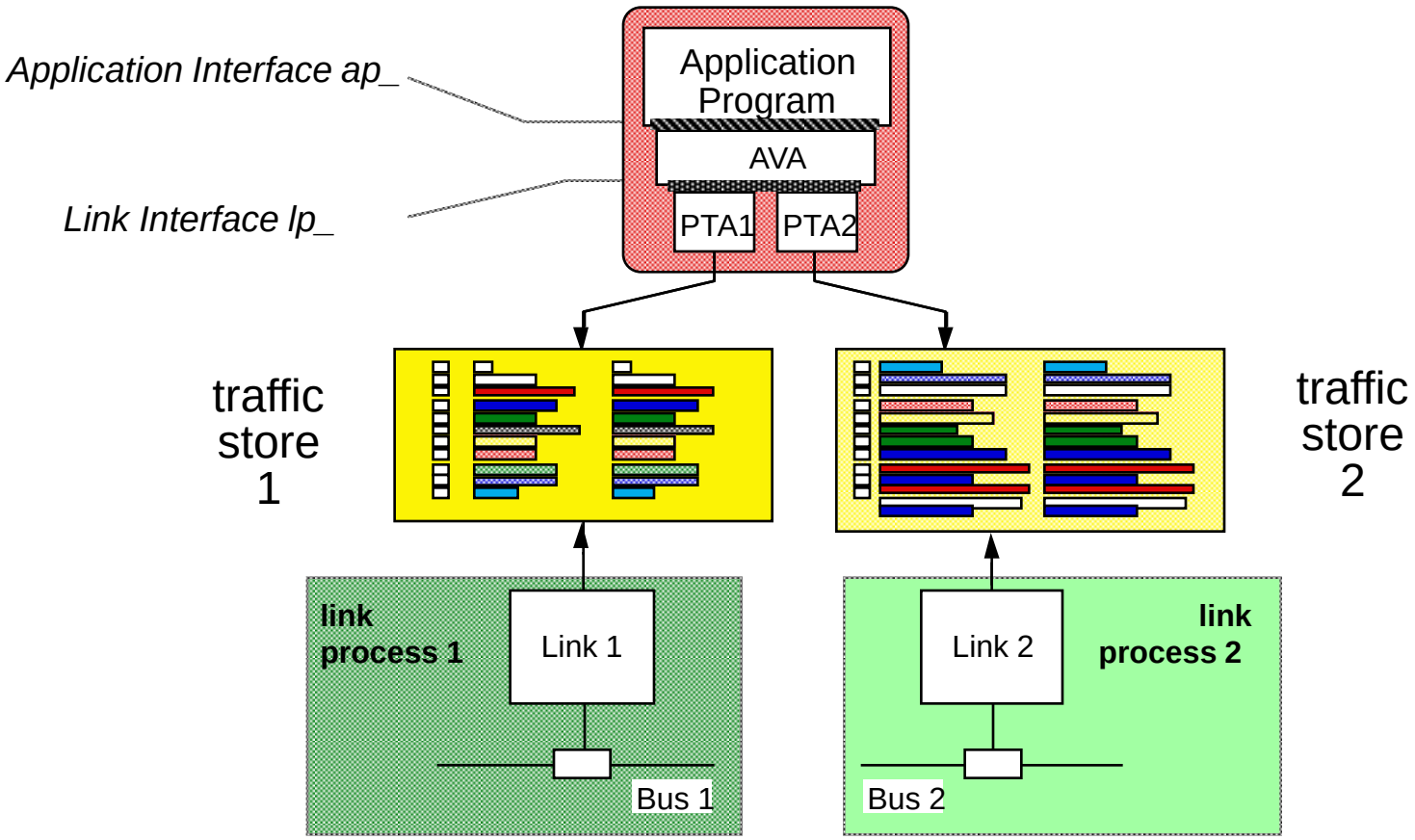

applications access the traffic store directly, as a shared memory.
(there may be only writer for a given port)

# Several Applications, One Traffic Store

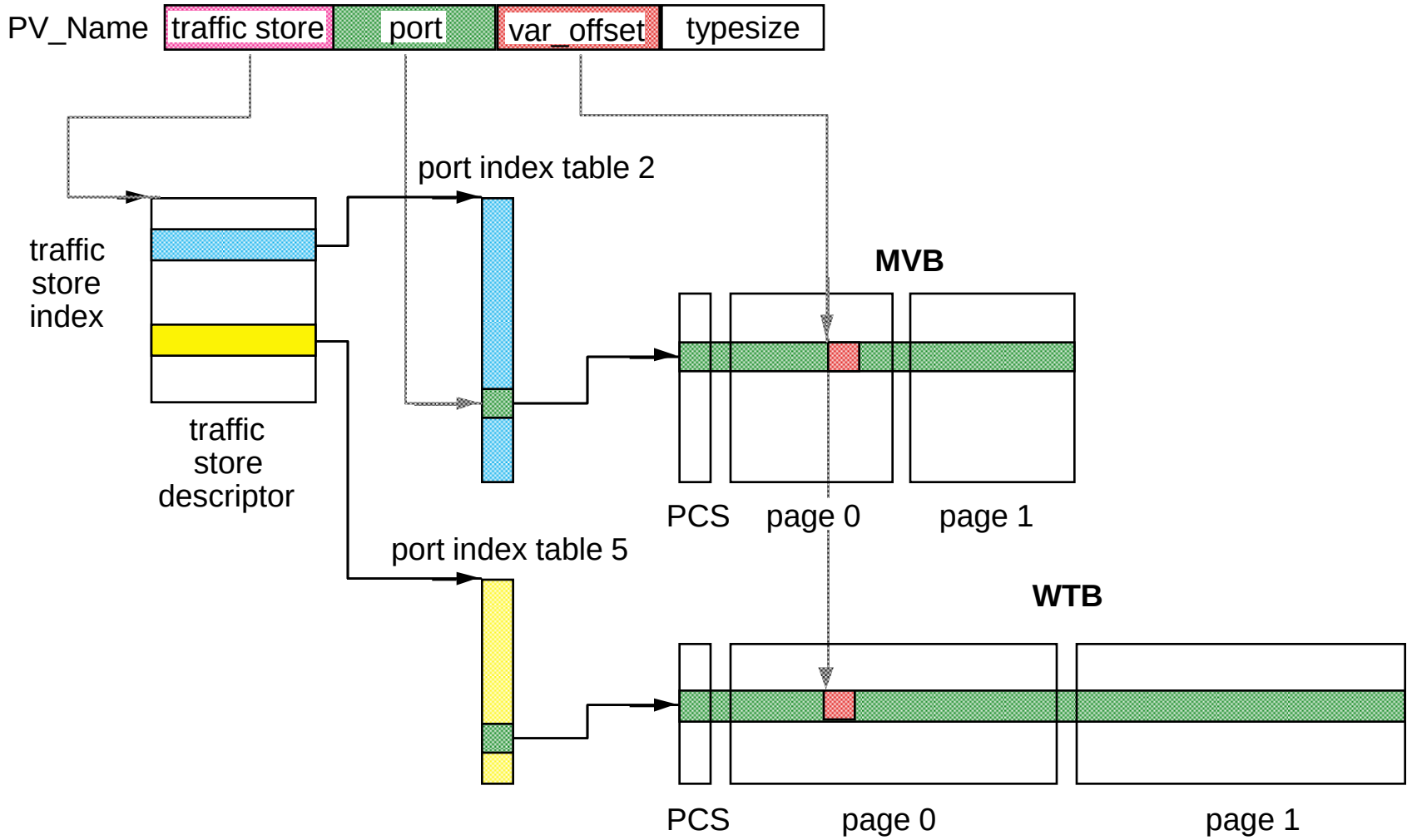



several processes may access the traffic store in parallel

# One Application, Several Traffic Stores

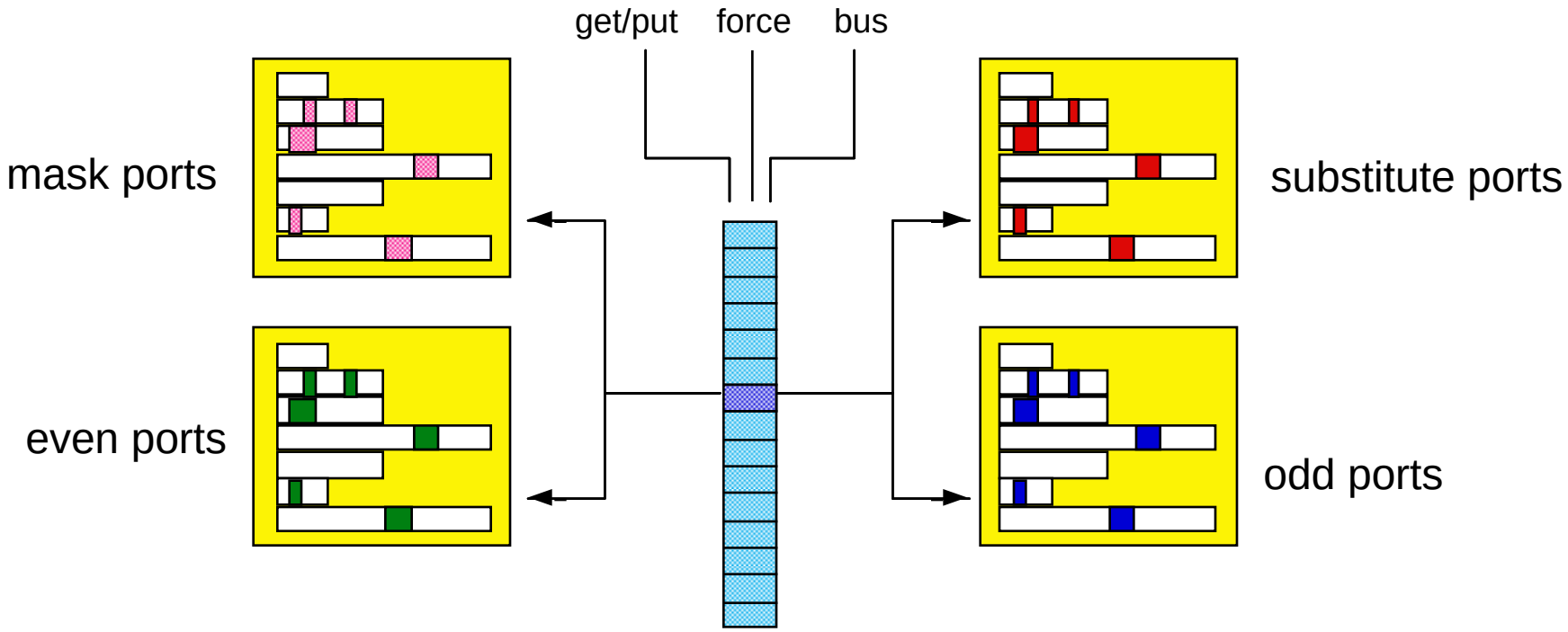


# Access To Several Traffic Stores



# Forcing

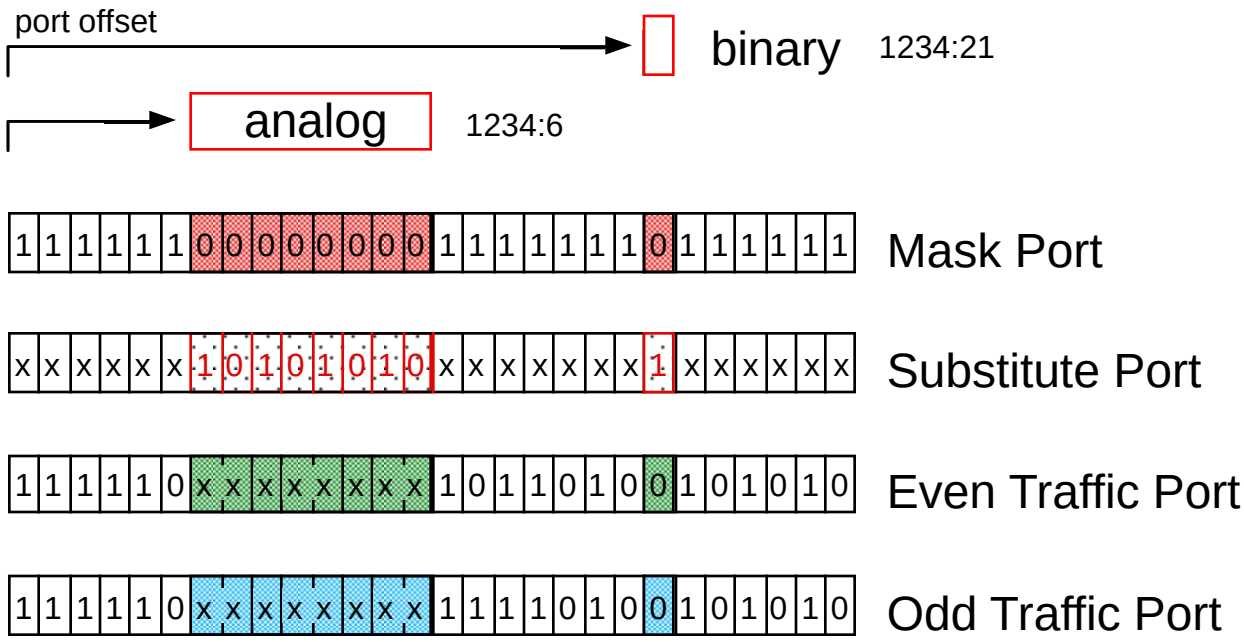
Variables can be forced individually for debugging and commissioning




if a source variable is forced, its substitute value is copied to the bus and to every (local) application which reads it

if a sink variable is forced, its substitute value is read by local applications, remote appplications are not affected

# Forcing Operation




For each bit set to 0 in the Mask Port, the corresponding bit of the Substitute port is copied to the variable location

# Application Interface For Process Variables

1. General Principles

2. Variables
    1. Principle of cyclic Process Data broadcast
    2. Traffic Stores principle and implementation
    3. Process Variables and Datasets
    4. Software structure
    5. Application Layer Interface for Process Variables
    6. Networking

3. Messages
    1. Principle of Message Data communication
    2. Link Layer Interface
    3. Networking and Routing
    4. Transport protocol
    5. Software structure
    6. Application Interface

# AVA Procedures

**individual access**

| | |
|---|---|
| apd_put_variable (pv_name, producer) | *write a variable* |
| apd_get_variable (pv_name, consumer, check) | *read a variable* |
| apd_force_variable (pv_name, substitute) | *force a variable to a value* |
| apd_unforce_variable (pv_name) | *restore bus communication* |
| apd_unforce_all | *same for all variables* |
| apd_init | *initialize traffic store* |

**cluster access**

| | |
|---|---|
| ap_put_cluster (pv_list) | *read a cluster of variables* |
| ap_get_cluster (pv_list) | *write a cluster of variables* |

**pv-set access**

| | |
|---|---|
| ap_put_set (pv_list) | *read a set of variables* |
| ap_get_set (pv_list) | *write a set of variables* |

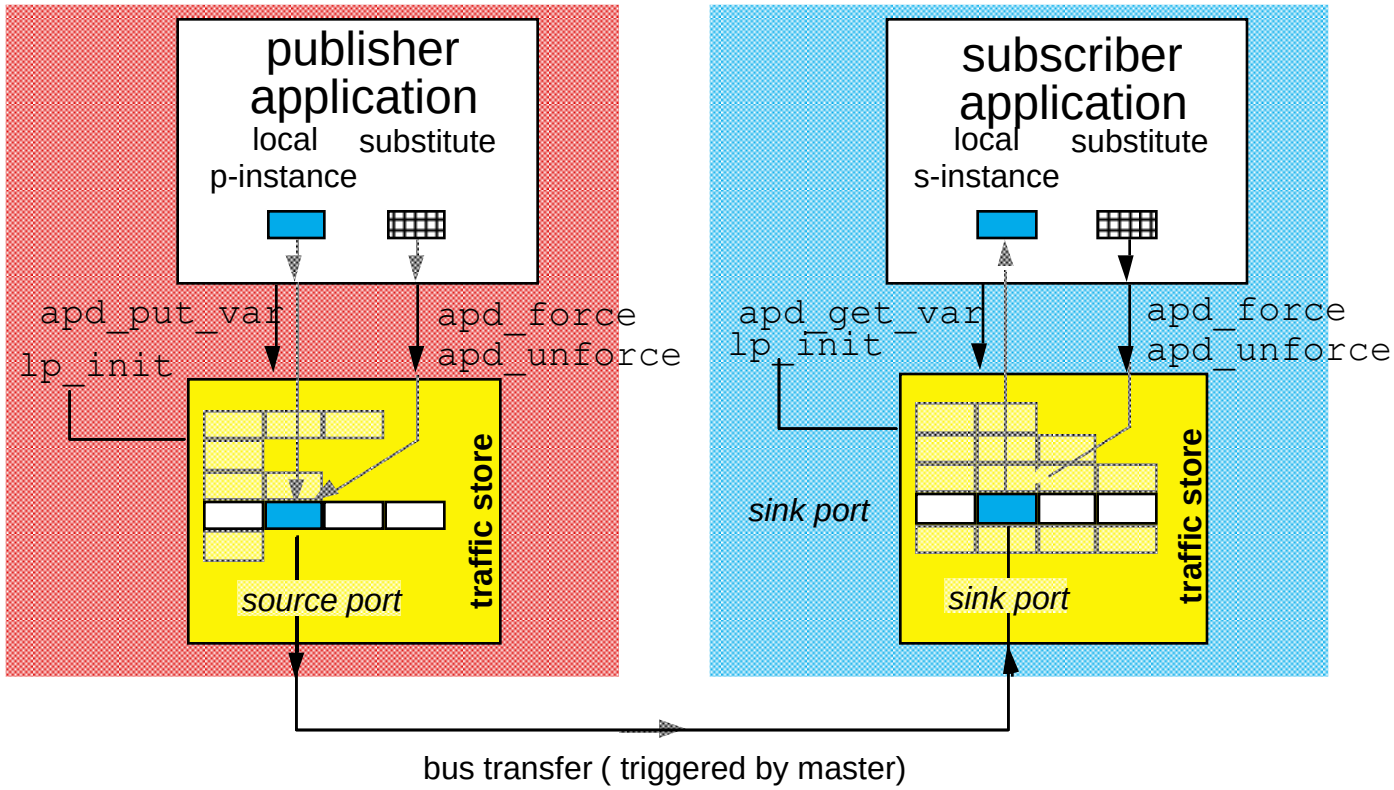
**dataset access (link layer)**

| | |
|---|---|
| ds_received (ds_name) | *signals reception of dataset* |
| ds_sent (ds_name) | *signals emission of dataset* |
| ds_subscribe_sent (ds_name, sent_ind) | *corresponding subscriptions* |
| ds_subscribe_received (ds_name, received_ind) | |

# Individual Access

Variable may be accessed individually, together with their check variable

Variable may be forced individually to a certain value (for test purpose)



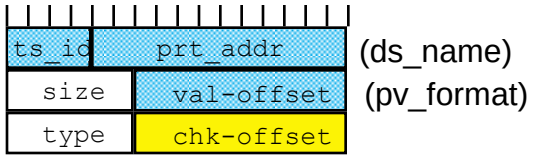

`apd_put_variable`
`apd_get_variable`

`apd_force_variable`
`apd_unforce_variable`

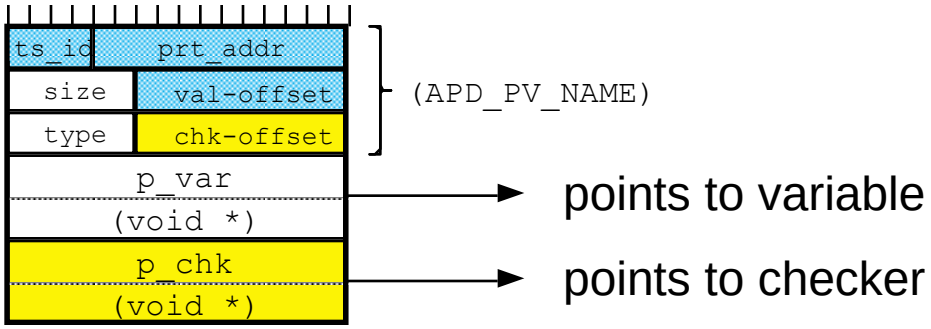
# Individual Access Procedures

## data structures

`struct STR_APD_PV_NAME (TYPE_APD_PV_NAME)`




`struct STR_APD_PV_DESCRIPTOR (TYPE_APD_PV_DESCRIPTOR)`




## procedures

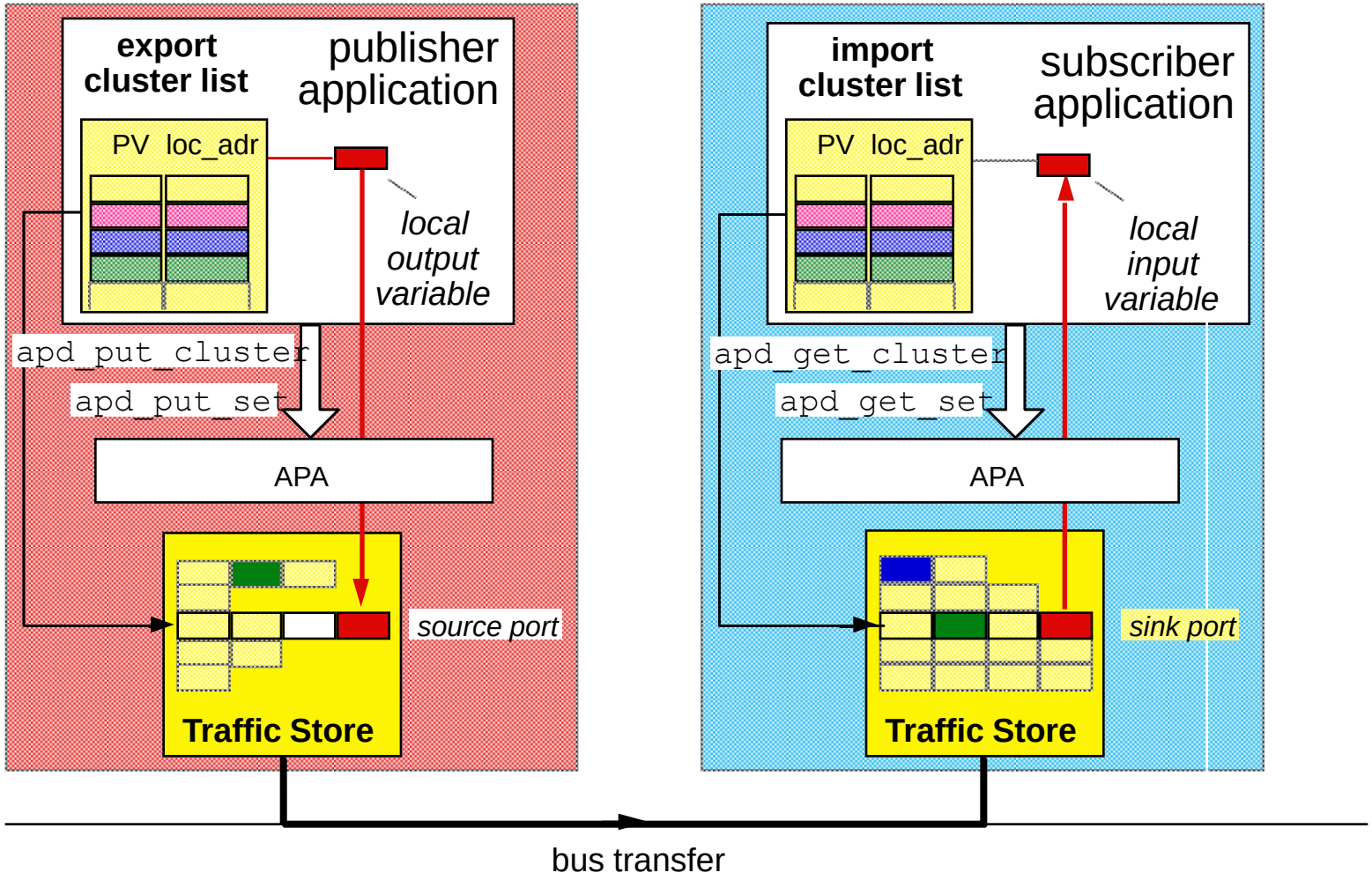
```
TYPE_APD_RESULT apd_get_variable( const struct STR_PV_DESCRIPTOR* var, unsigned short* fresh)
TYPE_APD_RESULT apd_put_variable( const struct STR_PV_DESCRIPTOR* var)
```

## example:

my_result = apd_get_variable( &apd_cpu1_b1, &frs_cpu1_t1)

# Cluster Access Procedures

It is fast and convenient to access groups of variables, from different ports.

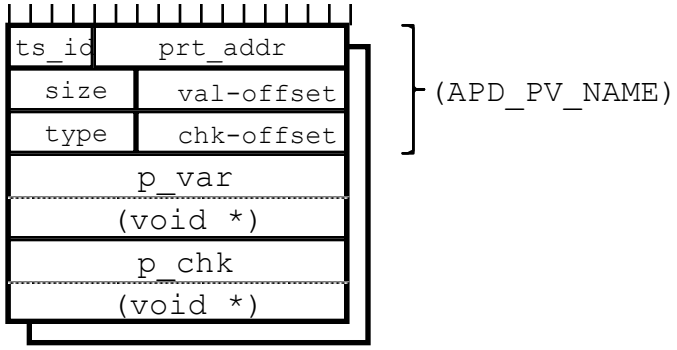


# PV_Set

A PV_set groups variables of the same port

## data structures

```
struct STR_APD_PV_DESCRIPTOR
       (TYPE_APD_PV_DESCRIPTOR)
```




**array of pv_descriptors**

## procedures

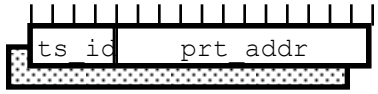
```
TYPE_APD_RESULT apd_get_set(
  const struct STR_PV_DESCRIPTOR* pvset,
  unsigned short* fresh,
  int nr_variables)

TYPE_APD_RESULT apd_put_set( const struct STR_PV_DESCRIPTOR* pvset, int nr_variables)
```

## example:

my_result = apd_get_set( &ads_cpu1_t1, &frs_cpu1_t1, 5)

# Dataset Access Procedures

```
struct STR_APD_DS_NAME
        (TYPE_APD_DS_NAME)
```




for refresh timer

array of dataset identifiers

To supervise their freshness,  data sets are grouped in a time-out set

With the BAP, the set must be explicitely refreshed.

This operation is not needed anymore with the MVBC.

# Synchronization

Ports of a traffic stores can be configured to raise an interrupt when addressed for sending or for receiving.

This feature is not intended to transmit Process Data by events, but to synchronize applications, e.g. for time distribution.

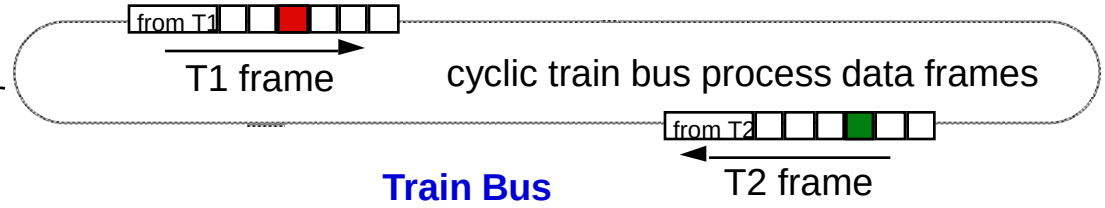
The same mechanism is used for message data

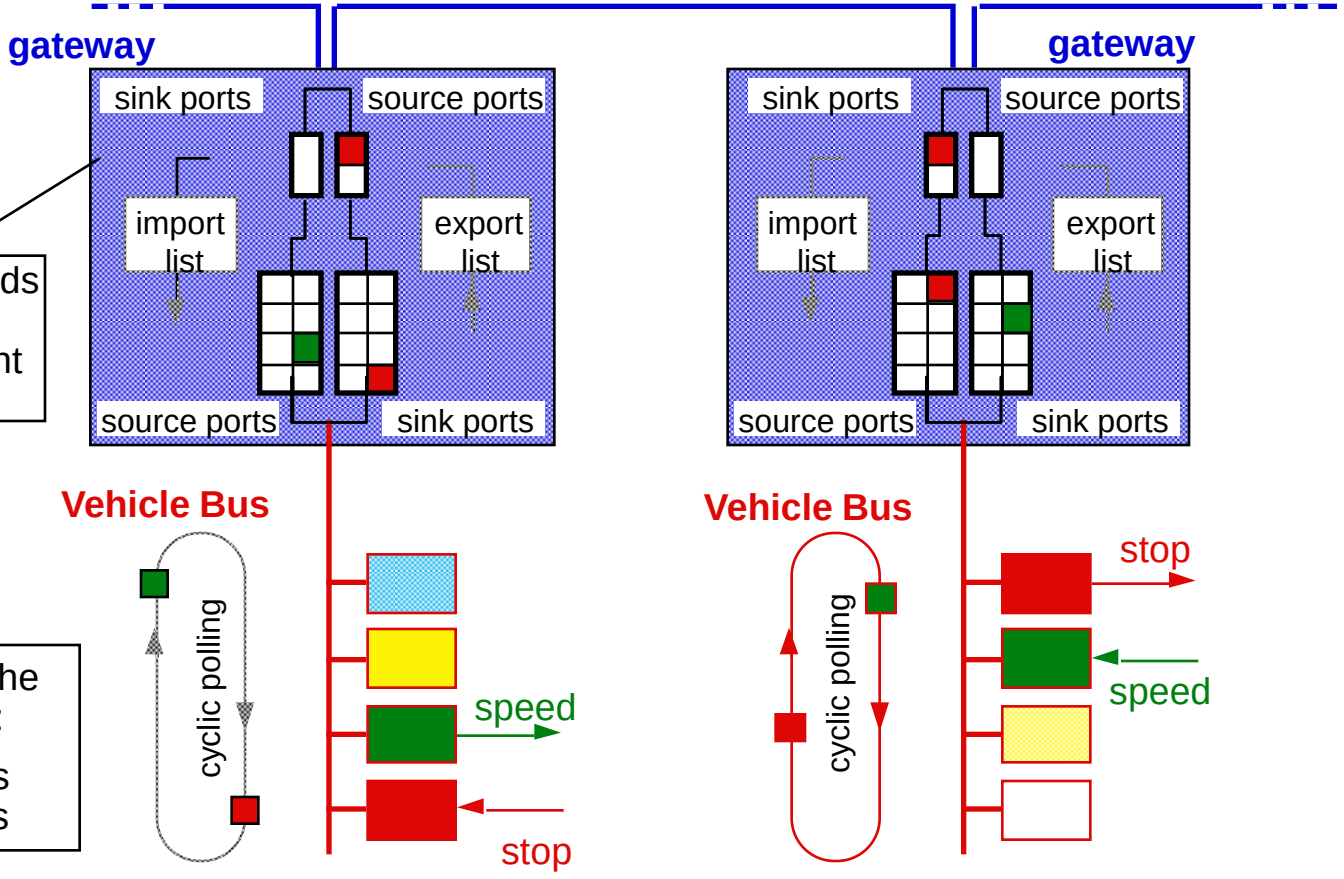
To this purpose, an application subscribes a procedure to the bus, which will be called when the port is written (or read).

| | |
|---|---|
| apd_subscribe | DS_NAME |
| apd_desubscribe | DS_NAME |
| apd_pass_subscription | DS_NAME |

# Process Data Networking

All busses operate cyclically.




gateway filters & forwards Process Data
= application-dependent marshalling

The total delay is the sum of 5 delays:

2 gateway delays and 3 bus delays