Data consistency & data store

FESA Advanced

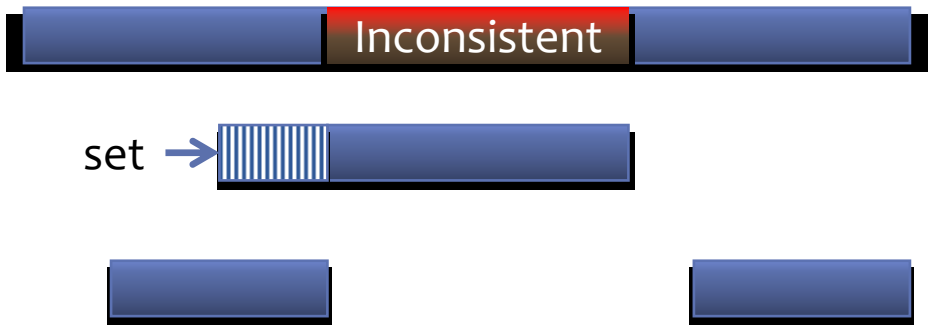
Agenda

- Problems with previous framework
- More problems with the new CPUs
- FESA 3 solutions
- Fields types
- Field type decision trees

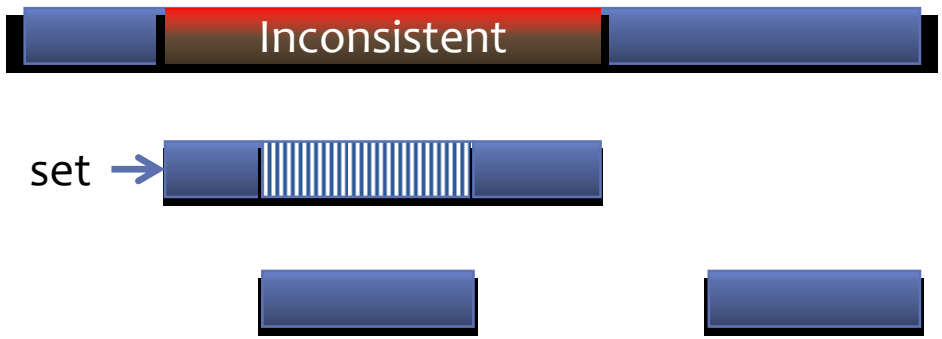
Typical priorities

- we typically give a lower prio to the server threads
- E.g. RT around 50 and server around 25
- Sometimes, we give even lower prio for some properties (FESA 2.10 background props)

Settings problem

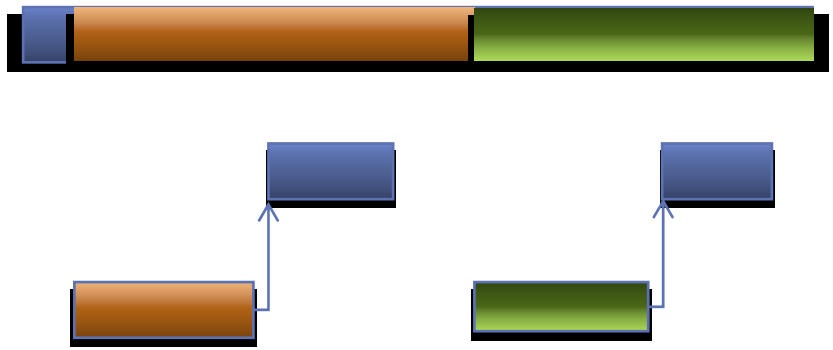


✓ The RT uses consistent settings

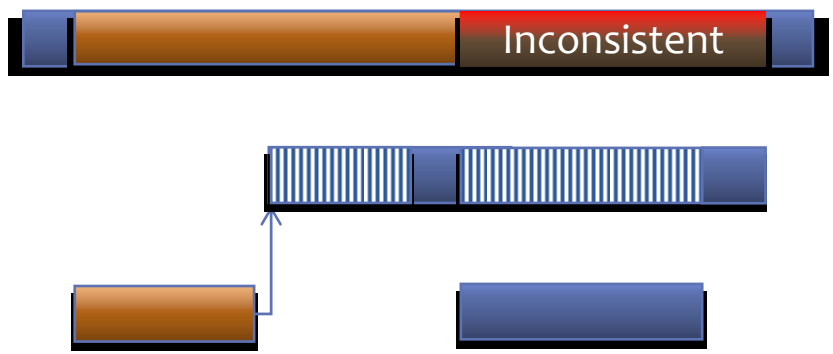


✗ The RT uses inconsistent settings

Acquisition problem



✓ The server sends consistent data



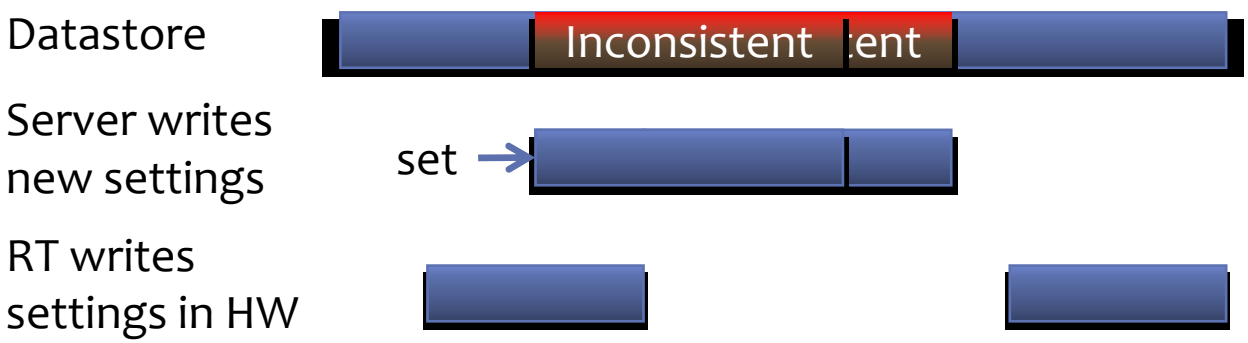
✗ The server sends inconsistent data

Agenda

- Problems with previous framework
- More problems with the new CPUs
- FESA 3 solutions
- Fields types
- Field type decision trees

New CPU
New problems

- Intel Core 2 DUO L7400 – 2 cores w/o hyperthreading
- 2 cores → 2 threads run in parallel whatever the prio



✘ The RT uses inconsistent settings

- Similar problem for the acquisition case
- Playing with prio no longer helps with data races

Agenda

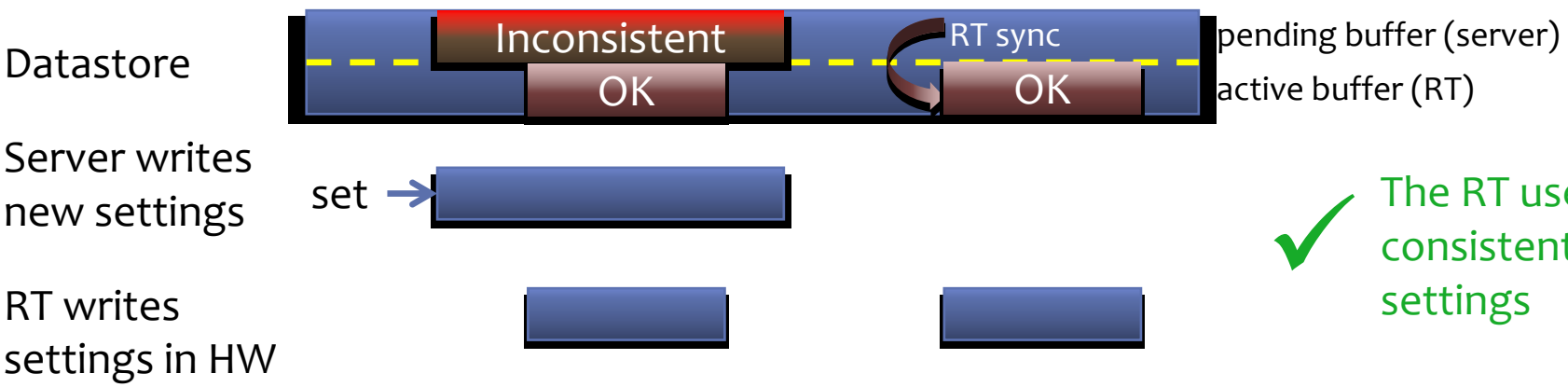
- Problems with previous framework
- More problems with the new CPUs
- FESA 3 solutions
- Fields types
- Field type decision trees

FESA 3 solutions Settings

- **Settings flow as defined in FESA 3 is**

  High-level → Server part → RT part → Hardware

- **Double-buffer mechanism to protect against data races**



- **RT sync done in the RT part**
- **Very cheap pointer swapping → very little jitter introduced**

FESA 3 solutions
Acquisition

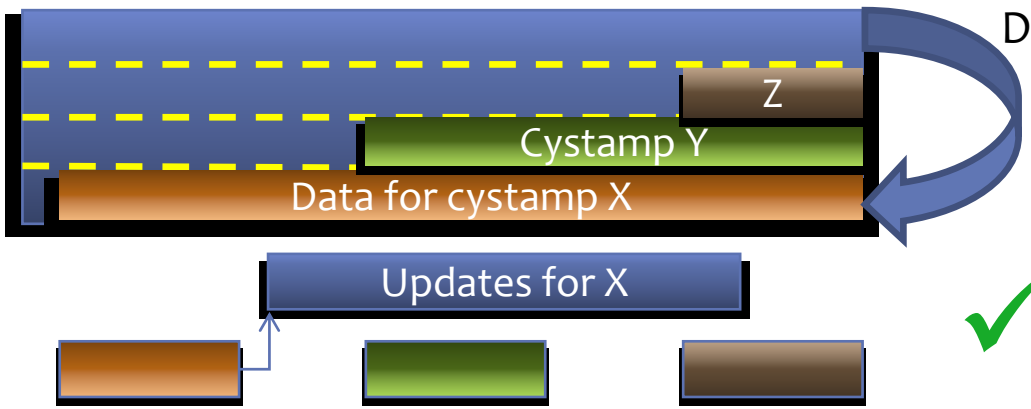
- **Acquisition flow as defined in FESA 3 is**

  Hardware → RT part → Server part → High-level

- **Rolling-buffer mechanism to protect against data races**

Datastore

Server updates clients
RT writes acquisition



The clients receive consistent acquisitions

- **Uses the cycle-stamp to select the buffer**
- **Implementation hidden → you don't have access to the other buffers**

Agenda

- Problems with previous framework
- More problems with the new CPUs
- FESA 3 solutions
- Fields types
- Field type decision trees

Field types

- There are 5 types of fields
  - Configuration
  - Setting
  - Unshared setting
  - Acquisition
  - Generic
- Configuration field
  - Read-only from anywhere
  - Default field in the configuration section

Field types

- Setting field
  - Shared and data-consistent (double-buffer)
  - Read-write from server
  - Read-only from RT
  - Default field in the setting section
  - Buffer swap by RT part
- Unshared setting field
  - Not visible from RT
  - Setting field's attribute "shared" = false

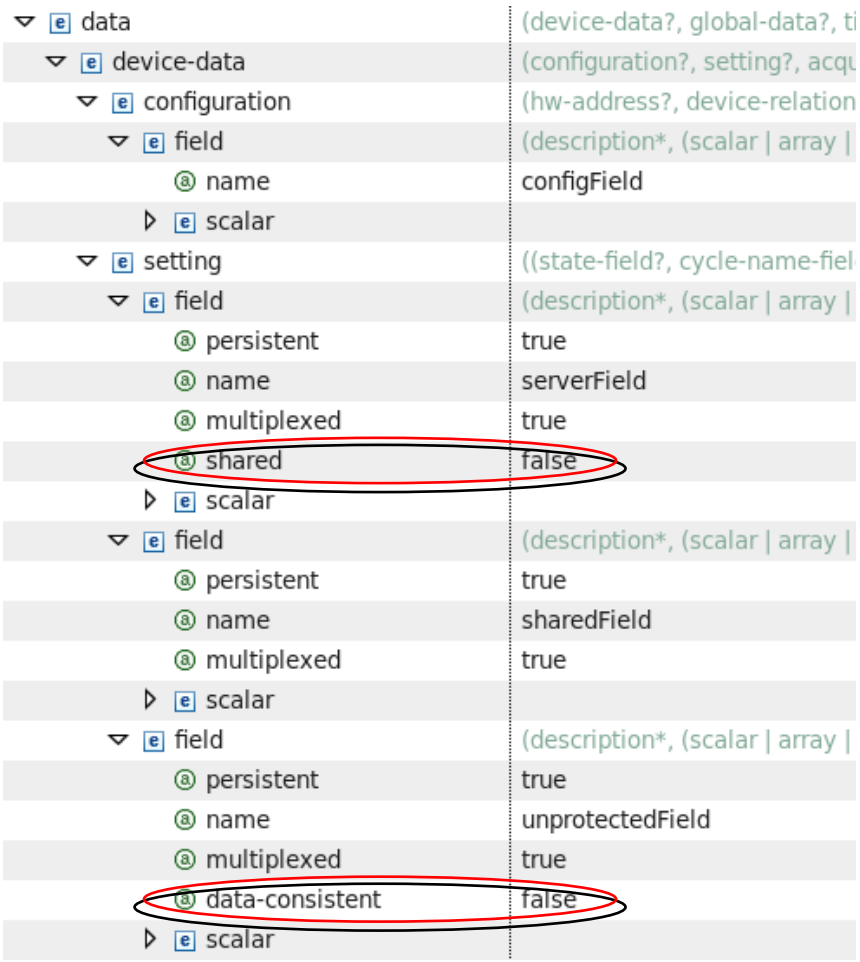
Field types

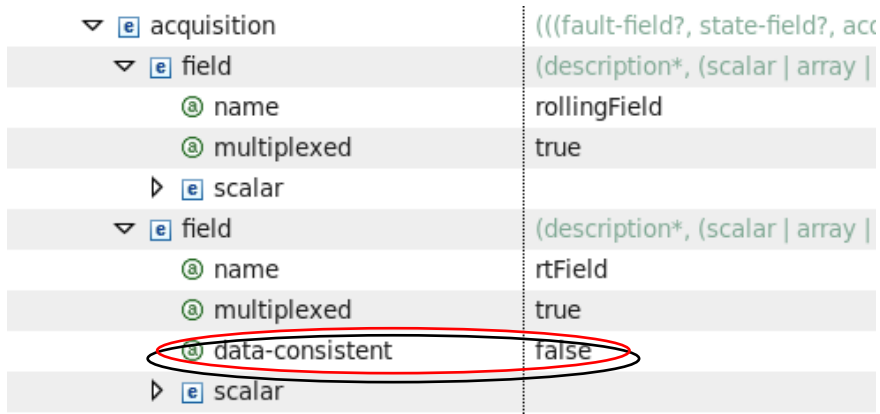
- Acquisition field
  - Shared and data-consistent (rolling-buffer)
  - Read-only from server
  - Read-write from RT
  - Default field in the acquisition section
  - Works with the cycle-stamp in the context
- Generic field
  - Accessible from anywhere and without any protection
  - Any field's attribute "data-consistent" = false
  - Should be used for private fields only

An example

- 6 fields – all int32_t
- 1 configuration field – configField
- 3 setting fields
  - serverField – a server only field (i.e. not shared)
  - sharedField – a double-buffer field
  - unprotectedField – a generic field w/o protection
- 2 acquisition fields
  - rollingField – a rolling-buffer field
  - rtField – a generic field w/o protection (RT use only)

An example Design





An example device.h


```
class Device : public fesa::AbstractDevice
{
public:
	Device();

	ConfigFieldScalar<int32_t> configField;
	SettingFieldScalar<int32_t> serverField;
	SettingFieldScalar<int32_t> sharedField;
	GenericFieldScalar<int32_t>
unprotectedField;
	AcqFieldScalar<int32_t> rollingField;
	GenericFieldScalar<int32_t> rtField;
};
```

An example Device constructor


```
Device::Device() :
   configField("configField",this),
   serverField("serverField",true, false, this, true, false),
   sharedField("sharedField",true, false, this, true, true),
   unprotectedField("unprotectedField",true, this, true),
   rollingField("rollingField",true, this, false),
   rtField("rtField",true, this, false)
{
}
```

Note: double-buffer fields and unshared fields have same impl.
Only the last arg is different

Rolling buffer depth configuration

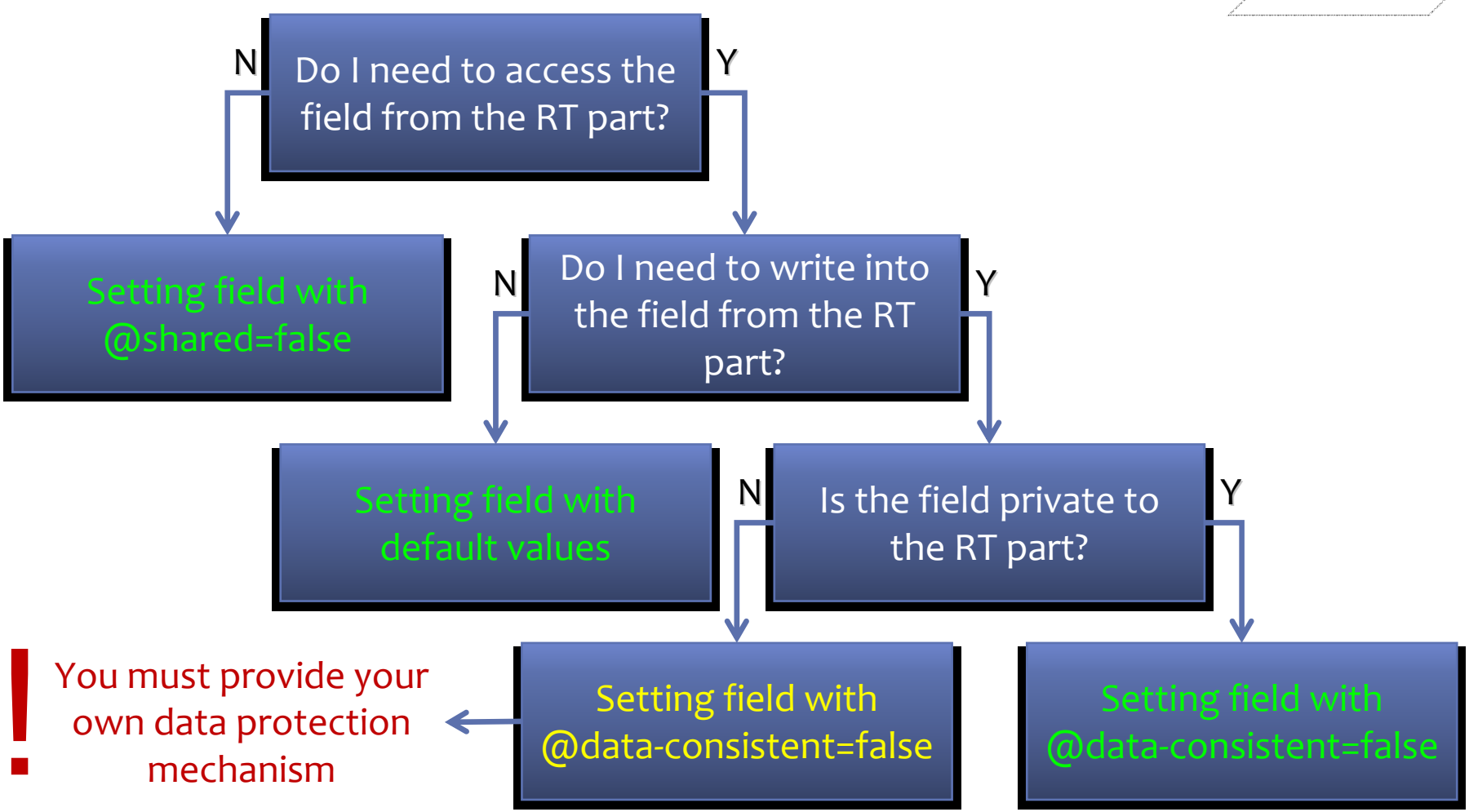
- Rolling buffer's size in the instantiation file
- Minimum 3 slots
- Maximum deps on the FEC memory

classes	(DatastoreExample)
DatastoreExample	(rolling-buffer, device-instance*, global-instance)
rolling-buffer	
depth	10
device-instance	(configuration)
global-instance	(configuration)

Agenda

- Problems with previous framework
- More problems with the new CPUs
- FESA 3 solutions
- Fields types
- Field type decision trees

Which setting field?



Which acquisition field?