Distributed Processing Environment (DPE)

1. Overview
2. Basic concepts
3. Use Cases
4. Introduction to Capsule Management
5. Introduction to Execution Management
6. Equipment Management
7. Introduction to Function Distribution Management
8. Introduction to State Register
9. Software Management
10. Introduction to Checkpointing and Activation of a Software Configuration

Building the Information Society

Copyright © 2006 TietoEnator Corporation

Distributed Processing Environment (DPE)

1. Overview

- General
- System requirements
- System views
- Application design

After the course the participant shall have received a brief description of the DPE used in WPP

Copyright © 2006 TietoEnator Corporation

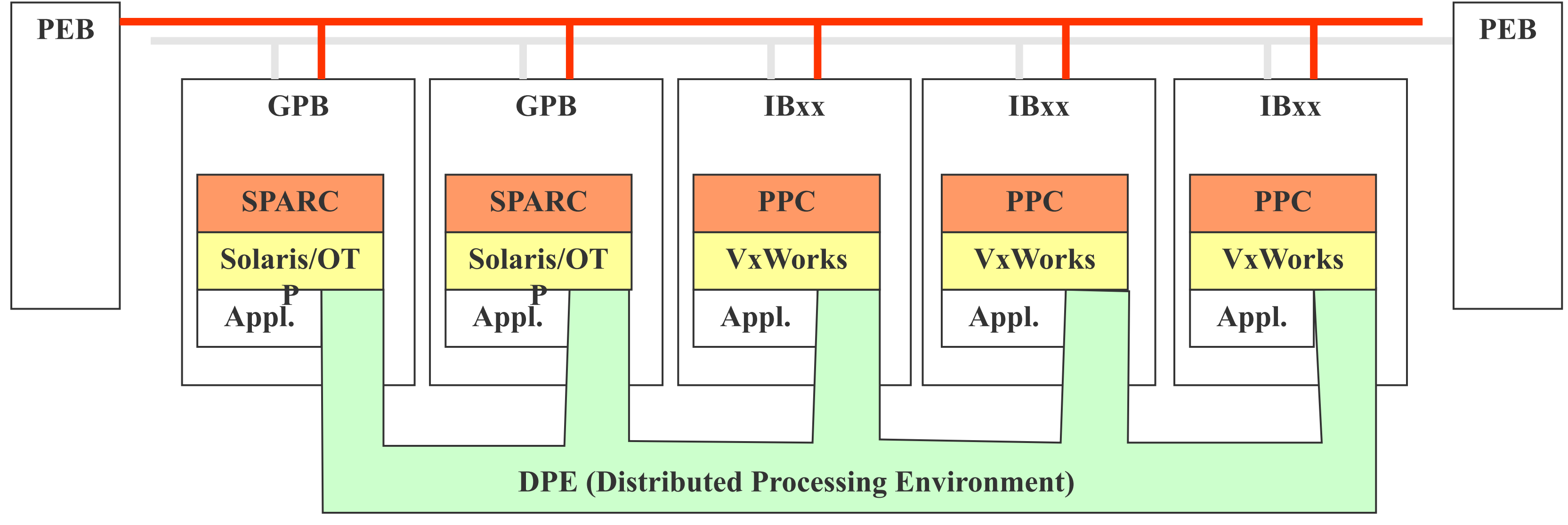
General

What DPE is all about...

- developing applications for a distributed platform
- continuous, reliable, robust operation
- a generic system
- “plug and play”

Copyright © 2006 TietoEnator Corporation

Distributed Processing Environment - DPE



- DPE supports distribution of the applications on the PIUs in the node
- Gives support for a continues, reliable and robust service even during hardware failures
- Supports ”plug-and-play” functionality
- Gives support for distributed applications

Building the Information Society

Copyright © 2006 TietoEnator Corporation

DPE Services

Service	Description
▪ State register	Synchronization and publication of data
▪ Software management	Installation of Software
▪ Function distribution management	Distribution of Software on Node Hardware
▪ Equipment management	Discover, supervise and shutdown of hardware
▪ Node management	Start, upgrade and shutdown of node
▪ Capsule management	Creation and supervision of capsules
▪ Execution management	Creation and supervision of block instances in capsules

Copyright © 2006 TietoEnator Corporation

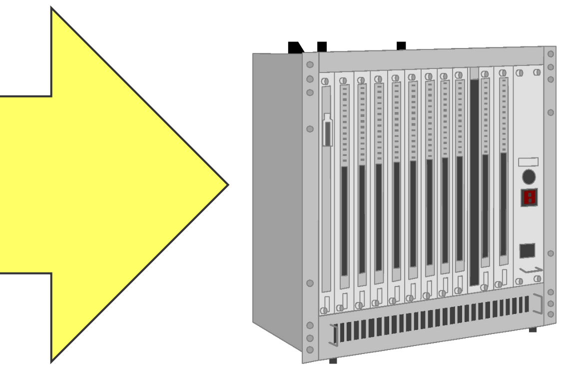
A node from a DPE perspective

SS7, Routing, PPDC, Node Dump, GPRS, OMS, ...			Applications
DPE			
Solaris	VxWorks	...	Operating systems
Sparc	PPC	...	Hardware

Copyright © 2006 TietoEnator Corporation

System Requirements

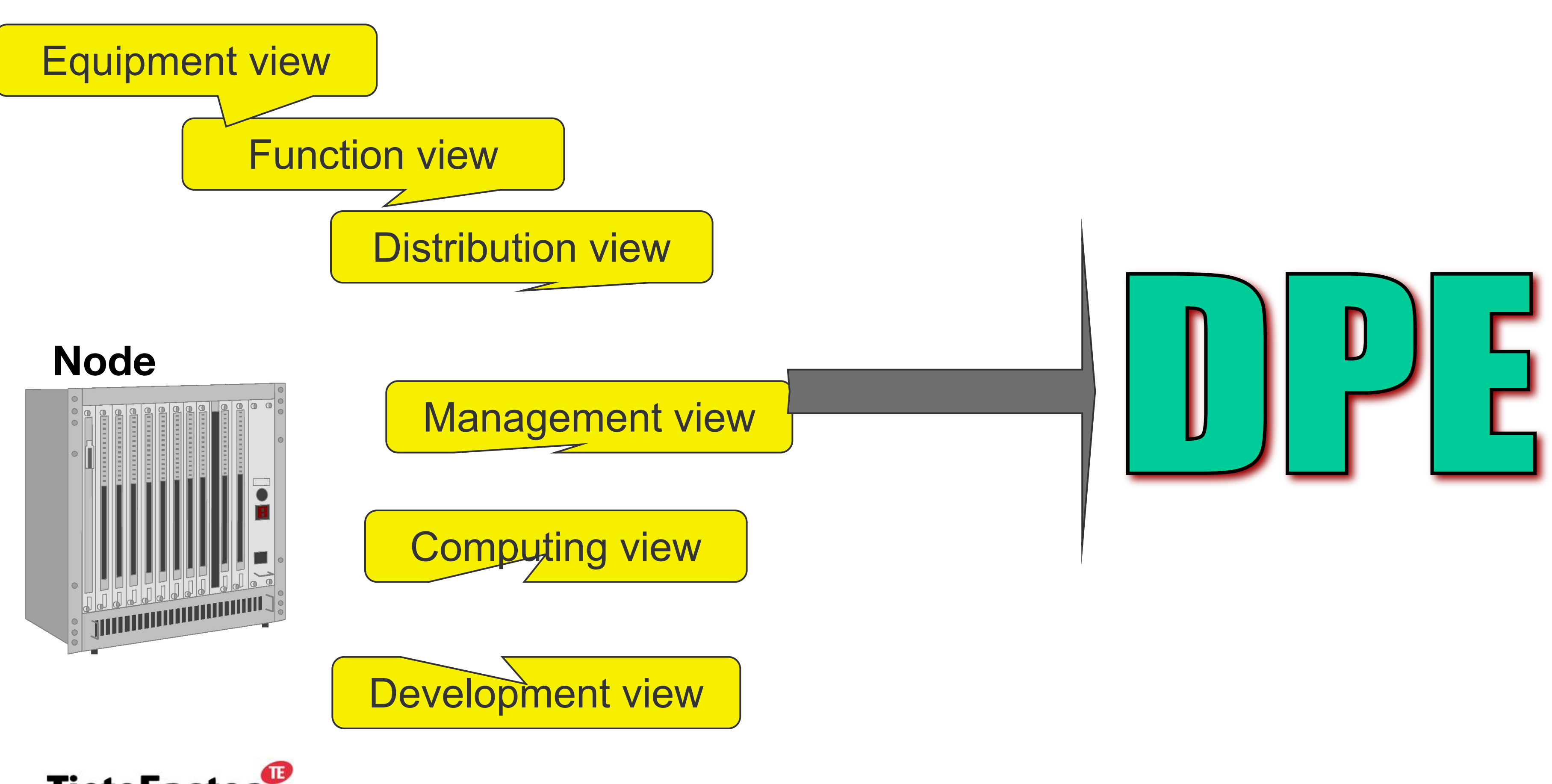
- High availability
- Follow evolution (HW & SW)
- Scalability
- "Plug & Play"
- Management for **one** system, not a set of boards



Building the Information Society

Copyright © 2006 TietoEnator Corporation

System views



TietoEnator
Building the Information Society

Copyright © 2006 TietoEnator Corporation

Equipment view

Equipment view

Function view

Distribution view

Node

Management view

Computing view

Development view

- A set of Plug-in-units (boards)
- Each Plug-in-unit may house several processors
- Plug-in-units may be removed/inserted while the Node is in operation
- Prepared & unprepared removal

Copyright © 2006 TietoEnator Corporation

Function view

Equipment view

Function view

Distribution view

Node

Management view

Computing view

Development view

- The full Node functionality may be broken down into smaller pieces : Function Blocks
- Reasons
  - Manage complexity
  - Different lifecycles
  - Reuse

Copyright © 2006 TietoEnator Corporation

Distribution view

Equipment view

Function view

Distribution view

Node

Management view

Computing view

Development view

- Different Function Blocks may have different distribution patterns
- Map Function on Computing resources
- Distribution may change over time, due to:
  - Redundancy and failover operations
  - Equipment administration
  - SW Upgrade/Update

Copyright © 2006 TietoEnator Corporation

Management view

Equipment view

Function view

Distribution view

Node

Management view

Computing view

Development view

- One SYSTEM
- Node level redundancy
- SW installation/activation
- Node configuration state
  - checkpoint state
  - revert to known state

Copyright © 2006 TietoEnator Corporation

Computing view

Equipment view

Function view

Distribution view

Node

Management view

Computing view

Development view

- A heterogeneous set of processors
- The processors are loosely inter-connected
- Local Operating System on each processor
- Local SW load operations - Local/Remote data source
- Local encapsulation of executing entities - Unix processes, RTOS task groups, ...

Copyright © 2006 TietoEnator Corporation

Development view

Equipment view

Function view

Distribution view

Node

Management view

Computing view

Development view

- Group related function blocks into an APPLICATION
- Different programming languages
- Integration of a set of APPLICATIONS into a full Node SW system

Copyright © 2006 TietoEnator Corporation

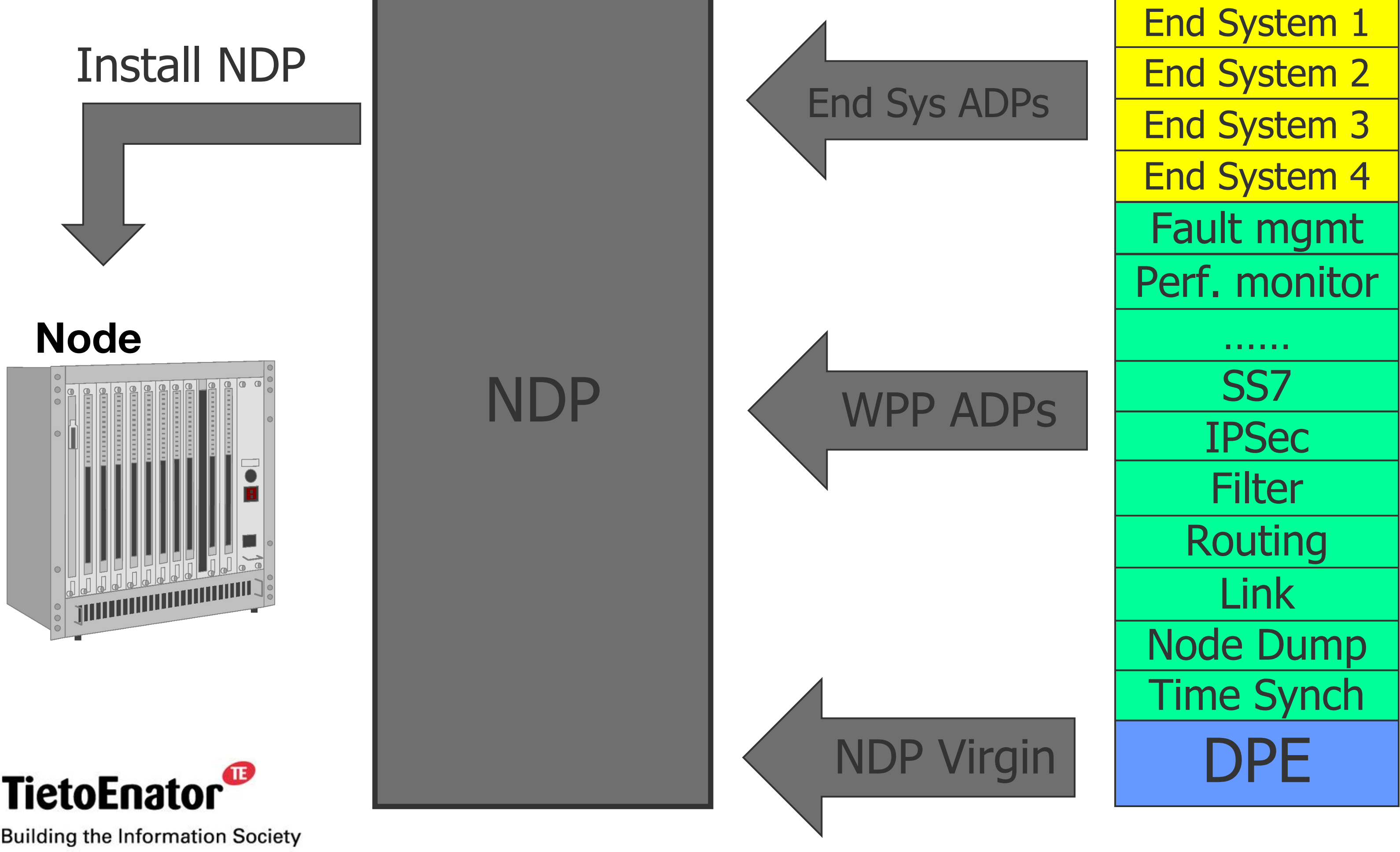
Different OS and program languages

	C	Erlang	*JAVA*
Solaris	x	x	
VxWorks	x		
Other OS			

Building the Information Society

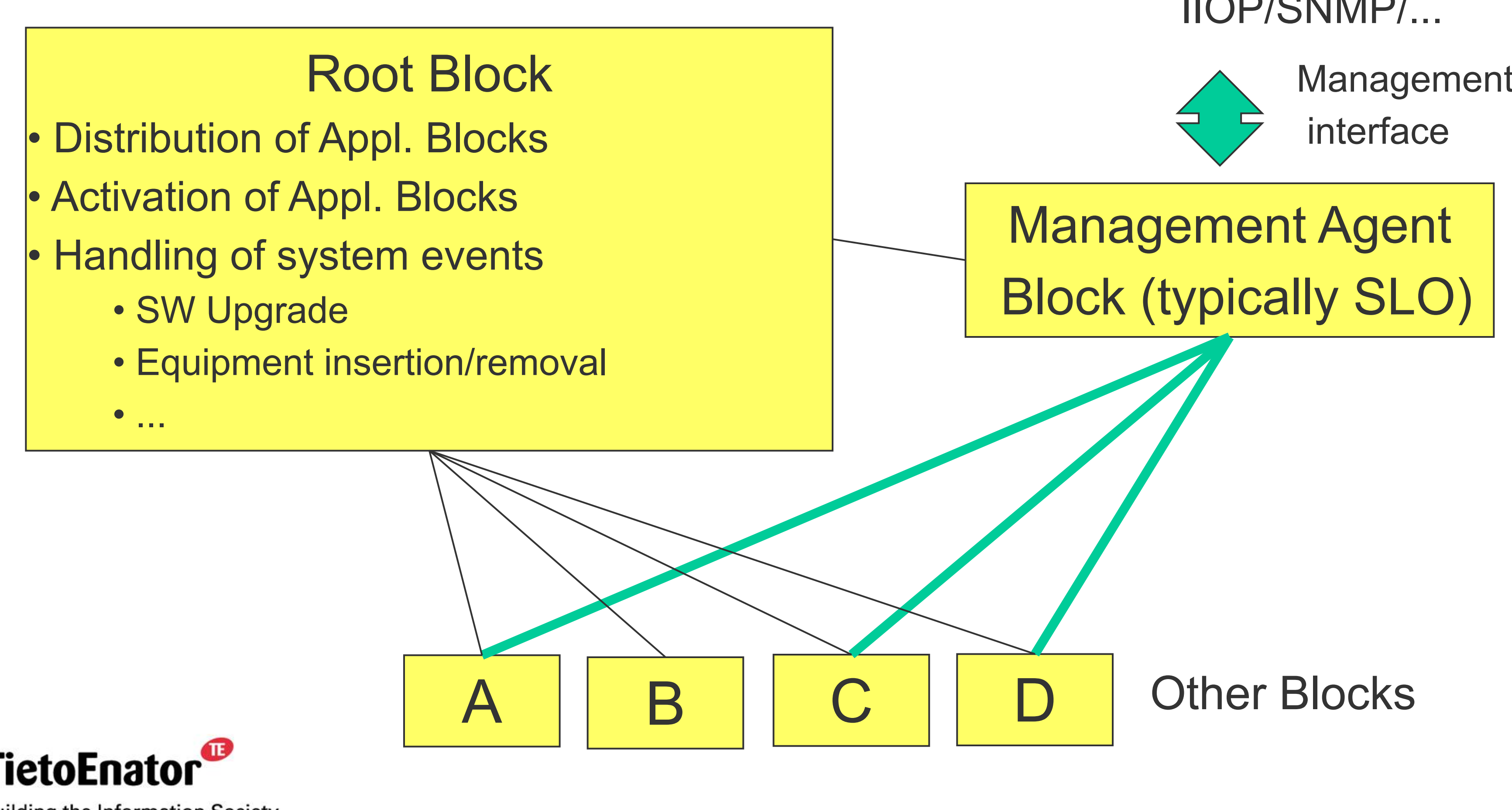
Copyright © 2006 TietoEnator Corporation

Development View, SW delivery example



Copyright © 2006 TietoEnator Corporation

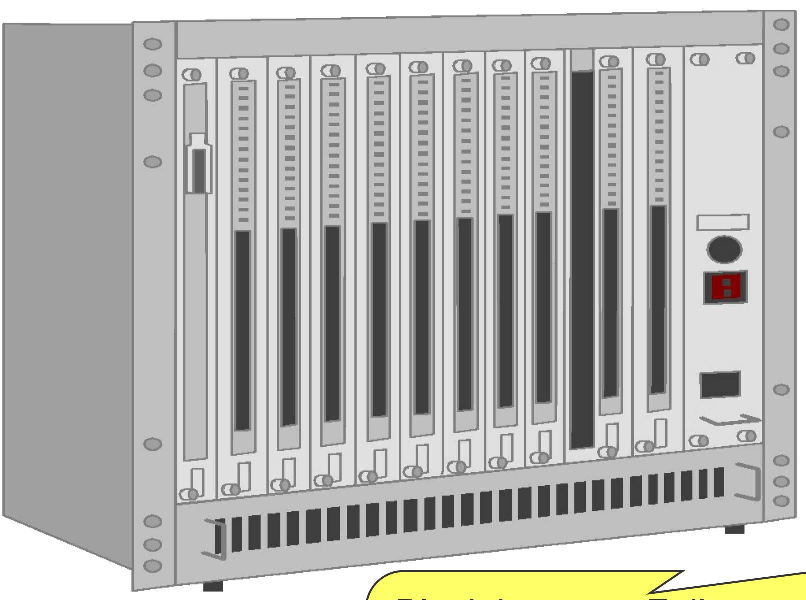
Typical Application structure



TietoEnator
Building the Information Society

Copyright © 2006 TietoEnator Corporation

System events



External events

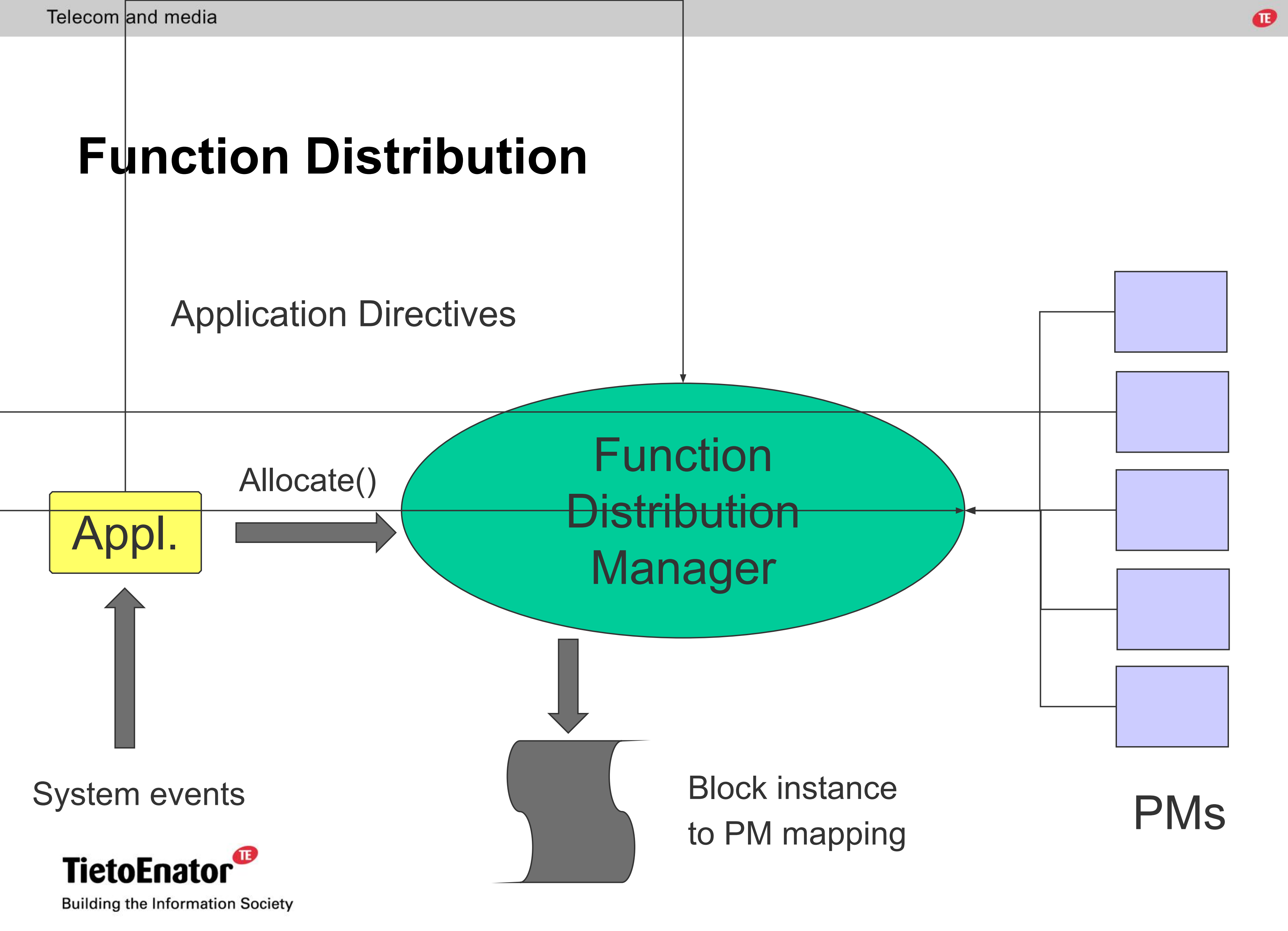
Start/Stop of system
SW Upgrade
Equipment insertion/removal
Revert to stored state

Block instance Failure,
Capsule Failure & PM Failure
Inter Appl. signals

Internal events

Copyright © 2006 TietoEnator Corporation

Function Distribution



Copyright © 2006 TietoEnator Corporation

Summary

DPE gives support for:

- Node start up/stop
- Distribution
- SW upgrade
- Node supervision
- State Register

Copyright © 2006 TietoEnator Corporation

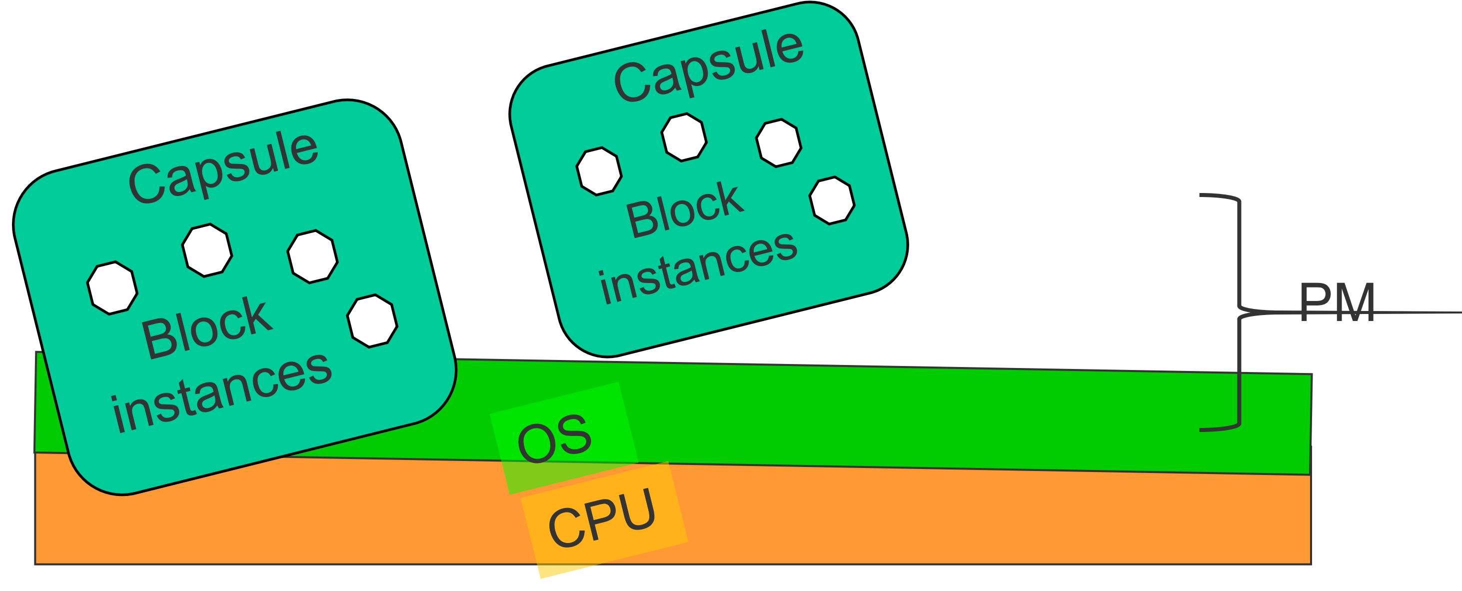
Distributed Processing Environment (DPE)

2. Basic concepts

- Processing module (*PM*)
- *Application*
  - *application instance, application structure tree (AST), application instance, application root instance.*
- *Block, block template*
- *Block instance*
- *Capsule*
- *Load unit*
- DPE architecture

Copyright © 2006 TietoEnator Corporation

Processing module (*PM*)



BI belonging to *application instance* **Application A**

BI belonging to *application instance* **Application B**

A *PM* is a hardware processor with operating system.
A *PM* can be used to run *capsules* containing *block instances.*

Building the Information Society

Copyright © 2006 TietoEnator Corporation

Application

- *Application*
  - A program that can be run within DPE.
  - An *application* may be thought of as a (structured) collection of *blocks*.
  - All *blocks* in an *application* are organized into an *Application Structure Tree* (*AST*).
- *Application instance*
  - A program running within DPE.
  - An *application instance* can be thought of as a collection of *block instances*.
  - The collection may evolve over time.

Copyright © 2006 TietoEnator Corporation

Application functionality structured into *block instances*

Root Instance App. A

Root Instance App. B

Capsule

Capsule

Capsule

Capsule

Capsule

A1 :2

B3 :3

A1 :1

B1 :1

B2 :1

B3 :2

B1 :2

B3 :1

PM1

PM2

PM3

PM4

Application A

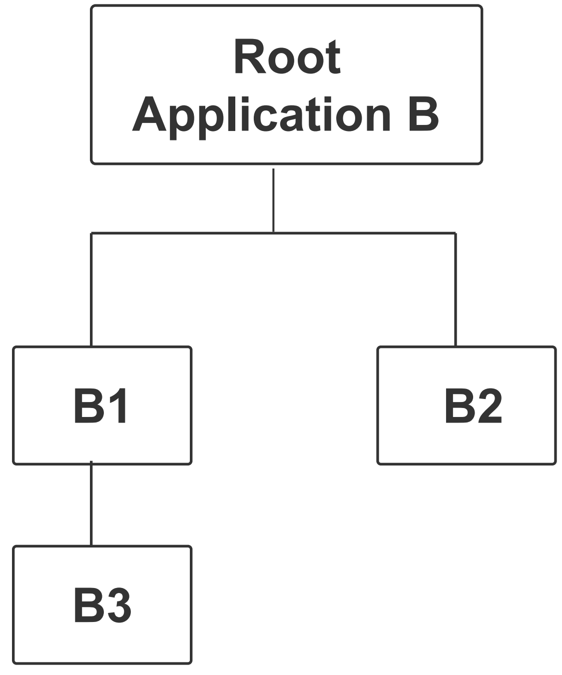
Application B

Building the Information Society

Copyright © 2006 TietoEnator Corporation

Application structure tree (AST)

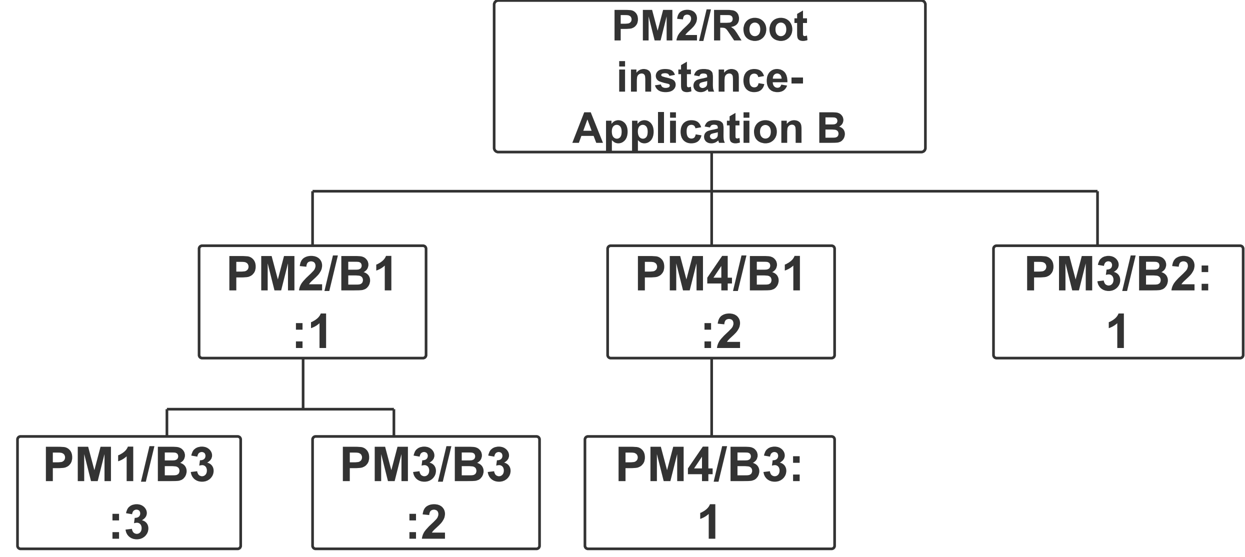
- A tree that describes the relations between *blocks* in an *application.*
- The root of a tree represents the total functionality of the *blocks* contained in that tree.
- *AST* does not change at runtime.



Copyright © 2006 TietoEnator Corporation

Application instance descriptor

- Describes the relations between *block instances* in an *application instance.*
- The root is called the *application root instance*. It is the only instance of the block in the root of an *AST.*
- The descriptor may change at runtime.



Copyright © 2006 TietoEnator Corporation

Block and *block template*

- *Block*
  - A functional unit within an *application*.
  - Smallest functional part that can be instantiated to a running entity on a particular processing module (*PM*).
- *Block template*
  - Executable code performing the functionality of a *block.*
  - Same block may have several *block templates*, each for a different type of execution environment (*Capsule*).

Copyright © 2006 TietoEnator Corporation

Implementation of a block instance in a C-Capsule on Solaris

C-Capsule

Block instance Descriptor
- Block Instance Name
- Pointer to Template
- Pointer to Data

Block Instance Data
- ……

Block Template Descriptor
- Block Name
- Block Identity
- Revision Name
- Application Name
- ...
- Function pointers to Mandatory Operations
- Function pointers to Installed Handlers

Load unit
- ………..
- Mandatory Operations
- Handlers

Copyright © 2006 TietoEnator Corporation

Block Instance

- A *block instance*
  - resides in a particular *capsule.*
  - requires that the appropriate *block template* is present in the *capsule.*
  - is created based on the *block template.*
- A *block instance* is the executing form of a *block, e.g.,*
  - an Erlang process.
- Each *block instance* (BI)
  - belongs to a particular *application instance; and*
  - has an unique identity *(block instance name)* which is not reused.

Building the Information Society

Copyright © 2006 TietoEnator Corporation

Capsule

- A special protective execution environment in which block instances can be made to run in a certain processing module (*PM*).
- Makes it possible to use uniform interfaces between DPE and *applications* despite the differences in implementation languages, operating systems etc.

Copyright © 2006 TietoEnator Corporation

Capsule types are defined by:

- Hardware (SPARC, PowerPC, etc.);
- Operating system (Solaris, VxWorks, etc.);
- Language (interpreted Erlang, C, Java, etc.); and
- Design decisions when implementing the capsule.

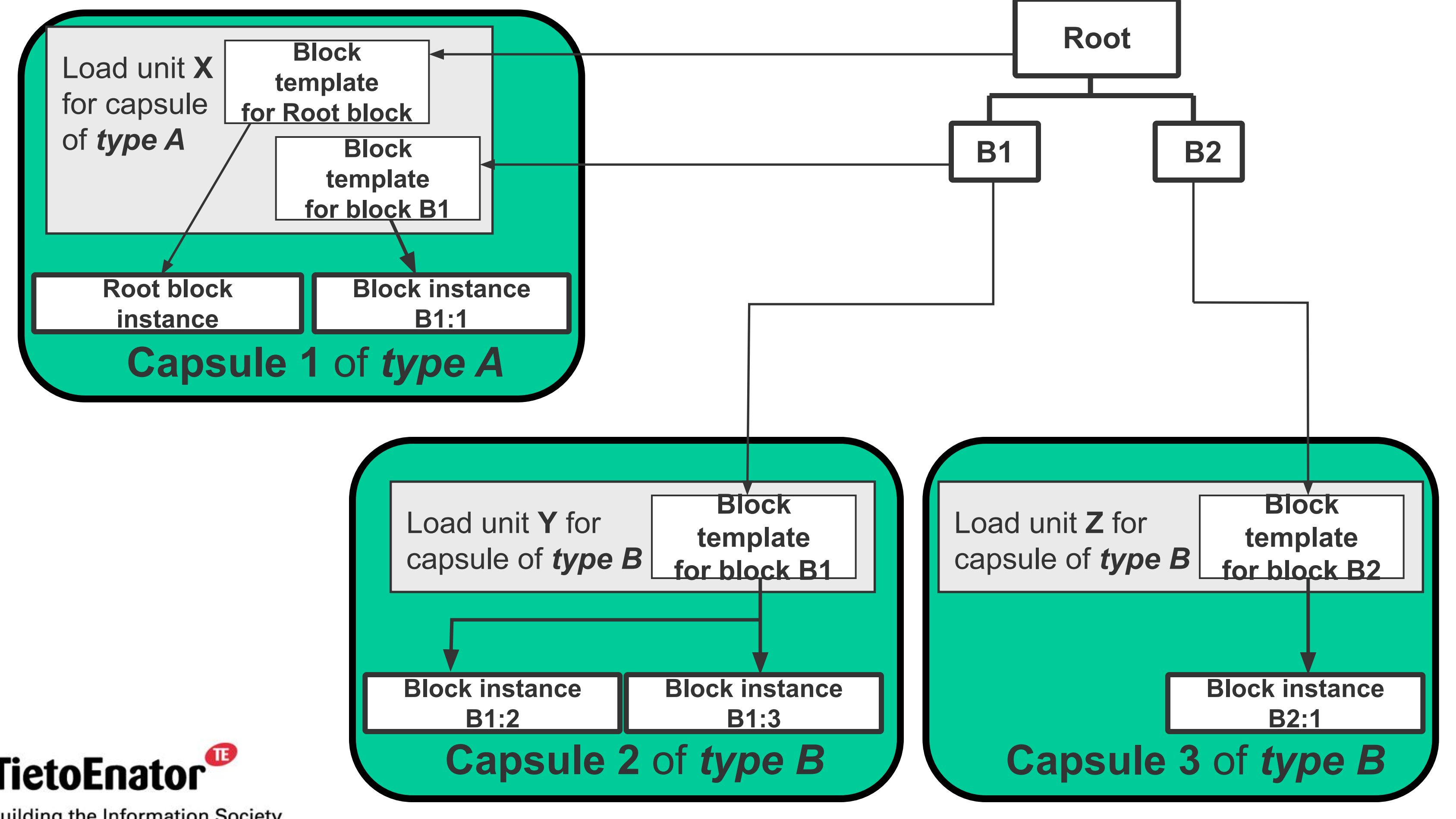
Copyright © 2006 TietoEnator Corporation

Load Unit

- An entity containing one or more *block templates.*
  - A C load unit is a file.
  - An Erlang load unit is a directory.
- Specific to a *capsule* type.
- When code is loaded into a *capsule*, it is always in the form of an entire *load unit.*
- After a *load unit* is loaded into a *capsule*, the capsule contains copies of the enclosed *block templates.*

Copyright © 2006 TietoEnator Corporation

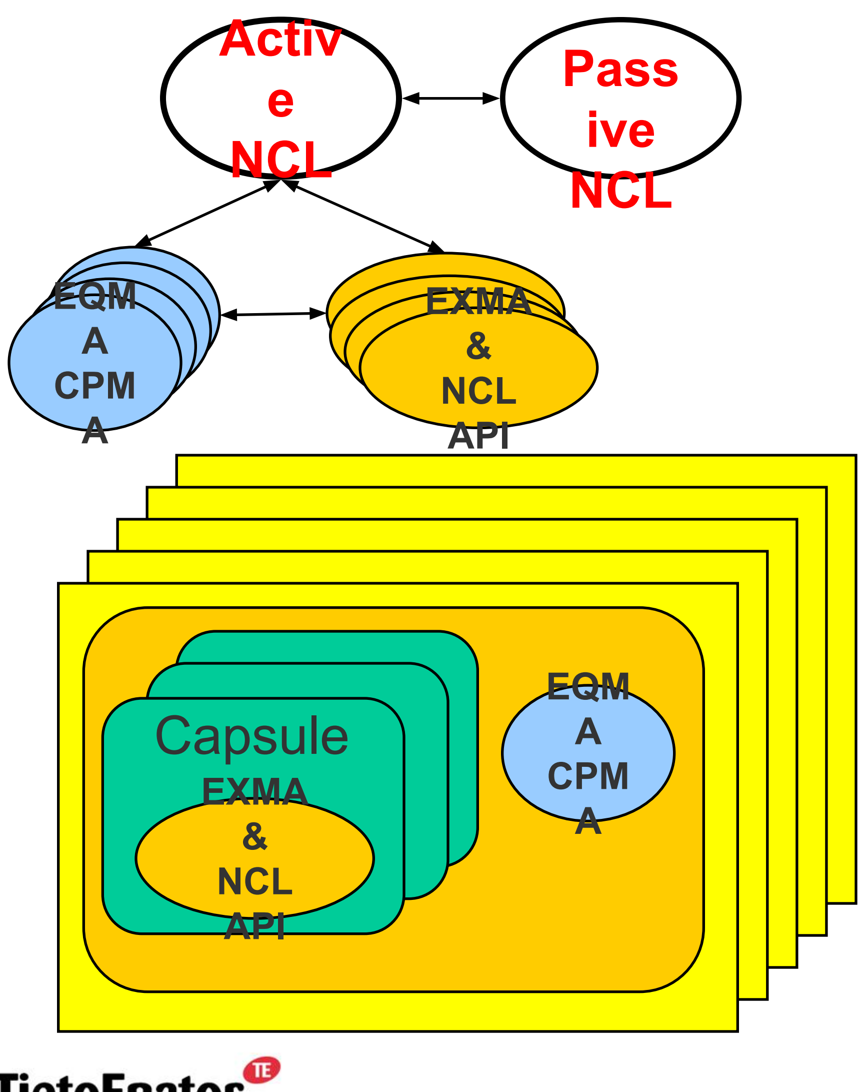
Blocks, load units, block templates, capsules and block instances - Relations

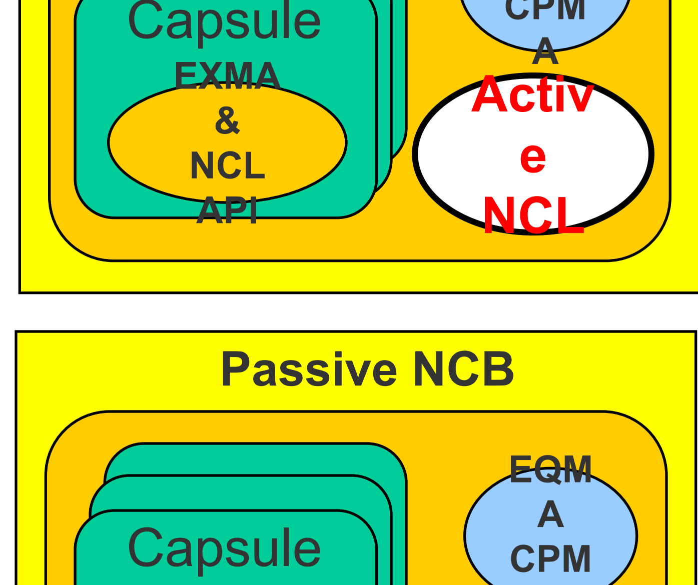


TietoEnator
Building the Information Society

Copyright © 2006 TietoEnator Corporation

DPE architecture - overview





Copyright © 2006 TietoEnator Corporation

DPE architecture - Node Control Logic (NCL)

- NCL is the DPE kernel.
- Two instances within the Node:
  - Active NCL, and
  - Passive NCL.
- The Boards where the two NCL instances reside are called Active NCB and Passive NCB (Node Control Board).
- The purpose of of the Passive NCL is to track the internal states of the Active NCL, in order to be able to take over if the Active NCL fails.

Copyright © 2006 TietoEnator Corporation

DPE architecture - Processor related parts

- EQMA - Equipment Management Agent
  - Located on all CPU:s
  - Responsible for all local equipment management operations
- CPMA - Capsule Management Agent
  - Located on all PM:s
  - Responsible for all local capsule management operations (create, delete, …)

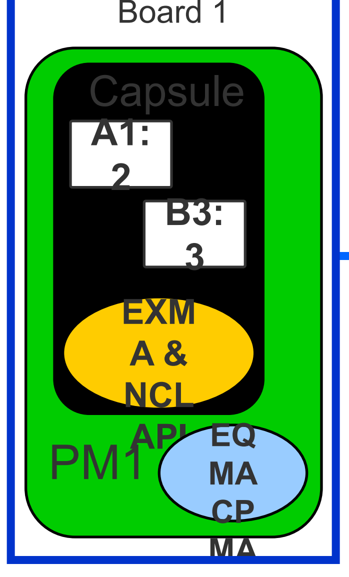
Copyright © 2006 TietoEnator Corporation

DPE architecture - Capsule related parts

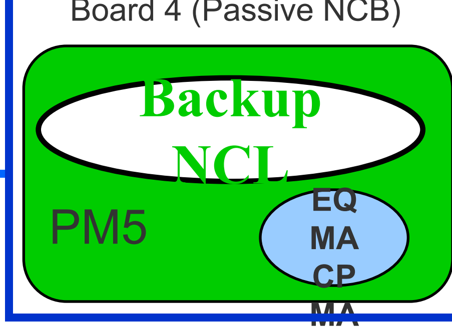
- EXMA - Execution Management Agent
  - Located in every Capsule
  - Responsible for operations related to Block Instances (create, start, stop, delete, …) within the specific Capsule.
- DPE API
  - Located in every Capsule
  - Allows Block Instances to interact with DPE (EXMA and NCL)
  - Location of NCL transparent to applications
  - Implementations for both C and Erlang

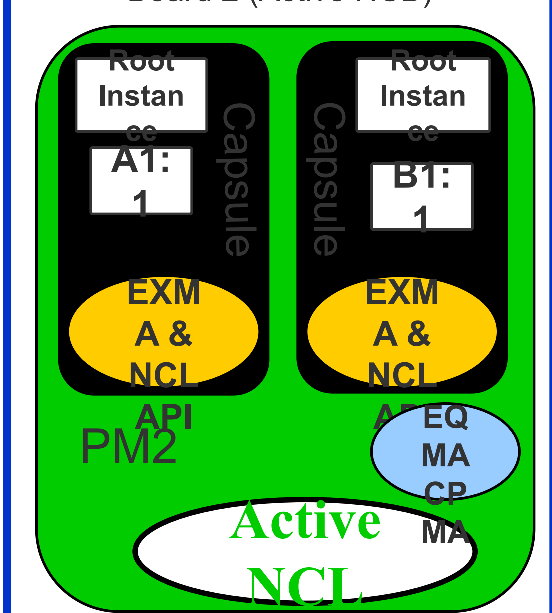
Copyright © 2006 TietoEnator Corporation

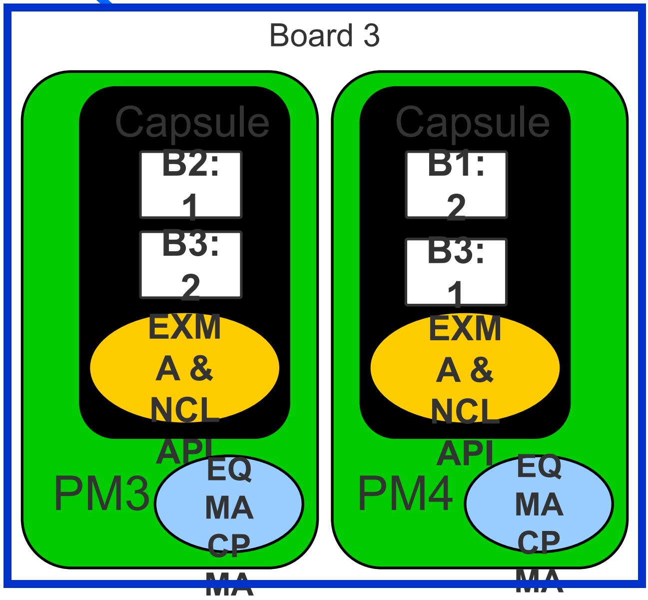
DPE architecture vs. example application instances



Intra node communication







Copyright © 2006 TietoEnator Corporation

Distributed Processing Environment (DPE)

3. Use Cases

- An example application
- Use cases:
  - Starting an application
  - Stopping an application
  - Blocking of a board
  - Addition of a board
  - Failing boards or PMs
  - Fail over
  - Software upgrade

Copyright © 2006 TietoEnator Corporation

An example application

Red I/F
Red I/F
Green I/F
Base Station
Red I/F
Green I/F
Red I/F
Base Station

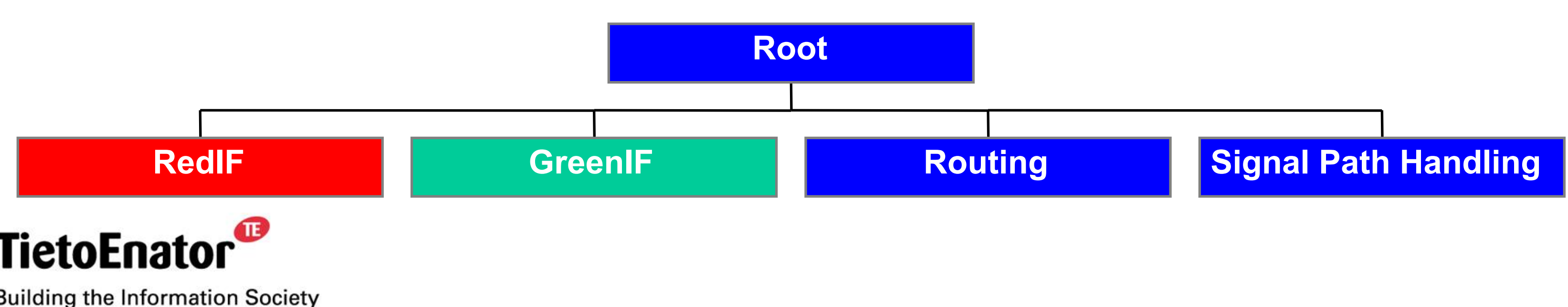
Network Node to be implemented using WPP

Other Network Nodes

Copyright © 2006 TietoEnator Corporation

Example application (Cont.) - Logical view

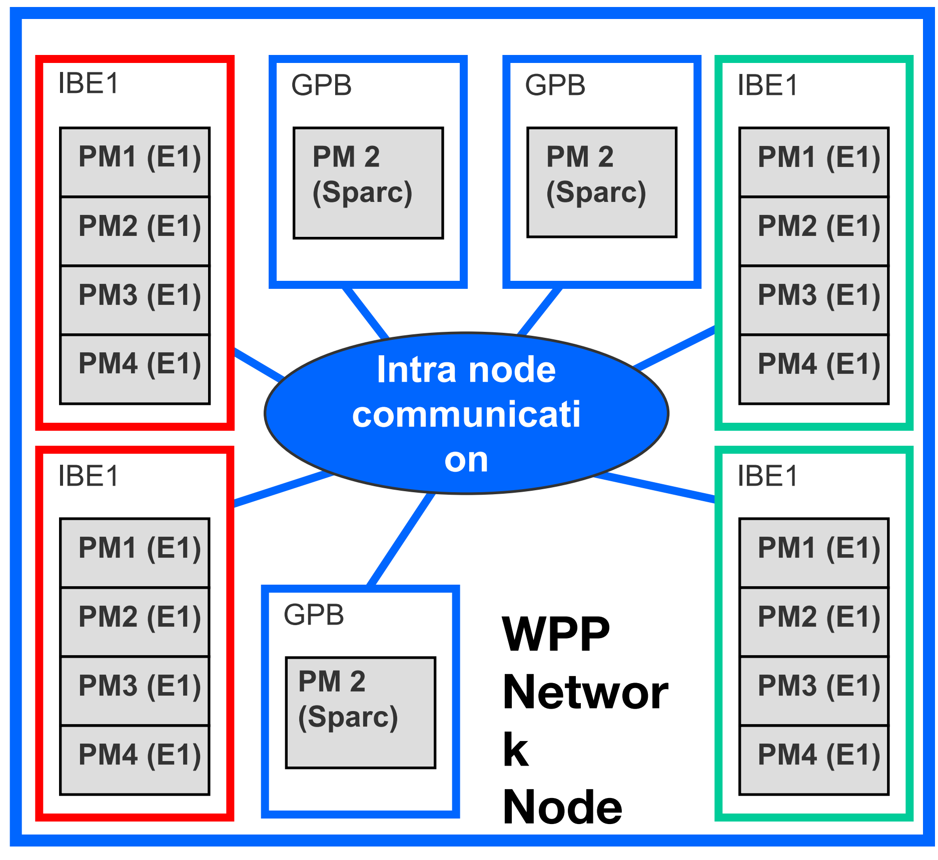
- Four major tasks:
  - handling of protocol interfacing base stations (Red I/F);
  - handling of protocol interfacing other nodes (Green I/F);
  - basic routing service (Routing); and
  - signal path handling (SignalPath).
- Chosen application structure tree (AST).



Copyright © 2006 TietoEnator Corporation

Example Application (Cont.) - Computing resources

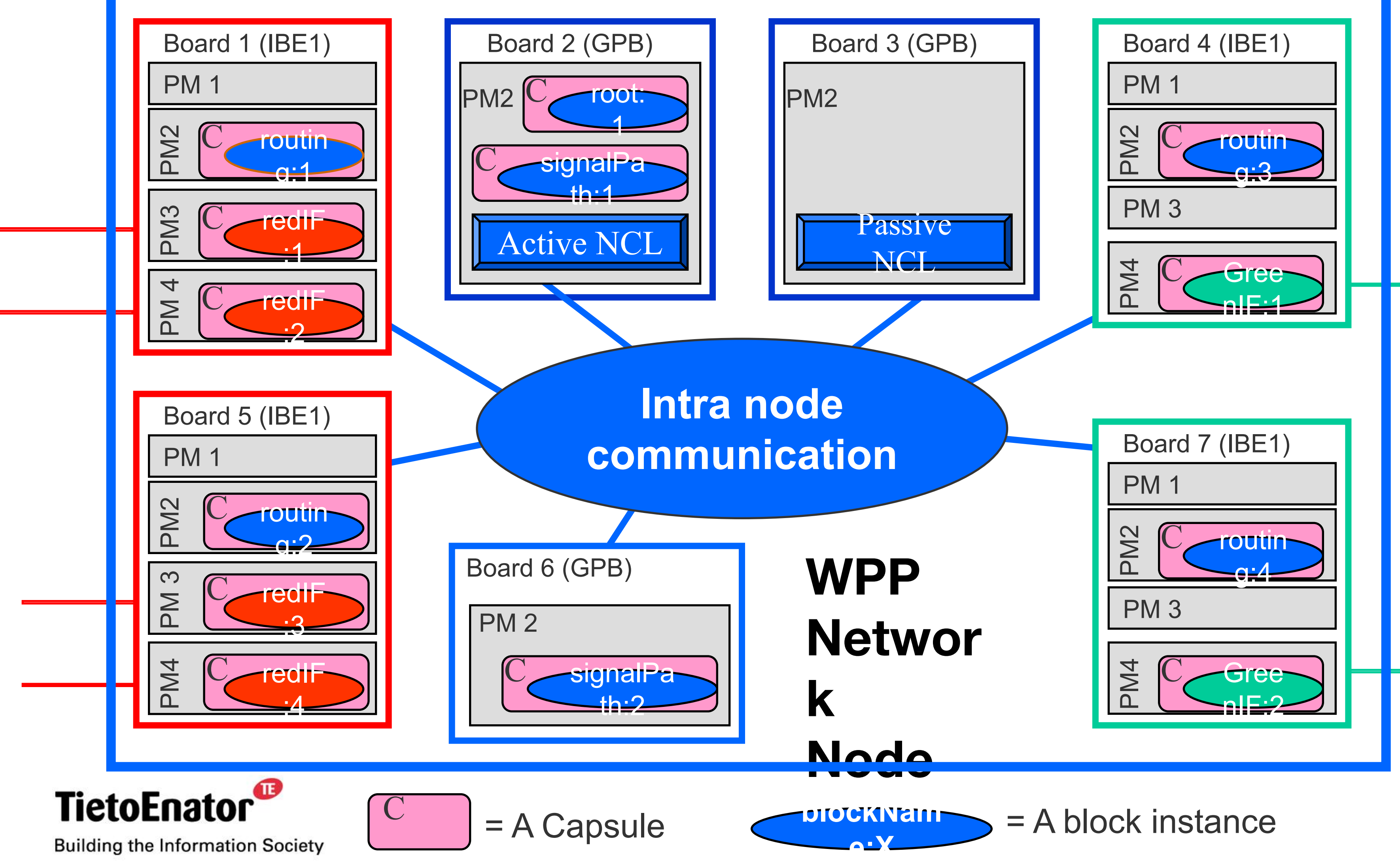
- 3 General Processing Boards (GPB)
  - 1 x UltraSparc
  - Solaris
- 4 Interface Board E1 (IBE1)
  - 4 x E1 with 2 Mbit/s (Sends and receives packages)
  - VxWorks
- Intra node communication
  - Ethernet (2x) between boards



Building the Information Society

Copyright © 2006 TietoEnator Corporation

Example application (Cont.) - Mapping



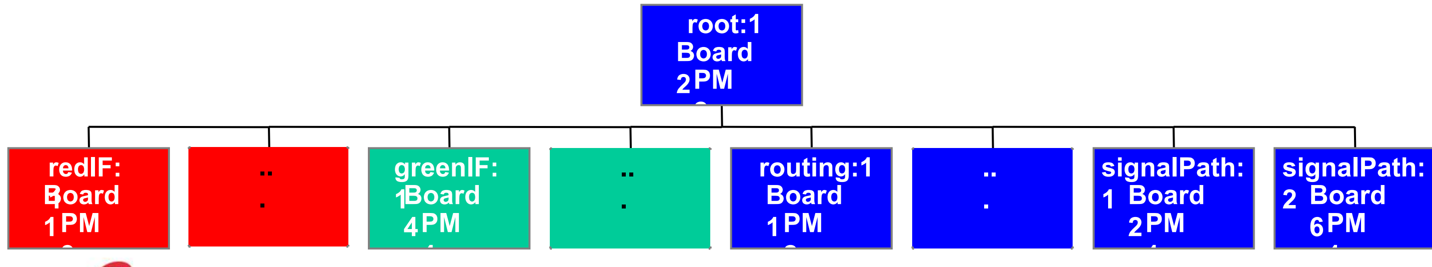
Copyright © 2006 TietoEnator Corporation

Starting an application

- DPE creates application root instance (root).

root:1
Board 2
PM

- DPE starts root.
- Root defines application directives.
- Root asks DPE
  - for an allocation suggestion.
  - to create block instances.
  - to start block instances.
- Root notifies DPE when the distribution is complete



TietoEnator
Building the Information Society

Copyright © 2006 TietoEnator Corporation

Stopping an application

- DPE requests the application root instance (root) to stop the application.
- Root requests DPE to
  - stop the block instances.
  - delete the stopped block instances.
- Root notifies DPE when the application is stopped.
- DPE stops the root and assures that it is deleted.

Copyright © 2006 TietoEnator Corporation

Blocking of a board

- Operator requests DPE to block a board, and hence all PMs on that particular board.
- DPE informs application root instance (root) about blocking request of PMs where the application has running Block Instances (BIs).
- The root removes all BIs that belongs to the application from the PMs on the affected board.
- DPE informs the operator when the whole board has been blocked.

Copyright © 2006 TietoEnator Corporation

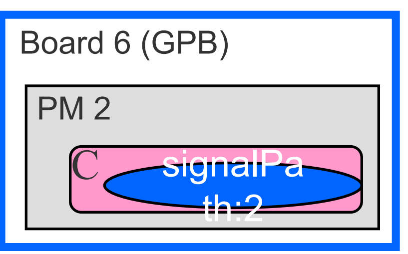
Note that a blocking request can be issued by:

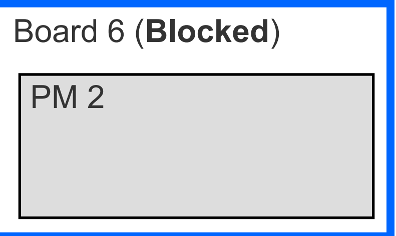
- an operator via GUI,
- an operator that presses the repair button on a board,
- an application.

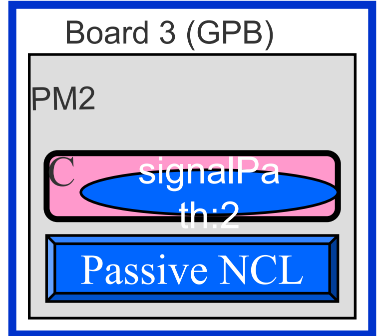
Copyright © 2006 TietoEnator Corporation

An example - Blocking `board 6`

- DPE informs root about blocking request affecting BI: `signalPath:2`.
- Root requests DPE to stop and delete `signalPath:2`.
- DPE informs operator about blocking completion.
- Root may reconfigure the application:
  - Root asks DPE for an allocation suggestion.
    - DPE suggest that `signalPath:2` is allocated on `PM2` in `board 3`.
  - Root asks DPE to create and start `signalPath:2`.







Building the Information Society

Copyright © 2006 TietoEnator Corporation

Addition of a Plug-In-Unit (PIU)

- An operator inserts a PIU into an empty slot
- DPE notifies all application root instances (roots) that new hardware is available
- The Roots investigates if the applications should reconfigure
- The Roots ask DPE for an allocation suggestion

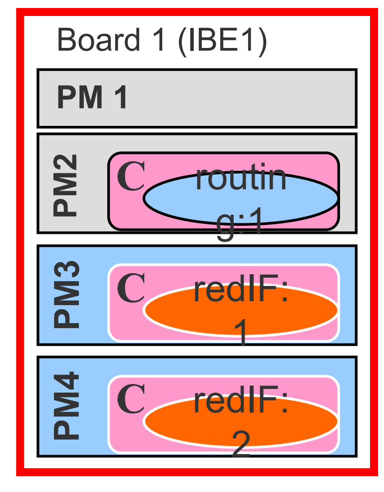
Copyright © 2006 TietoEnator Corporation

Failing boards or PMs

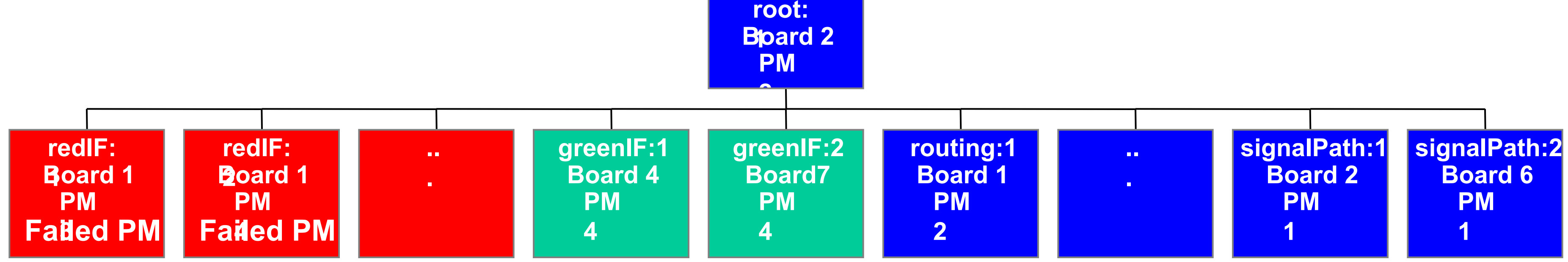
- Failures are caused by a wide variety of events
- The result of these failures is that one or more block instances die.
- DPE detects failures and sends messages to the affected application root instances
  - the message contains a set of failed block instances
- These roots may then initiate application-specific recovery actions.

Copyright © 2006 TietoEnator Corporation

An example - PM3 and PM4 on board 1 has failed



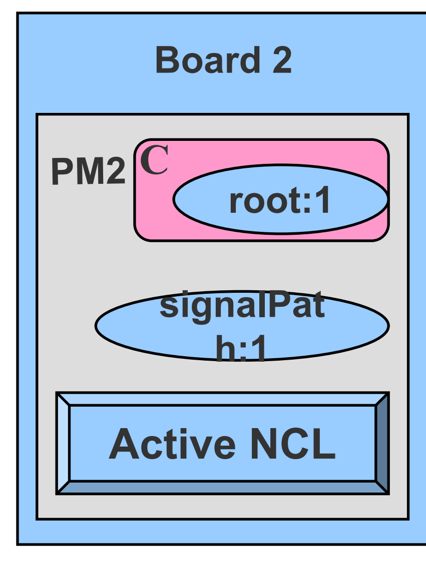
- DPE detects this, and:
  - notifies `root:1` that the block instances `redIF:1` and `redIf:2` have died.
- `root:1` may then initiate application-specific recovery actions.



Copyright © 2006 TietoEnator Corporation

But .. What if the Active NCB (`Board 2`) fails?

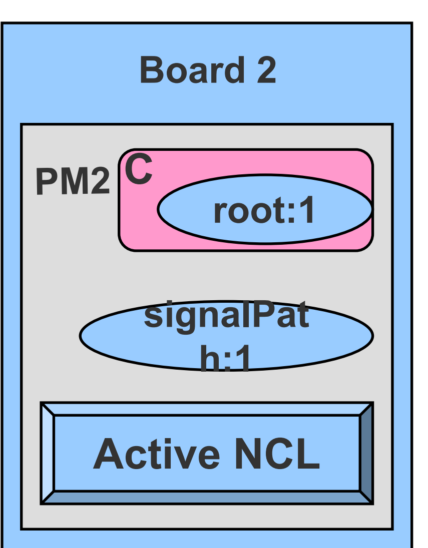
- DPE performs a fail-over operation, which consists of replacing the active NCL with the passive NCL:
  - All the NCL data structures are replicated;
  - the replicated data is used to re-establish the state of NCL. The operations of DPE can continue almost without disruption.
- After fail-over, NCL will automatically restart all application root instances.

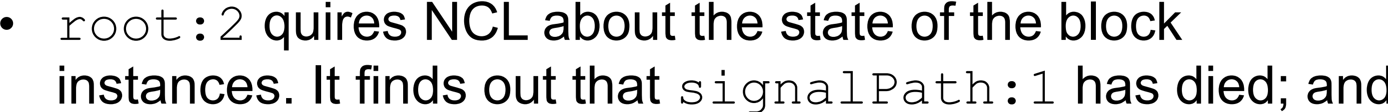


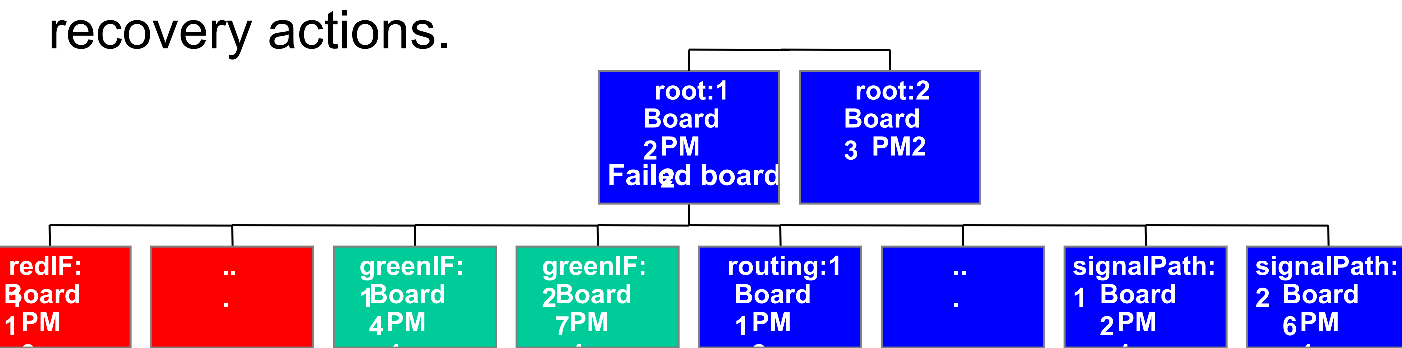
Copyright © 2006 TietoEnator Corporation

An example: `Board 2` has failed

- `root:1`, `signalPath:1` and `Active NCL` die
- DPE detects this, and:
  - performs a fail-over operation (active NCL is replaced with the passive NCL);
  - `root:1` is restarted as `root:2` on `PM1` in `board 3;`
  - `root:2` quires NCL about the state of the block instances. It finds out that `signalPath:1` has died; and
  - `root:2` may then initiate application-specific recovery actions.







Building the Information Society

Copyright © 2006 TietoEnator Corporation

Software upgrade

- DPE provides support for:
  - upgrading to a new software configuration;
  - falling back to a previously check-pointed software configuration;
  - introducing patches.
- The method to activate a software configuration depends on the circumstances.
- From an applications point of view, it does not matter which activation method that is used:
  - an application may either be stopped or remain running;
  - in each case it has to manage its own configuration data.

Copyright © 2006 TietoEnator Corporation

Distributed Processing Environment (DPE)

4. Introduction to Capsule Management

- Capsule Management
- Functionality and Behavior

Copyright © 2006 TietoEnator Corporation

Capsule management

- Capsule Management manages the creation and deletion of capsules
- It is implemented by the following software entities:
  - the *Capsule Manager*, or *CPM*, which is a part of NCL
  - the *Capsule Management Agents* (or *CPMA*s): there is one such agent in each active PM.

Copyright © 2006 TietoEnator Corporation

An example of CPM and CPMAs

Board 1

Capsule

PM1 CPMA

Intra node communication

Board 2

Capsule

CPMA

PM2

CPM

NCL

Board 3

Capsule

PM3 CPMA

Capsule

PM4 CPMA

Copyright © 2006 TietoEnator Corporation

Capsule

- Special protective environment for locating a *block instance* in a certain processing module (*PM*)
- Make possible to use uniform interfaces between DPE and *applications* despite the differences in implementation languages, operating systems etc.
- *Capsule* type defined by:
  - Hardware (SPARC, PowerPC, etc.)
  - Operating system (Solaris, RTOS, etc.)
  - Language (interpreted Erlang, C, Java, etc.)
  - Design decisions when implementing the *capsule*

Copyright © 2006 TietoEnator Corporation

Capsule Attributes

- MultipleLoadUnits
  - Can the capsule contain more than one load unit?
- Loadable
  - Is it possible to add new load units after the capsule has been created?
- MultiThreaded
  - Does the capsule support multiple block instances?

Copyright © 2006 TietoEnator Corporation

Capsule attributes (cont’d)

- Osunloadable
  – Is it possible for the OS to unload (shut down) the capsule
- cachedSendmsg
  – Does the capsule type take advantage of the efficient protocol for DPE_SendMessage

Copyright © 2006 TietoEnator Corporation

Capsule types

- `Solaris_C_Capsule`
  MultipleLoadUnits No
  Loadable No
  MultiThreaded Yes
  Osunloadable Yes
  cachedSendmsg Yes

- `RTOS_C_Capsule`
  MultipleLoadUnits No
  Loadable No
  MultiThreaded Yes
  Osunloadable No
  cachedSendmsg Yes

- `Erlang_Capsule`
  MultipleLoadUnits Yes
  Loadable Yes
  MultiThreaded Yes
  Osunloadable Yes
  cachedSendMessage No

- `JAVA_Capsule`
  MultipleLoadUnits Yes
  Loadable Yes
  MultiThreaded Yes
  Osunloadable Yes
  cachedSendmsg No

TietoEnator
Building the Information Society

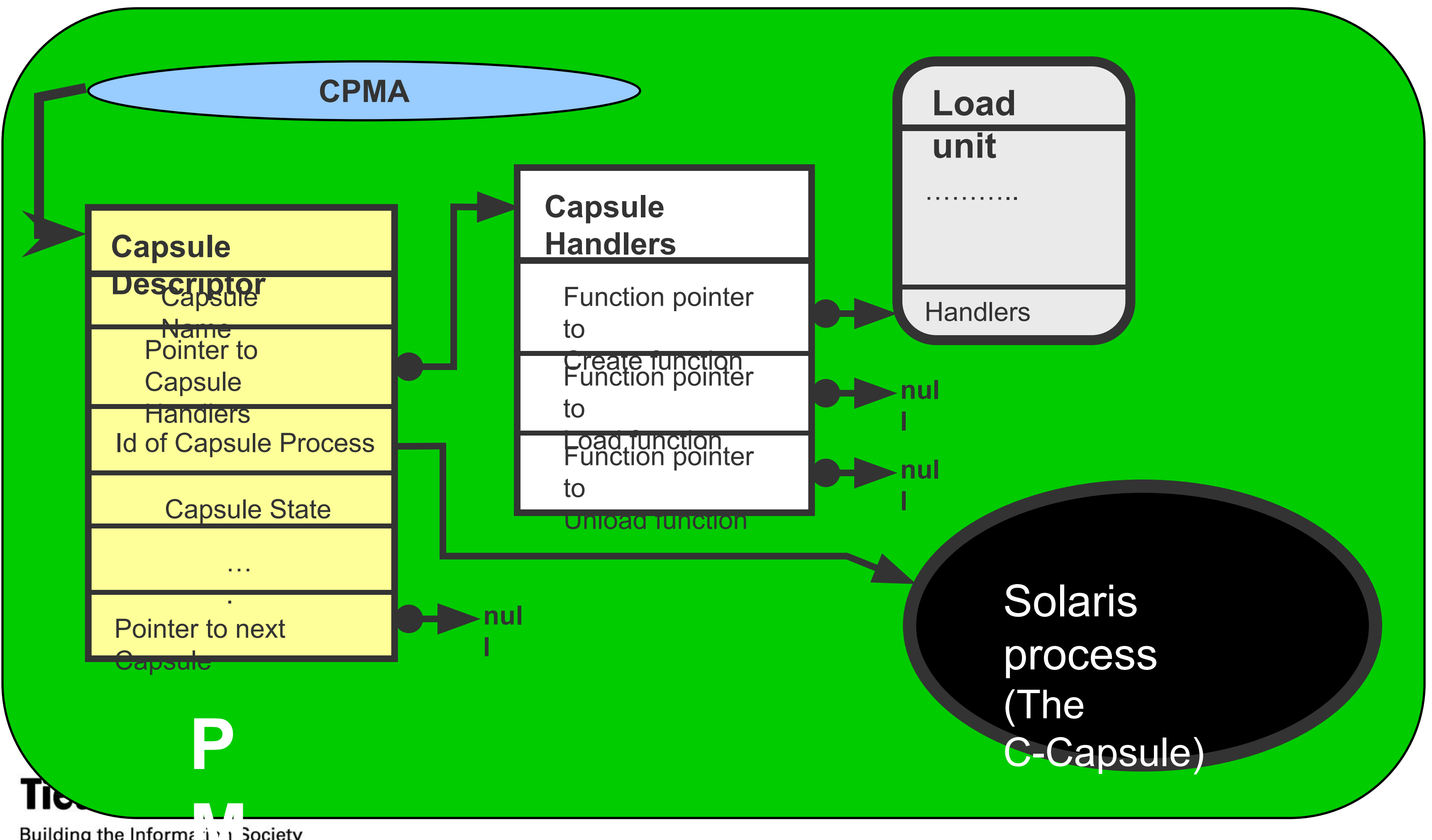
Copyright © 2006 TietoEnator Corporation

New RTOS C-capsule types

- Using WPP5.0 it is possible to further specify the processor, an RTOS capsule may execute on.
  - RTOS_C_Capsule_*
- Where * is the type of board, f.I: Ibxx, Ibxx_860, Ibxx_craneboard, PEB, etc.

Copyright © 2006 TietoEnator Corporation

Implementation of a C-Capsule on Solaris



Building the Information Society

Copyright © 2006 TietoEnator Corporation

Functionality and Behavior

- Capsule management provides operations that
  - create, delete, and monitors capsules
  - loads load units into the capsule (*only for Erlang and JAVA*)
  - unloads load units from the capsule (*only for Erlang and JAVA*)
  - enable communication with entities inside the capsule
- The block instance management uses these functions to ensure :
  - that a capsule will exist in the correct location
  - that the capsule is loaded with the block templates that are needed to create a given set of block instances

Copyright © 2006 TietoEnator Corporation

Distributed Processing Environment (DPE)

5. Introduction to Execution Management

- Block instance management
- Description of block, block template, load unit, block instance and capsule
- Functionality and behavior
- Operations available to applications

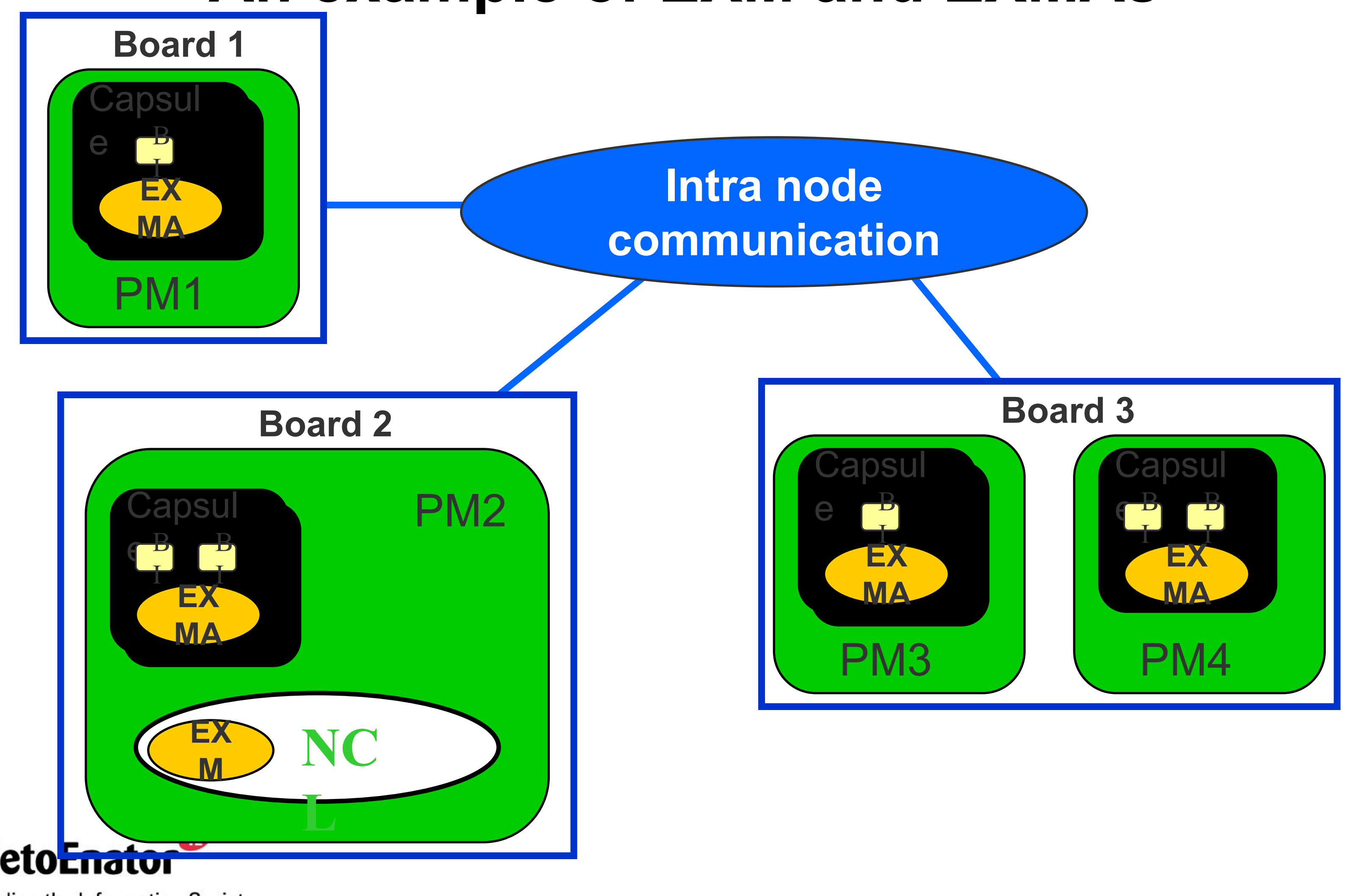
Copyright © 2006 TietoEnator Corporation

Block instance management

- Execution Management manages block instances.
  - A block instance is the basic functioning unit of an application.
- It is implemented by the following software entities:
  - the ***Execution Manager***, or ***EXM***, which is a part of NCL;
  - the ***Execution Management Agents*** (or ***EXMA***s): there is one such agent in each existing capsule.

Copyright © 2006 TietoEnator Corporation

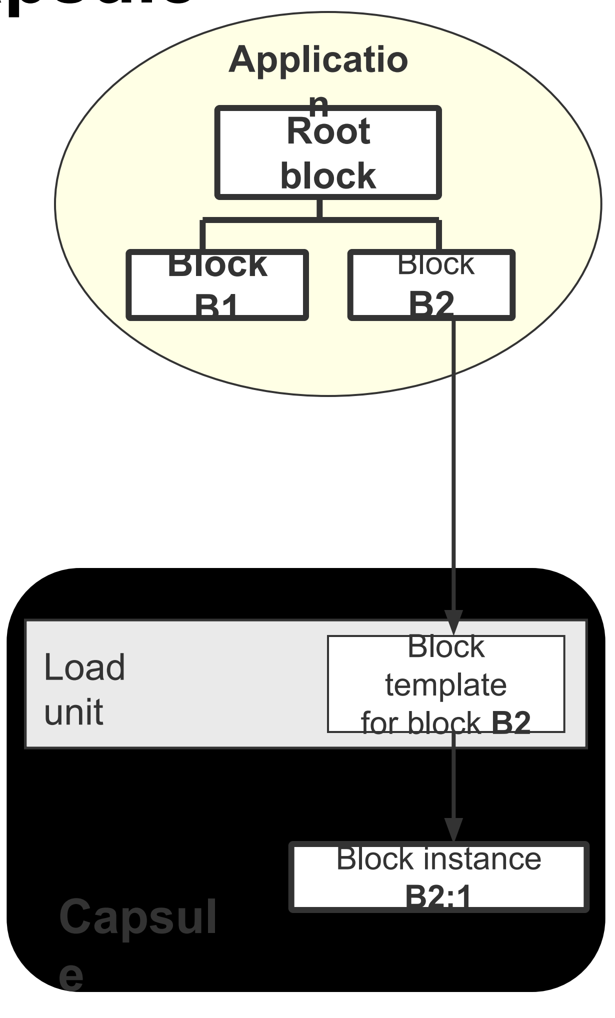
An example of EXM and EXMAs



TietoEnator
Building the Information Society

Copyright © 2006 TietoEnator Corporation

Description of block, block template, load unit, block instance and capsule



An application is a collection of *blocks*.

A *block template* is the executable code performing the functionality of a block.

A load unit is an entity containing one or more *block templates.*

A *block instance* is the executing form of a *block.*

A *capsule* is a special execution environment for *block instances*.

Copyright © 2006 TietoEnator Corporation

Block **and** ***block template***

- *Block*
  - A functional unit within an *application*.
  - Smallest functional part that can be instantiated to a running entity on a particular processing module (*PM*).
- *Block template*
  - Executable code performing the functionality of a *block.*
  - Same block may have several *block templates*, each for a different type of execution environment (*Capsule*).

Copyright © 2006 TietoEnator Corporation

Block Instance

- A *block instance:*
  - resides in a particular *capsule;*
  - requires that the appropriate *block template* is present in the *capsule; and*
  - is created based on the *block template.*
- A *block instance* is the executing form of a *block, e.g.,*
  - an Erlang process.
- Each *block instance* (BI):
  - belongs to a particular *application instance; and*
  - has an unique identity *(block instance name)* which is not reused.

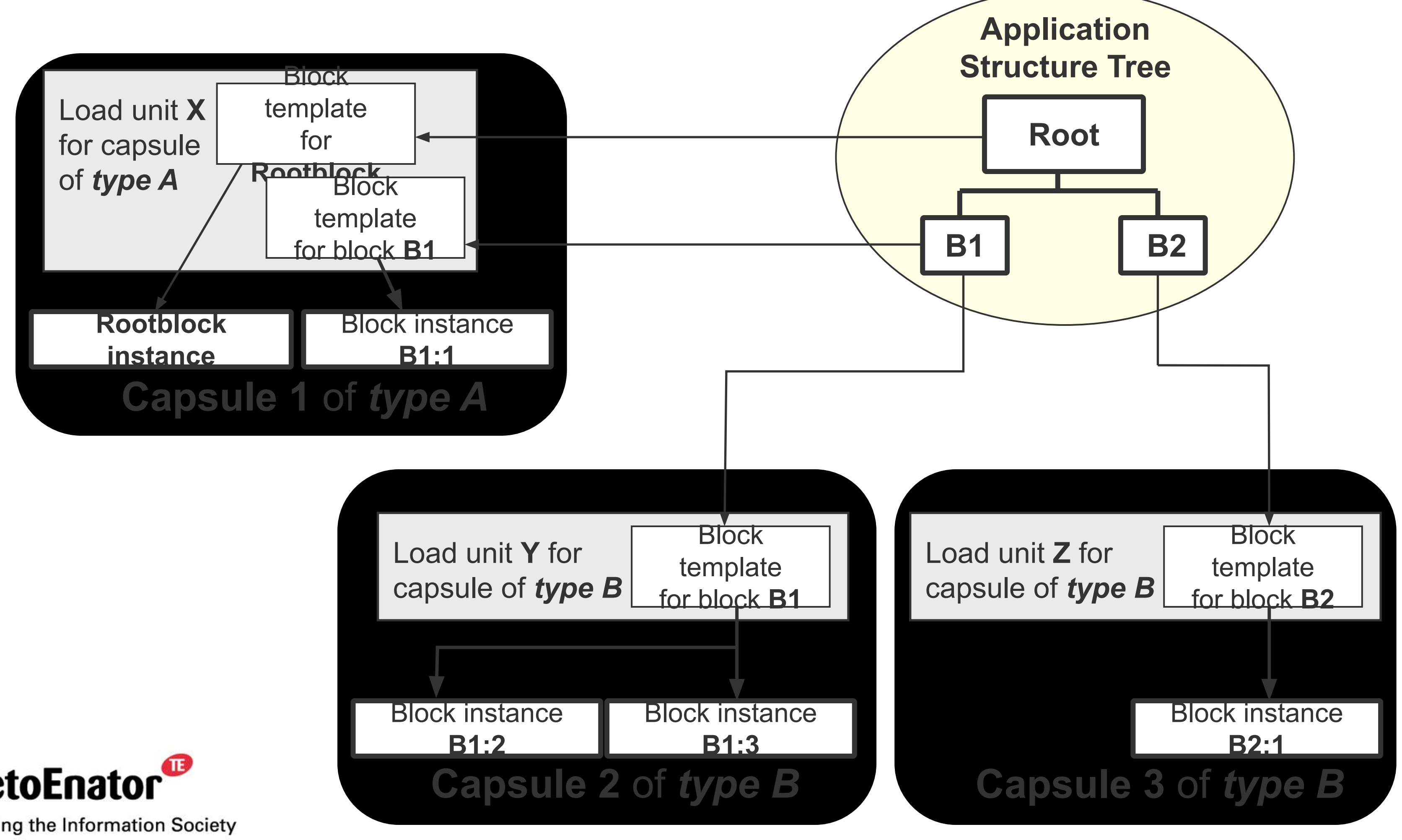
Copyright © 2006 TietoEnator Corporation

Load Unit

- An entity containing one or more *block templates.*
  - A C load unit is a file.
  - An Erlang load unit is a directory.
- Specific to a *capsule* type.
- When code is loaded into a *capsule*, it is always in the form of an entire *load unit.*
- After a *load unit* is loaded into a *capsule*, the capsule contains copies of the enclosed *block templates.*

Copyright © 2006 TietoEnator Corporation

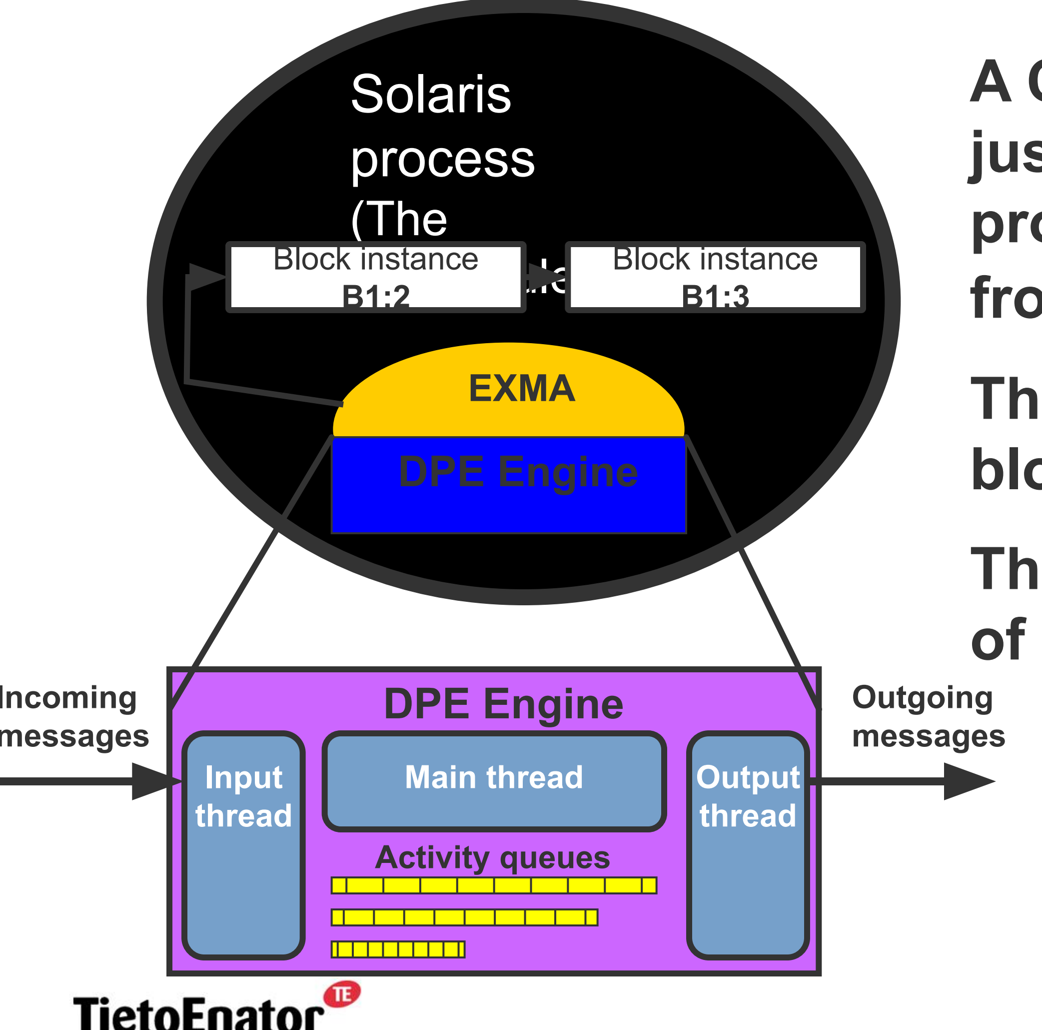
Summary of relation between block, load unit, block template, capsule and block instance



TietoEnator
Building the Information Society

Copyright © 2006 TietoEnator Corporation

The internals of a C-Capsule on Solaris



A C-Capsule on Solaris is just a process executing a process image constructed from a load unit.

The EXMA maintains a list of block instance descriptors.

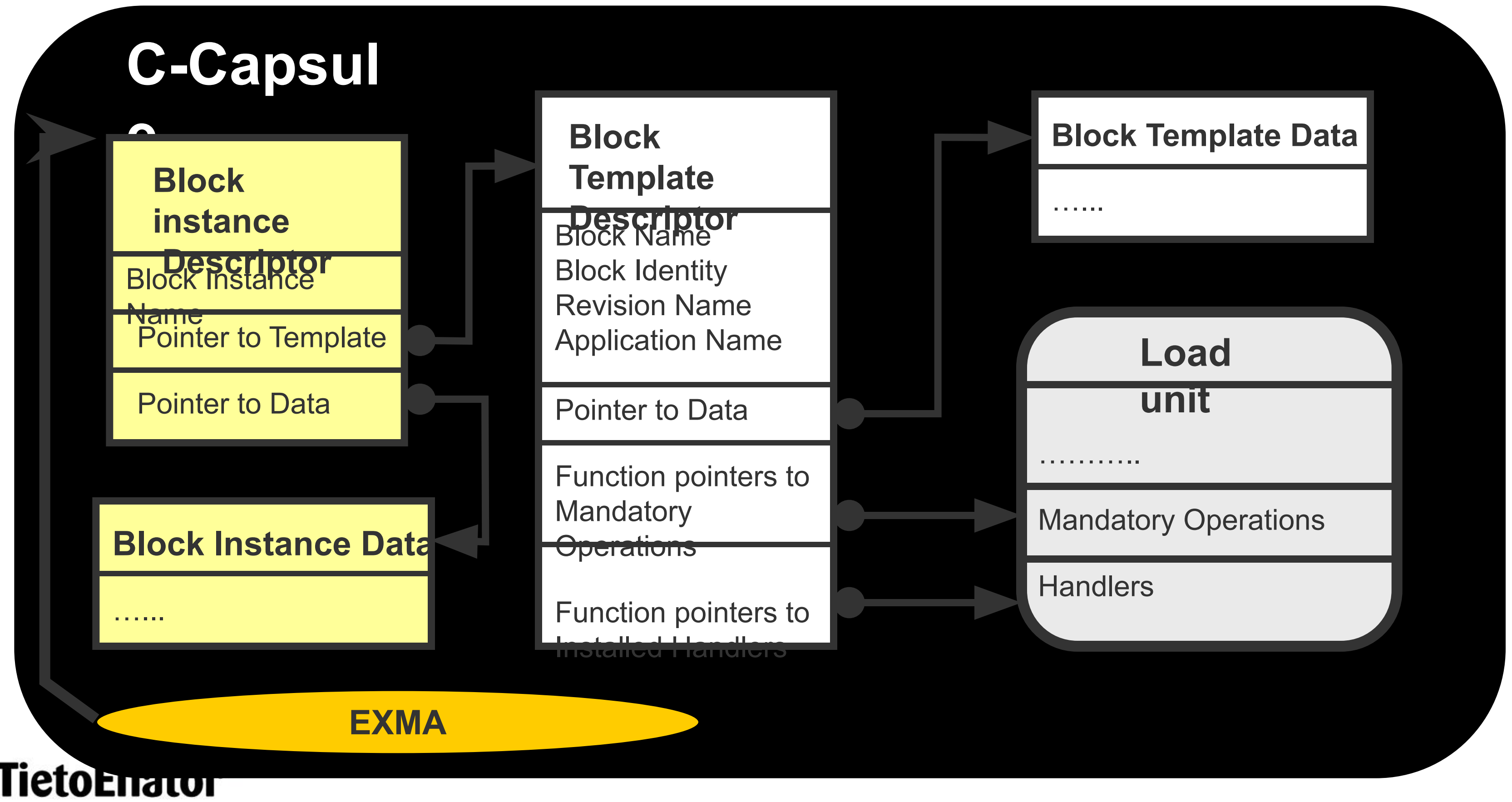
The EXMA executes on top of the DPE Engine.

Note that the DPE Engine only can be used by DPE.

TietoEnator
Building the Information Society

Copyright © 2006 TietoEnator Corporation

Implementation of a block instance in a C-Capsule on Solaris



TietoEnator
Building the Information Society

Copyright © 2006 TietoEnator Corporation

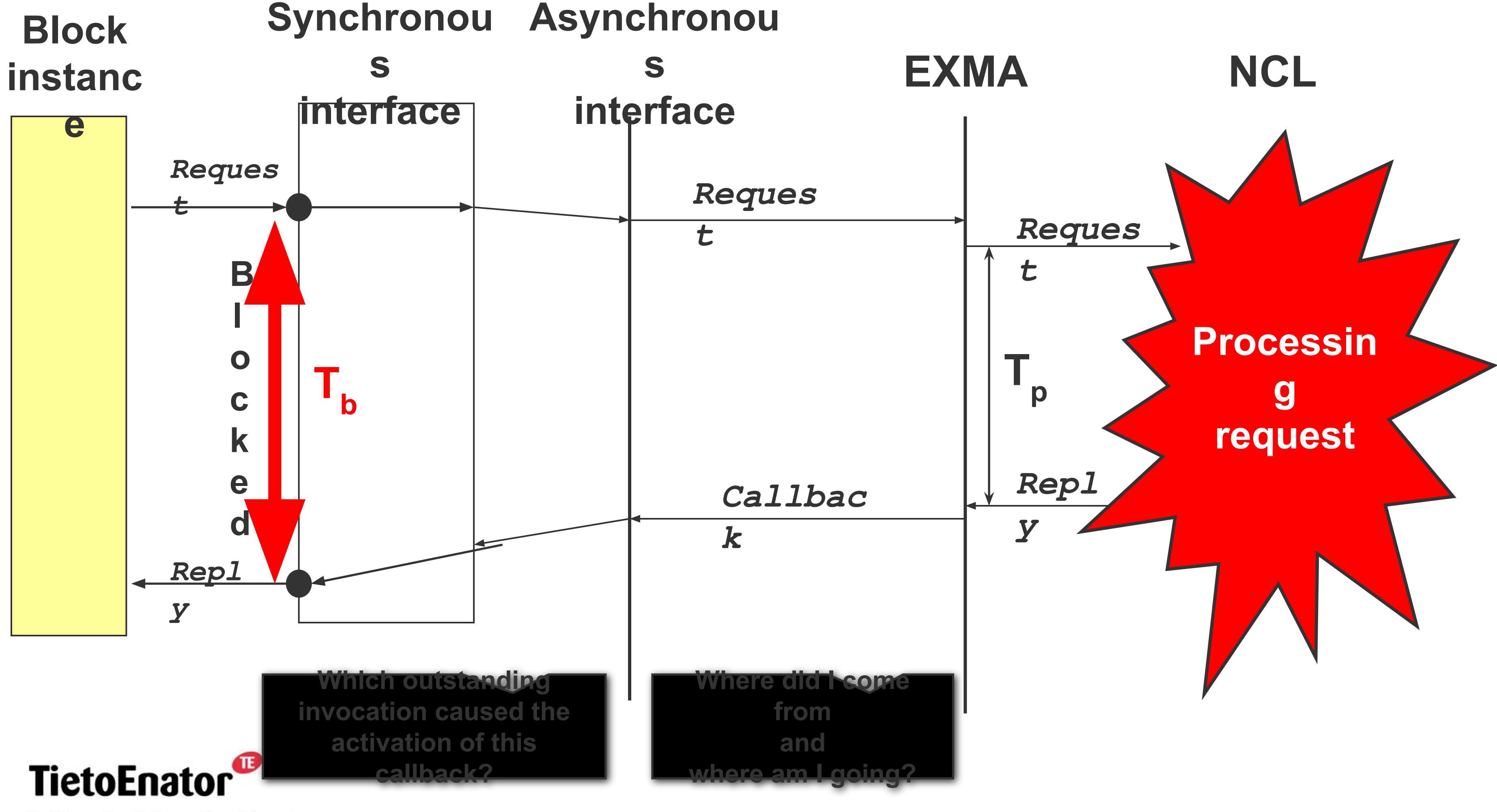
Functionality and behavior

- Block instance management provides operations that:
  - create, start, stop, delete, and kill block instances;
  - enable communication between block instances; and
  - monitors block instances.
- Application request to DPE for block instance management is:
  - directed to EXM, which carries out the request by means of the various local EXMAs.

- What type of interface does DPE provide?

Copyright © 2006 TietoEnator Corporation

Asynchronous vs. synchronous interface



TietoEnator
Building the Information Society

Copyright © 2006 TietoEnator Corporation

Asynchronous vs. synchronous interface (cont.)

Asynchronous

☺ Pros

- Enables quick response to various events
- A capsule with an asynchronous interface is easy to implement

☹ Cons

- Requires great care from the application programmers
- Difficult to provide a structured application program

Synchronous

☺ Pros

- Enables structured sequential application programs

☹ Cons

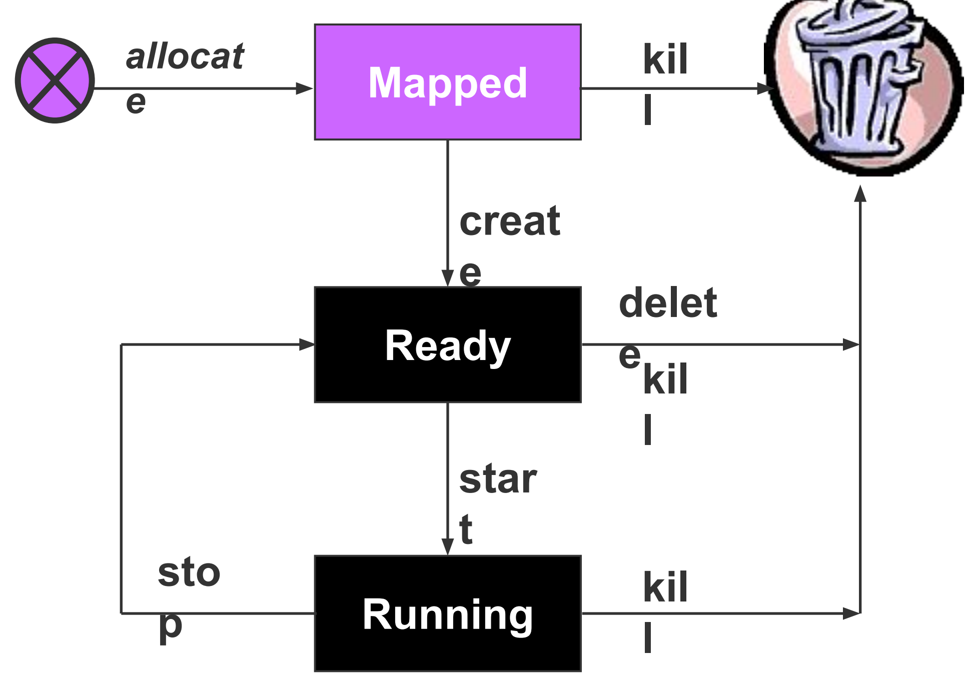
- A call is blocked during the time (T_b) DPE processes the request
- A capsule with a synchronous interface is more complex to implement

An useful interface should provide both asynchronous and synchronous operations!

TietoEnator
Building the Information Society

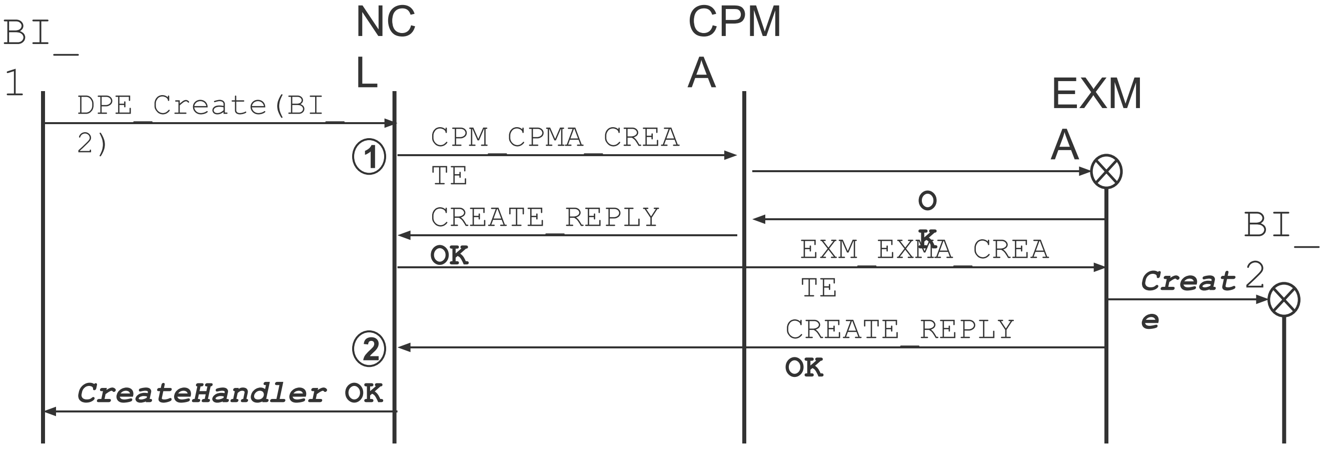
Copyright © 2006 TietoEnator Corporation

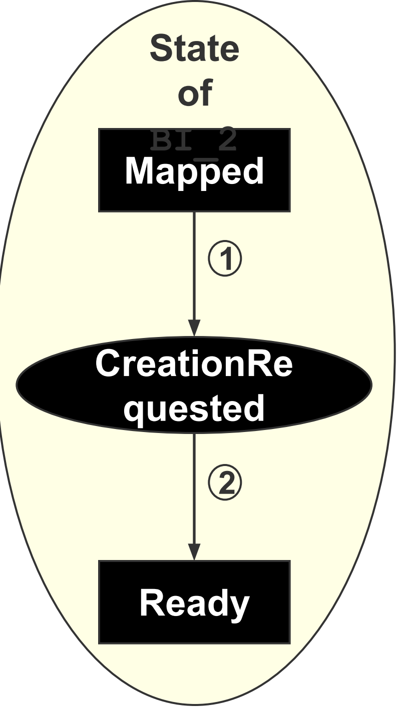
The main states of a block instance



Copyright © 2006 TietoEnator Corporation

Creating a block instance





Block instance `BI_2` **must** be in state Mapped.

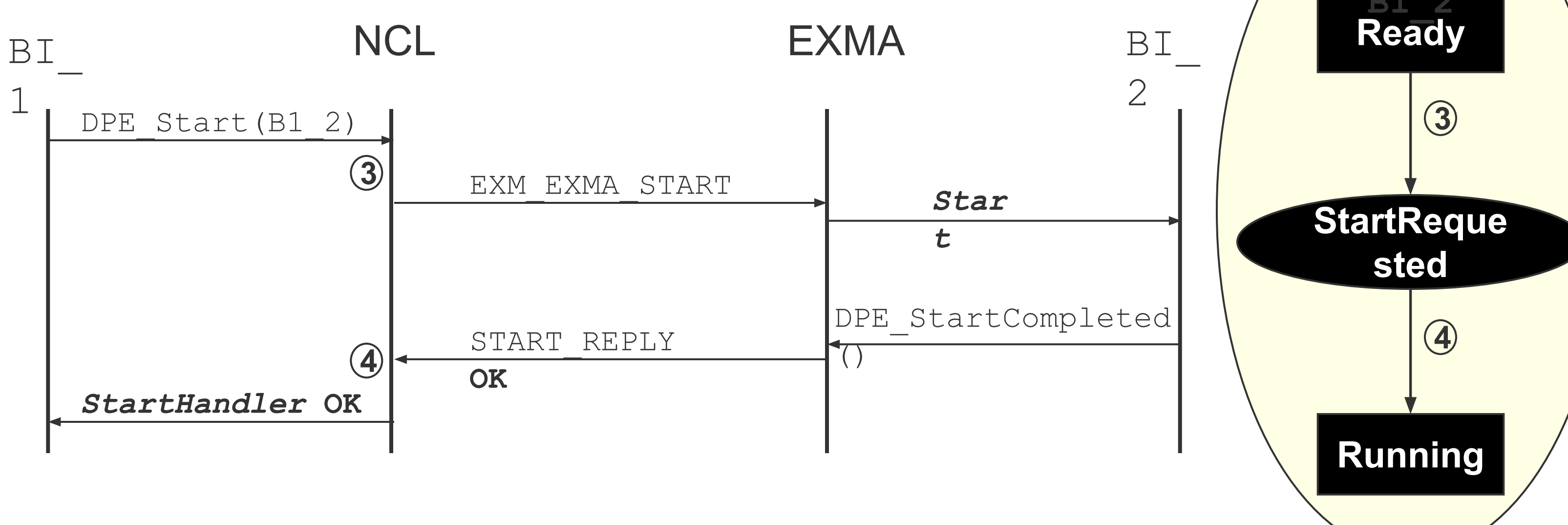
① NCL sets the transient state CreationRequested on `BI_2`.

② NCL sets the state of `BI_2` to Ready.

Building the Information Society

Copyright © 2006 TietoEnator Corporation

Starting a block instance



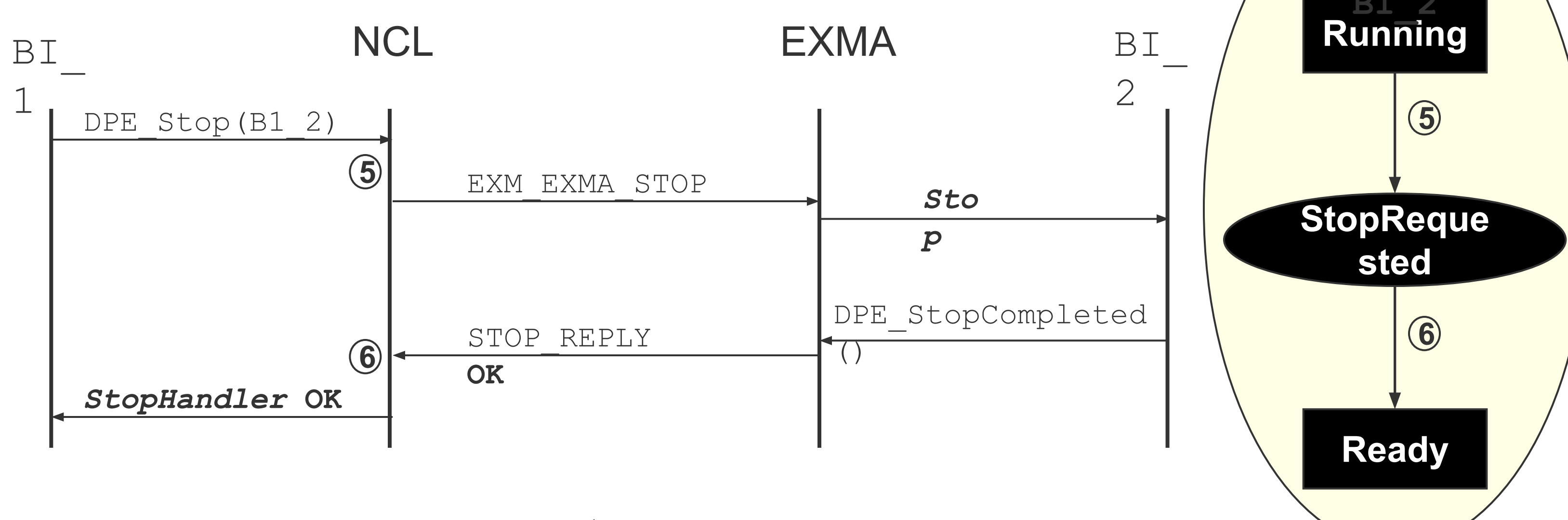
Block instance BI_2 **must** be in state Ready.

③ NCL sets the transient state StartRequested on BI_2.

④ NCL sets the state of BI_2 to Running.

Copyright © 2006 TietoEnator Corporation

Stopping a block instance



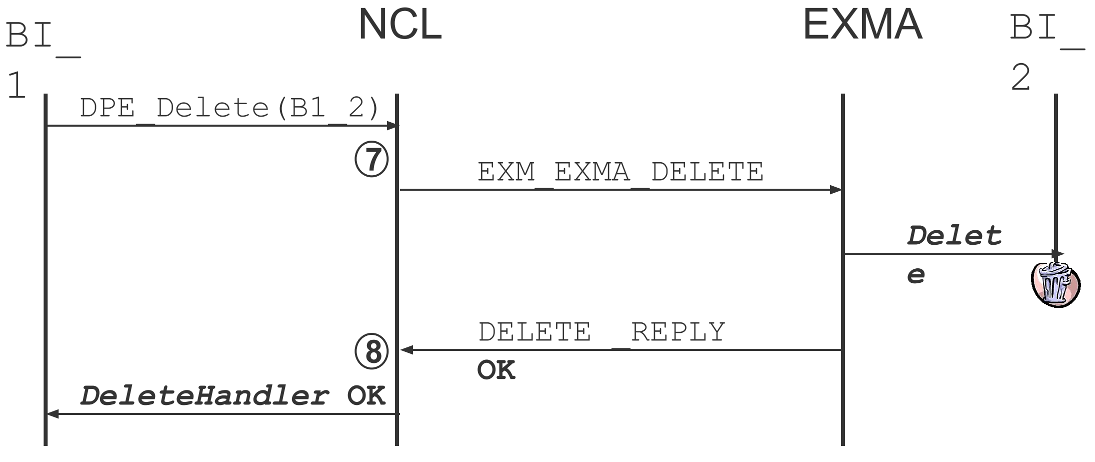
Block instance `BI_2` **must** be in state Running.

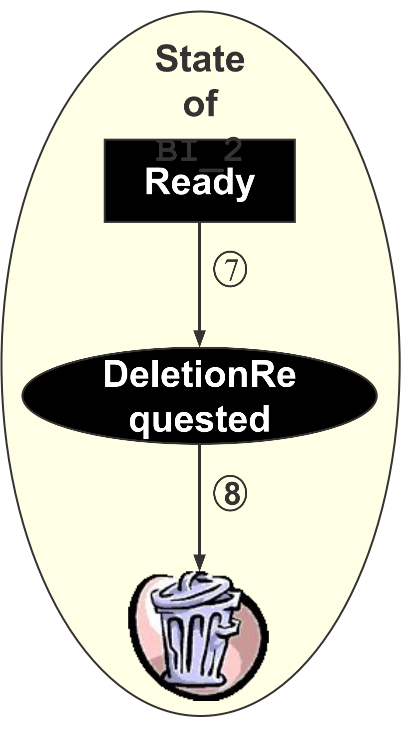
⑤ NCL sets the transient state StopRequested on `BI_2`.

⑥ NCL sets the state of `BI_2` to Ready.

Copyright © 2006 TietoEnator Corporation

Deleting a block instance





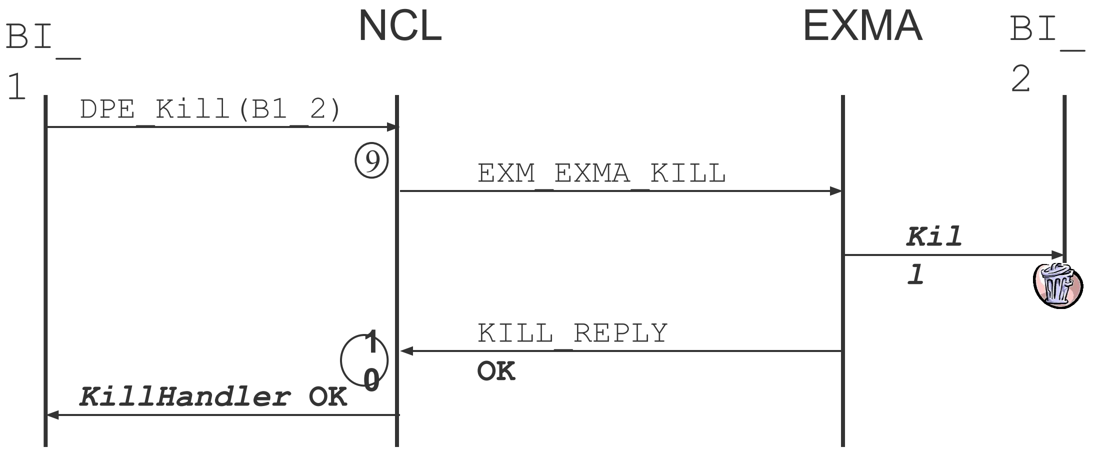
Block instance `BI_2` **must** be in state Ready.

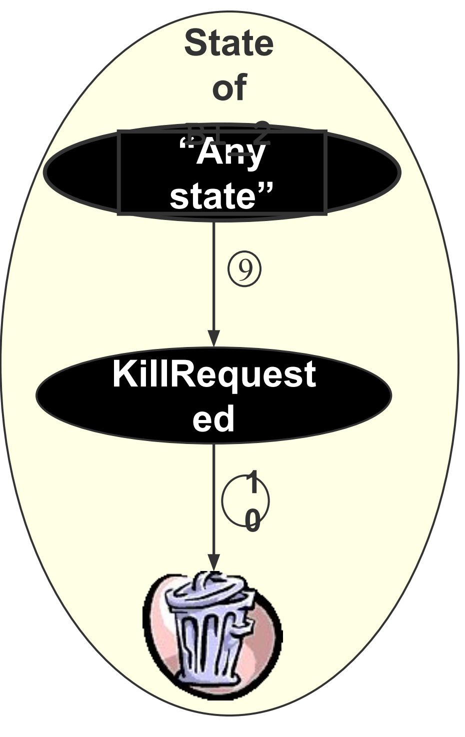
⑦ NCL sets the transient state DeletionRequested on `BI_2`.

⑧ NCL removes the block instance `BI_2` from its register.

Copyright © 2006 TietoEnator Corporation

Killing a block instance





Block instance BI_2 can be in any state.

9 NCL sets the transient state KillRequested on BI_2.

10 NCL removes the block instance of BI_2 from its register.

Copyright © 2006 TietoEnator Corporation

All states of a block instance

allocat
e

Mapped

①

CreationReq
uested

②

Ready

⑦

DeletionReq
uested

⑧

1
0

KillRequ
ested

⑨

"Any
state"

⑥

③

StopRequ
ested

StartRequ
ested

④

⑤

Running

Copyright © 2006 TietoEnator Corporation

Communication between block instances

- Messages can be sent between block instances via:
  - DPE_SendMessage in C; and
  - send_message in Erlang.
- No automatic confirmation is provided to the sender.
  - ⇒ An application is free to define its own protocol.
- Provides an easy way of using block instance names for addressing recipients of communication.
  - Note that a restarted block instance will get a new block instance name.

Copyright © 2006 TietoEnator Corporation

Monitoring of block instances

- Every block instance in a capsule is monitored by the EXMA in that capsule:
  - The `Monitor()` function is called;
  - The block instance is expected to invoke `DPE_BlockInstanceAlive()`; and
  - If this is not done, within a certain amount of time, EXMA will inform NCL that the block instance has died.
- NCL informs the *application root instance* that the block instance has died.
  - A failed *application root instance* will not be notified. Instead it is restarted by NCL.

 Copyright © 2006 TietoEnator Corporation

Operations available to applications

- create Creates a set of block instances.
- start Starts a set of block instances.
- stop Stops a set of block instances.
- delete Deletes a set of block instance.
- kill Kills a set of block instances.
- sendMessage Sends a message to a block instance.

Copyright © 2006 TietoEnator Corporation

Distributed Processing Environment (DPE)

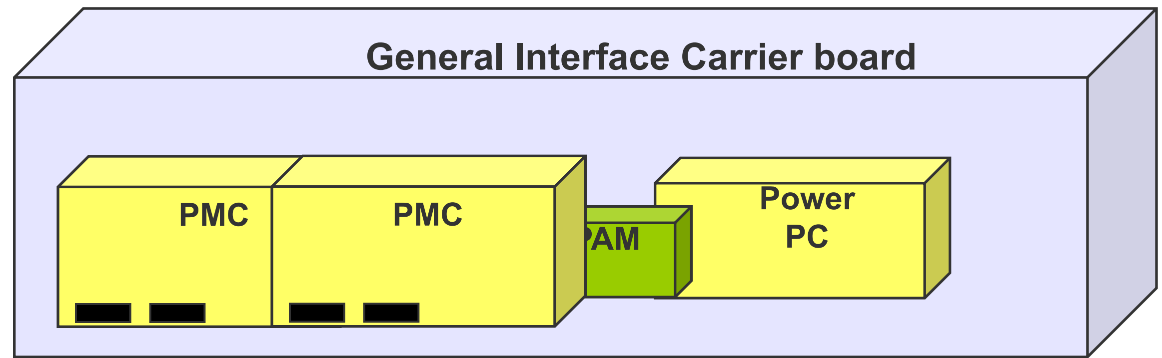
6. Equipment Management

- Part of Node Control Logic (NCL)
- Map of hardware (HW)
- Used by Function Distribution Manager (FDM)
- Crane Board Dictionary (CBD)
- HW supervision
- HW control
- Auxiliary services

Copyright © 2006 TietoEnator Corporation

Node Hardware

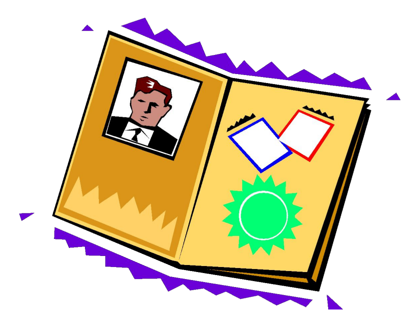
- Magazine
- Plug-In-Units (PIUs)



- Subboards
- Function Elements (FE)
  - Processing Module (PM)
  - IO card

Copyright © 2006 TietoEnator Corporation

Equipment ID



- Specifies location of equipment
- Magazines, PIUs, subboards, FEs are numbered
- Form is Mag.Slot.SubPos.ElemPos
- PIU located by Mag.Slot
- PM located by Mag.Slot.SubPos.ElemPos
- Empty equipment ID denotes the entire node

Copyright © 2006 TietoEnator Corporation

Equipment ID - examples

- { 1.5 } - PIU inserted in slot 5 of magazine 1
- { 2.4.2.1 } - PM located in position 1 on subboard position 2 on PIU inserted in slot 4 of magazine 2
- { } - The entire node

Copyright © 2006 TietoEnator Corporation

Node hierarchy visualized

M3

3.1

3.1.2

PEB

SB_PEB_1

PowerPC

3.1.2.1

Copyright © 2006 TietoEnator Corporation

Node hierarchy visualized

3.3

3.3.2

M3

GPB

SB_Sparc_1

Sparc

3.3.2.1

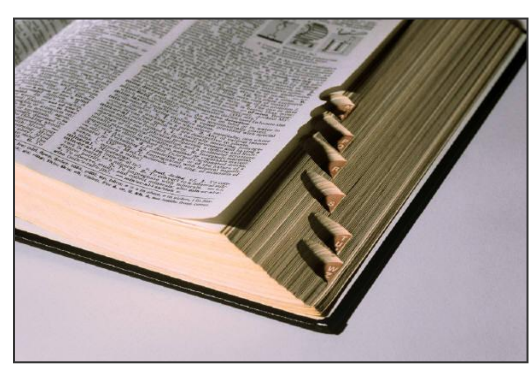
Copyright © 2006 TietoEnator Corporation

Equipment Products Table File

- .ept is delivered in NDP Core
- Defines all types of PIUs
    - Including subboards and FEs for PIU-types respectively
- Used by EQM to initialize Table of Equipment Products (TEP)
- Must NOT be modified by application developers

Copyright © 2006 TietoEnator Corporation

Crane Board Dictionary (CBD)



- Dynamic map maintained by DPE
- Associates logical names (named sets) with PIU locations
  - End-system applications can distribute functions over various sets PIUs without modifications in the source code
- Specified in a configuration file with extension .cbd
  - Specified as a named set of PIU selection criteria
  - PIU Location Criteria
  - PIU Type Criteria
- Logical names may occur in Application directives

Copyright © 2006 TietoEnator Corporation

CBD configuration file examples

- Location criteria:
  - location(AllowedNCBs 1.20)
  - location(AllowedNCBs 2.20)

- Type criteria:
  - type (TS_Solaris GPB)
  - type (TS_VxWorks PEB)
  - type (TS_VxWorks IBE1)
  - type (TS_VxWorks IBEN)
  - type (TS_VxWorks IBAM)

Copyright © 2006 TietoEnator Corporation

EQM, EQMA and EQSA

- A DPE slave process called EQMA (Equipment Management Agent) runs on every PM
- EQM detects remote hardware by catching broadcast messages emitted by EQMAs
- EQM monitors remote processors by sending poll messages to EQMAs. If an EQMA sends timely poll replies, EQM believes its processor is alive
- If EQM believes a remote processor is down, it does not send poll messages to its EQMA
- EQMAs use poll messages to maintain watchdogs
- EQSAs (EQM Sub-Agents) handle HW specific operations

Copyright © 2006 TietoEnator Corporation

TEIE and TCIE

- **T**able of **E**xpected **I**nstalled **E**quipment
  - Describes the “ideal” state of all the equipment in the node.
  - A piece of equipment not listed in TCIE is called ***`Foregin`*** and will not be available for use be DPE applications.
  - TEIE is loaded from the file `gsn.teie` and only changed when an operator invokes the CLI command `scale_up`.
- **T**able of **C**urrently **I**nstalled **E**quipment
  - Describes the current state of all the equipment in the node (as known to NCL).
  - Updated by EQM.
  - Can be queried by application instances, as well as by the various DPE services.
  - CLI command `list_eq`.

TietoEnator
Building the Information Society

Copyright © 2006 TietoEnator Corporation

AEA (All Equipment Available)

- Start-up HW detection process takes time
- How does EQM know when all hardware has been found?
  - NDM waits for AEA to be set, or for timeout
  - EQM enters information about detected hardware into TCIE
  - AEA is set by EQM when contents of TCIE = contents of TEIE
  - Applications started when all equipment present in last hardware snapshot has been found

Copyright © 2006 TietoEnator Corporation

Equipment State

- Operational (up or down)
- Administrative (deblocked, blocked, foreign)
- A PM which is up and deblocked can be used by applications
- A PM which is foreign is not part of the current node size and thus not available for use by applications.

Copyright © 2006 TietoEnator Corporation

Administrative State

- Deblocked (can be used), blocked (can not be used)
- Administrative State of PMs and PIUs can be set by applications
- Operators can block PIUs via GUI or the Repair Request button
- Applications must be prepared to clear BIs from PM that are to be blocked

Copyright © 2006 TietoEnator Corporation

Equipment Management

ERICSSON

- Logout
- Performance
  - Types
  - Groups
  - Jobs
- Subscriber
- Node Operation
  - Equipment
  - Execution
  - Software
  - Node Properties
- Interfaces
- Security
- Charging
- Logging
- Index
- Help

Equipment Elements

- NE
  - Magazine 1
    - Slot 1: Board
      - Pos 2
        - FE 1: PM
    - Slot 2: Board
    - Slot 3: Board
    - Slot 4: Board
    - Slot 5: Board
    - Slot 6: Board
    - Slot 8: Board
    - Slot 10: Board
    - Slot 12: Board
    - Slot 14: Board
    - Slot 16: Board
    - Slot 18: Board
    - Slot 20: Board
    - Slot 21: Board

Selected Equipment

EqId: 1/1/2/1
Class: PM
State: Up
Unblocked

Restart | Delete

Save | Refresh | Help

CLIENT: 12:35 — 0
6
NODE: 12:35 — 0
Alarms... — 0

```
5: Operation Refresh successful.
6: Operation GetInfo successful.
7: Operation GetInfo successful.
8: Operation GetInfo successful.
```

Applet com.ericsson.pcs.fwResultViewer.ResultViewer started

Building the Information Society

Copyright © 2006 TietoEnator Corporation

Equipment supervision

- When EQM detects that a PM has gone down it informs all applications that had BIs running on that PM. Applications may reallocate these BIs to other PMs
- When EQM detects new PMs, application root instances are informed
- Applications may request to be informed when operational or administrative state of a particular PM changes
- A state register variable is set every time the set of PIUs associated with a logical name changes

Copyright © 2006 TietoEnator Corporation

Auxiliary services

- These services can be used by applications that need more detailed information about the hardware in a node
- Think twice about using these services – using them is against the spirit of DPE

Copyright © 2006 TietoEnator Corporation

Auxiliary services... Hardware information

- List equipment
- Get equipment attributes (operational state, administrative state, etc)
- Get IP address of a processor

Copyright © 2006 TietoEnator Corporation

Auxiliary Services... Crane Board Dictionary

- Get PIUs associated with CBD logical name
- Add or Remove
  - Type Criteria
  - Location Criteria

  to a Logical Name
- Run-time changes of CBD are not persistent

Copyright © 2006 TietoEnator Corporation

Auxiliary services... Control Equipment

- Block or Deblock a PIU
- Restart Function Elements such as processors
- Reset the whole node
- Remove entry for equipment that is known to be down

Copyright © 2006 TietoEnator Corporation

Auxiliary Services... Active and Backup NCL

- CBD logical name – Active NCB
- CBD logical name – Backup NCB
- State Register Variable – Backup Available

Copyright © 2006 TietoEnator Corporation

Summary

- EQM maps available hardware
  - State, Availability, Supervision
- Function Elements = Processors and IO cards
- Equipment ID = Mag.Slot.SubPos.ElemPos
- EQMAs run on every processor
- Avoid having applications using Auxiliary services

Copyright © 2006 TietoEnator Corporation

Distributed Processing Environment (DPE)

7. Introduction to Function Distribution Management

- Purpose of FDM
- Services provided by FDM
- Principles of FDM

Copyright © 2006 TietoEnator Corporation

Purpose of FDM

- To support applications in distributing their block instances, so that:
  1) required functionality is provided;
  2) fault tolerance is guaranteed;
  3) available resources are utilized to meet capacity requirements.
- This typically means:
  1) distribution of block instances to certain, dedicated PMs;
  2) distribution of ”hot stand-by” instances of blocks;
  3.1) distribution of block instances to each board of a certain type;
  3.2) sharing of ”heavy” capsules, e.g., Erlang capsules.

! The same application code must be executable on several different node configurations!

Copyright © 2006 TietoEnator Corporation

Services provided by FDM

- FDM provides an API for applications to:
  - specify adequate distributions of block instances;
  - generate adequate distributions of block instances.

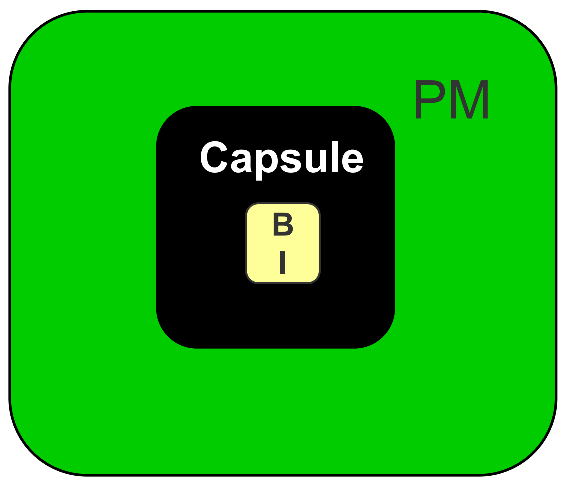
Copyright © 2006 TietoEnator Corporation

Some Properties of Block Instance Names

A ***block instance name*** includes information about:

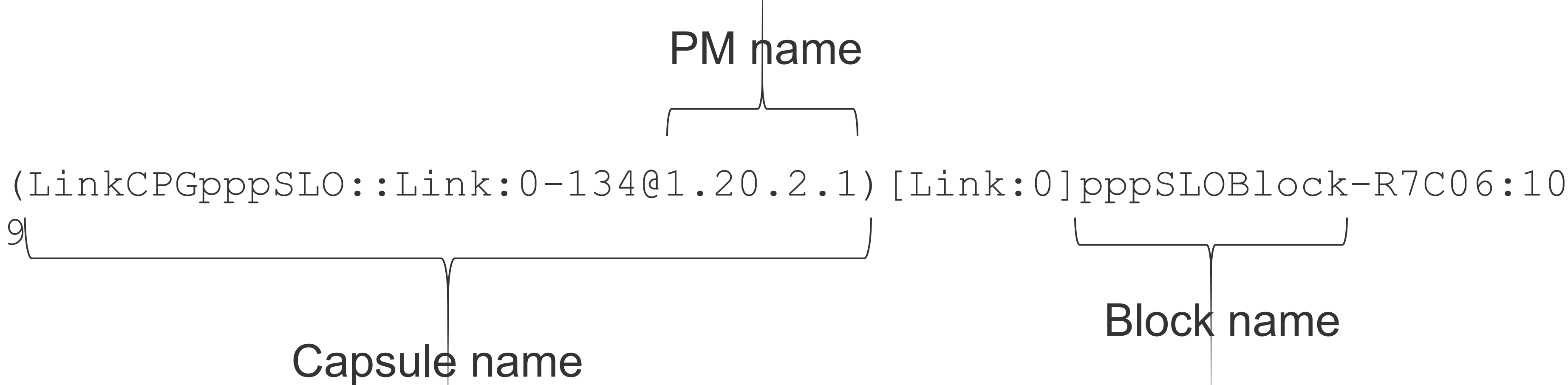
- the position of its PM;
- the name of the capsule in which it resides.

→ The name of a block instance determines its location.



Copyright © 2006 TietoEnator Corporation

Textual representation of a block instance name



Copyright © 2006 TietoEnator Corporation

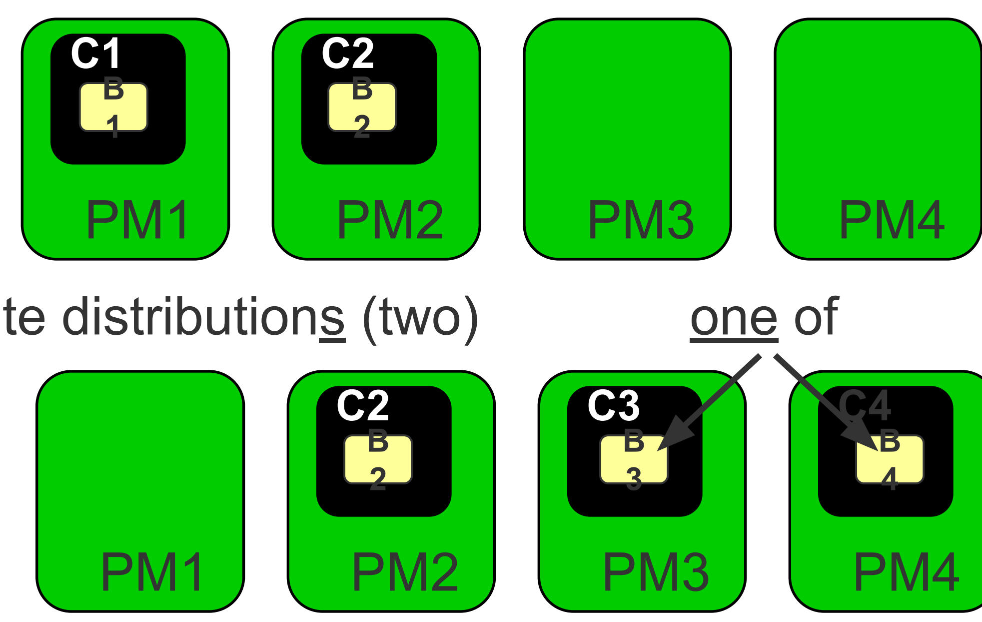
Definitions

- ***Distribution***
  - a set of block instance names.
- ***Current distribution***
  - the set of names of block instances which are in any state except for “Mapped” (i.e., which “exist”).
- ***Adequate distribution***
  - a distribution (i.e., a set of block instance names).
- ***Distribution difference***
  - two sets of names of block instances that must be created and deleted, respectively, in order for the current distribution to become an adequate distribution.

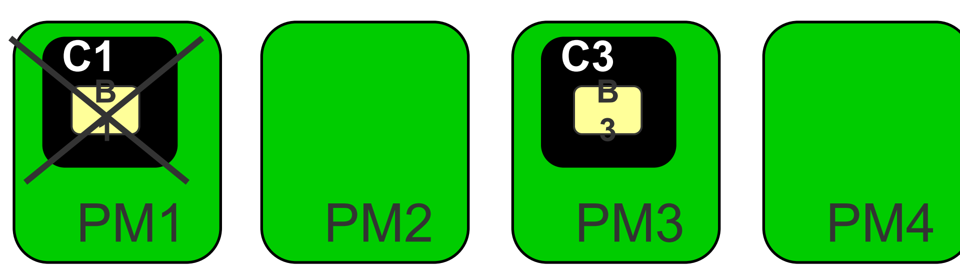
Copyright © 2006 TietoEnator Corporation

Example

Current distribution



Distribution difference (one of two possible)



Copyright © 2006 TietoEnator Corporation

Services Provided by FDM (again)

- Declaration of application directives
  - application directives specify adequate distributions.
- Allocation
  - allocation generates a distribution difference as result.

! The distribution difference should be considered a suggestion: DPE does not create a block instance until requested.

! No capsule is created until creation of a block instance in that capsule is requested.

Copyright © 2006 TietoEnator Corporation

FDM – Basic Principles

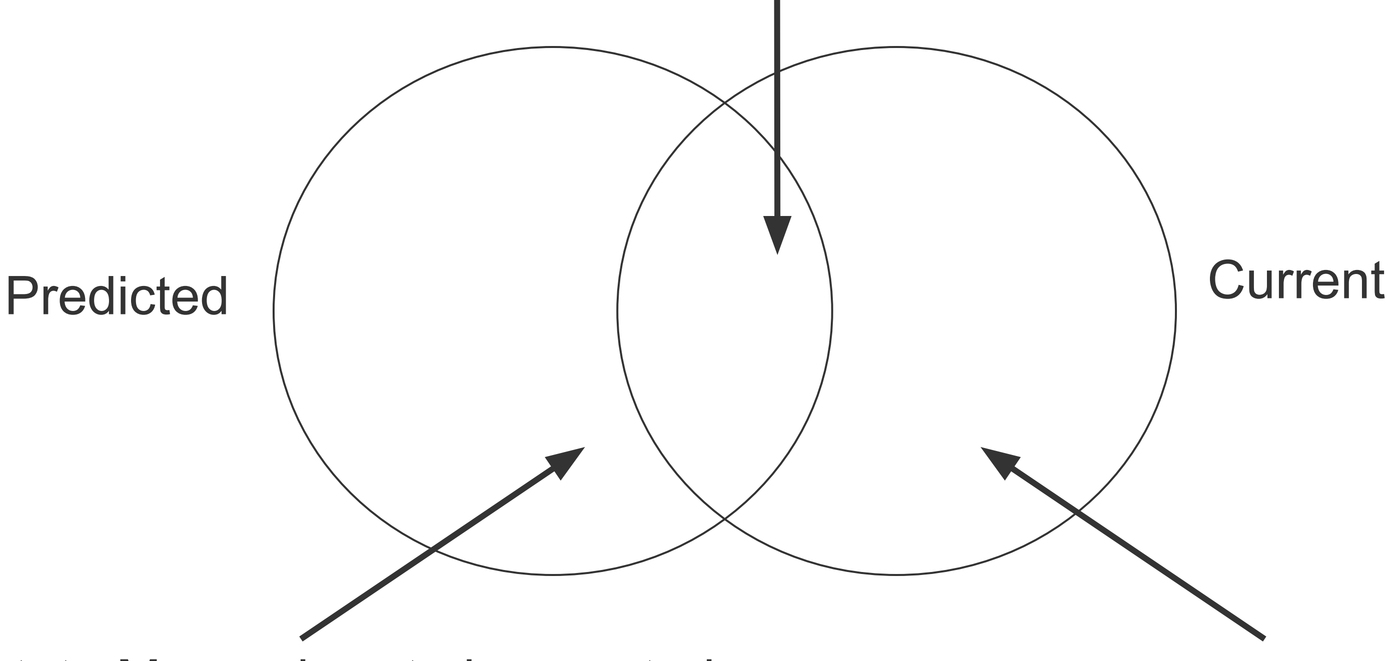
- Three types of application directives
  - PM Group (PMG) → constraint on PMs;
  - Capsule Group (CPG) → constraint on capsules;
  - Block Instance Group (BIG) → constraint on block instances.
- A combination of groups → adequate distributions.
- Allocation → infers ***current*** and generates ***predicted*** content: PMG → content = set of PM names;

  CPG → content = set of capsule names;

  BIG→ content = set of block Instance names;
- Distribution difference = predicted ”–” current content of BIG.

Copyright © 2006 TietoEnator Corporation

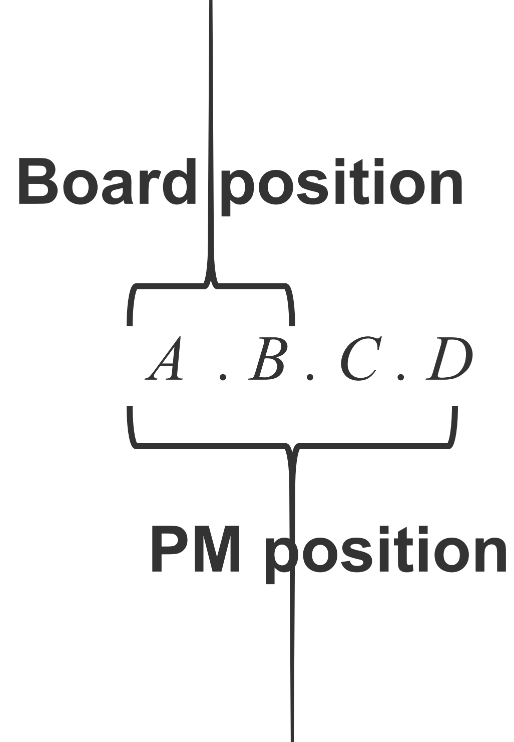
Current and Predicted Content of a BIG



Copyright © 2006 TietoEnator Corporation

Some Properties of Board and PM Positions

- The position of a board is specified by two numbers.
- The position of a PM is specified by four numbers.



Copyright © 2006 TietoEnator Corporation

Crane Board Dictionary (CBD) in a Nutshell

- CBD maps ***logical names*** to sets of board positions.
- Each set of board positions is either defined by:
  - an explicit enumeration of positions, or;
  - a ***board type***.
- Only positions of detected boards are included.
- CBD is initialized from a configuration file.
- There is an API for dynamically updating CBD.
- Predefined logical names: ”**`AllCraneBoards`**”
  ”**`ActiveNCB`**”
  ”**`BackupNCB`**”

Copyright © 2006 TietoEnator Corporation

PM Groups (PMG)

- Two criteria for including a PM in the PMG's content:
  - Supported capsule types;
  - Allowed PM positions (four types):
    - (basic) Explicit enumeration of PM positions;
    - (basic) All PMs in boards of a logical name;
    - (basic) Relative positions in boards of a logical name:

```
Logical name (in CBD): { 1.1,  1.2 }
Relative positions: { 3.1,  4.2 }

        →  PM positions = { 1.1.3.1, 1.1.4.2,
                            1.2.3.1, 1.2.4.2 }
```

    - (included) PM positions to which capsules of a CPG are allocated.

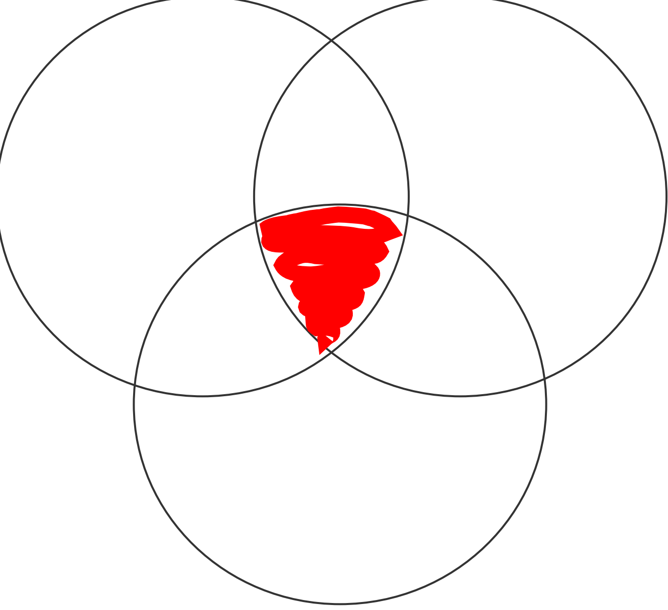
Building the Information Society

Copyright © 2006 TietoEnator Corporation

PMG Selection Criteria - Illustration

PMs that support specified capsule types

PMs that reside in specified positions



PMs that are *up* and *deblocked*

= PMs that Allocate includes in the predicted content of a PMG

Copyright © 2006 TietoEnator Corporation

Example – PMG1

PMG1 will contain all PMs that support Erlang capsules.

```
Name:          PMG1
Capsule types: { Erlang_Capsule }
Type:          All PMs in boards
Logical name:  AllCraneBoards
```

Copyright © 2006 TietoEnator Corporation

Example - PMG2

PMG2 will contain all PMs that:

1) support C **and** Erlang capsules, and;
2) are in relative position 2.1 of;
3) boards associated with the logical name `MyBoards`.

```
Name:              PMG2
Capsule types:        { Solaris_C_Capsule,
                     Erlang_Capsule }
Type:              Selected PMs in boards
Logical name:         MyBoards
Relative positions:   { 2.1 }
```

Copyright © 2006 TietoEnator Corporation

Capsule Groups (CPG)

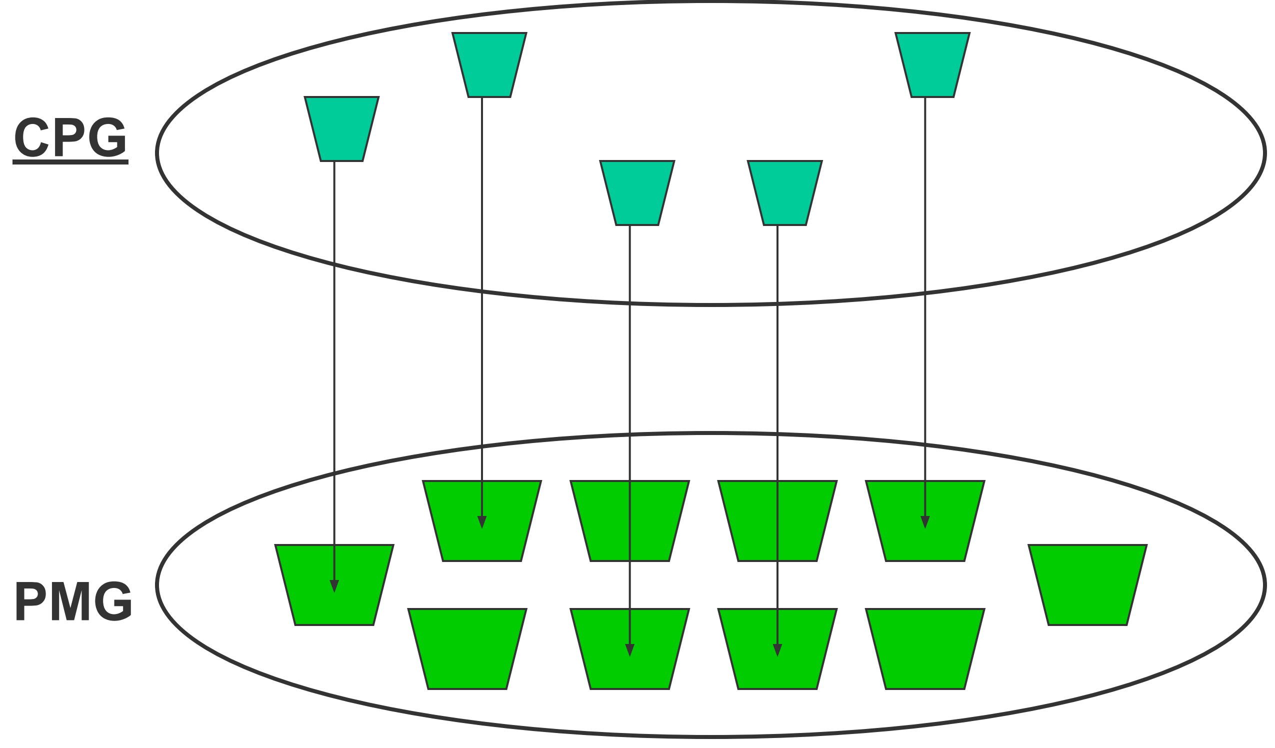
- Two types of CPGs
  - Basic: PMG + capsule type + a size ***N***
    should contain ***N*** capsules of the specified type.
    At most one capsule per PM in the specified PMG.

  - Included: another CPG + PMG + a size ***N***
    should contain ***N*** of the capsules included in the other CPG.
    Each capsule must reside on some PM in the specified PMG.

! ***N*** < 0 means ”all”.

Copyright © 2006 TietoEnator Corporation

CPG Selection Criteria (Basic) - Illustration



Copyright © 2006 TietoEnator Corporation

CPG Selection Criteria (Included) - Illustration

CPG

CPG (super)

PMG

PMG (filter)

Copyright © 2006 TietoEnator Corporation

Example – CPG1

The Capsule Group CPG1 should contain one Erlang capsule on each PM in PMG1.

```
Name:               CPG1
Type:               Basic
PM Group            PMG1
Capsule type:          Erlang_Capsule
Size:               -1 (i.e., all PMs)
```

Copyright © 2006 TietoEnator Corporation

Example – CPG2

The Capsule Group CPG2 should contain two of the capsules in CPG1 that reside on PMs in PMG2.

```
Name:               CPG2
Type:               Included
CPG (super group):CPG1
PMG (filter):        PMG2
Size:               2
```

Copyright © 2006 TietoEnator Corporation

Block Instance Groups (BIG)

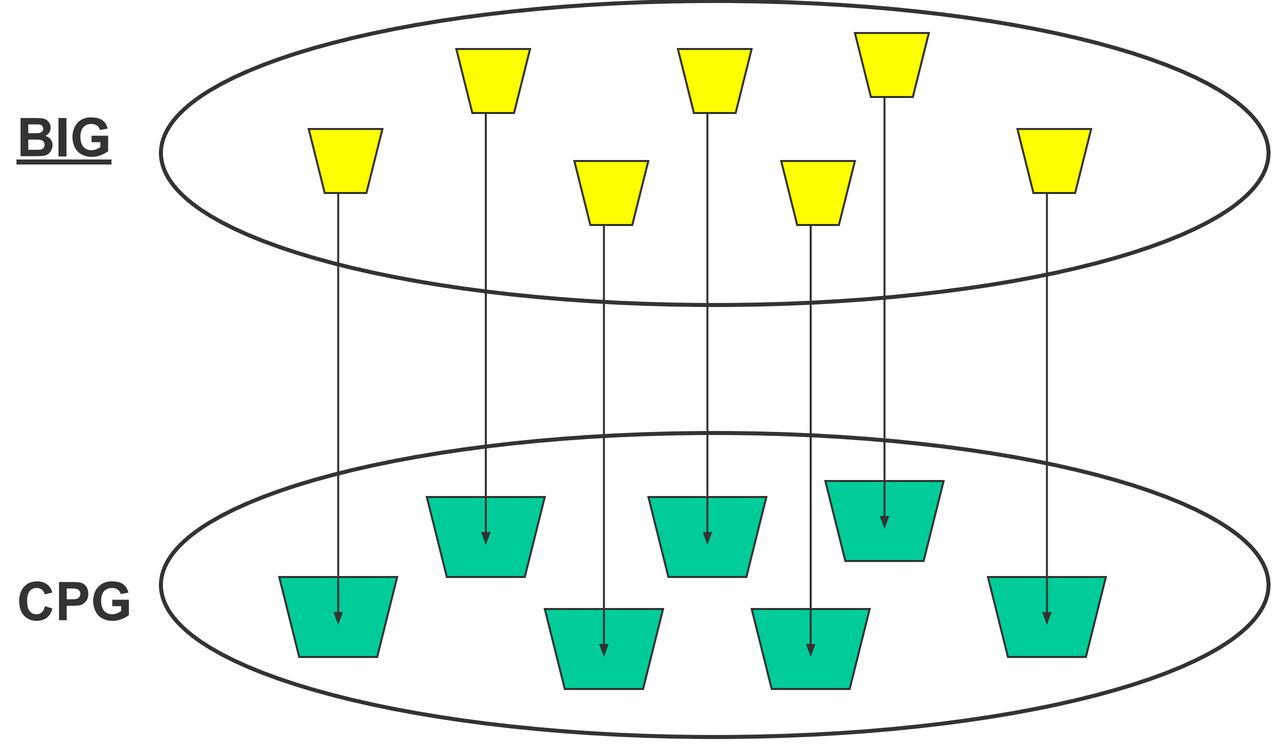
- Only one type of BIG
  - Block name + CPG + weight

    should contain exactly one instance of the block in each capsule of the CPG.

! The weight is used for rudimentary, static load balancing. It must be within the range 1 - 1000.

Copyright © 2006 TietoEnator Corporation

BIG Selection Criterion - Illustration



Copyright © 2006 TietoEnator Corporation

Example – BIG1 and BIG2

The Block Instance Group BIG1 should contain one instance of the block E1 in each capsule in CPG2

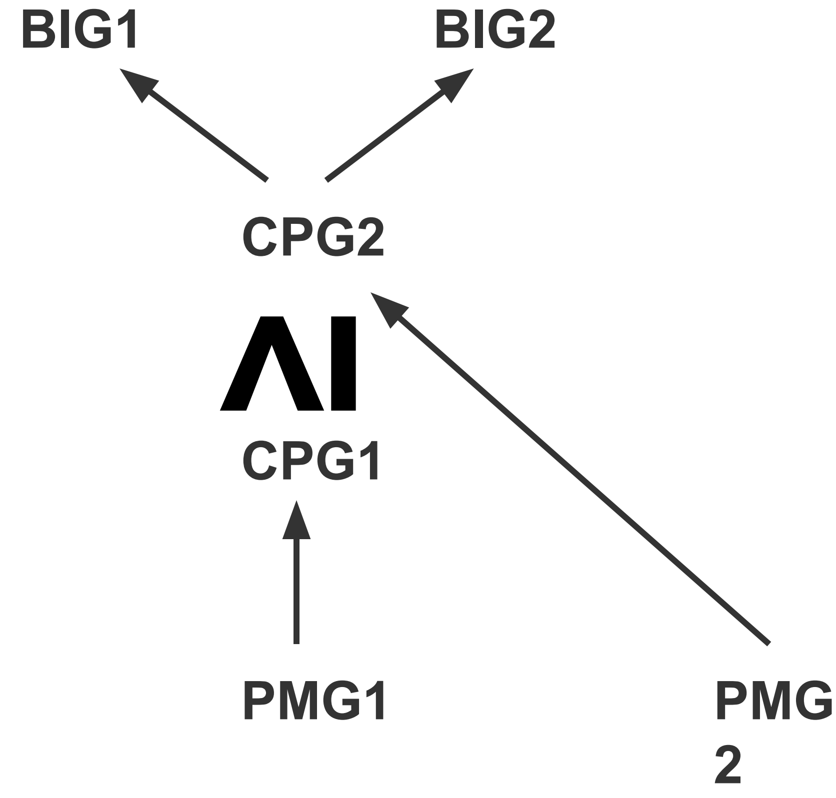
The Block Instance Group BIG2 should contain one instance of the block E2 in the same two capsules.

```
Name:          BIG1
Block name:       E1
CPG:           CPG2
Weight:        500
```

```
Name:          BIG2
Block name:       E2
CPG:           CPG2
Weight:        500
```

Copyright © 2006 TietoEnator Corporation

Example – Summary of Application Directives



Copyright © 2006 TietoEnator Corporation

Example – Predicted Contents

BIG1 BIG2

CPG1

CPG2

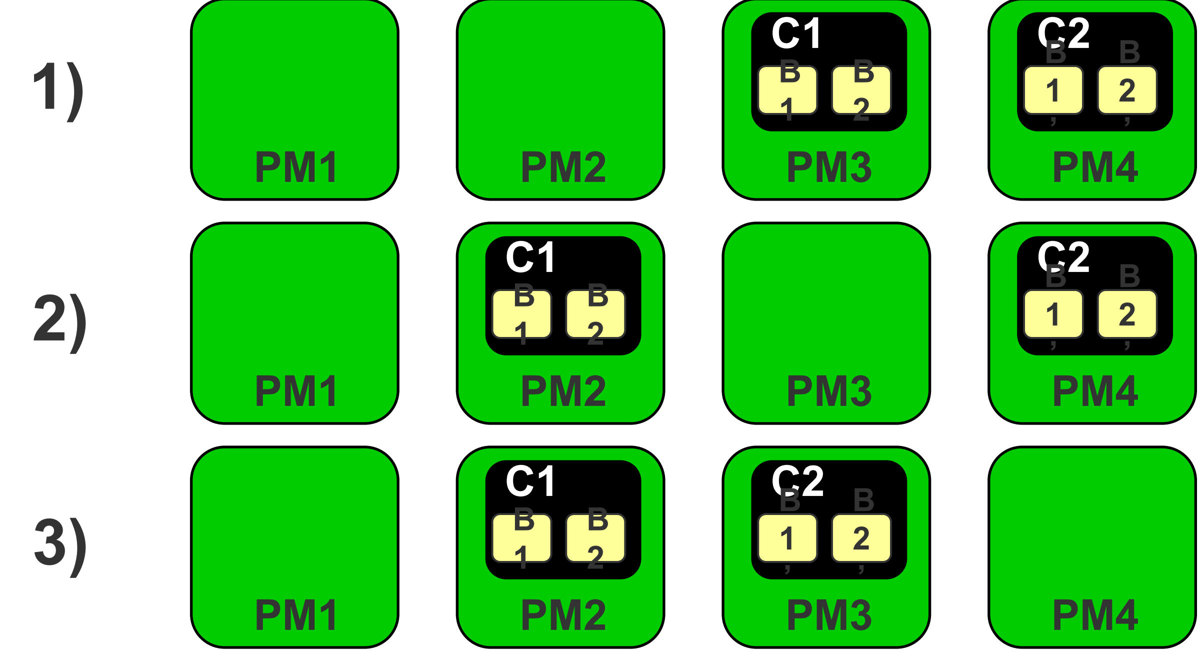
PMG1 PMG2

Copyright © 2006 TietoEnator Corporation

Example – Adequate Distributions

- PM1, PM2, PM3, and PM4 belong to PMG1
- PM2, PM3, PM4 belong to PMG2

→ adequate distributions:



Copyright © 2006 TietoEnator Corporation

Example – PMG3

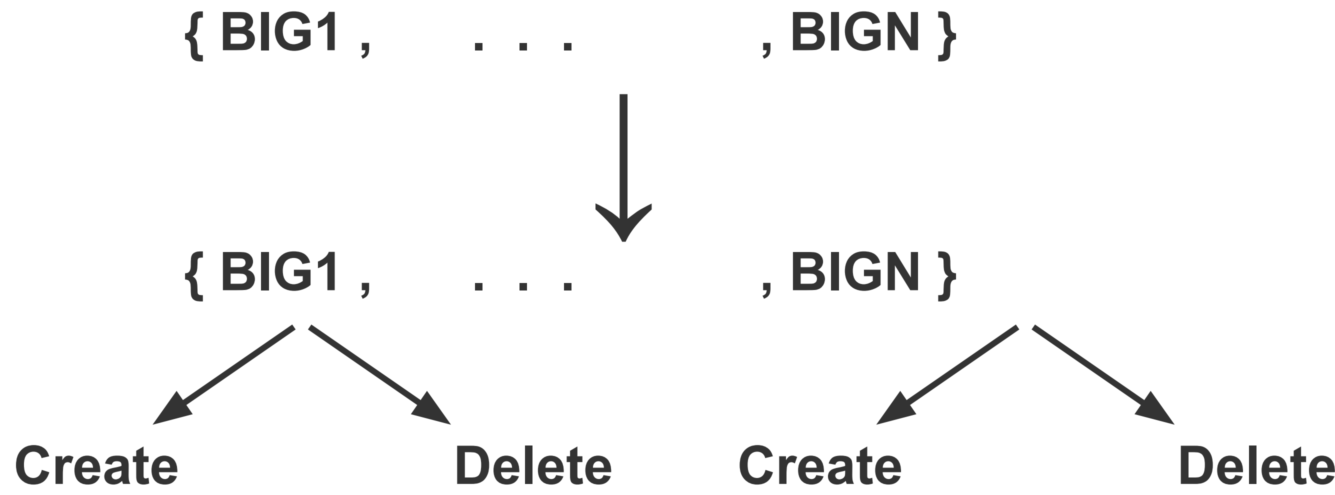
PMG3 will contain exactly those PMs on which some capsule in CPG2 resides.

```
Name:          PMG3
Type:          Included
CPG indicator: CPG2
```

Copyright © 2006 TietoEnator Corporation

Allocation

- Input: a set of BIG names
- Output: a distribution difference for each BIG



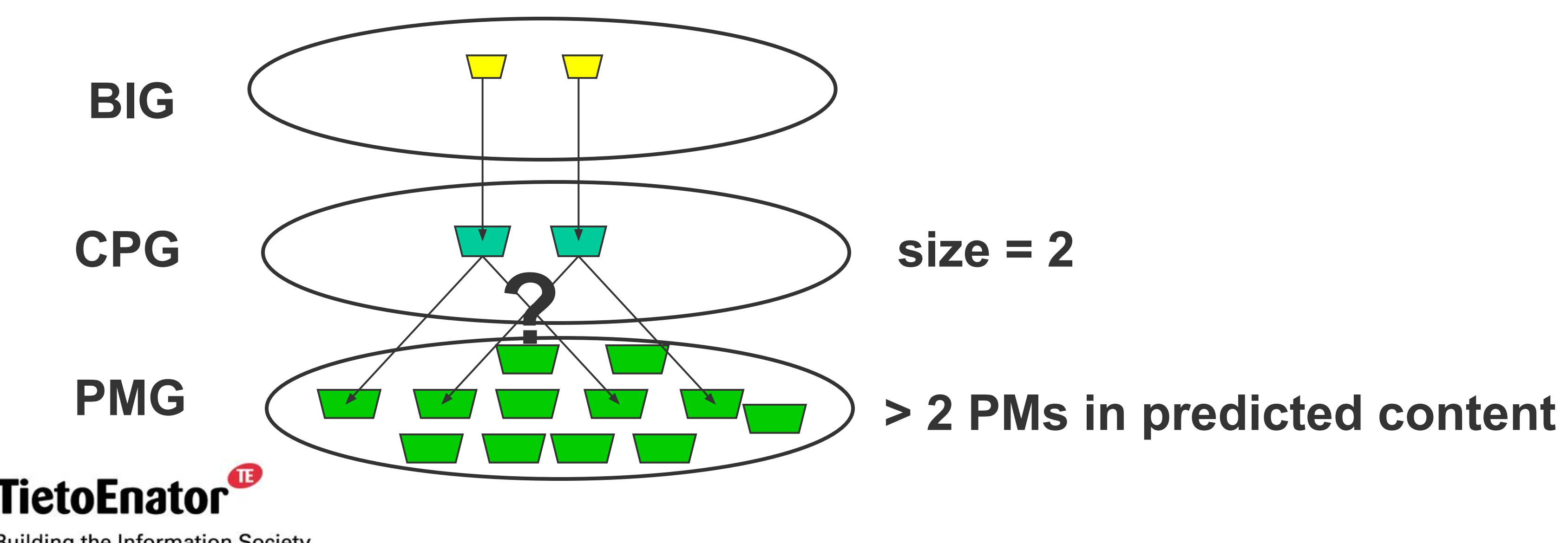
! Possible to specify the set of all BIGs of an application as input

Copyright © 2006 TietoEnator Corporation

Load Balancing

- Static balancing of ***predicted*** load
- Based on weights specified for BIGs
- Applicable only when there is a choice

Example



Copyright © 2006 TietoEnator Corporation

Priorities

Priorities applied by Allocate (decreasing order):

1) Satisfaction of application directives;
2) Minimize modification of current distribution;
3) Uniform load balancing.

Copyright © 2006 TietoEnator Corporation

Scopes of Group Names

- Each group name (of PMG, CPG, BIG) has a ***scope***
  - NCL Group ”belongs” to NCL;
  - Global Group has no specific ”owner”;
  - Application Group ”belongs” to one application.
- Purpose
  - To avoid name conflicts;
  - To indicate which entities may use the group.
- Example

```
LinkBIGpppSLO::Link:0
```

Copyright © 2006 TietoEnator Corporation

Predefined Capsule Groups

- For each capsule type, that:
  - is loadable;
  - has a name of the form ”***Name***_Capsule”

  there is a CPG called ***Name*** with global scope.
- The CPG contains one capsule per PM that supports the type.

  Example:

  ```
  ”Erlang_Capsule”  →  ”Erlang”
  ”Java_Capsule”    →  ”Java”
  ```

  cf. The example definition of CPG1

Copyright © 2006 TietoEnator Corporation

Error situations

- Errors when declaring a PMG, CPG or BIG
  - name is already in use;
  - group depends on undeclared entity;
  - incompatibilities (e.g., mismatching capsule types).
- Errors during allocation
  - insufficient number of PMs.

! The application directives of an application exist until the application has been stopped.

! Application directives cannot be modified.

Copyright © 2006 TietoEnator Corporation

Distributed Processing Environment (DPE)

8. Introduction to State Register

- The Purpose of the State Register
- Operations available to applications

Copyright © 2006 TietoEnator Corporation

The Purpose of the State Register

The purpose of the State Register is to provide a general and extensible mechanism for:

- Synchronisation
  - Applications may need to synchronise their activities with other applications.
- Publication of information
  - Applications may wish to publish data that is visible to other applications.
- Safe storage of data
  - Applications may want stored data to be maintained even in the case of SW or HW failure.

Copyright © 2006 TietoEnator Corporation

The State Register

- A set of State Variables.
- Managed by NCL.
- Applications are able to publish and modify a State Variable.
- Applications are able to subscribe to modifications of a State Variable.
- All modifications of the State Register are replicated to the Backup NCL.

Copyright © 2006 TietoEnator Corporation

The State Variable

Each entry in the State Register has the following properties:

- the name of a state variable.
- the status of the state variable: TRUE or FALSE.
- the value of the state variable, only defined if Status =TRUE.
- a set of block instance names, called the subscribers to the state variable.
- a set of block instance names, called the providers of the service represented by the state variable.
- a set of block instance names, called the acknowledgement requesters of the state variable.

Copyright © 2006 TietoEnator Corporation

An example of the State Register

Name *(string)*	Status *(boolean)*	Value *(byte array)*	Providers *(set of BI:s)*	Subscribers *(set of BI:s)*
"APPL_A"	**TRUE**	**"Name"**	**Root instance of A**	**B1, C2**
"SS#7"	**TRUE**	**NULL**	**C3**	
B_STATE	**FALSE**	**...**	**...**	
A_PARAM	**TRUE**	**"Value"**	**A0**	**A1,A2,A3**
DPE_SR_Node Up	**TRUE**	**NULL**	**All root instances**	**X,Y,Z**

Building the Information Society

Copyright © 2006 TietoEnator Corporation

Subscribers, an example

Name *(string)*	Status *(boolean)*	Value *(byte array)*	Providers *(set of BI:s)*	Subscribers *(set of BI:s)*
"APPL_A"	**TRUE**	**"Name"**	**Root instance of A**	**B1, C2**
"SS#7"	**TRUE**	**NULL**	**C3**	
B_STATE	**FALSE**	**...**	**B0**	
A_PARAM	**TRUE**	**"Value"**	**A0**	**A1,A2,A3**
...	**...**	**...**	**...**	**...**

A0

NCL

A_PARAM, TRUE, ...

A1

A2

A3

Building the Information Society

Copyright © 2006 TietoEnator Corporation

Providers, an example

Name (string)	Status (boolean)	Value (byte array)	Providers (set of BI:s)	Subscribers (set of BI:s)
"APPL_A"	**TRUE**	**"Name"**	**Root instance of A**	**B1, C2**
"SS#7"	**TRUE**	**NULL**	**C3**	
B_STATE	**FALSE**	**...**	**...**	
A_PARAM	**FALSE**	**"Value"**	**...**	**A1,A2,A3**
...	**...**	**...**	**...**	**...**

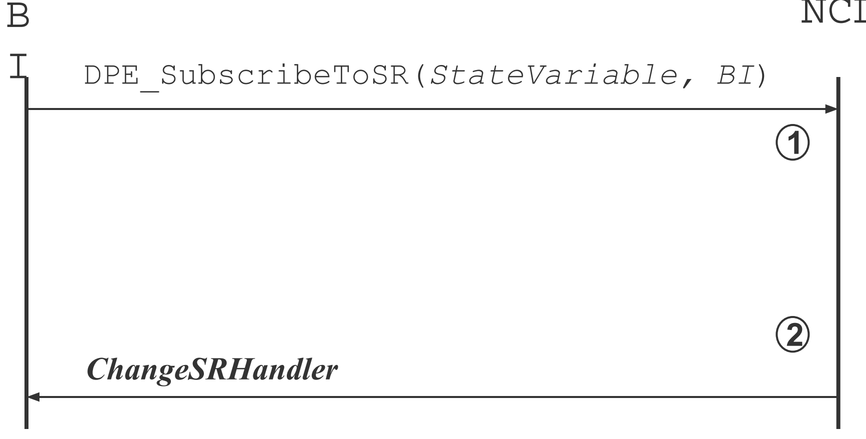
Copyright © 2006 TietoEnator Corporation

Operations available to applications

- Subscribe to a State Variable
- Unsubscribe to a State Variable
- (Re)initialise a State Variable, with or without acknowledgement request
- Set the value of a State Variable
- Reset a State Variable
- Request information about the subscribers
- Request information about the providers

Copyright © 2006 TietoEnator Corporation

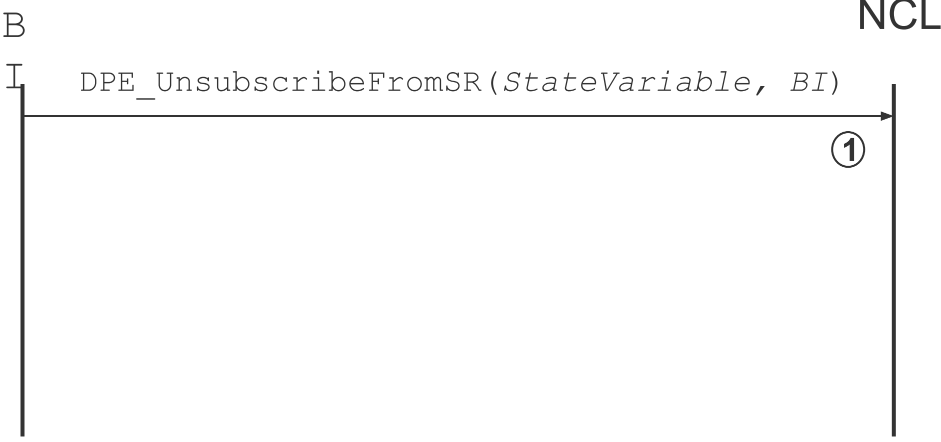
Subscribe to a State Variable



1. NCL adds *BI* to the set of subscribers to *StateVariable*
2. NCL responds to BI with information about the State Variable

Copyright © 2006 TietoEnator Corporation

Unsubscribe from a State Variable

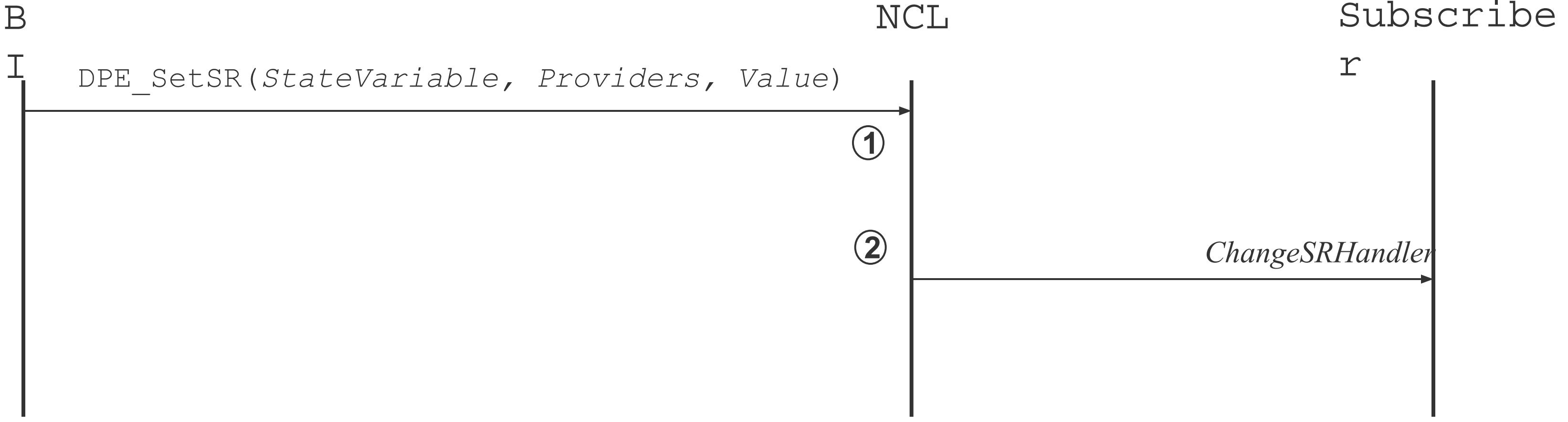


1 NCL removes *BI* from the set of subscribers to *StateVariable*

No notification is expected

Copyright © 2006 TietoEnator Corporation

(Re)initialize a State Variable

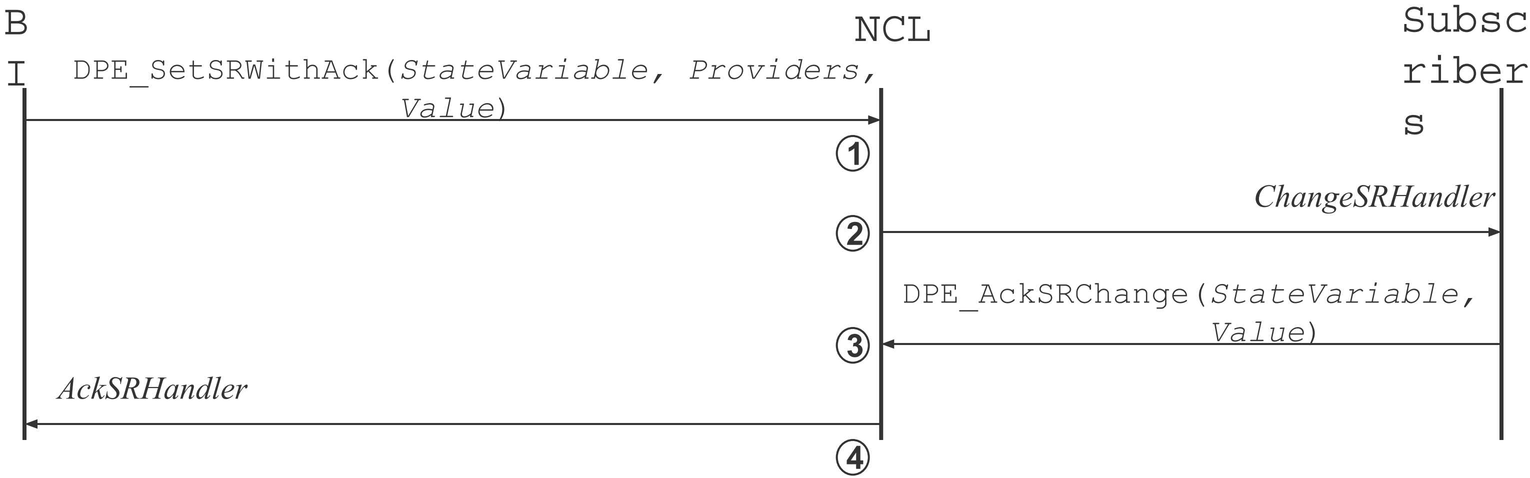


① NCL will set the value of *StateVariable* together with its set of *Providers*

② NCL will notify all subscribers to *StateVariable*

Copyright © 2006 TietoEnator Corporation

(Re)initialize a State Variable with acknowledgements

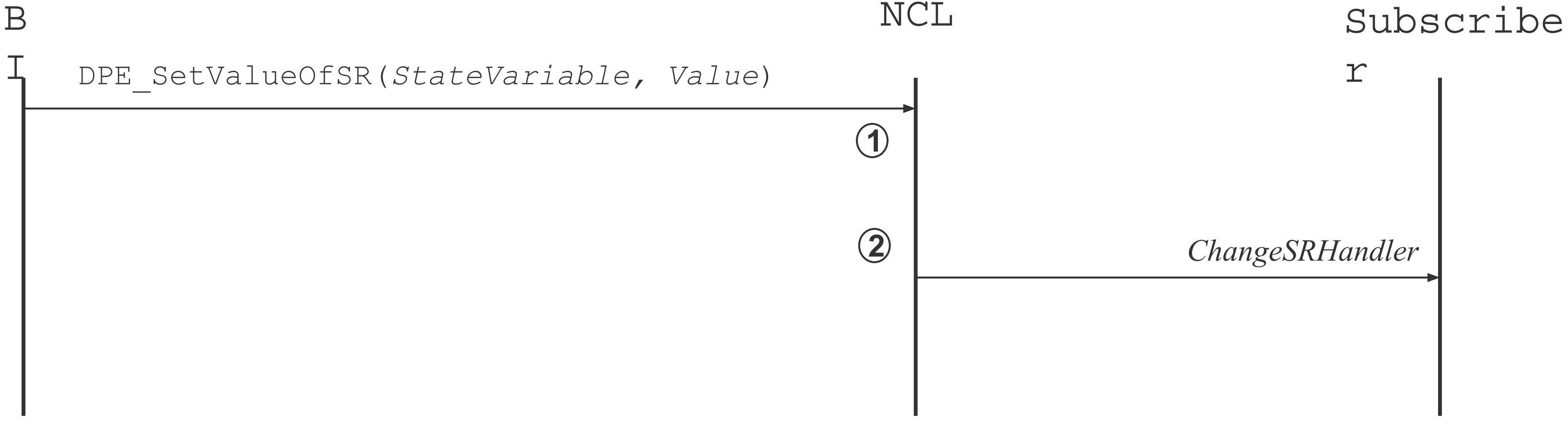


1. NCL will set the value of *StateVariable* together with its set of *Providers*
2. NCL will notify all subscribers to *StateVariable*
3. The subscribers will acknowledge the change
4. NCL will notify the modifying block instance when all subscribers have acknowledged

Building the Information Society

Copyright © 2006 TietoEnator Corporation

Set the value of a State Variable

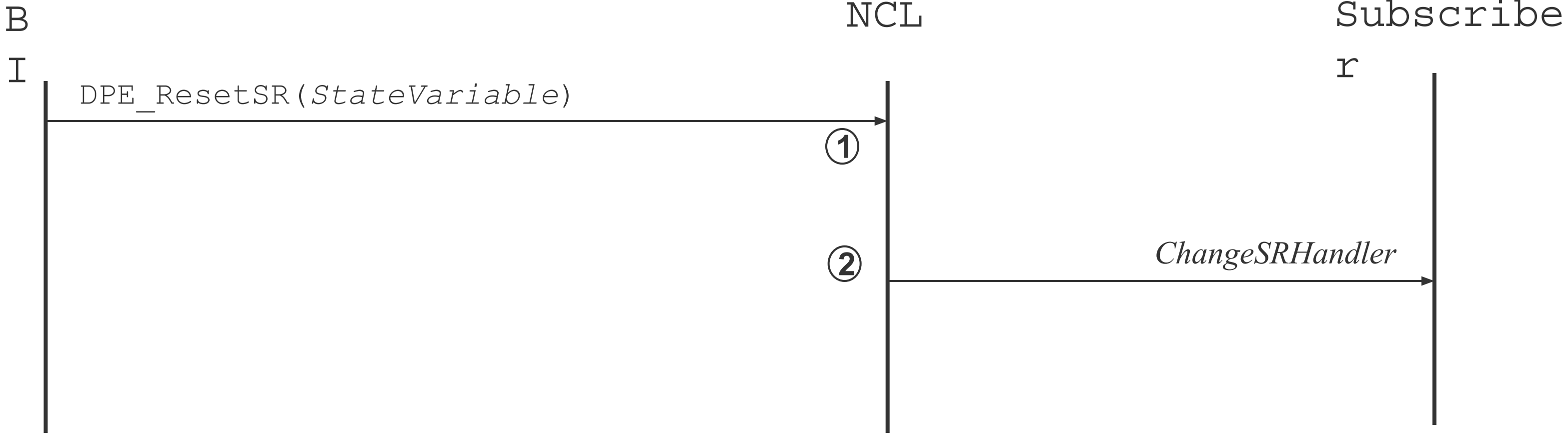


1. NCL will set the value of *StateVariable*

2. NCL will notify all subscribers to *StateVariable*

Copyright © 2006 TietoEnator Corporation

Reset a State Variable

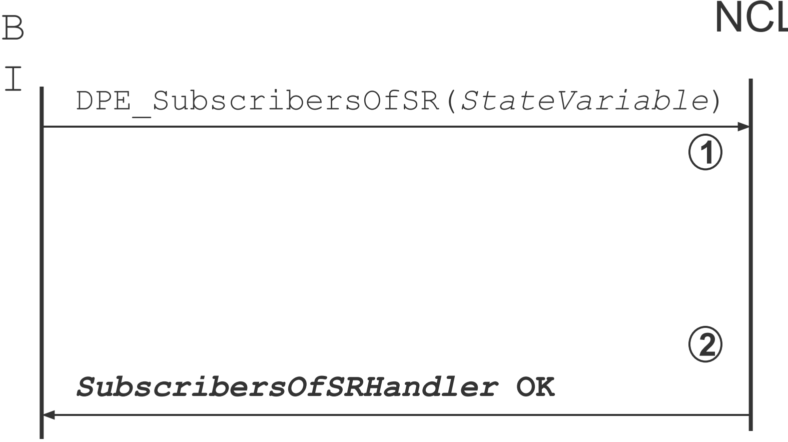


① NCL will set the status of *StateVariable* to FALSE

② NCL will notify all subscribers to *StateVariable*

Copyright © 2006 TietoEnator Corporation

Request information about the subscribers

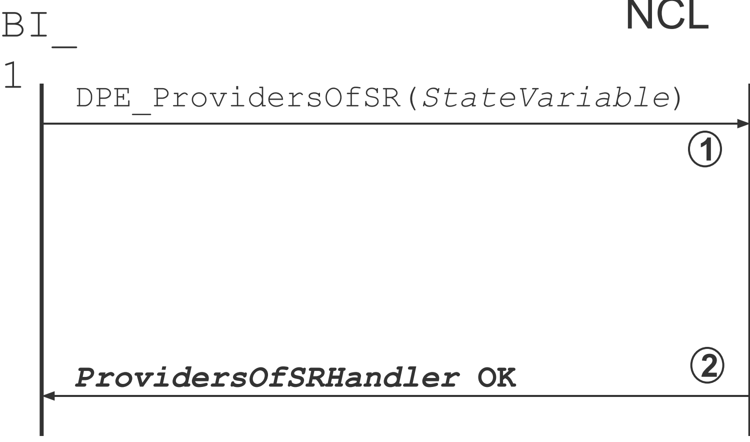


1. NCL will retrieve the set of subscribers to *StateVariable* from the State Register.

2. NCL responds to BI with the set of subscribers.

Copyright © 2006 TietoEnator Corporation

Request information about the providers



1. NCL will retrieve the set of providers to *StateVariable* from the State Register.

2. NCL responds to BI with the set of providers.

Copyright © 2006 TietoEnator Corporation

State Variables owned by NCL

- APPL_<X>
- DPE_ SR_NodeUp
- DPE_SR_DpeRoot

Copyright © 2006 TietoEnator Corporation

State Variables owned by NCL, continued

- DPE_SR_CurrentSC
- DPE_SR_NextSC
- DPE_SR_PreviousSC

Copyright © 2006 TietoEnator Corporation

State Variables owned by NCL, continued

- DPE_SR_BackupNCLAvailable
- DPE_SR_ActiveNCLAvailable
- DPE_SR_NewPMsAvailable
- DPE_SR_CBDChanged
- DPE_SR_StoppingAllApplications

Copyright © 2006 TietoEnator Corporation

Distributed Processing Environment (DPE)

9. Software Management

- Software Management Services
- Delivery Packages
  - ADP
  - NDP
  - DDP
- NDP installation
- Structure of the file system
- Software Configurations

Copyright © 2006 TietoEnator Corporation

Software Management Services

- Installation of software
- Removal of installed software
- Management of software configurations
  - Activation of a SC
  - Checkpoint of a SC
  - Verify consistency of a SC
  - Set a SC to be used for next restart
  - Set a SC to be used as default

Copyright © 2006 TietoEnator Corporation

Delivery Packages

- A compressed archive file (*tar*-file)
- Three types of delivery packages:
  - Application Delivery Package (ADP)
  - Node Delivery Package (NDP)
  - Development Delivery Package (DDP)

Copyright © 2006 TietoEnator Corporation

Application Delivery Package (ADP)

- Mandatory files:
  - Load units
  - Application structure tree file
  - Block to load unit map file
  - Upgrade action file
- Optional files:
  - Boot files
  - Application specific configuration data
  - Application specific web server data

Copyright © 2006 TietoEnator Corporation

Application Delivery Package (ADP), cont'd

```
ApplicationName
        ├── ApplicationStructureTree.ast
        ├── BlockToLoadUnitMap.blu
        └── UpgradeActionFile.uac

BootFiles    LoadUnits    ApplicationData    Webbase
```

Copyright © 2006 TietoEnator Corporation

Node Delivery Package (NDP)

- Mandatory files:
  - Boot files
  - NCL load units
  - ADPs
  - Crane Board dictionary definitions file
  - Capsule attributes file
  - Node attributes file
  - Product definition file
  - Persistent Application file
  - Scripts
  - Dynamic link libraries

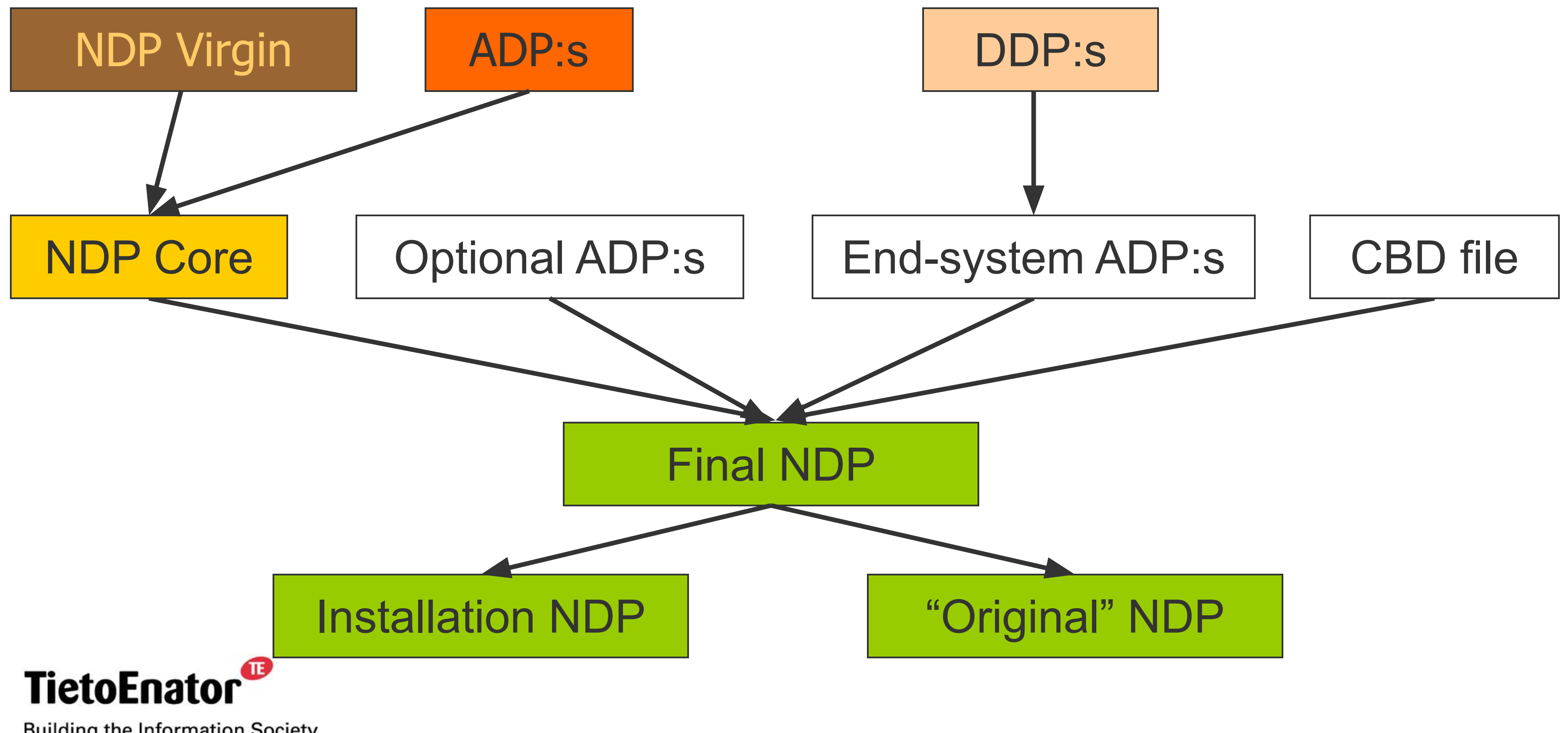
Copyright © 2006 TietoEnator Corporation

Node Delivery Package (NDP), cont'd

```
                      NDPName
                         |
                         |---- [revision_of_DPs]
                         |
   +--------+--------+---+------+-----------------+
   |        |        |          |      |          |
Config     Lib   BootFiles     NCL    ADPs      Scripts
                                       |
                                       |---- PersistentApplications.pap
                                       |
                            +----------+----------+
                            |                     |
                   ADPName_1.tar.Z       ADPName_2.tar.Z
```

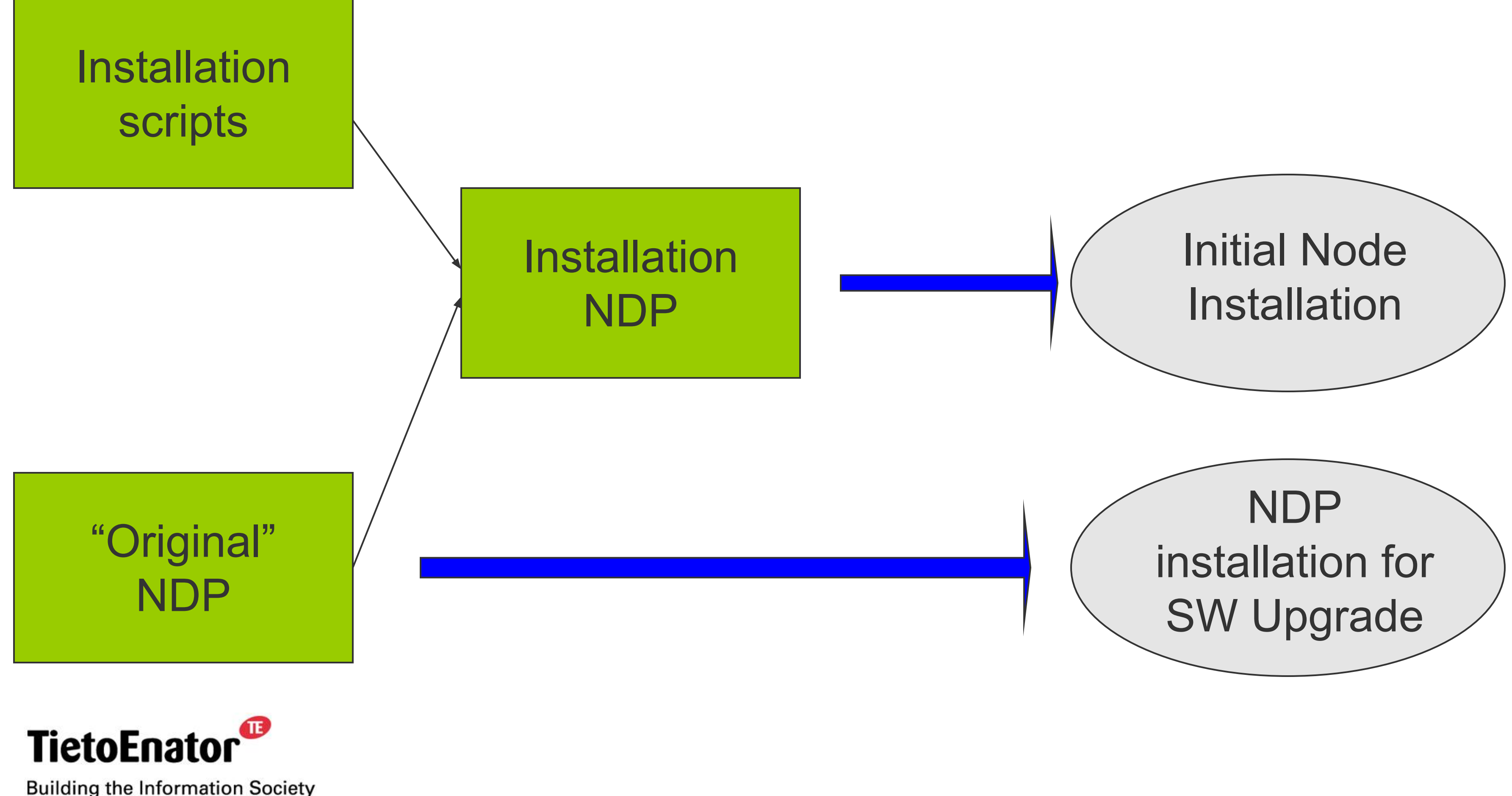
Copyright © 2006 TietoEnator Corporation

The process of building a final NDP



Copyright © 2006 TietoEnator Corporation

Installation NDP vs “Original” NDP



Copyright © 2006 TietoEnator Corporation

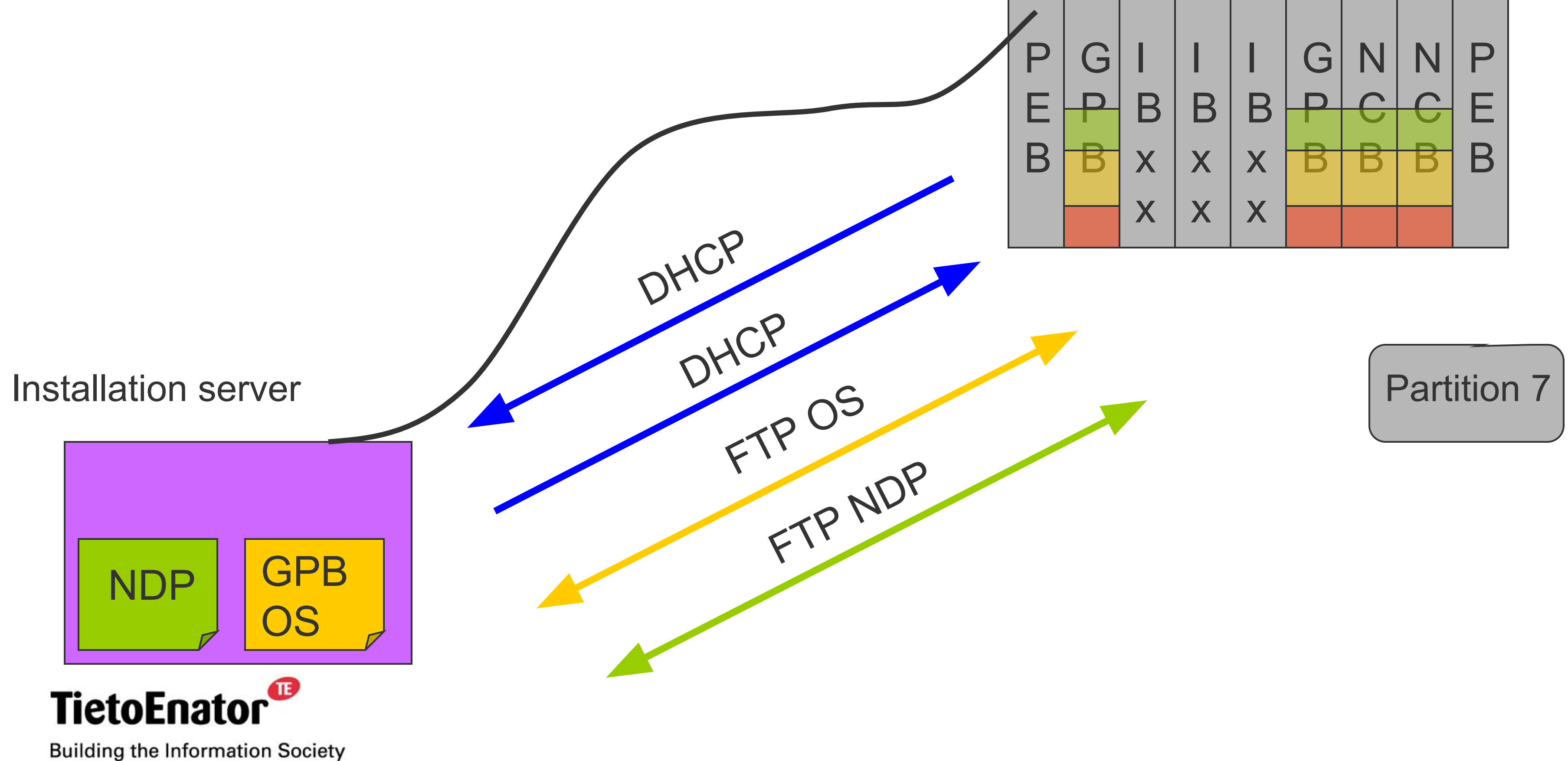
Initial Node installation

- An external installation server is needed
- Connected to the internal node network via the PEB
- The installation server is a Dynamic Host Configuration Protocol (DHCP) and FTP server

Copyright © 2006 TietoEnator Corporation

Initial Node installation, cont'd

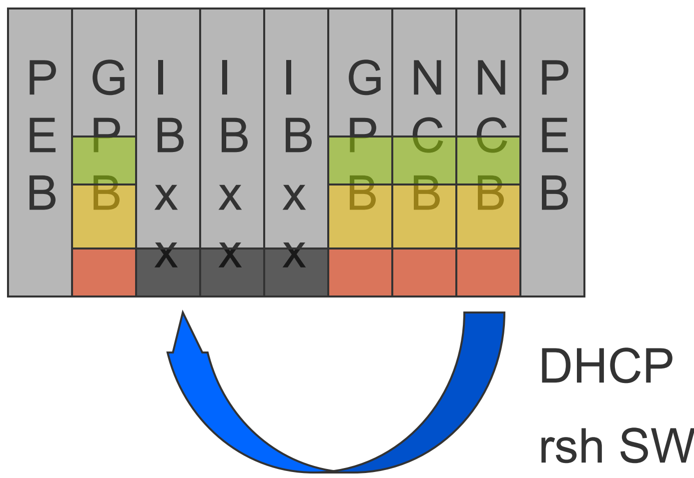
- SW installed from external server



Copyright © 2006 TietoEnator Corporation

IBxx installation

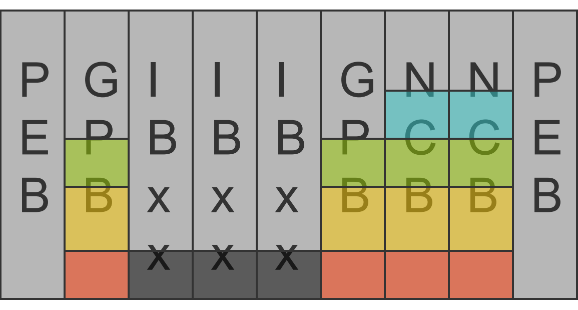
- The active NCB acts as installation server
- IBxx boards get their SW from the active NCB



Copyright © 2006 TietoEnator Corporation

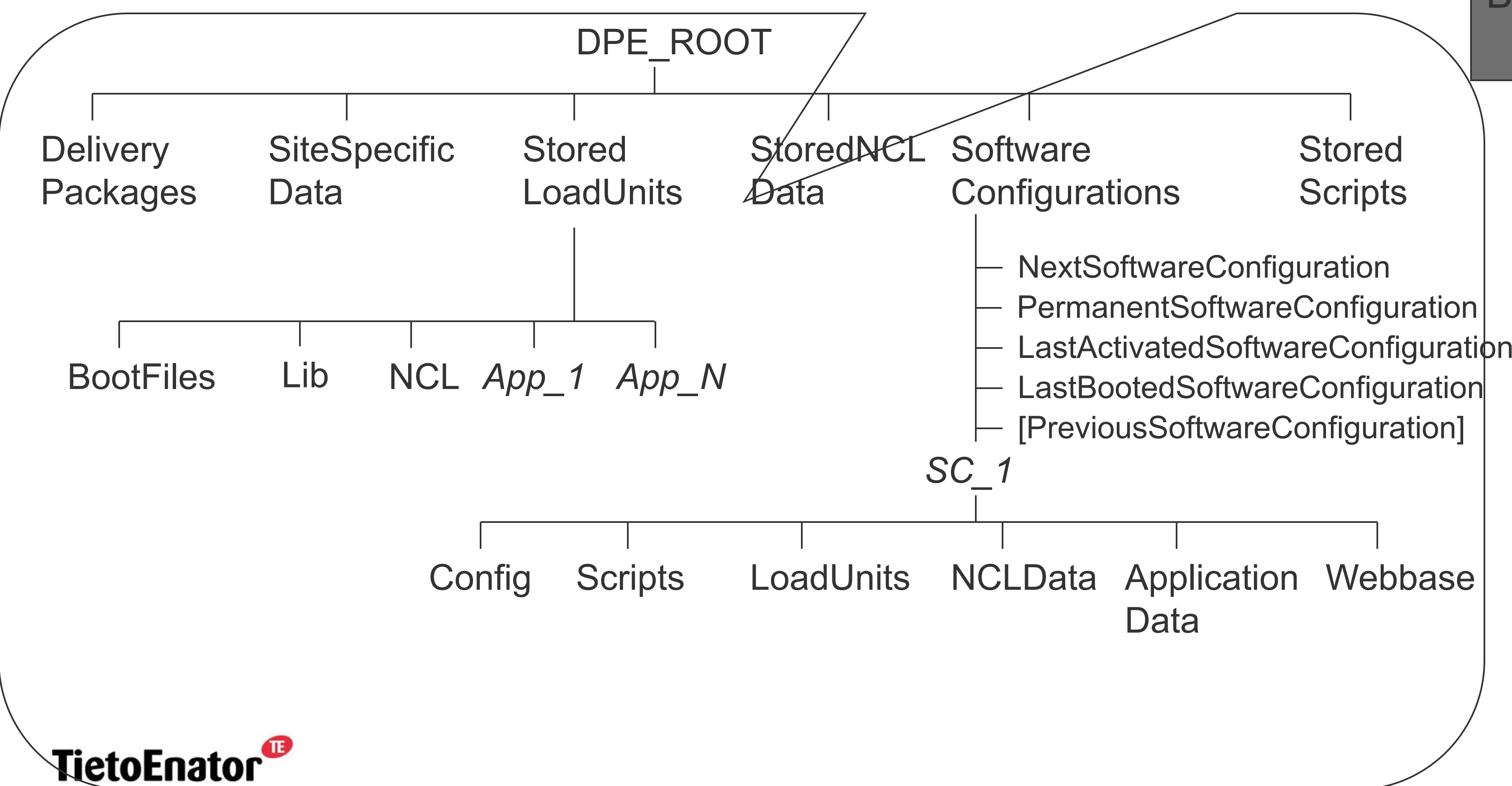
NDP installation for SW Upgrade

- The node SW remains running
- Operator installs new NDP via the GUI



Copyright © 2006 TietoEnator Corporation

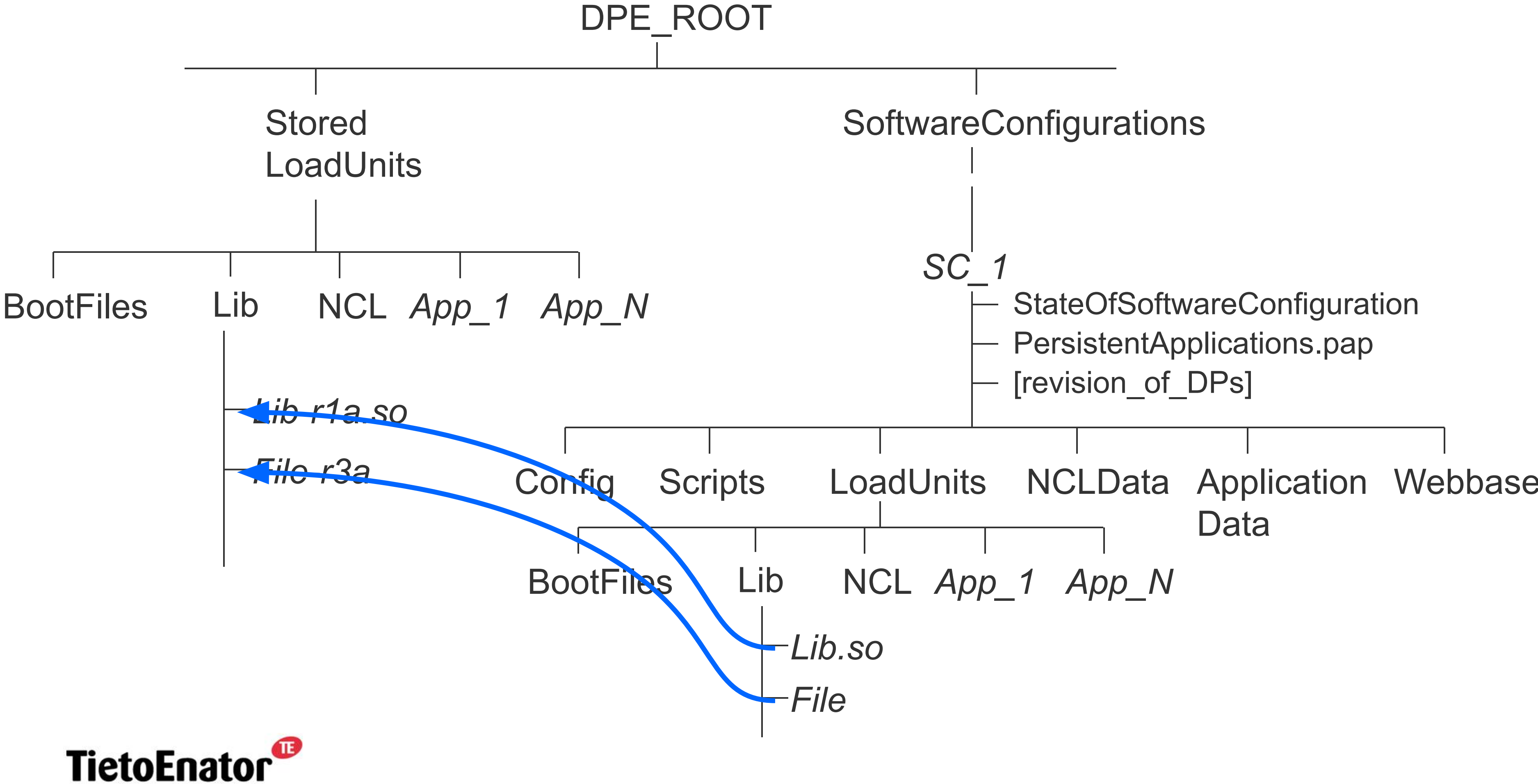
Structure of the file system



TietoEnator
Building the Information Society

Copyright © 2006 TietoEnator Corporation

Software Configuration (SC)



TietoEnator
Building the Information Society

Copyright © 2006 TietoEnator Corporation

Software Configuration (SC), cont’d

DPE_ROOT

Stored LoadUnits

Lib

Lib_r1a.so

File_r3a

SoftwareConfigurations

SC_1

LoadUnits

Lib

Lib.so

File

SC_1_cp

LoadUnits

Lib

Lib.so

File

Copyright © 2006 TietoEnator Corporation

Software Configurations (SCs)

DPE_ROOT

Software Configurations

- NextSoftwareConfiguration
- PermanentSoftwareConfiguration
- LastActivatedSoftwareConfiguration
- LastBootedSoftwareConfiguration
- [PreviousSoftwareConfiguration]

SC_1 *SC_1_cp* *SC_2*

Building the Information Society

Copyright © 2006 TietoEnator Corporation

Distributed Processing Environment (DPE)

10. Introduction to Checkpointing and Activation of a Software Configuration

- Activation of a software configuration
- Check pointing
- An application perspective of upgrade

Copyright © 2006 TietoEnator Corporation

Type of software configuration

- Installed
  - The software configuration was unpacked from a Node Delivery Package (NDP).
- Patched
  - The software configuration was generated by applying a patch (SuperCP) to another software configuration.
- Checkpointed
  - The software configuration was generated by checkpointing another software configuration while it was active.

Building the Information Society

Copyright © 2006 TietoEnator Corporation

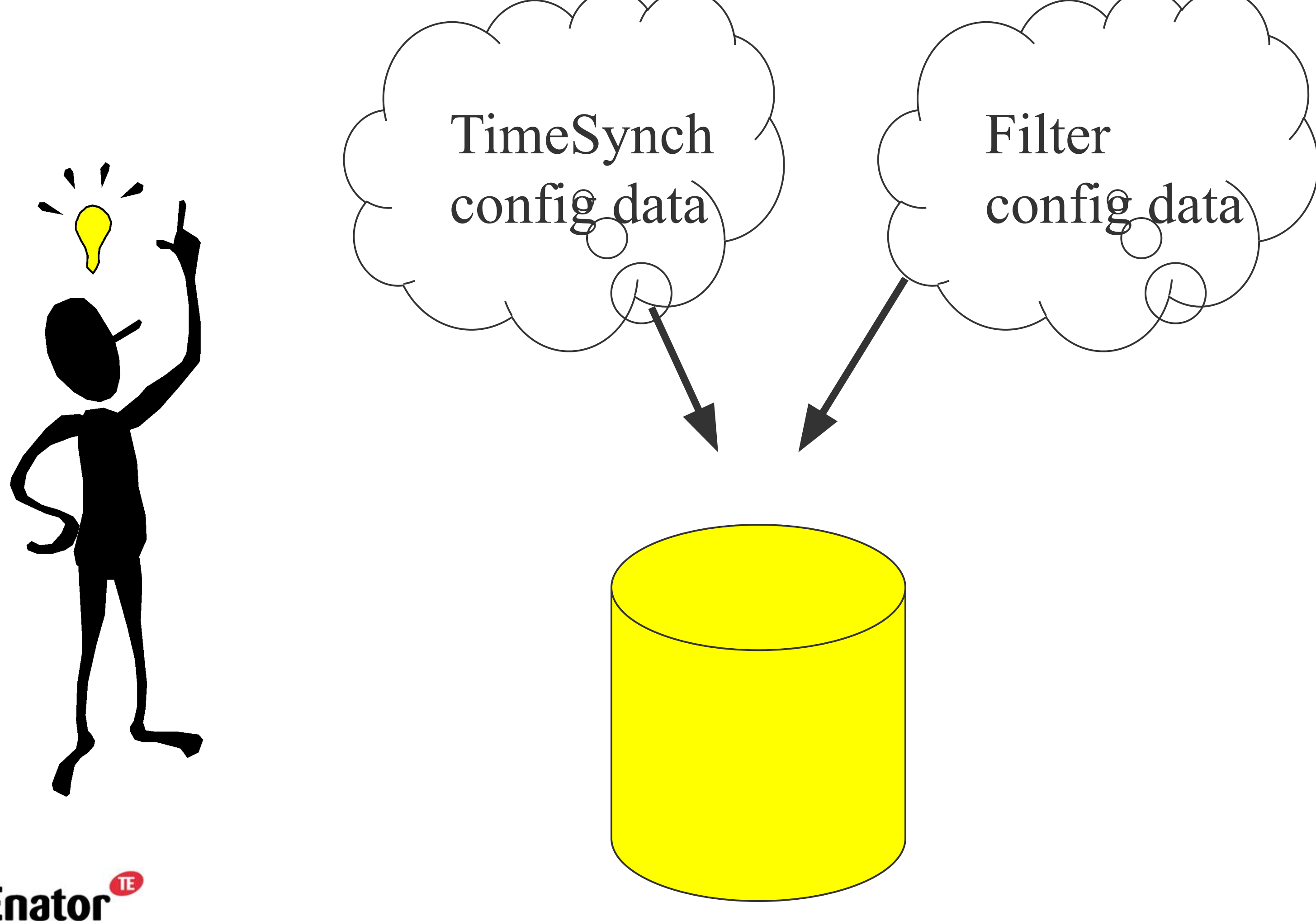
Software Configuration Activation Methods

- RebootNode:
  - The entire node is rebooted, starts up on the new SC.
- RestartDPE:
  - DPE (NCL, agents, all DPE applications), and VxWorks PMs are restarted.
- RestartApplications:
  - VxWorks PMs and DPE applications are restarted (smooth restart).
- RestartPatched:
  - Block instances with patched load units are restarted (smooth restart).
- ManualStart:
  - Only change the current SC, restarts nothing.

Building the Information Society

Copyright © 2006 TietoEnator Corporation

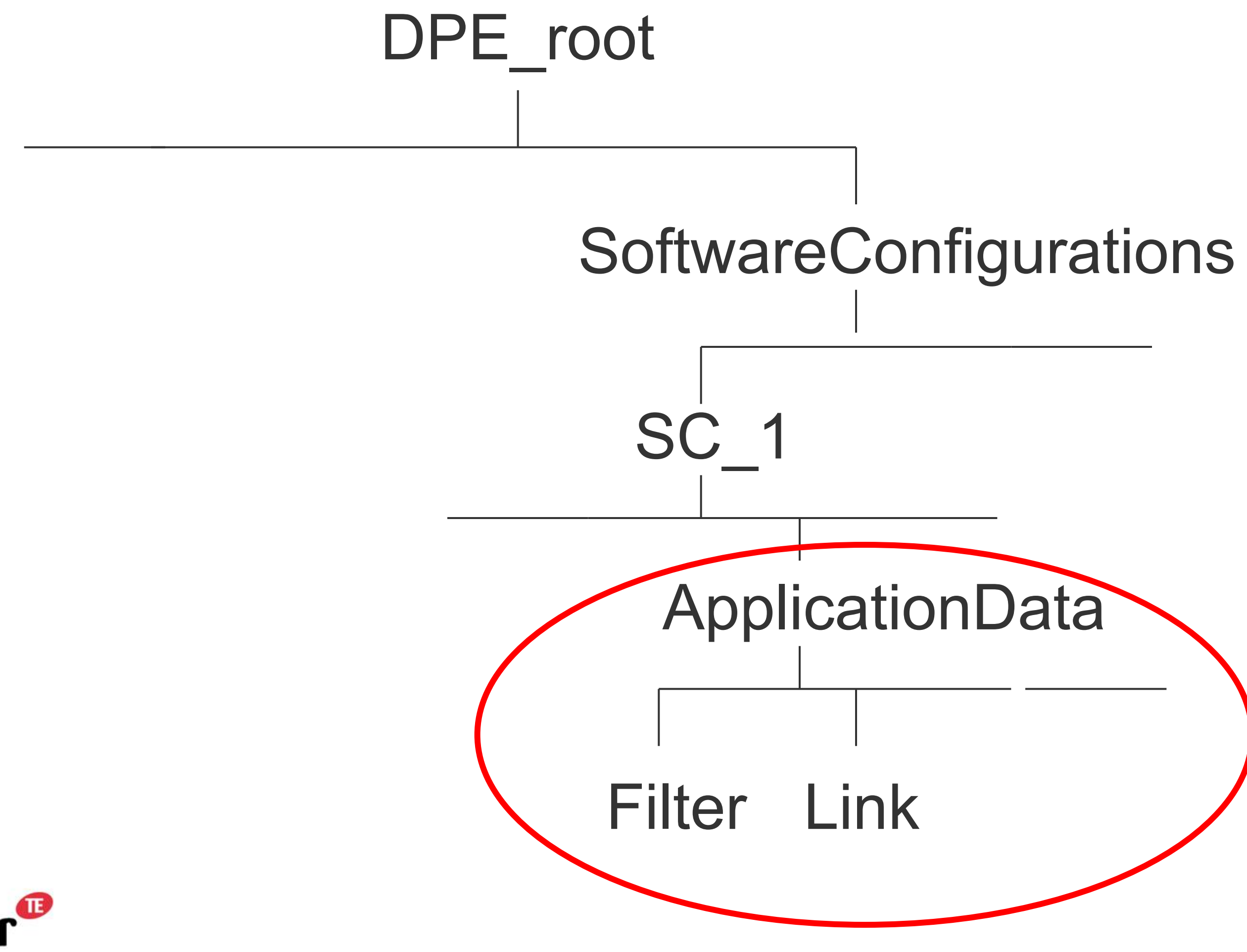
Checkpointing



TietoEnator
Building the Information Society

Copyright © 2006 TietoEnator Corporation

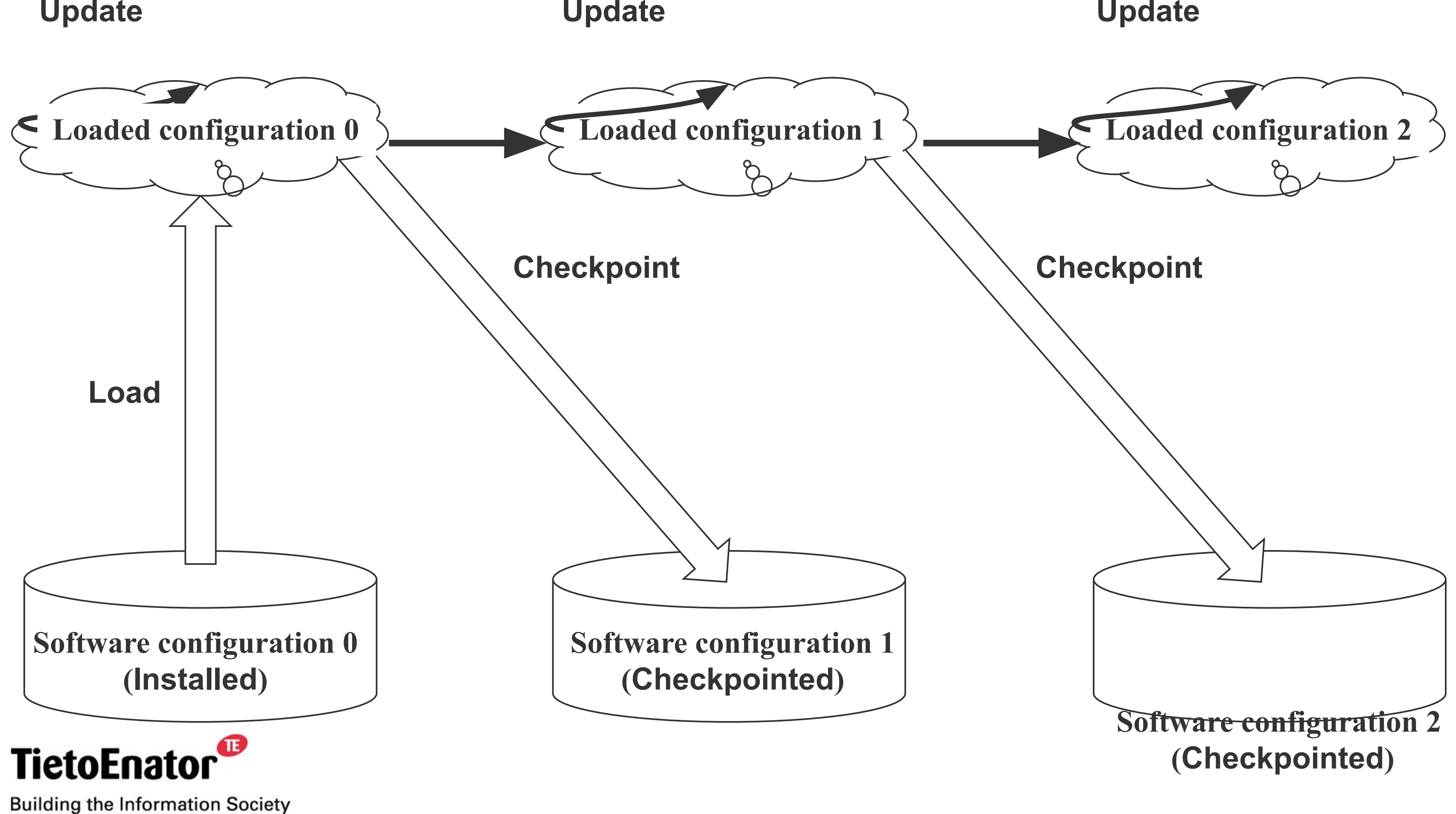
Dedicated place for configuration data



TietoEnator
Building the Information Society

Copyright © 2006 TietoEnator Corporation

The relation between loading, updating and check-pointing configuration data



TietoEnator
Building the Information Society

Copyright © 2006 TietoEnator Corporation

Storing of configuration data = checkpointing

Software Management SLO

DPE

Root Block Instance of `MyApp`

DPE_SubscribeToSR(APPL_SR_CheckpointOk, "SWM_SLO")

DPE_SubscribeToSR(SWCM_SR_NextSC, "RBI_MyApp")

DPE_SetValueOfSR(SWCM_SR_NextSC, Value)

ChangeSRHandler(..)

Store configuration data

DPE_SetValueOfSR(APPL_SR_CheckpointOk, Value)

ChangeSRHandler(..)

MyApp is supposed to store its configuration data to the directory:

<DPE_Root>/`SoftwareConfigurations`/<SC>/`ApplicationData/MyApp`

Building the Information Society

Copyright © 2006 TietoEnator Corporation

An application perspective on upgrade / update

- With activation method RebootNode, RestartDPE, an application will be:
  - stopped with reason `Upgrade`
  - restarted with reason `Upgrade` or `InitialStart`
- The application must properly manage its configuration data.
- For activation methods RestartApplications, RestartPatched, a stopped application will allways be restarted with reason `InitialStart`.

Copyright © 2006 TietoEnator Corporation

Loading of configuration data

Load configuration data

Is the start reason InitialStart?

1

Yes

Use State variable DPE_SR_CurrentSC

Load data from current (or last activated) software configuration

No

2

Is the start reason Upgrade?

Yes

Use State variable DPE_SR_PreviousSC

Load (and convert) data from previous software configuration

No

Failure

Building the Information Society

Copyright © 2006 TietoEnator Corporation

Upgrading

Upgrade

SC_old

SC_new

Copyright © 2006 TietoEnator Corporation

The upgrade action file

- NCL uses this file to determine the revision of an application.
- The following is an example of the content of a valid upgrade action file:

```
# Upgrade action file for Application App.

  This: PA2 .     # The current revision of App is PA2.

  PA1  RestartMe .   # Upgrading from PA1 to PA2 should be
          # done using action "RestartMe"
  PA2  Internal .    # Upgrading from PA2 to PA2 should be
          # done using action "Internal"
```

Copyright © 2006 TietoEnator Corporation

Upgrading from SC1 to SC2: Example 1 (`App1`)

Software configuration 2
SC2 (Installed)

```
#Upgrade actions for App1
This: PA2 .
PA1: RestartMe .
PA2: RestartMe .
```

Activation method:

RestartDPE

Upgrade of `App1`

Stopped with reason Upgrade.

Started on SC2 with reason Upgrade.

Loads configuration data from SC1.

Reports DistributionComplete to NCL when `App1` is properly started.

Copyright © 2006 TietoEnator Corporation

Upgrading from SC1 to SC2: Example 2 (`App1`)

Software configuration 2
SC2 (Patched)

```
#Upgrade actions for App1
This: PA2 .
PA1: RestartMe .
PA2: RestartMe .
```

Activation method:

RestartDPE

Upgrade of `App1`

Stopped with reason Upgrade.

Started on SC2 with reason InitialStart.

Loads configuration data from SC2.

Reports DistributionComplete to NCL when `App1` is properly started.

Building the Information Society

Copyright © 2006 TietoEnator Corporation

Monitoring of the upgrade process

- Timeouts used to monitor the upgrade process:
  - PrepareForStopTimeout
  - StopApplicationsTimeout
  - ShutDownSmoothUpdateableTimeout
  - InitialDistributionTimeout
  - SoftwareUpgradeTimeout
- If any of these timeouts expire, DPE is restarted on the permanent software configuration (fallback).
  - Applications are then started with start reason InitialStart.

Copyright © 2006 TietoEnator Corporation

State registers updated by DPE during upgrade

- “DPE_SR_CurrentSC”
- “DPE_SR_PreviousSC”
- “DPE_SR_TypeOfCurrentSC”
- “DPE_SR_PrepareForStop”
  - An application must acknowledge this SR when it is ready to stop. E.g., when charging data have been saved to disk.

Copyright © 2006 TietoEnator Corporation