Consafe Logistics
Our business, your advantage

Sattmate WCS

Warehouse Control System in Ada
Björn Lundin
2015-06-23 Madrid

SUPPLY CHAIN
EXECUTION

Who are we ?

- Consafe Logistics AB
  - Head Office Lund, Sweden

Sweden	~ 250 employees
Denmark	~ 30 employees
Netherlands	~ 30 employees
Norway	~ 30 employees
Poland	~ 30 employees

- We make and adapt WMS and WCS system
- In total we are about 50 using the Ada based systems, the rest is administrative, or working with a c-based WMS

Who am I ?

- Björn Lundin
- Master of Science in Mechanical Engineering
- Chief Architect Automation
- Technical responsible for our Warehouse Control System, Sattmate WCS
- Employed since 1997

Who do we deliver to?

▪ SKF	WCS	2009
▪ Husqvarna	WCS	2012
▪ The Absolut Company	WMS/WCS	1998
▪ Heineken	WMS/WCS	~2000
▪ ICA, Dagab (Swedish grocery retailer)	WMS/WCS	~2005
▪ COOP, Netto, SuperGros (Danish grocery retailers)	WMS/WCS	~2002
▪ Rema 1000 (Norwegian grocery retailer)	WMS/WCS	~2000
▪ Kesko (Finnish grocery retailer)	WMS/WCS	~2005
▪ SCA (paper industry)	WMS/WCS	~1998
▪ Iggesund (paper mill)	WCS	2011
▪ SNA/Snap On/Sandvik (tools manufacturer)	WMS/WCS	~1995
▪ Astra Zeneca (pharmaceutical manufacturer)	WCS	~2010
▪ Canon	WMS/WCS	~1995
▪ Ahlsell, BA (Building contractor retailers)	WCS	2012

What does a WCS do ?

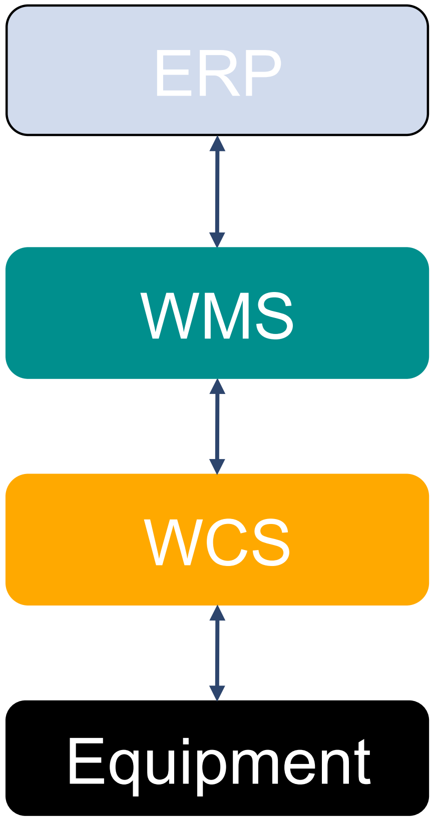
WMS versus WCS: What is the difference?

WMS – **What, when and where**

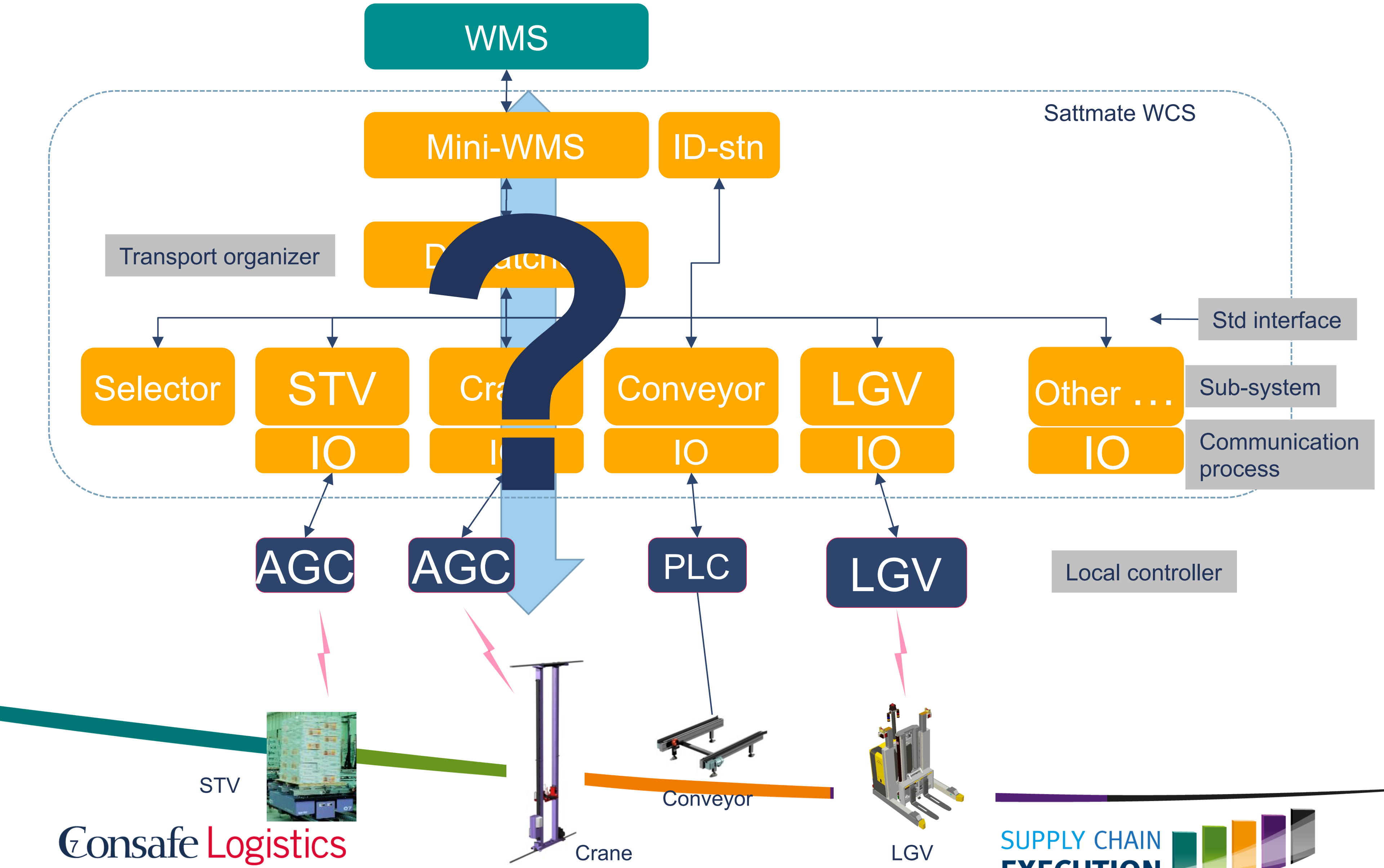
- Handles orders, articles, balances
- Selects pallets for picking, replenishment and full pallet output.
- Performs picking

WCS - **How**

- Handles transports wrt automation equipment
- Performs and coordinates transports through the automation system



WCS – architecture principles

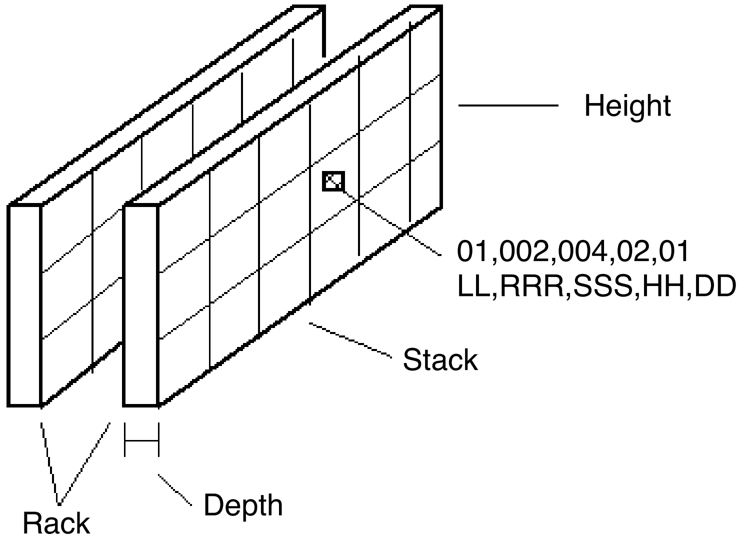


Consafe Logistics
Our business, your advantage

SUPPLY CHAIN EXECUTION

Assignments and Locations – like Files and Processes to unix

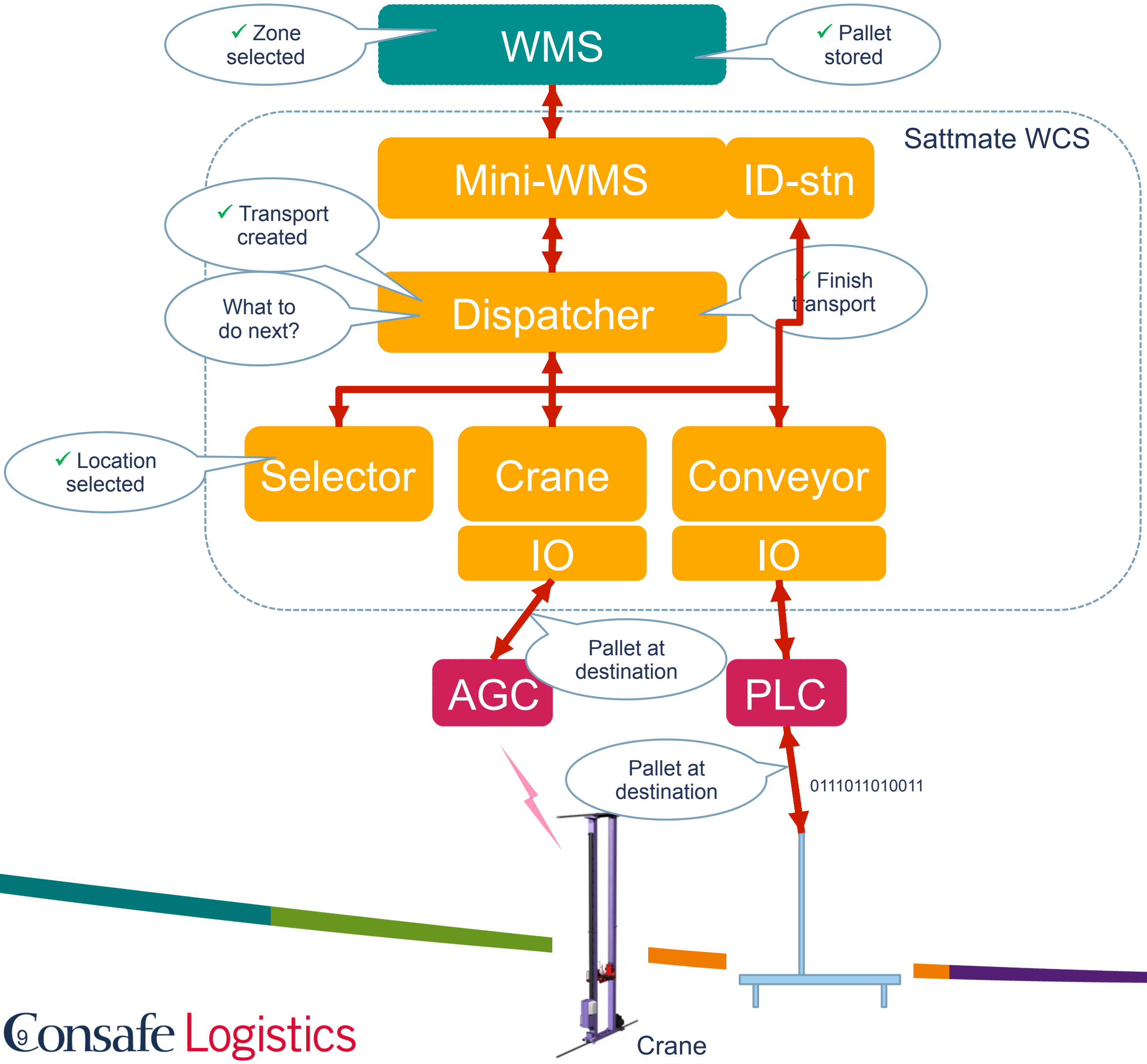
- Location – a coordinate within the warehouse store/rack/stack/level/depth
- Assignment – Movement of a pallet from location A to B, via other locations
- Assignment sequences – definition of legal moves.
  XML-based file describing how a pallet moves from area A to Area B, and what sub systems are involved



Consafe Logistics
Our business, your advantage

SUPPLY CHAIN EXECUTION

Pallet input example – WCS selects location

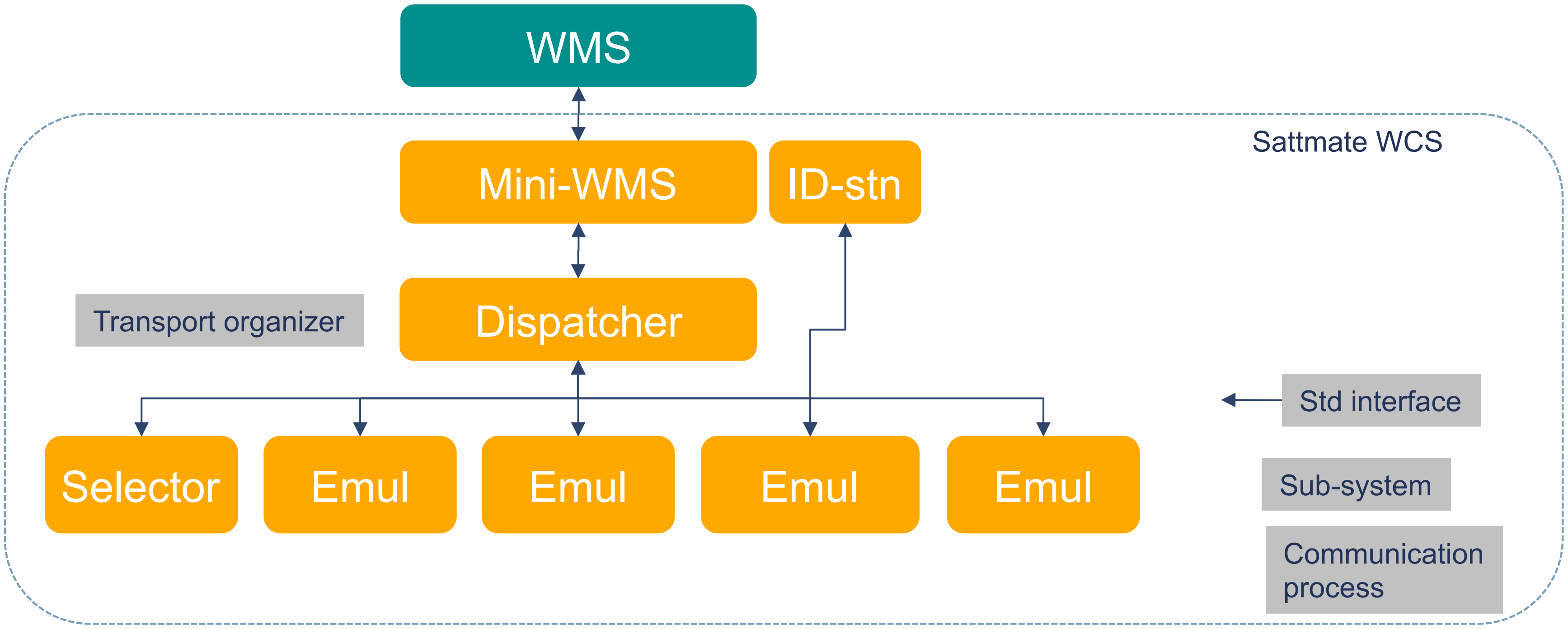


Example:
A typical system with a crane high-bay and a conveyor system in front of the high-bay and location selection in WCS.

Consafe Logistics
Our business, your advantage

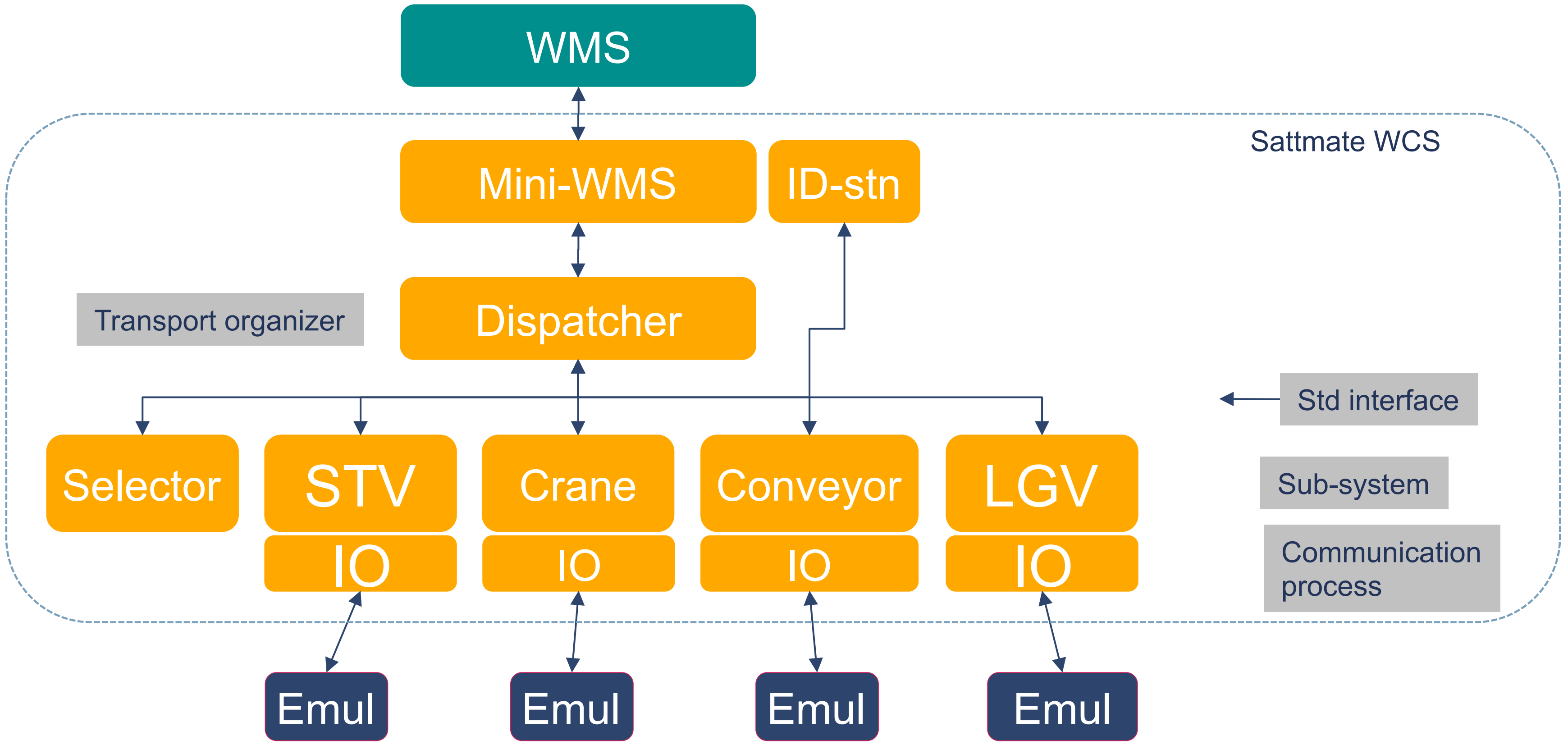
SUPPLY CHAIN EXECUTION

WCS – testing and emulation



Simple emulation testing
Used to test assignment sequence logic

WCS – testing and emulation



Full emulation testing
Used for full system testing

Consafe Logistics
Our business, your advantage

SUPPLY CHAIN EXECUTION

Std Interface from other slides – what is that?

```
package Process_Io is
  type      Message_Type  is private;
  procedure Receive   (Message   : out Message_Type;
                       Time_Out : in  Duration := 0.0);

  function  Identity (Message   : Message_Type) return Identity_Type;

  generic
    Identity          : Identity_Type;
    type Data_Type is private;
    Data_Descriptor : Data_Descriptor_Type;
  package Generic_Io is
    procedure Send (Receiver : in  Process_Type;
                    Data     : in  Data_Type);
    function Unpack(Message  : Message_Type) return Data_Type;
    function  Pack (Data     : Data_Type) return Message_Type;
  end Generic_Io;
end Process_Io;
```

```
package Wcs_Messages is
   Wcs_Delete_Request_Asm_Message :
           constant Wcs_Message_Id_Type  := 28004;
   type Wcs_Delete_Request_Asm_Record is record
      Bldid          : Integer_4              := 0;
      Basmid         : Integer_4              := 0;
   end record;

   package Wcs_Delete_Request_Asm_Package is new
       Process_Io.Generic_Io
          (Identity        => Wcs_Delete_Request_Asm_Message,
           Data_Type       => Wcs_Delete_Request_Asm_Record,
           Data_Descriptor => Integer_4_Type &
                              Integer_4_Type);
   procedure Send (Receiver  : in  Process_Type;
                   Data      : in  Data_Type);
   function Unpack(Message   : Process_Io.Message_Type)
         return Wcs_Delete_Request_Asm_Record
             renames Wcs_Delete_Request_Asm_Package.Unpack;
end Wcs_Messages;
```

To send a message

```
declare
  Delete_Record : Wcs_Messages.
       Wcs_Delete_Request_Asm_Record := (
           Bldid   => 123_456,
           Basmid  => 456_789);
  Receiver : Process_Io.Process_Type := (
          ("WCS_BOOKER"),("localhost"));
begin
  Wcs_Messages.Send(Receiver, Delete_Record);
end;
```

```
Sattmate_Sql_Session.Open;
Wcs_Process_Config.Initiate;
loop -- hang here until telegram received
   Process_Io.Receive(Telegram);
   case Process_Io.Identity (Telegram) is
      when Wcs_Delete_Request_Asm_Message =>
        -- unpack and treat msg
          Wcs_Process_Services.Handle_Message(
              Wcs_Messages.Unpack(Telegram));
      when Core_Messages.Exit_Signal => exit;
      when others => null;
   end case;
end loop;
Sattmate_Sql_Session.Close;
```

Ada libs ? Yes - in some sense

```
source
├───agc
│   ├───global
│   └───local
│       ├───admin
│       ├───emul
│       ├───io
│       └───test
├───autostore
│   ├───global
│   └───local
│       ├───admin
│       ├───emul
│       ├───io
│       └───test
│
```

- All files under a **local** directory can see the corresponding **local** files
- All files can see all files under a **global** directory
- A sub system exports **global** definitions via the **global** hiearchy
- Orange files sees all orange files **and** all black files
- Red files sees all red files **and** all black files
- Black files see **only** black files
- Orange files do **not** see red files and vice versa

Consafe Logistics
Our business, your advantage

SUPPLY CHAIN
EXECUTION

Repository in XML

- A repository is a data source where global definitions are defined
  - Database Table definitions
  - Database View definitions
  - Client-server messages
  - Terms definitions and translations
  - Coded values (enumerations) definitions, translations and integer to use for each value
- Tool to generate SQL DDL statements for supported databases
- Tool to generate Ada code for DML statements in database
- Tool to generate Ada and C# code (stubs) for client-server messages

Autogenerated packages – DB Access 1

```
package Table_Bload is
type Data_Type is tagged record
  Bldid :  Integer_4  := 0 ; -- Primary Key
  Bldsta : Integer_4  := 0 ; --
  Bldtyp : Integer_4  := 0 ; --
  Bwmsid : String (1..35) := (others => ' ') ; -- non unique
  Xlocnam :String (1..20) := (others => ' ');--
  Bcrets : Time_Type  := Time_Type_First ; --
  Bcartyp :Integer_4  := 0 ; --
  Bcawei : Float_8  := 0.0 ; --
end record;
```

Autogenerated packages – DB Access 2

```
procedure Read(Data        : in out Table_Bload.Data_Type;
               End_Of_Set  : in out Boolean);

function  Get(Bldid : Integer_4) return
                    Table_Bload.Data_Type;

procedure Delete(Data : in     Table_Bload.Data_Type);
procedure Update(Data : in out Table_Bload.Data_Type);
procedure Insert(Data : in out Table_Bload.Data_Type);
procedure Read_One_Bwmsid(Data : in out
                         Table_Bload.Data_Type;
                      End_Of_Set : in out Boolean);
```

Autogenerated packages – DB Access 3

```
function To_String(Data : in Table_Bload.Data_Type) return String;
function To_Xml(Data    : in Table_Bload.Data_Type) return String;
function To_Map (Data   :    Table_Bload.Data_Type'class) return
some_map;

package Bload_List_Pack is new
Simple_List_Class(Table_Bload.Data_Type);
procedure Read_List(Stm  : in Sql.Statement_Type; List : in out
Bload_List_Pack.List_Type);

package Bload_List_Pack2 is new
Ada.Containers.Doubly_Linked_Lists(Table_Bload.Data_Type);
procedure Read_List(Stm  : in Sql.Statement_Type; List : in out
Bload_List_Pack2.List);
```

Autogenerated packages – DB Access in use - 1

```
declare
  Bload_Data   : Table_Bload.Data_Type;
  End_Of_Set   : Boolean := False;
  Transaction : Sql.Transaction_Type;
  use Wcs_Types ;
begin
  Transaction.Start;
  Bload_Data.Bldid := 123_456; -- primary key
  Bload_Data.Read(End_Of_Set);
  if not End_Of_Set then
     Bload_Data.Bldsta := Wcs_Load_Status(Reserved);
     Bpload_Data.Update;
  end if;
  Transaction.Commit;
end;
```

Features of Ada we rely on - Enumerations

- Often defined via xml – used by GUI too

```
type wcs_load_status_Type is (
 created,     waiting_for_wmsid,
 wmsid_set,   reserved,
 stored,      out_of_store,
 shipped);

 for wcs_load_status_Type'Size use Integer_4'Size;
 for wcs_load_status_Type use (
   created   => 1,     waiting_for_wmsid => 2,
   wmsid_set => 3,     reserved          => 4,
   stored    => 5,     out_of_store      => 6,
   shipped   => 7);

 function wcs_load_status (X: wcs_load_status_Type) return Integer_4;
 function wcs_load_status (X: Integer_4) return wcs_load_status_Type;
```

Autogenerated packages – DB Access in use - 2

```
declare
  Bload_Data   : Table_Bload.Data_Type;
  Bload_List   : Table_Bload_Bload_List_Pack.List;
  Transaction  : Sql.Transaction_Type;
  Statement    : Sql.Statement_Type ;
  use Wcs_Types ;
begin
  Transaction.Start;
  Statement.Prepare("select * from BLOAD where BLDSTA = :STATUS");
  Statement.Set("STATUS",Wcs_Load_Status(Stored));
  Table_Bload.Read_List(Statement, Bload_List);
  for Load of Bload_List loop
    Load.Delete;
  end loop;
  Transaction.Commit;
end;
```

Autogenerated packages - Service call packages

- GUI in C#
- Calls services via socket – message is in xml
- C# struct/class for calling service is auto-generated
- On Ada side, conversion from xml to Ada record is auto-generated
- Ada side performs action stated in message and replies on socket
- Ada code is converted to xml – auto-generated
- C# receives the reply and converts it to a C# struct/class

- Message format on socket is always xml
- Each call on the Ada side is a separate procedure

Features of Ada we rely on - Record layouts

As we are leaving the really low levels, this becomes less important, but some binary I/O protocols still have definitions like this

Note the commented mirrored version for PPC

```
  subtype SIGNAL_TYPE is INTEGER range 8#600#..8#777#;
  type SIGNAL_STATUS_TYPE is  (OFF, ON);
  for SIGNAL_STATUS_TYPE use  (OFF => 0, ON => 1);
  for SIGNAL_STATUS_TYPE'SIZE use 1;

  subtype SIX_BITS is INTEGER range 0..2#111111#;

  type ENQ_246_TYPE is record
      SIGNAL : SIGNAL_TYPE;
      FILL   : SIX_BITS;
      STATUS : SIGNAL_STATUS_TYPE;
  end record;

  for ENQ_246_TYPE'alignment use 2;
  for ENQ_246_TYPE use record -- V5.2 VAX & Intel
      SIGNAL at 0 range 0..8;
      FILL   at 0 range 9..14;
      STATUS at 0 range 15..15;
  end record;

--    for ENQ_246_TYPE use record -- V5.2 RS/6000
--        STATUS at 0 range 0..0;
--        FILL   at 0 range 1..6;
--        SIGNAL at 0 range 7..15;
--      end record;

  for ENQ_246_TYPE'SIZE use 2*8;
```

Features of Ada we rely on

- Packages – around 1500 whereof about 350 autogenerated
- Generics – lists/stacks/sorters
- Separates
- Backwards compatibility
  Long lived systems may be created with Ada83 but maintained with Ada95/Ada05/Ada12 compilers.
  Verdix/Alsys/Object Ada/Gnat

- File header excerpt from a common types spec

```
--     VERSION                3.0
--     AUTHOR          Henrik Dannberg     3-DEC-1989
…
--9.4.1-8146           Björn Lundin        23-sep-2005
```