NATO OTAN

NURC- *Partnering for Maritime Innovation*

NURC

Hardware-in-the-Loop Testing

Stephanie Kemna, Arjan Vermeij, Michael J. Hamilton

{kemna,vermeij,hamilton}@nurc.nato.int

MOOS-DAWG 2011
MIT, Cambridge, MA, USA

NATO Undersea Research Centre: Underwater acoustics and ASW

1959: SACLANT

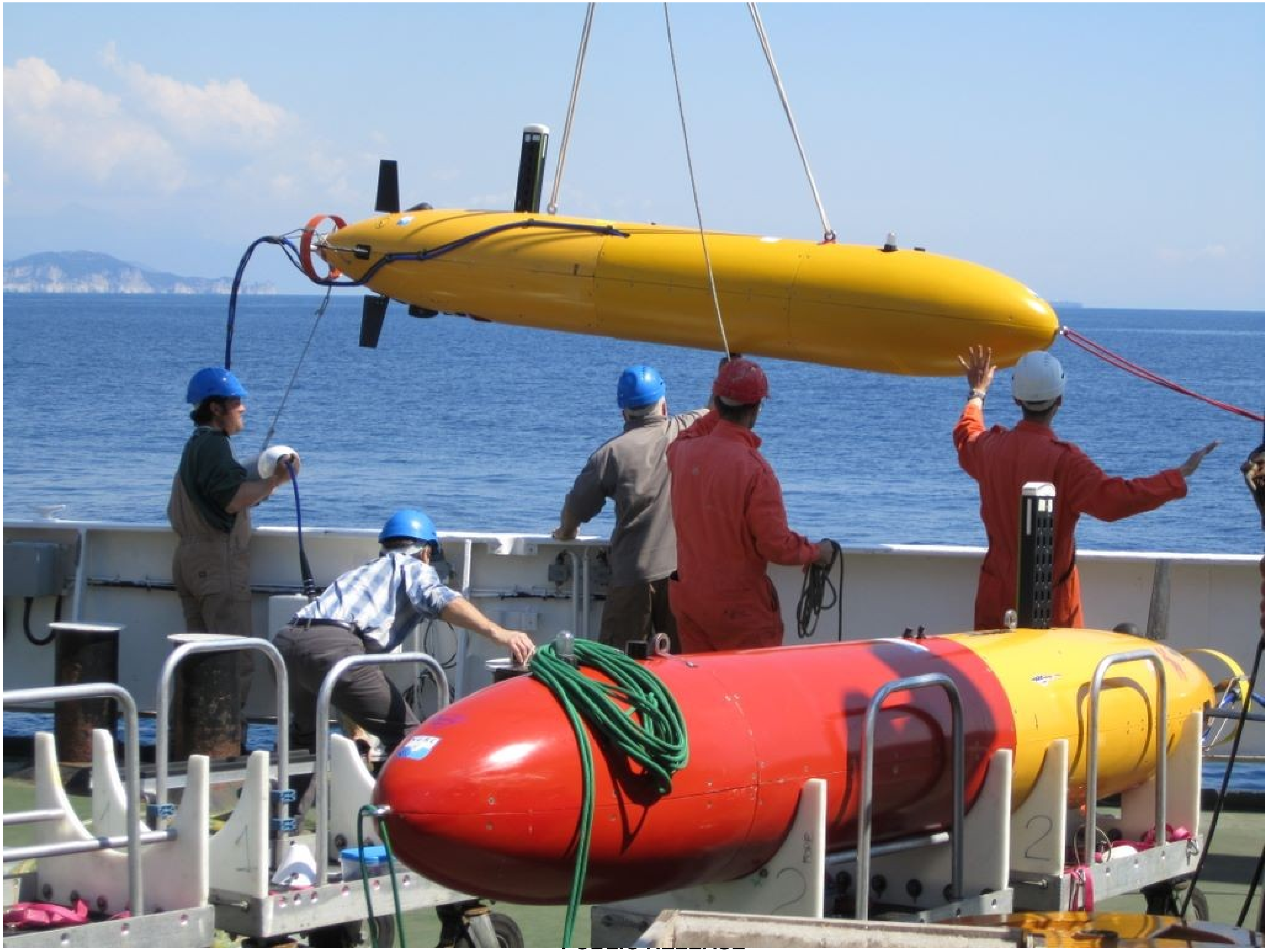
NATO maritime and transformational requirements

Seagoing research: Maritime innovation in NATO Nations

- Cooperative AntiSubmarine Warfare
- Autonomous Naval Mine Countermeasures
- Ship and Port Protection
- Marine Mammal Risk Mitigation
- Maritime Situational Awareness
- Environmental Knowledge & Operational Effectiveness

PUBLIC RELEASE

OEX AUVs: Groucho & Harpo



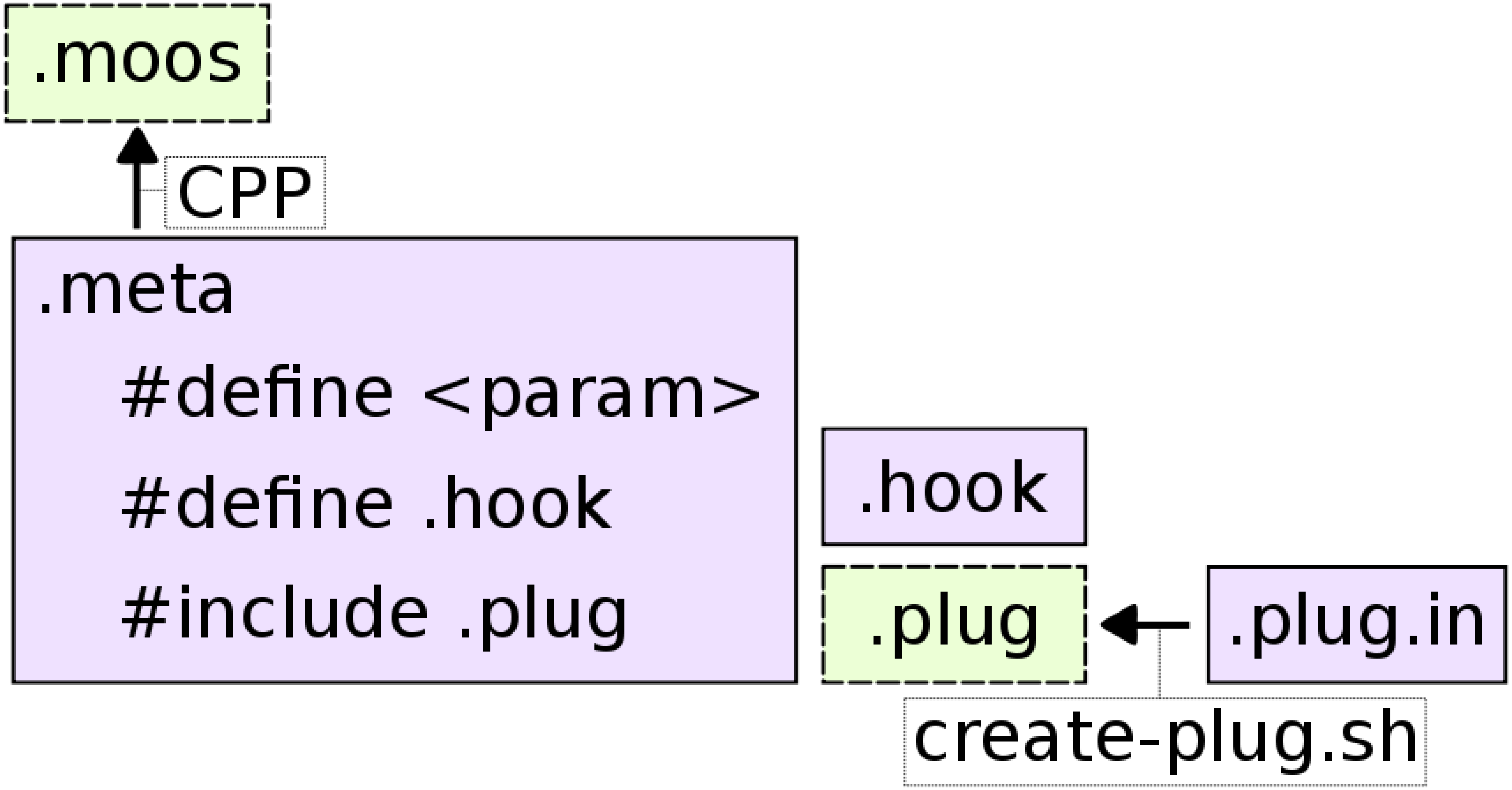
Hardware-in-the-Loop (HIL)

- Verify computational load
- Benchmark performance
- Reduce errors at sea
  - → backseat in runtime and simulation as similar as possible
  - → avoid missing packages
  - → check system configuration

Overview

- NURC's mission file generation
- Previous simulations – single PC
- HIL requirements, set-up and lessons learned
- Conclusions / Summary

Mission file generation at NURC



.moos

CPP

.meta

#define <param>

#define .hook

#include .plug

.hook

.plug

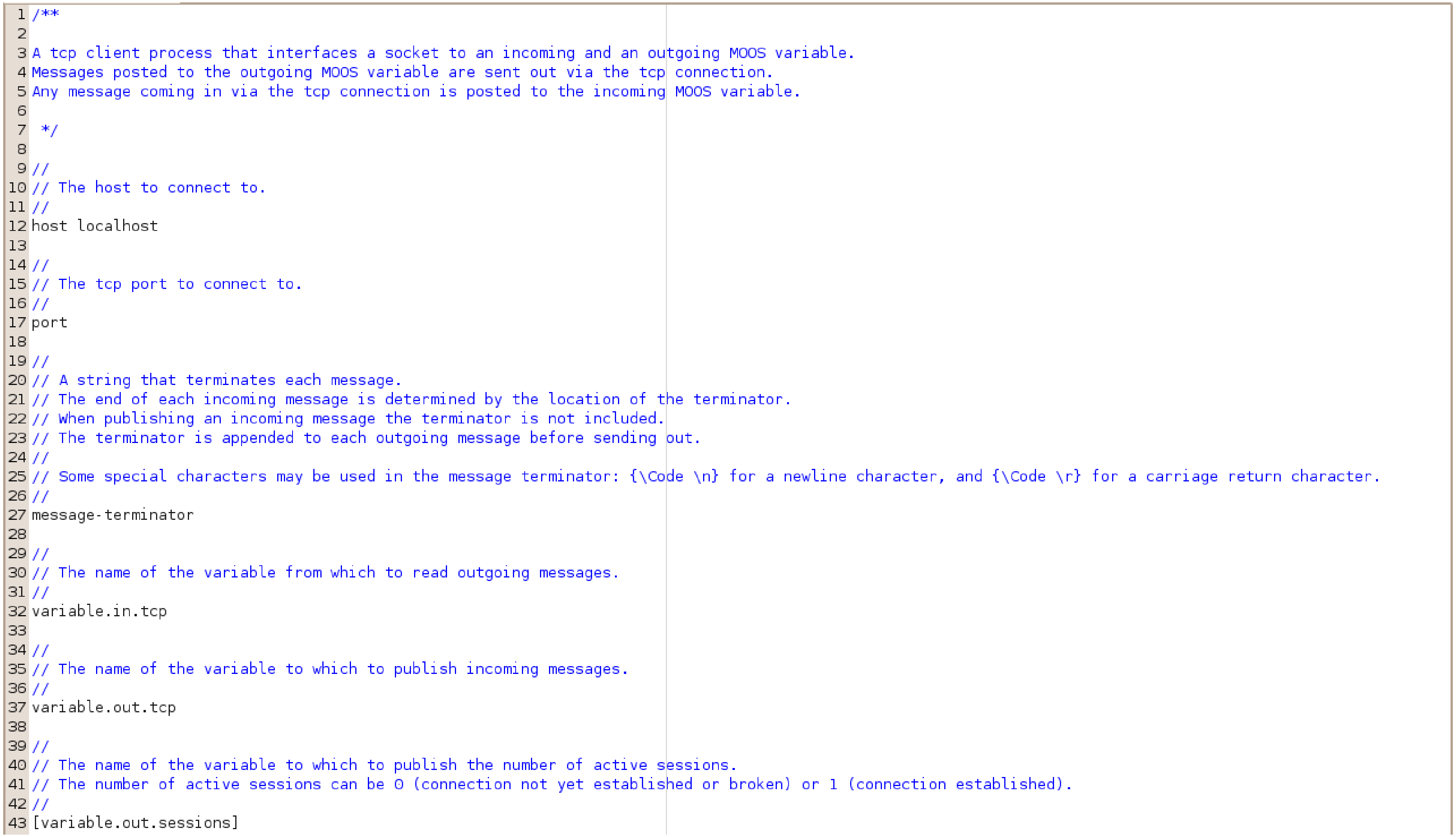
.plug.in

create-plug.sh

.plug.in

- standard parameters added automatically: AppName, AppTick, CommsTick, verbose, velvet line with NewConsole and WriteToFile.
- other parameters (one per line):
  - default values: after parameter name
  - optional parameters in square brackets

.plug.in example



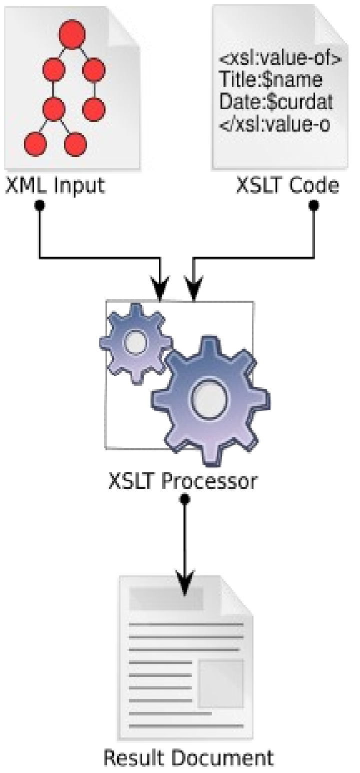
```
/**

A tcp client process that interfaces a socket to an incoming and an outgoing MOOS variable.
Messages posted to the outgoing MOOS variable are sent out via the tcp connection.
Any message coming in via the tcp connection is posted to the incoming MOOS variable.

 */

//
// The host to connect to.
//
host localhost

//
// The tcp port to connect to.
//
port

//
// A string that terminates each message.
// The end of each incoming message is determined by the location of the terminator.
// When publishing an incoming message the terminator is not included.
// The terminator is appended to each outgoing message before sending out.
//
// Some special characters may be used in the message terminator: {\Code \n} for a newline character, and {\Code \r} for a carriage return character.
//
message-terminator

//
// The name of the variable from which to read outgoing messages.
//
variable.in.tcp

//
// The name of the variable to which to publish incoming messages.
//
variable.out.tcp

//
// The name of the variable to which to publish the number of active sessions.
// The number of active sessions can be 0 (connection not yet established or broken) or 1 (connection established).
//
[variable.out.sessions]
```

create-plug.sh

```
$ create-plug.sh <pCamelCase>.plug.in
```

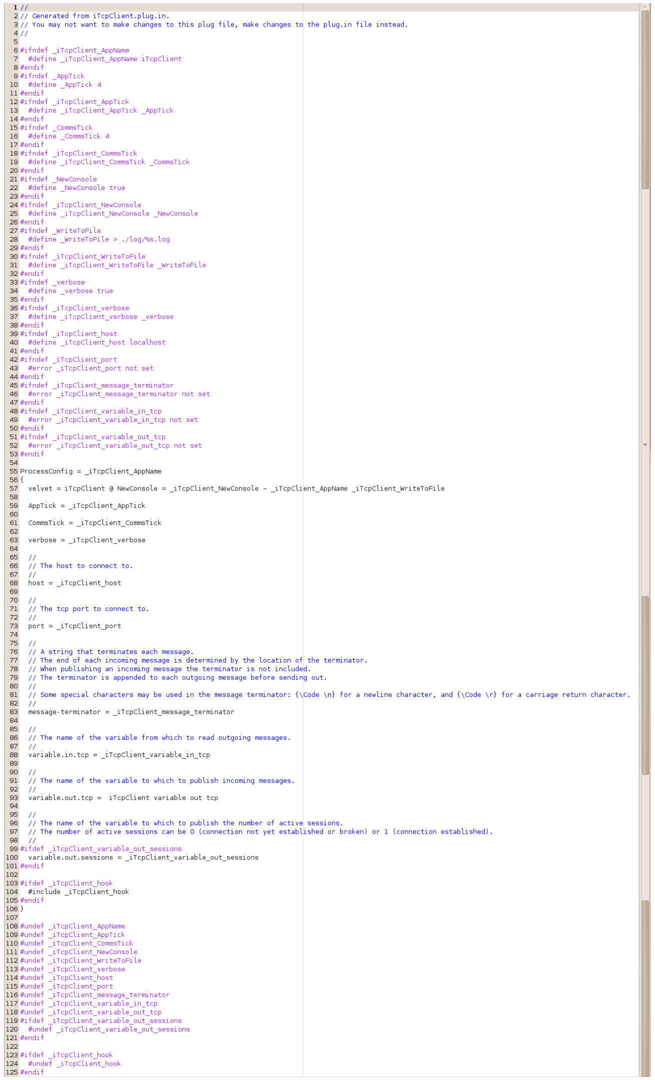
- takes .plug.in
- internally changes .plug.in to .xml (Perl)
- then it uses
  - .xsl style-sheet (main conversion rules)
  - xsltproc (command line XSLT processor)
  - sed

  to convert the .xml into a .plug



http://en.wikipedia.org/wiki/XSLT

generated .plug

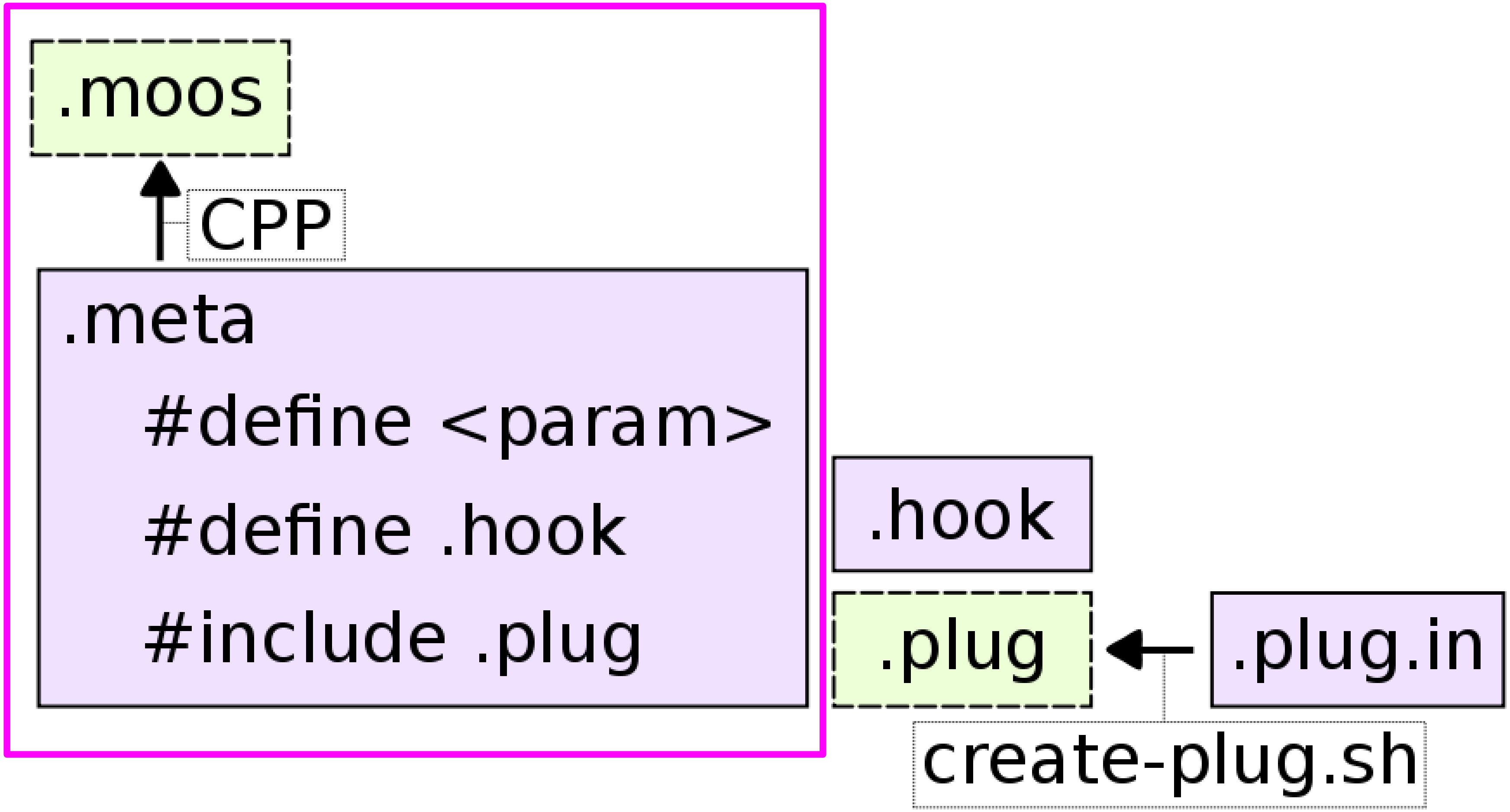


```
//
// Generated from iTcpClient.plug.in.
// You may not want to make changes to this plug file, make changes to the plug.in file instead.
//

#ifndef _iTcpClient_AppName
  #define _iTcpClient_AppName iTcpClient
#endif
#ifndef _AppTick
  #define _AppTick 4
#endif
#ifndef _iTcpClient_AppTick
  #define _iTcpClient_AppTick _AppTick
#endif
#ifndef _CommsTick
  #define _CommsTick 4
#endif
#ifndef _iTcpClient_CommsTick
  #define _iTcpClient_CommsTick _CommsTick
#endif
#ifndef _NewConsole
  #define _NewConsole true
#endif
#ifndef _iTcpClient_NewConsole
  #define _iTcpClient_NewConsole _NewConsole
#endif
#ifndef _WriteToFile
  #define _WriteToFile > ./log/%s.log
#endif
#ifndef _iTcpClient_WriteToFile
  #define _iTcpClient_WriteToFile _WriteToFile
#endif
#ifndef _verbose
  #define _verbose true
#endif
#ifndef _iTcpClient_verbose
  #define _iTcpClient_verbose _verbose
#endif
#ifndef _iTcpClient_host
  #define _iTcpClient_host localhost
#endif
#ifndef _iTcpClient_port
  #error _iTcpClient_port not set
#endif
#ifndef _iTcpClient_message_terminator
  #error _iTcpClient_message_terminator not set
#endif
#ifndef _iTcpClient_variable_in_tcp
  #error _iTcpClient_variable_in_tcp not set
#endif
#ifndef _iTcpClient_variable_out_tcp
  #error _iTcpClient_variable_out_tcp not set
#endif

ProcessConfig = _iTcpClient_AppName
{
  velvet = iTcpClient @ NewConsole = _iTcpClient_NewConsole ~ _iTcpClient_AppName _iTcpClient_WriteToFile

  AppTick = _iTcpClient_AppTick

  CommsTick = _iTcpClient_CommsTick

  verbose = _iTcpClient_verbose

  //
  // The host to connect to.
  //
  host = _iTcpClient_host

  //
  // The tcp port to connect to.
  //
  port = _iTcpClient_port

  //
  // A string that terminates each message.
  // The end of each incoming message is determined by the location of the terminator.
  // When publishing an incoming message the terminator is not included.
  // The terminator is appended to each outgoing message before sending out.
  //
  // Some special characters may be used in the message terminator: {\Code \n} for a newline character, and {\Code \r} for a carriage return character.
  //
  message-terminator = _iTcpClient_message_terminator

  //
  // The name of the variable from which to read outgoing messages.
  //
  variable.in.tcp = _iTcpClient_variable_in_tcp

  //
  // The name of the variable to which to publish incoming messages.
  //
  variable.out.tcp =  iTcpClient variable out tcp

  //
  // The name of the variable to which to publish the number of active sessions.
  // The number of active sessions can be 0 (connection not yet established or broken) or 1 (connection established).
  //
#ifdef _iTcpClient_variable_out_sessions
  variable.out.sessions = _iTcpClient_variable_out_sessions
#endif

#ifdef _iTcpClient_hook
  #include _iTcpClient_hook
#endif
}

#undef _iTcpClient_AppName
#undef _iTcpClient_AppTick
#undef _iTcpClient_CommsTick
#undef _iTcpClient_NewConsole
#undef _iTcpClient_WriteToFile
#undef _iTcpClient_verbose
#undef _iTcpClient_host
#undef _iTcpClient_port
#undef _iTcpClient_message_terminator
#undef _iTcpClient_variable_in_tcp
#undef _iTcpClient_variable_out_tcp
#ifdef _iTcpClient_variable_out_sessions
  #undef _iTcpClient_variable_out_sessions
#endif

#ifdef _iTcpClient_hook
  #undef _iTcpClient_hook
#endif
```

Advantages to generating .plug from .plug.in

- reduce typing/human errors
- reduce .plug development time
- be sure that every parameter is definable
- simplify maintenance
- allows for generating documentation from .plug.in

Mission file generation at NURC



From .meta to .moos

- Makefile → CPP
  - using #define, #ifdef, #include
  - usage of INCLUDE_DIRS
    (easy to include mission specific or simulation/runtime specific paths)
- Make aborts & generates error if parameter values are missing

Overview

- NURC's mission file generation
- Previous simulations – single PC
- HIL requirements, set-up and lessons learned
- Conclusions / Summary

Previous simulations: single PC

- One .moos file for simulation, generated from .meta
- For each asset, all processes run within the same MOOS community
- No proper testing of incoming connections

HIL requirements

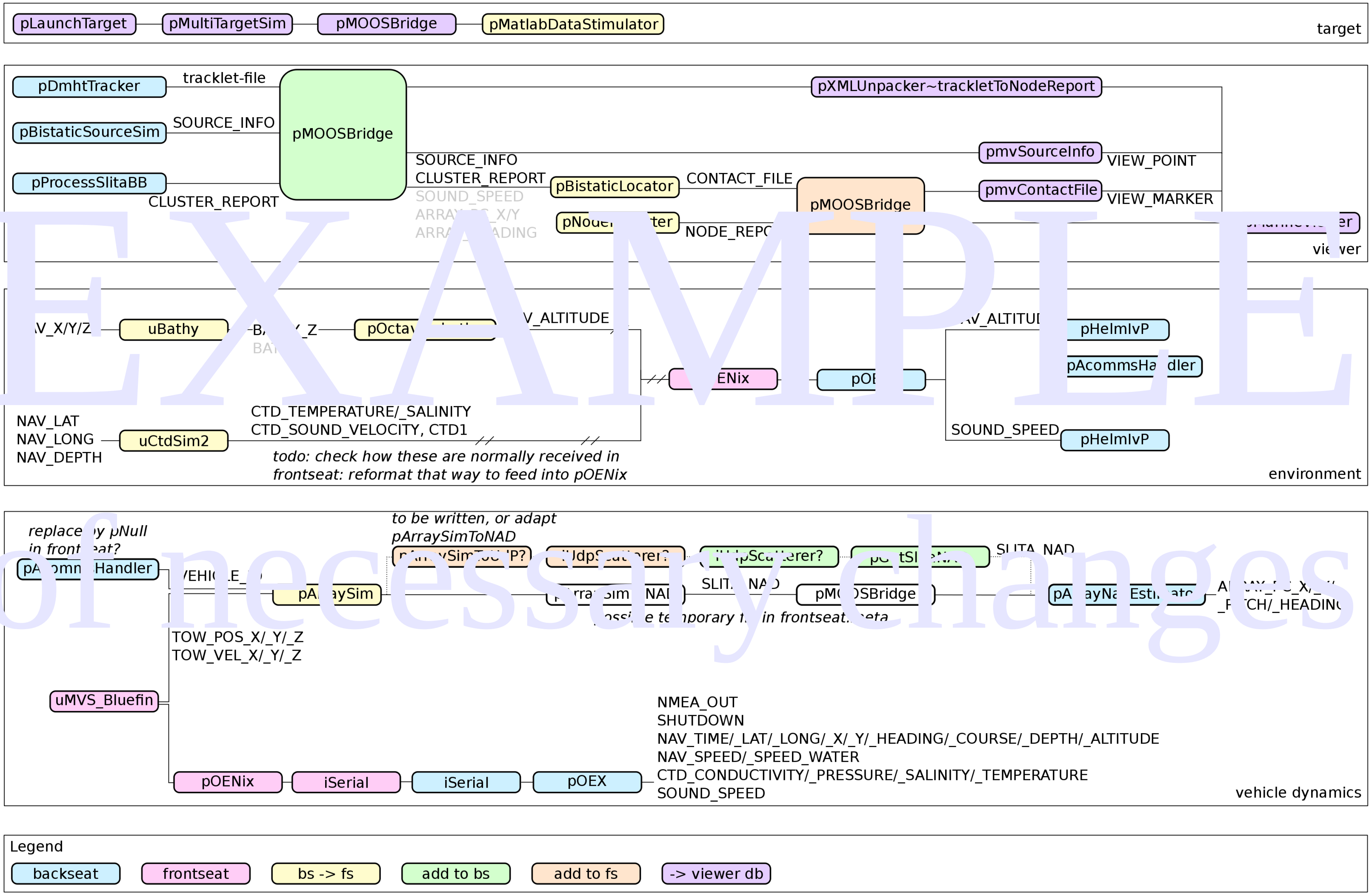
- backseat .meta file same between simulation, HIL simulation and runtime
- Test incoming connections (serial, UDP, TCP) in simulation as in runtime
- all simulator processes should be in a different .meta file, run on a separate computer if testing HIL

PUBLIC RELEASE

HIL – General Conventions

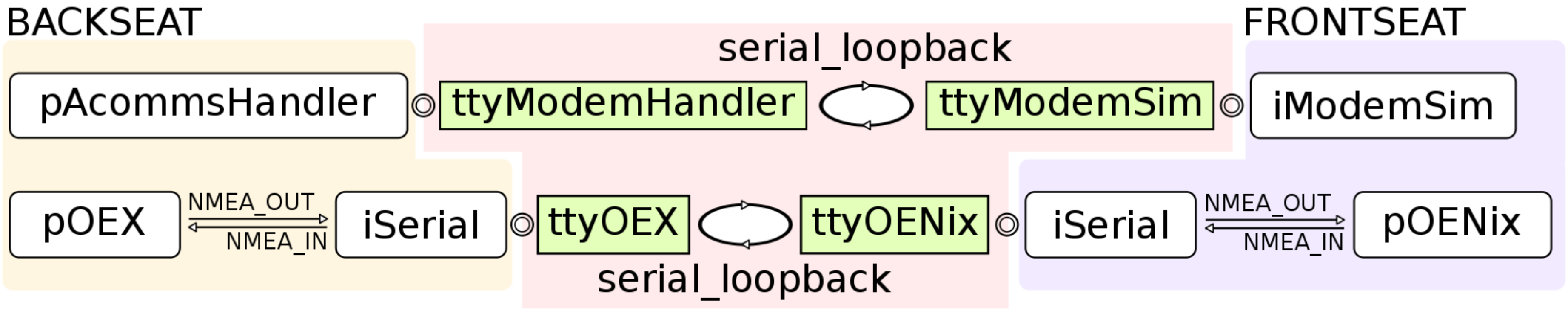
- backseat .meta file: only processes that are run in runtime (at sea)
- frontseat .meta file: processes that simulate the vehicle's frontseat, the modem, the normally acquired environmental data, etc.
- viewer .meta file: visualization and shared objects (equal for all simulated vehicles)

frontseat, backseat & viewer processes, and their connections (draft)



HIL – dealing with serial_loopback

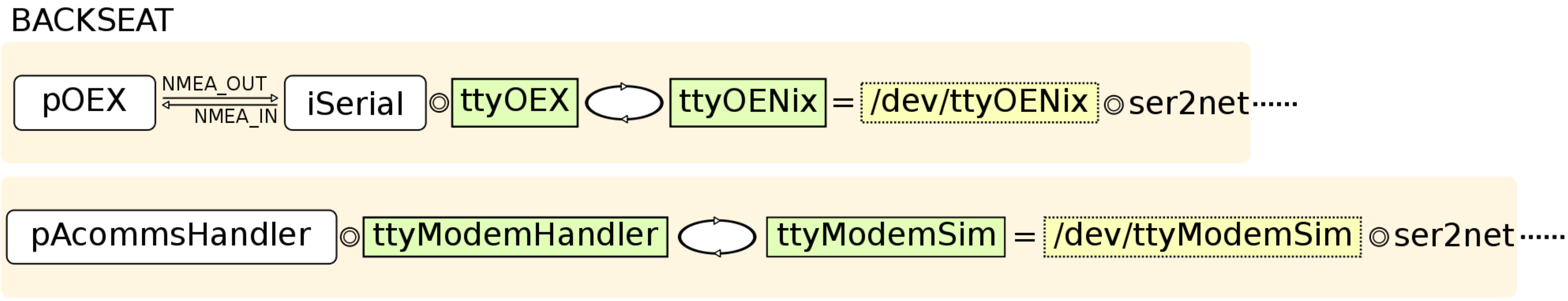
- The difficult cases: serial_loopback
  - pAcommsHandler
  - pOEX (vehicle interface)
- If split on 1 pc:



Serial port to network: ser2net

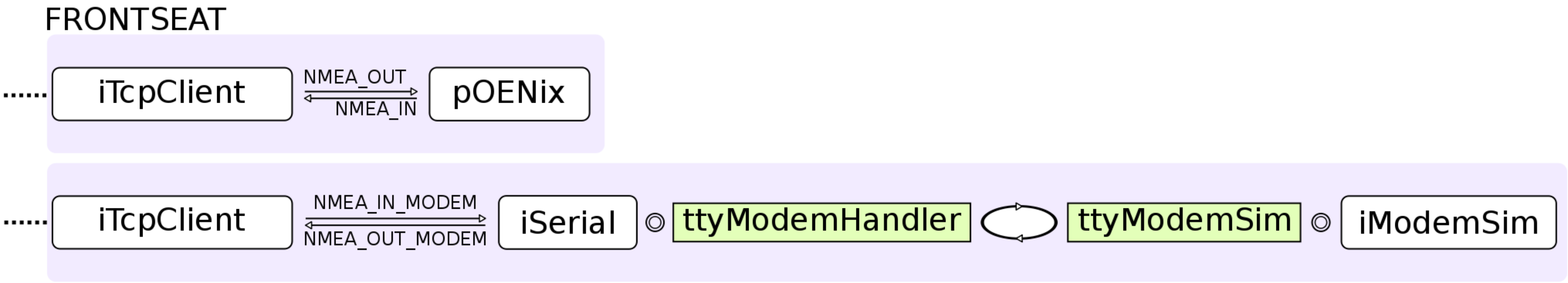
"The ser2net daemon allows telnet and tcp sessions to be established with a unit's serial ports"

http://linux.die.net/man/8/ser2net



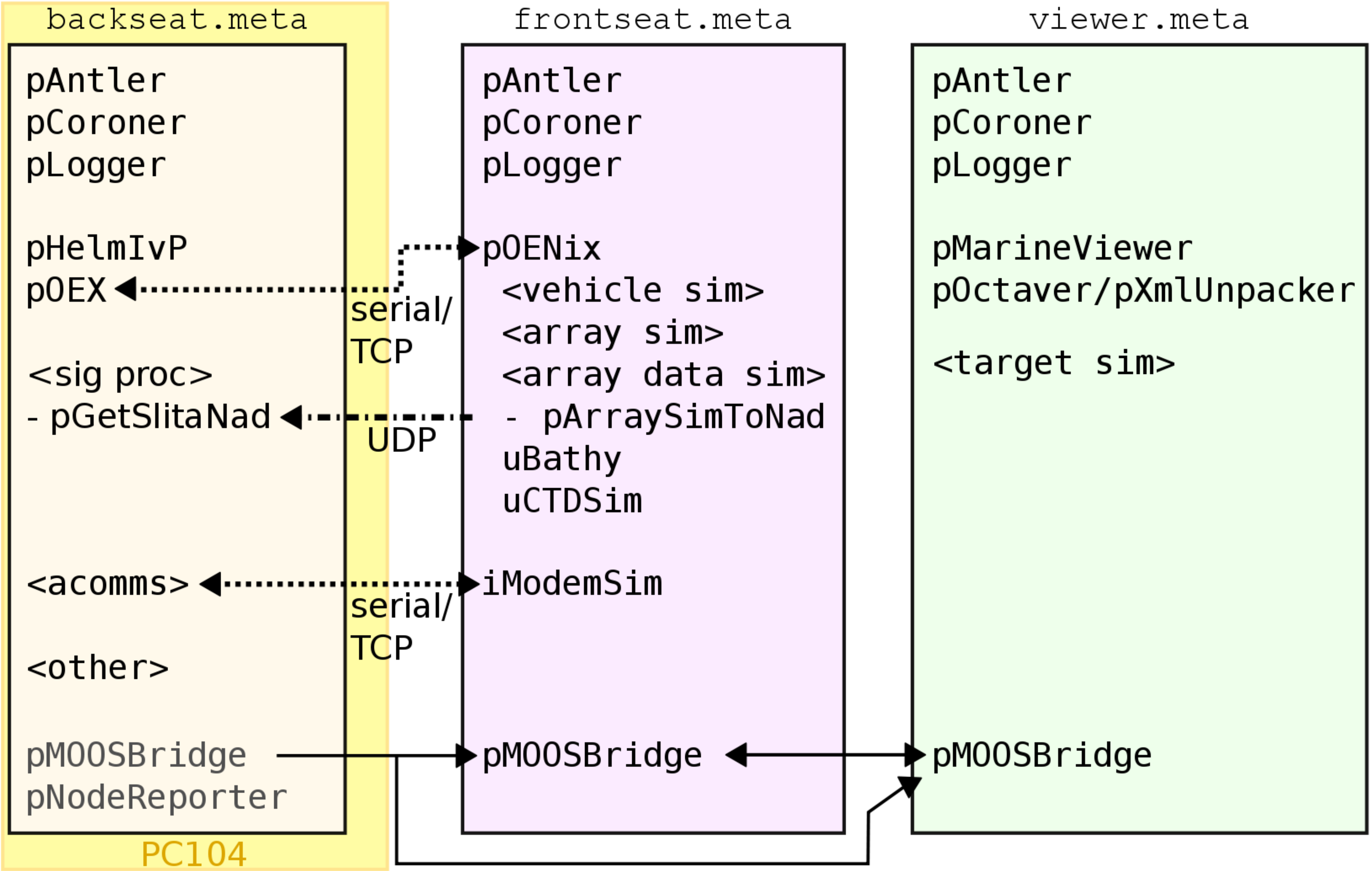
```
# initialize the ser2nets
ser2net -C <local-ip>,<port1>:telnet:600:/dev/ttyOENix:38400
ser2net -C <local-ip>,<port2>:telnet:600:/dev/ttyModemSim:38400
```

Serial port to network: ser2net (2)



- The advantage of modularity / keeping interface processes separate

Current HIL setup



PUBLIC RELEASE

Modularity

- iSerial
- iTcpClient
- iUdpScatter

Summary

HIL simulation helps in

- understanding differences between runtime and simulation
- reducing errors
- error analysis, if something still goes wrong
- increasing trust

Conclusion / Lessons Learned

- modularity
- generation of files
- trust

Distribution

- To workshop participants
- Please use it, and give feedback!

Questions