WiSE-MNet
(Wireless Multimedia Sensor Networks)

Overview & hands-on

Juan C. SanMiguel

Centre for Intelligent Sensing
Queen Mary University of London

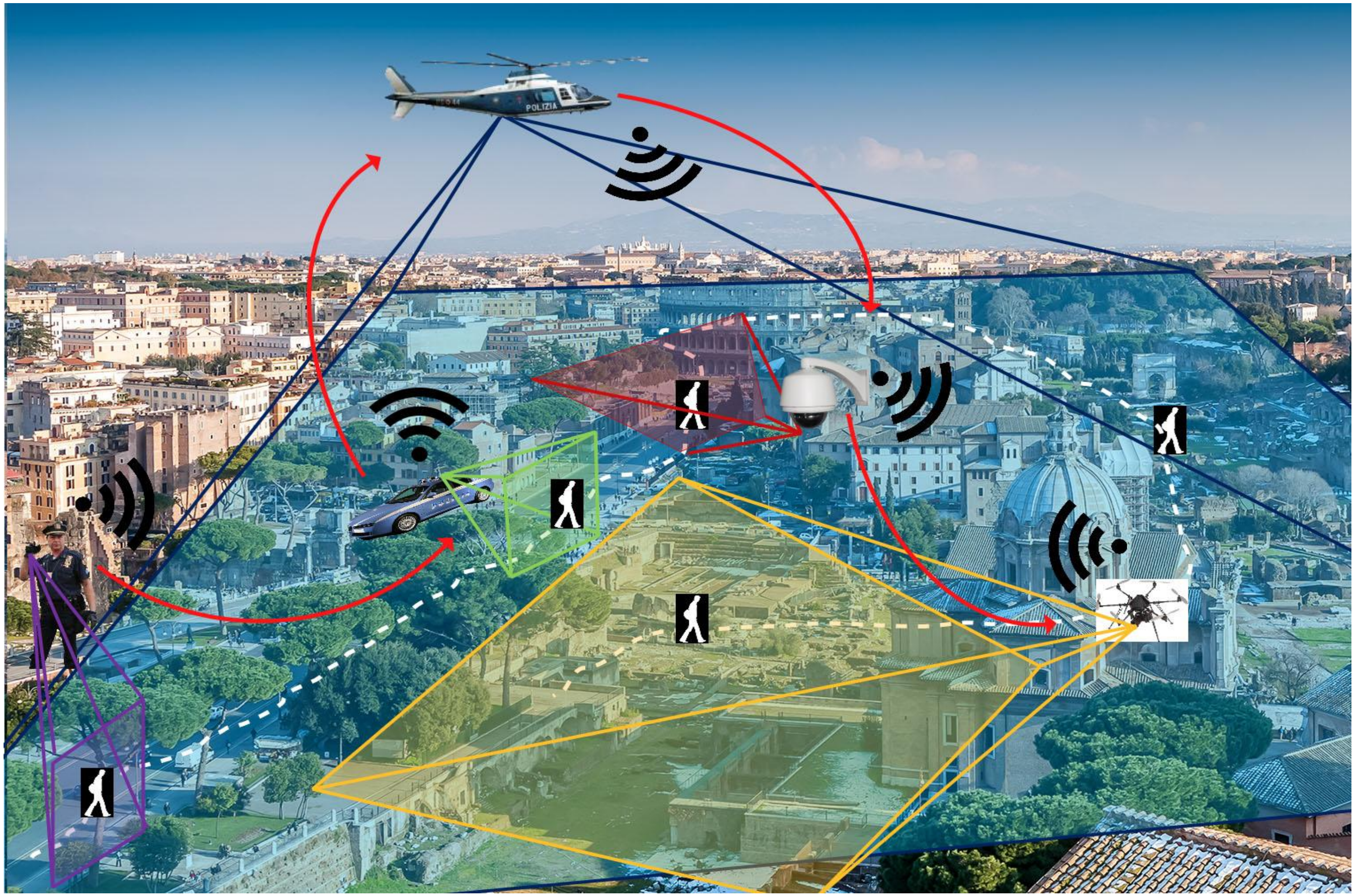
CIS centre for intelligent sensing

Queen Mary University of London

Outline

- Introduction
- Existing simulators
- Basics
- Description (components and extensions)
- Hands-on (installation, GUI and running an app)
- Conclusions

Introduction: why simulators for Camera Networks



DESIRES

Cameras
- Sensing
- Processing
- Communication
- Battery-powered
- …

Network
- Collaboration
- Adaptation
- Scalability
- …

Reproducibility

External factors

CIS centre for intelligent sensing

Queen Mary University of London

Existing simulators for camera networks

	Code	Type	Resources	Processing	Comms	Extendable	Focus
OVVV [CVPR2007]	C++ (Win)	3D	-	-	-	No	Virtual worlds
SLCNR* [IEEE JETCAS2013]	C++ (Win)	3D	-	Vision routines	-	Yes	Virtual worlds
CAMSIM [SISO2013]	Java	2D	-	-	Protocols	Yes	Coordination
WSVN** [WMCNC2010]	C++ (Linux)	2D	Battery, clock, memory	-	Wireless, MAC	~	Video monitoring
M3WSN** [Simutools13]	C++ (Linux)	2D	Battery, clock, memory	-	Wireless, MAC	Code not released	Multimedia TX
WiSE-Mnet** [SSPD2011]	C++ (Linux)	2D	Battery, clock, memory	Cameras & trackers	Wireless, dummy	Yes	Camera networks

*Paid license required

**Requires the libraries: Omnet++ and Castalia

CIS centre for intelligent sensing

Queen Mary University of London

Outline

- Introduction
- Existing simulators
- Basics
- Description (components and extensions)
- Hands-on (installation, GUI and running an app)
- Conclusions

WiSE basics: Omnet++

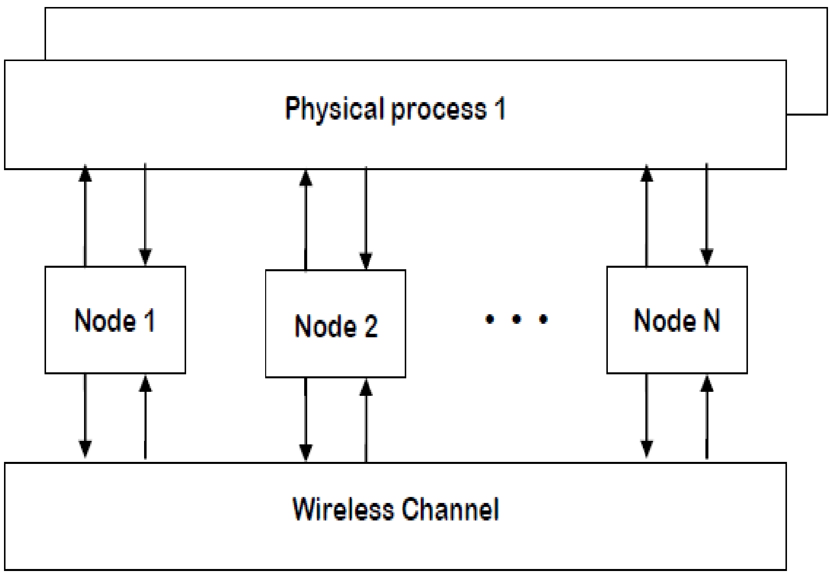
- Generic discrete-event simulation engine
- Generic modules interactions can be defined
  - behaviour is coded in C++
  - interconnections/composition specified through a Network Description (NED) language
  - parameters can be set through configuration files
- Highly flexible and extensible with external libraries
- Network elements
  - nodes, protocols, channels
  - provided (externally) as simulation models (INET, MiXiM, Castalia)

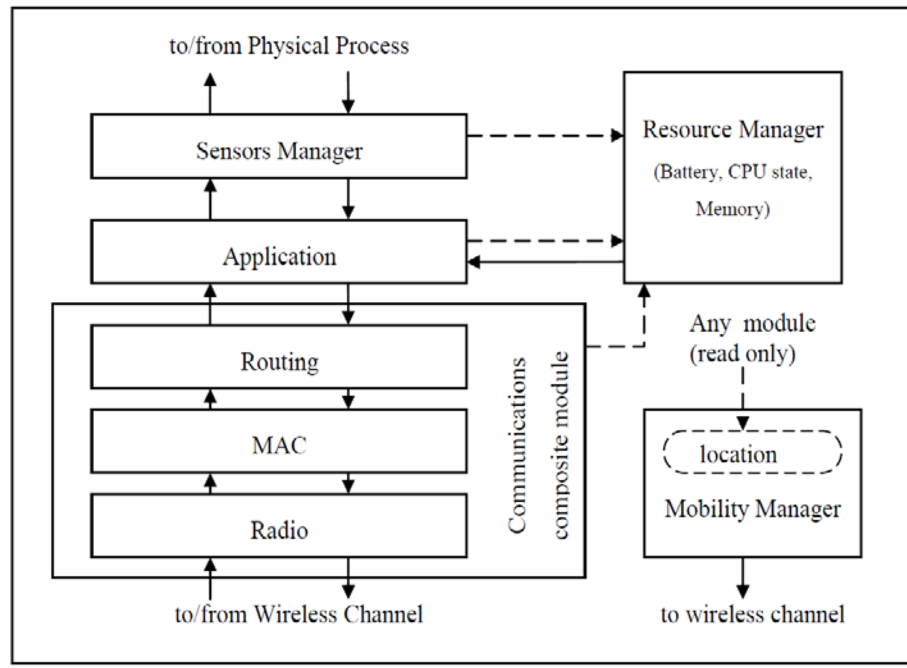
http://www.omnetpp.org

Slide credit: C. Nastasi & A. Cavallaro, 2011

WiSE basics: Castalia

- Wireless sensor networks (WSNs), body area networks (BANs) and networks of low-power embedded devices
- Defines the wireless environment and the node architecture



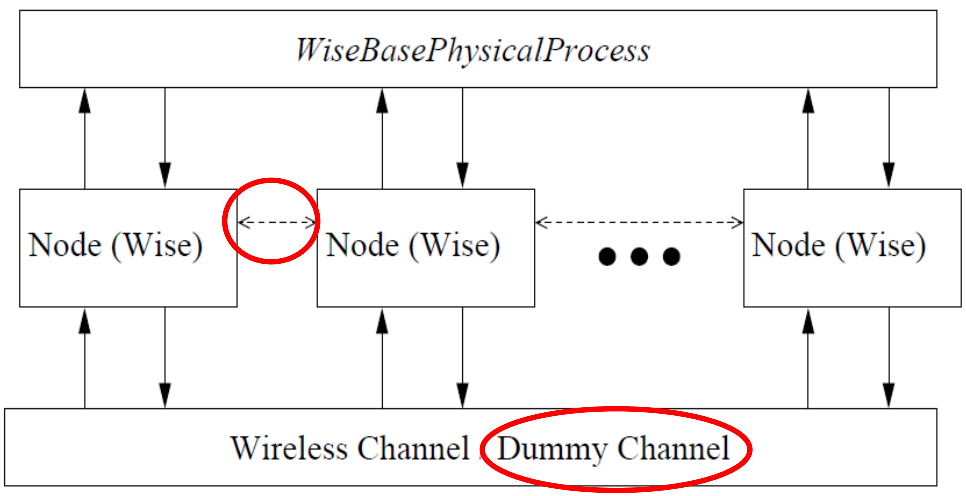


http://castalia.npc.nicta.com.au/

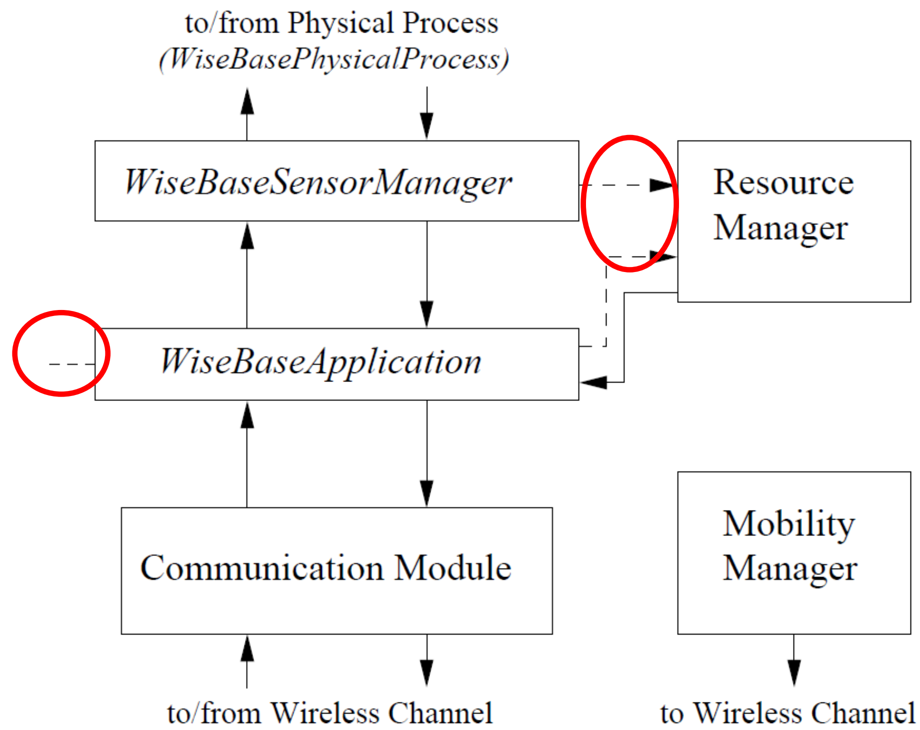
WiSE basics: architecture

- WiSE extends Castalia for Wireless Camera Networks

WiSEXXX files → extensions



(a) Network Model



(b) Node (Wise) Model

WiSE basics: discrete event simulation

- Every sensor/node is independent
- There is no linear script (Matlab) or main (C/C++ projects)
- Omnet++ automatically starts nodes and physical processes

- Communication: message exchange between nodes
- Processing: received messages in discrete units

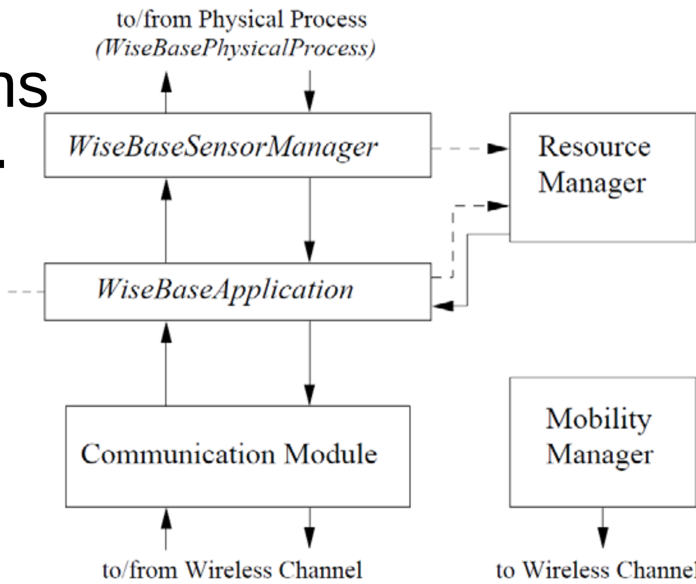
Tic Toc example
More info at http://goo.gl/L3SYBo

Outline

- Introduction
- Existing simulators
- Basics
- Description (components and extensions)
- Hands-on (installation, GUI and running an app)
- Conclusions

WiSE components: NED files

- Describe internal/external connections



(b) Node (Wise) Model

```
module Node {

 parameters:  //basic parameters
        double xCoor = default (0);
        double yCoor = default (0);
        //...

 gates: //connections from/to other modules
        output toWirelessChannel;
        input fromWirelessChannel;

        output toPhysicalProcess[];
        input fromPhysicalProcess[];

 submodules: //submodules of the node
        Communication: node.communication.CommunicationModule;
        MobilityManager: <MobilityManagerName> like node.mobilityManager.iMobilityManager;
        ResourceManager: node.resourceManager.ResourceManager;
        SensorManager: <SensorManagerName> like wise.node.sensorManager.WiseBaseSensorManager

 connections allowunconnected:  //connections between node and submodules
        Communication.toNodeContainerModule --> toWirelessChannel
        fromWirelessChannel --> Communication.fromNodeContainerModule
        Application.toSensorDeviceManager --> SensorManager.fromApplicationModule;
        Communication.toApplicationModule --> Application.fromCommunicationModule
        SensorManager.toApplicationModule --> Application.fromSensorDeviceManager;
        //...

        ResourceManager.toSensorDevManager --> SensorManager.fromResourceManager;
        //...
}
```

Source code in .h, .c and .cc files

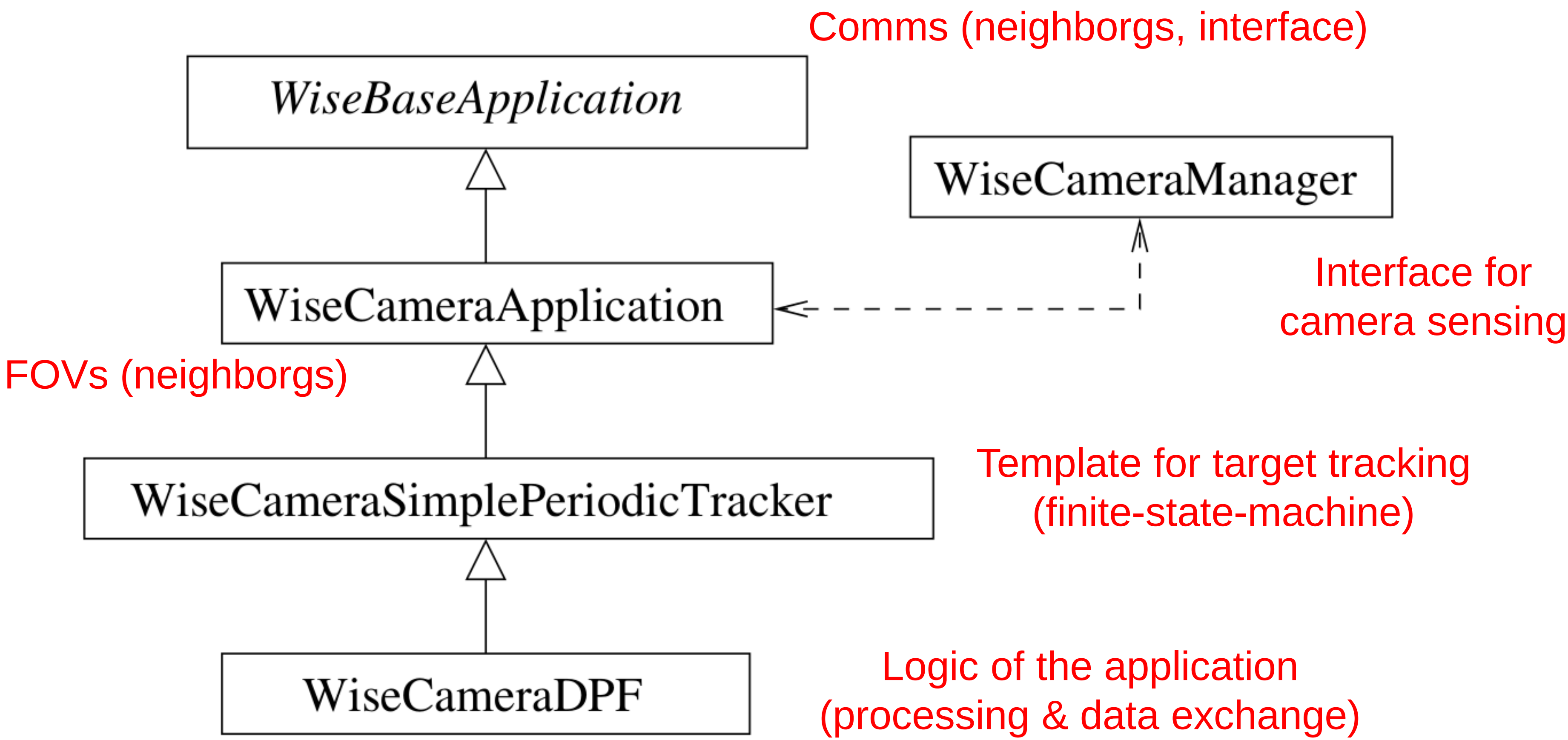
WiSE components: sensing

- Moving targets are represented as "Physical processes"
  - 2D targets --> moving squares (position and size)
  - Type of motion (linear, random,...)
  - Frequency for updating position
- Sensing
  - Return data only if target is within Field of View (but reads all!)
  - *WiseCameraSimplePeriodicTracker* class implements iterative sensing --> frequency to be set

Target position update might be different to sensing frequency

WiSE components: applications

- Sensor logic (processing, receive/send messages,...)
- Application class hierarchy



WiSE components: application (initialization)

- *Startup()* method
- Set timers to define node's behaviour (e.g., sensing rate)
- Initialize variables

```
#include "WiseAppTest.h"
using namespace std;

Define_Module(WiseAppTest);

#define ALARM_SEND_PACKET 0
#define ALARM_SENSOR_SAMPLE 1
#define LOGGER logger << "[" << simTime() << "] @ " << self << " : "

ofstream WiseAppTest::logger;

void WiseAppTest::startup()
{
	// This function is called upon simulation start-up
	if (!logger.is_open())
		logger.open("myLog.txt");
	LOGGER << "WiseAppTest::startup() called" << endl;

	// Set alarm to send a packet (0 delay -> NOW).
	setTimer(ALARM_SEND_PACKET, 0);
	// Set alarm to request a sample to the sensor manager (in 8
	setTimer(ALARM_SENSOR_SAMPLE, 8);
}

void WiseAppTest::finishSpecific()
{
	// This function is called when simulation is finishing
	LOGGER << "WiseAppTest::finishSpecific() called" << endl;
}
```

WiSE components: application (processing)

Response to received message (on-demand task)

fromNetworkLayer()

```
void WiseAppTest::fromNetworkLayer(WiseApplicationPacket * rcvPacket,
                                    const char *src, double rssi, double lqi)
{
        // Function called when a packet is received from the network
        // layer of the communication module
        LOGGER << "WiseAppTest::fromNetworkLayer() called" << endl;

        // Print some packet info: sender ID, RSSI, LQI, payload(hex)
        LOGGER << "\tRx from" << string(src) << " with rssi=" << rssi <<
                " lqi=" << lqi << endl << "\tPayload[] = " << hex;
        for (unsigned c = 0; c < 100; c++)
                logger << (unsigned int) rcvPacket->getPayload(c) << " ";
        logger << dec << endl;
        // Calculate application-to-application communication delay and print it
        ApplicationInteractionControl_type ctl = rcvPacket->getApplicationInteractionControl();
        double l = 1000 * SIMTIME_DBL(simTime() - ctl.timestamp);
        LOGGER << "\t app2app delay = " << l << endl;
}
```

Timer callback (repetitive task)

timerFiredCallback()

```
void WiseAppTest::timerFiredCallback(int index)
{
        // Called when an alarm expires:
        LOGGER << "WiseAppTest::timerFiredCallback() called";
        switch (index) {
        case ALARM_SENSOR_SAMPLE:// alarm was for sensor reading:
                // query the sensor manager a new sample (image)
                requestSensorReading();//call the sensor reading function
                break;
        case ALARM_SEND_PACKET:// alarm was a send packet: create a simple packet of 19200
                // bytes, put some payload and broadcast it.
                WiseApplicationPacket * pkt = new WiseApplicationPacket("Test Pkt",APPLICATION_PACKET);
                // set packet details
                // ...
                toNetworkLayer(pkt, BROADCAST_NETWORK_ADDRESS); //send a message to network
                break;
        default:
                // unexpected alarm ID: generate and error
                opp_error("WiseAppTest::timerFiredCallback(): BAD index");
        }
}
```

CIS centre for intelligent sensing

WiSE components: application (communication)

- Via packets
  - Defined in *.msg files
  - Contains the variables
  - Depends on application

WiseCameraICFMsg.msg

```
cplusplus {{
        #include "WiseApplicationPacket_m.h"
        #include "WiseDefinitionsTracking.h"
        #include <opencv.hpp>
}};

class WiseApplicationPacket;
class noncobject cv::Mat;

packet WiseCameraICFMsg extends WiseApplicationPacket {

        unsigned long trackingCount;
        unsigned long iterationStep;
        unsigned int targetID;
        unsigned int TypeNeighbour;

        cv::Mat matrix; // OpenCV matrix
}
```

- Send packets to network:
  - Specific nodes (in WiseBaseApplication.cc)

    toNetworkLayer()

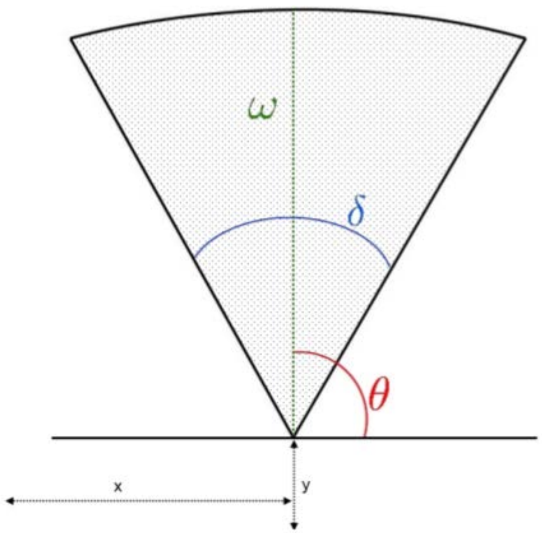
  - Comms/vision graph (In WiseCameraSimplePeriodicTracker.cc)

    send_messageNeighboursCOM()
    send_messageNeighboursFOV()

- Channel: wireless (real) and dummy (ideal)

WiSE extensions

- Capturing from Video files
- Directional sensing (2D FOV)
- Communication/Vision graphs
- Buffer for synchronized comms
- Algorithms
  - Single target tracking
    - Kalman Consensus Filter (KFC)
    - Information Weighted Consensus Filter (IWCF)
  - Multiple target tracking
    - Information Weighted Consensus Filter (IWCF-NN)



Outline

- Introduction
- Existing simulators
- Basics
- Description (components and extensions)
- Hands-on (installation, GUI and creating/running an app)
- Conclusions

WiSE hands-on: installation

1. Install dependencies of Omnet++
2. Install Omnet++
3. Install dependencies of OpenCV
4. Install OpenCV
5. Download WiSE package*
6. Setup a project using the WiSE package*

*Identical installation for Castalia (not required as it is included in WiSE)

Only runs in Linux!!!
(can run in Windows without OpenCV)

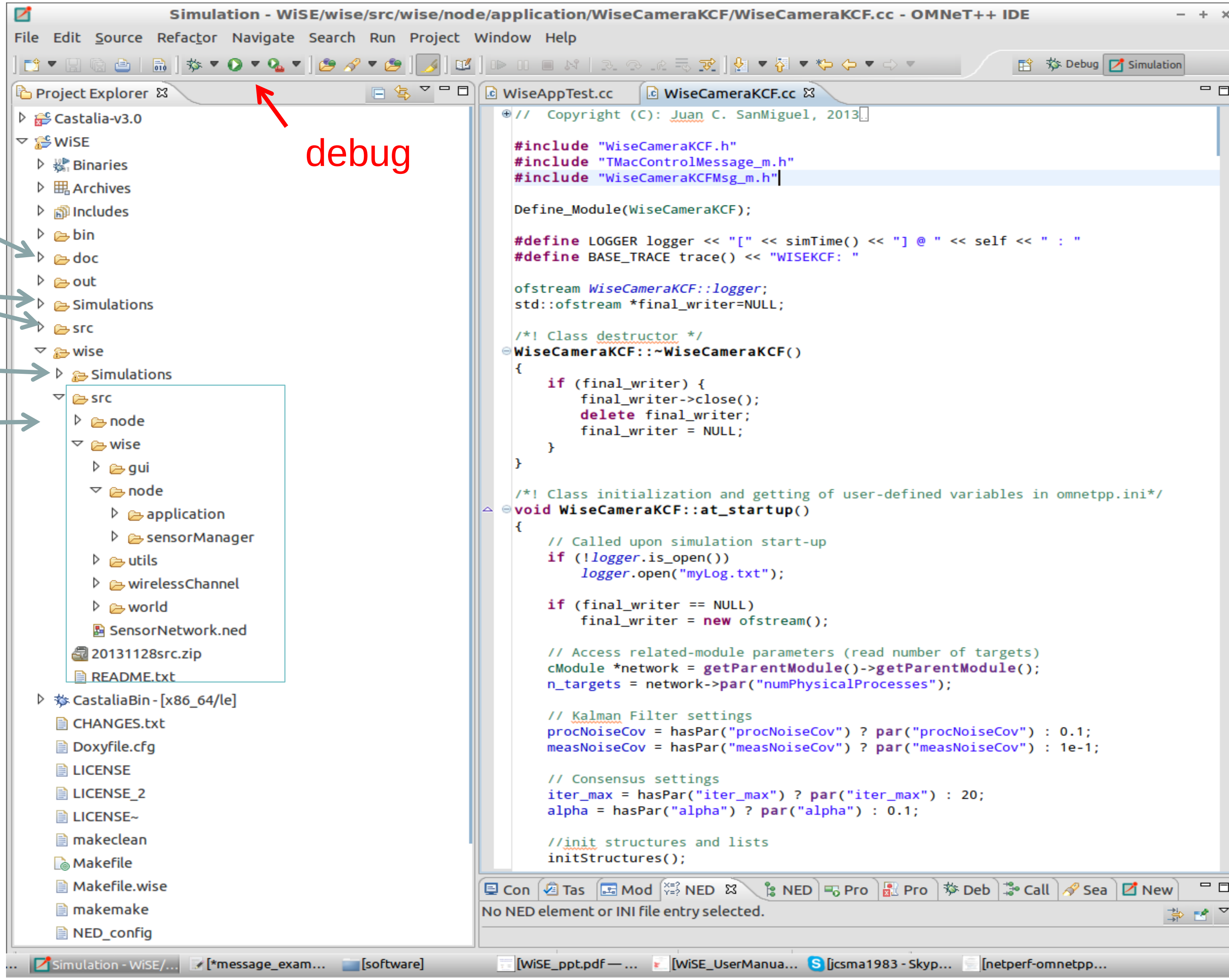
WiSE hands-on: GUI

Code docs

Castalia

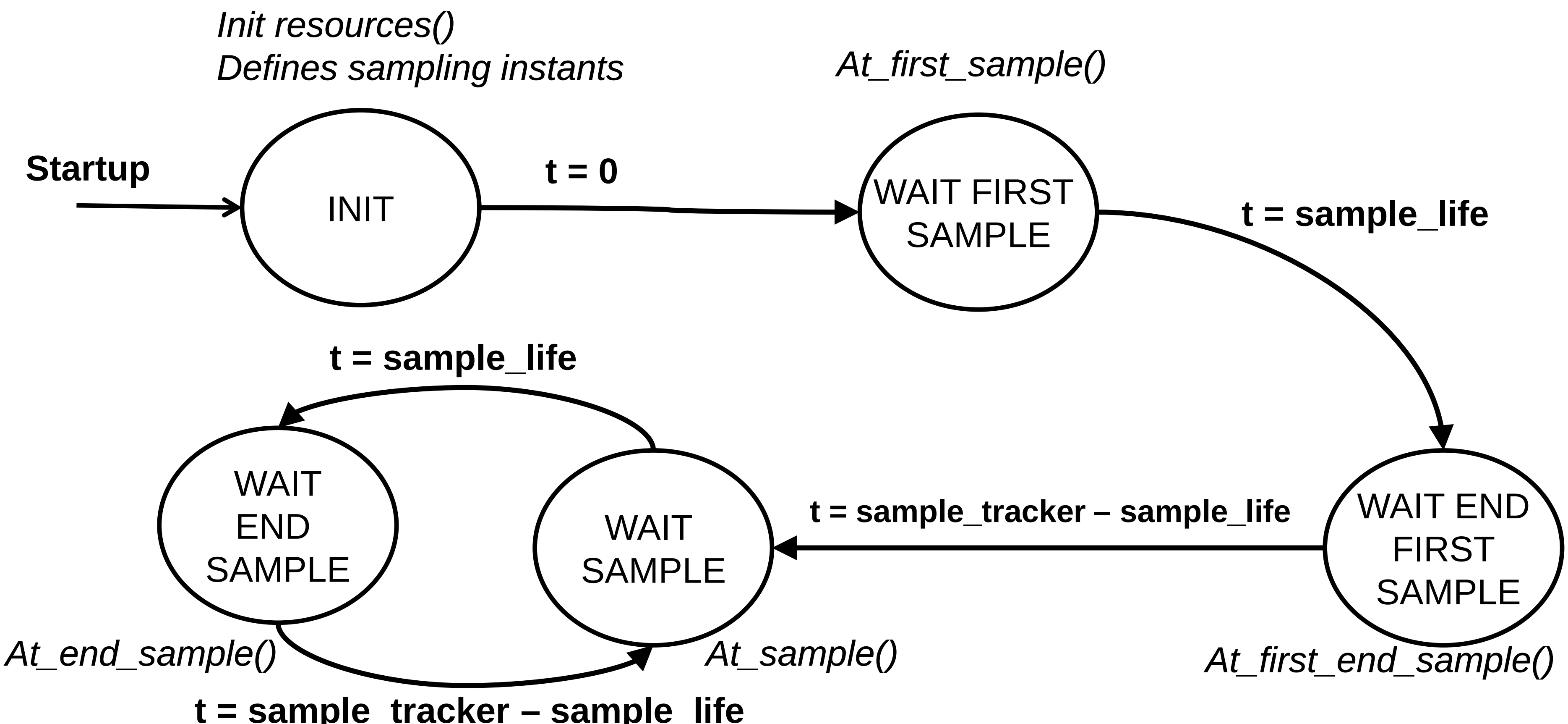
Simulations

WiSE code



WiSE hands-on: creating an app (1/2)

- <u>New trackers</u> as derived classes of WiseCameraSimplePeriodicTracker



WiSE hands-on: creating an app (2/2)

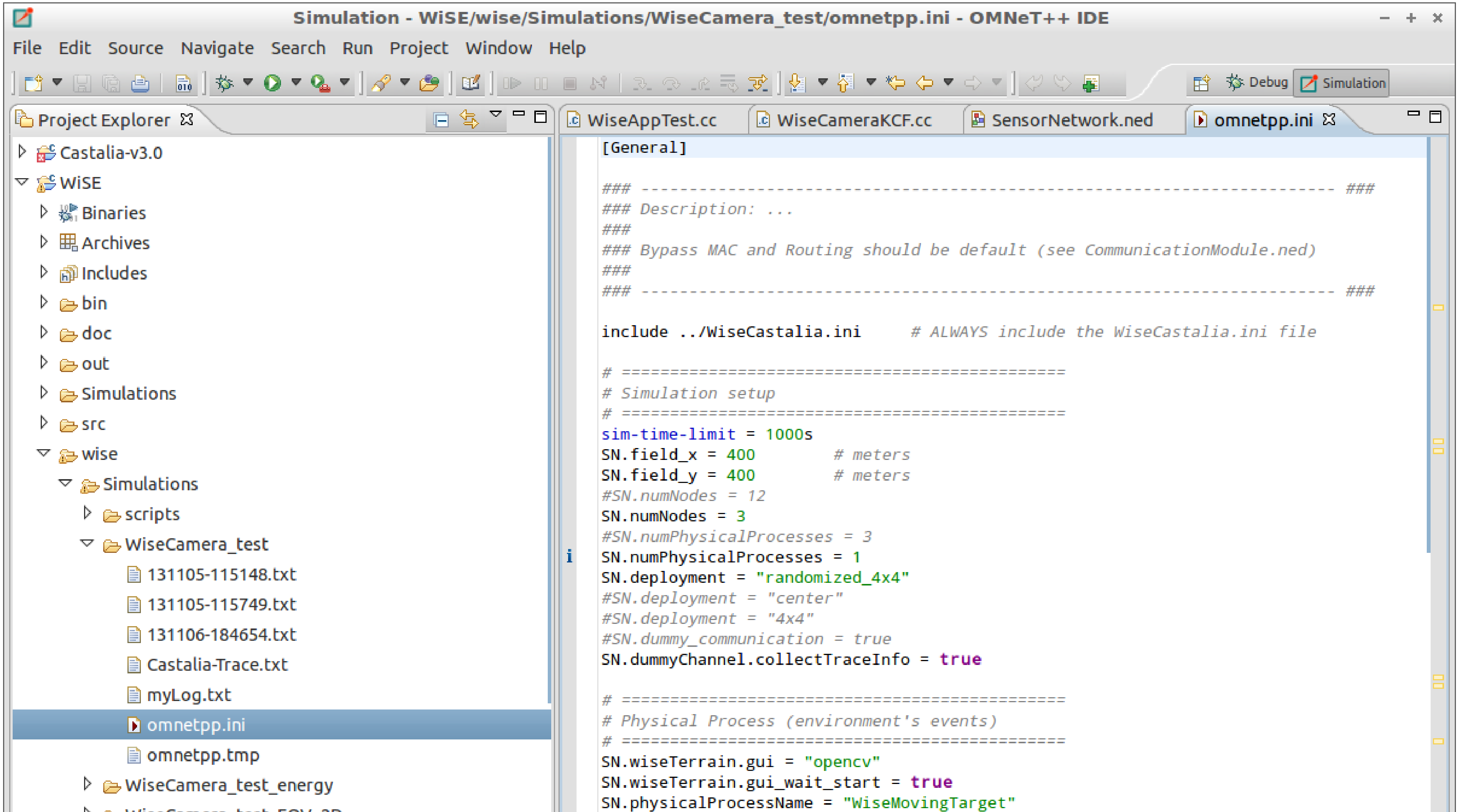
```
#include "WiseCameraSimplePeriodicTracker.h"
#include "WiseCameraICFMsg_m.h"
#include "WiseDefinitionsTracking.h" //include for definitions of states and measurements
#include "WiseCameraICF_utils.h" //include specific-structures for single-target tracking of ICF

#define MAX_SIZE_BUFFER 10

/*! \class WiseCameraICF
 *  \brief This class implements distributed Single-target tracking based on ICF
 */
class WiseCameraICF : public WiseCameraSimplePeriodicTracker
{

private:
    // Define variables
    // ...

protected:
    // Functions to be implemented from WiseCameraSimplePeriodicTracker class
    virtual void at_startup();                          //!< Init internal variables.
    virtual void at_timer_fired(int index) {} ;        //!< Response to alarms generated by specific tracker.
    virtual void at_tracker_init();                    //!< Init resources.
    virtual void at_tracker_first_sample();          //!< Operations at 1st example.
    virtual void at_tracker_end_first_sample();    //!< Operations at the end of 1st example.
    virtual void at_tracker_sample();                 //!< Operations at the >1st example.
    virtual void at_tracker_end_sample();           //!< Operations at the end of >1st example.

    // Functions to be implemented from WiseBaseApplication class
    virtual void process_network_message(WiseApplicationPacket *); //!< Processing of packets received from network.
    virtual void handleDirectApplicationMessage(WiseApplicationPacket *); //!< Processing of packets received from network
    virtual void make_measurements(const std::vector<WiseTargetDetection>&);  //!< Conversion of camera detections into b
                                                                              //lists of measurements for tracking.
```

Functions to implement from tracking template

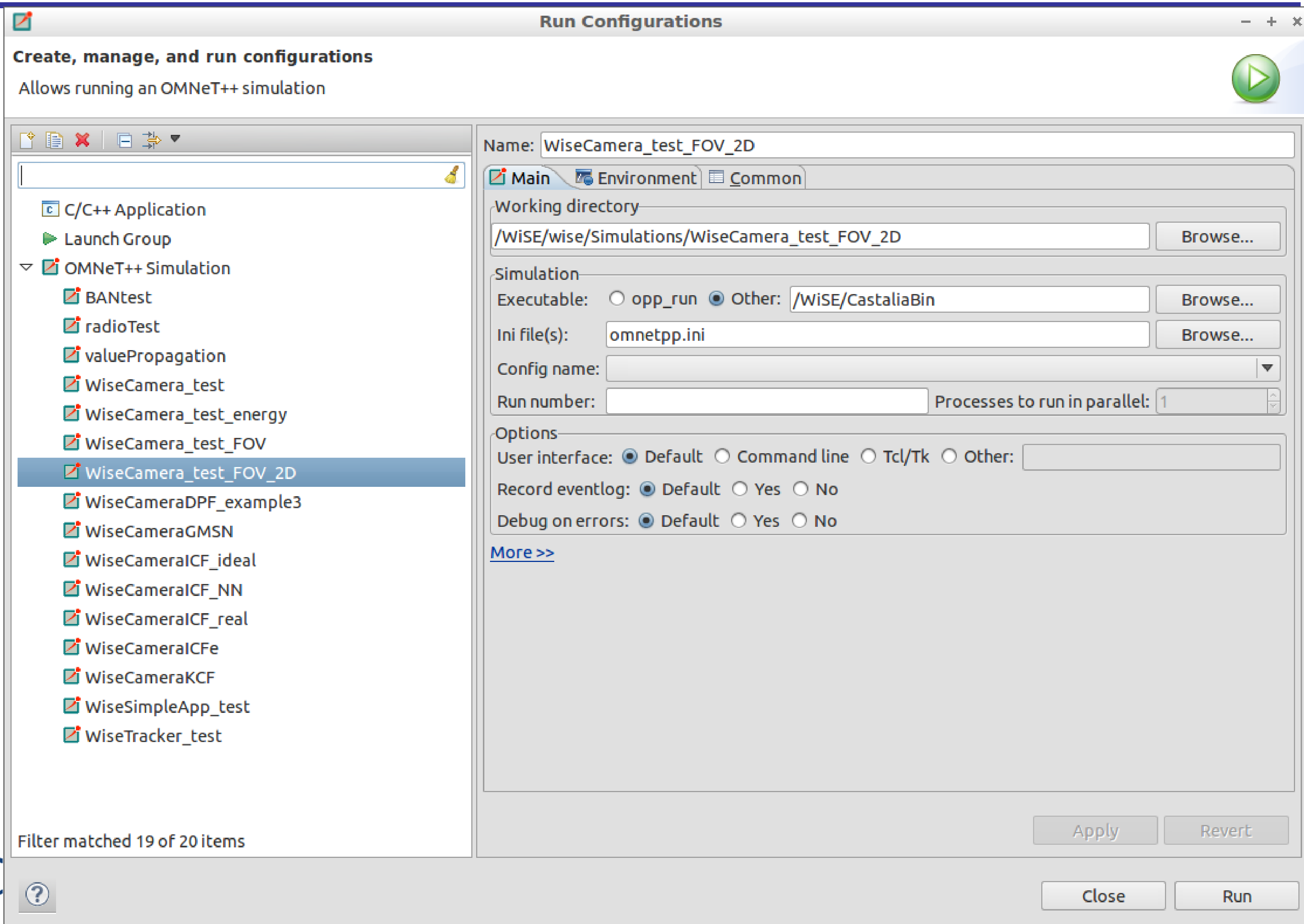
Functions to implement from base template

WiSE hands-on: running an app (1/2)

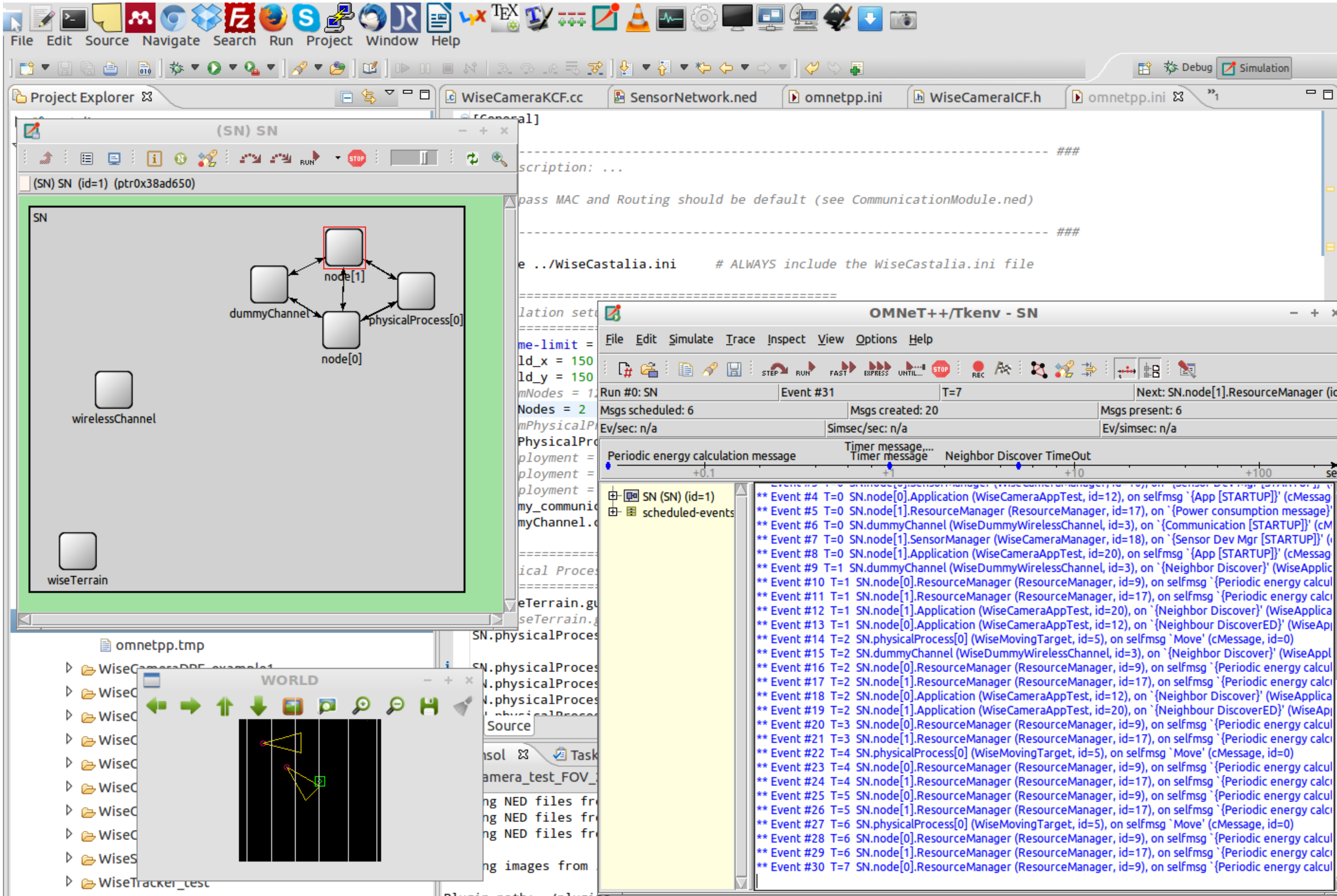
- Configuration (ini files)



WiSE hands-on: running an app (2/2)



WiSE hands-on: example



Conclusions

- WiSE enables research on camera networks via simulations of realistic environments
  - Resource constraints
  - Coordination among cameras
  - Real communication protocols
  - Image/Video processing tools
- Expertise required
  - C/C++ language
  - Linux programming skills (gcc compiler)
  - Non-linear design (i.e. collaborative processing)
- Ongoing work: develop resource-limited scenarios

References

OVVV: G. Taylor, A. Chosak, and P. Brewer, "*OVVV: Using virtual worlds to design and evaluate surveillance systems*," pp. 1-8, CVPR 2007. http://development.objectvideo.com/

SLCNR: W. Starzyk and F. Qureshi, "*Software laboratory for camera networks research*," IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 62(2): 284–293, June 2013. http://vclab.science.uoit.ca/~faisal/projects/vvs/index.html

CAMSIM: L. Esterle, P. R. Lewis, H. Caine, X. Yao, and B. Rinner, "*CamSim: A Distributed Smart Camera Network Simulator*," in Proc. of the IEEE Int. Conf. on Self-Adaptive and Self-Organizing Systems Workshops, pp. 1-2, Sept. 2013 https://github.com/EPiCS/CamSim

WSVN: A. Pham, C.; Makhoul, "*Performance study of multiple cover-set strategies for mission-critical video surveillance with wireless video sensors*," in IEEE Int. Conf. on Wireless and Mobile Computing, Networking and Communications., pp. 208-216, Oct. 2010, http://web.univ-pau.fr/~cpham/WSN-MODEL/wvsn.html

M3WSN: D. Rosario, Z. Zhao, C. Silva, E. Cerqueira, and T. Braun, "*An OMNeT++ framework to evaluate video transmission in mobile wireless multimedia sensor networks*," in Proc. of the Int. ICST Conf. on Simulation Tools and Techniques, pp. 277–284, Mar. 2013. http://cds.unibe.ch/research/M3WSN/

WiSE: C. Nastasi, A. Cavallaro*, "WiSE-MNet: an experimental environment for Wireless Multimedia Sensor Networks*"**,** Proc. of Sensor Signal Processing for Defence (SSPD), London, UK, 28-29 September, 2011 http://www.eecs.qmul.ac.uk/~andrea/wise-mnet.html

References

Implemented algorithms

KFC: Reza Olfati-Saber, J. Alex Fax, and Richard M. Murray. Consensus and cooperation in networked multi-agent systems. In Proceedings of theIEEE, 2007

IWCF: A. T. Kamal, J. A. Farrell, A. K. Roy-Chowdhury *Information Weighted Consensus Filters and their Application in Distributed Camera Networks*, , IEEE Transactions on Automatic Control, 2013

ICF-NN: A. T. Kamal, J. A. Farrell, A. K. Roy-Chowdhury, *Information Consensus for Distributed Multi-Target Tracking*,, IEEE Conf. on Computer Vision and Pattern Recognition, 2013

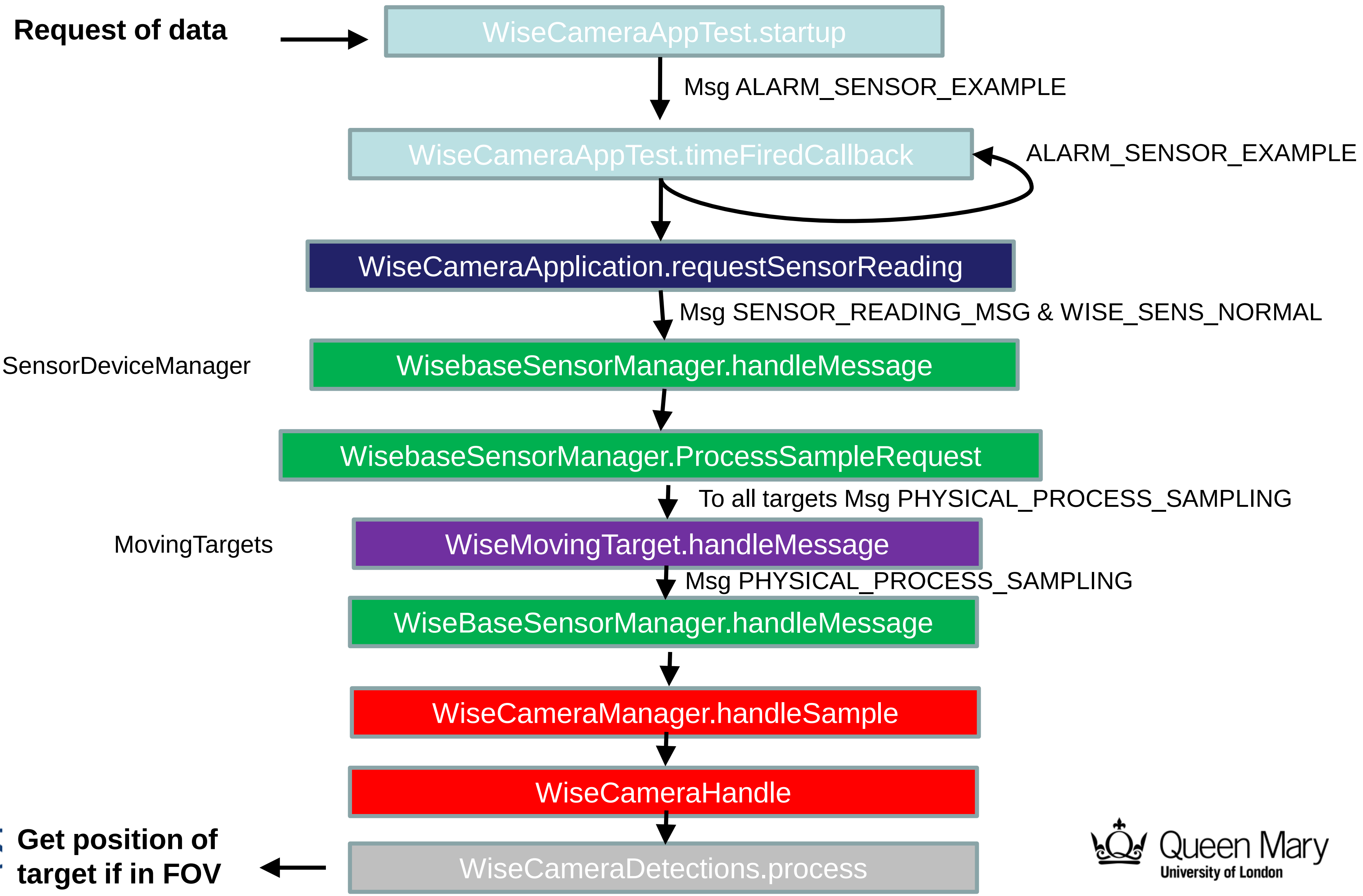
Additional resources: links

- Tutorials Omnet++
  - http://www.omnetpp.org/doc/omnetpp/tictoc-tutorial/
  - http://titania.ctie.monash.edu.au/netperf/netperf-omnetpp-ide-getting-started.pdf
  - http://web.univ-pau.fr/~cpham/ENSEIGNEMENT/PAU-UPPA/PROTOCOLES/omnetp.pdf

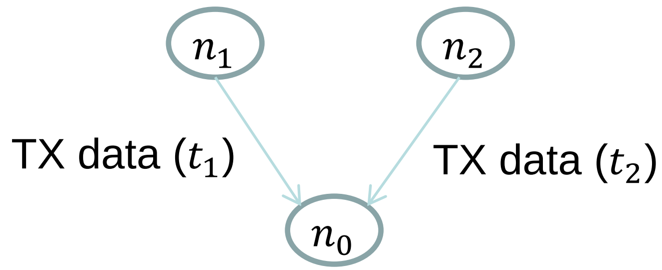
- Tutorials Castalia
  - http://castalia.npc.nicta.com.au/documentation.php

- Tutorials WiSE
  - http://www.eecs.qmul.ac.uk/~andrea/wise-mnet.html

Additional resources: capturing data pipeline



Additional resources: synchronization problem



Implicit coordination strategies require to fuse all neighbour data of the same iteration (ie, iterations of the consensus approach)
→ Data from same iterations received at different time instants ($t_1 \neq t_2$)

Solution (for consensus)

Implement a buffer that stores the data of different iterations. Each node will:
- Save up to MAX_TAM_SIZE iterations of consensus in the buffer
- When receiving data, it will be stored in the corresponding buffer position
- When the last data for an iteration of consensus is received, perform iteration and free the buffer position for other future iterations