A DISTRIBUTED INFRASTRUCTURE FOR THE SYNCHRONIZED ACQUISITION OF SENSOR DATA

-ICS 214b class project-

Winter 2006

Chi Fai Chan

Hojjat Jafarpour

Shengyue Ji

Kuan Sung Lee

Daniel Massaguer

Rares Vernica

{chancf, hjafarpo, shengyue, klee10, dmassagu, rvernica}@uci.edu

Project advisors: Utz Westermann and Prof. Sharad Mehrotra

RESPONSPHERE

The Place

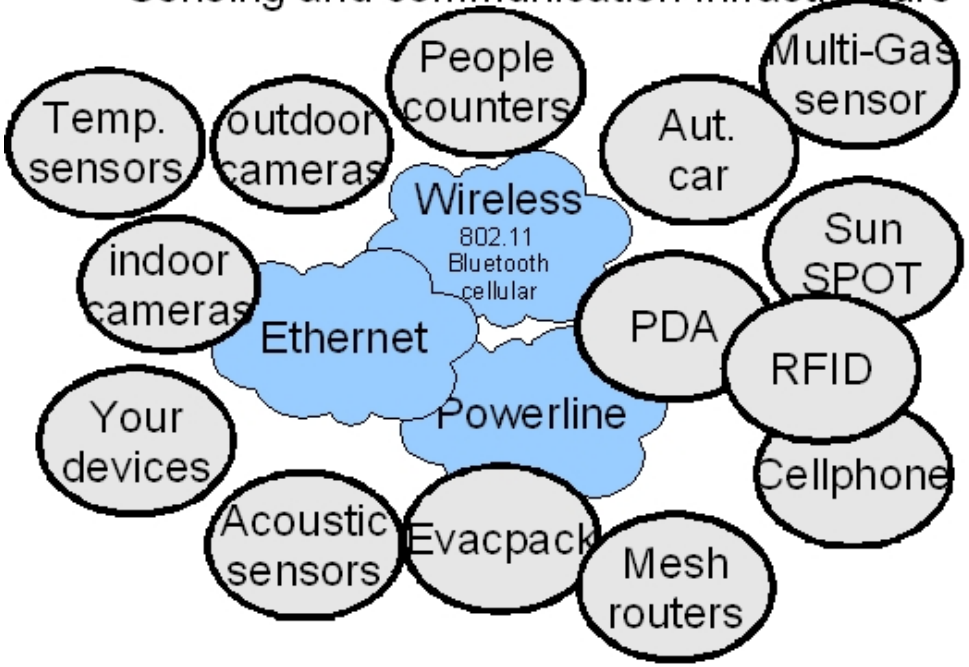
responsphere
infrastructure

RESPONSPHERE

The Place

responsphere infrastructure

Sensing and communication infrastructure

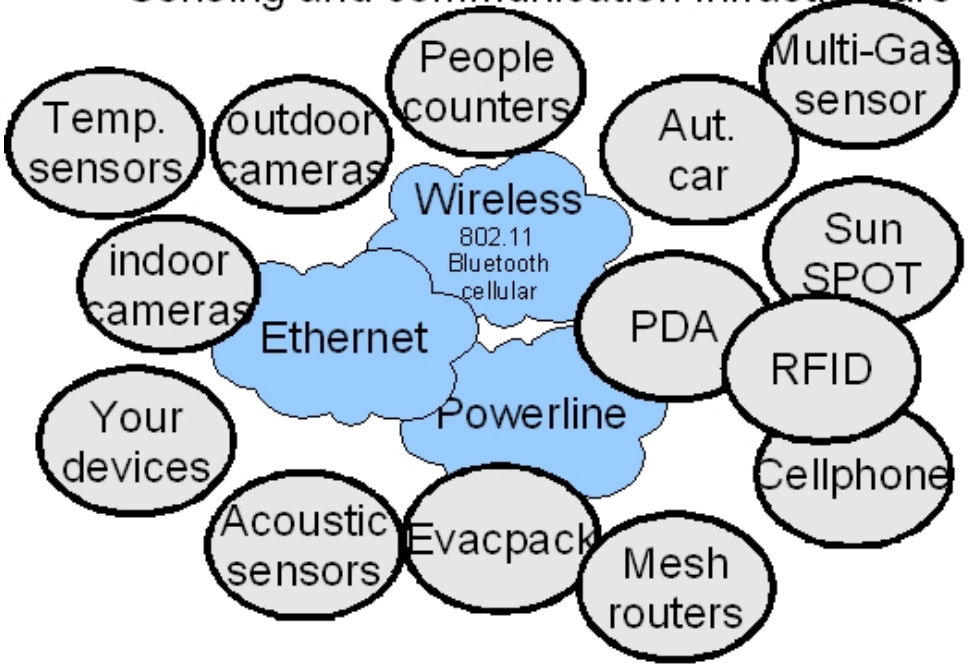


RESPONSPHERE

The Place

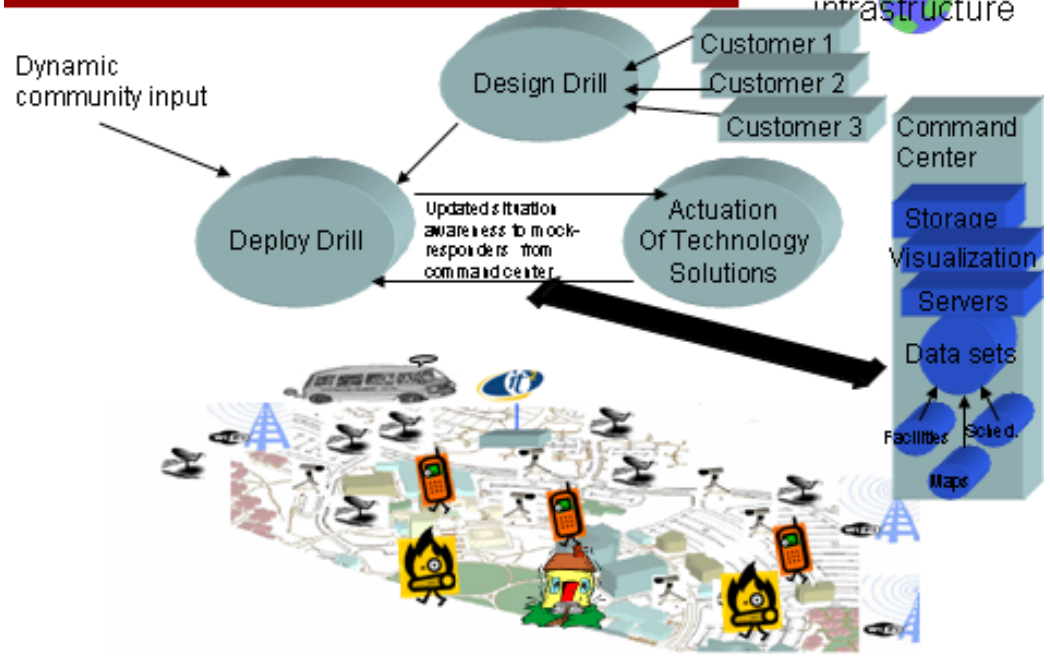
responsphere infrastructure

Sensing and communication infrastructure



The Paradigm

responsphere infrastructure

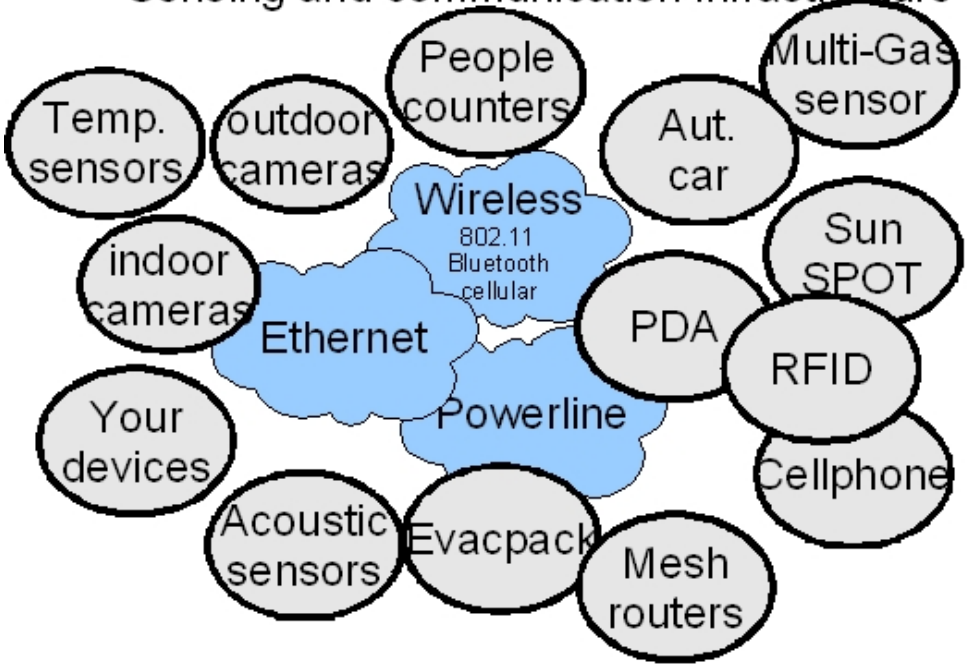


RESPONSPHERE

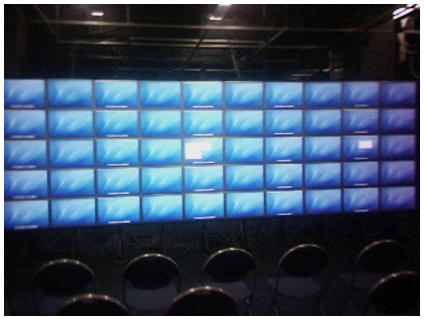
The Place

responsphere infrastructure

Sensing and communication infrastructure



Computing, visualization, and datasets

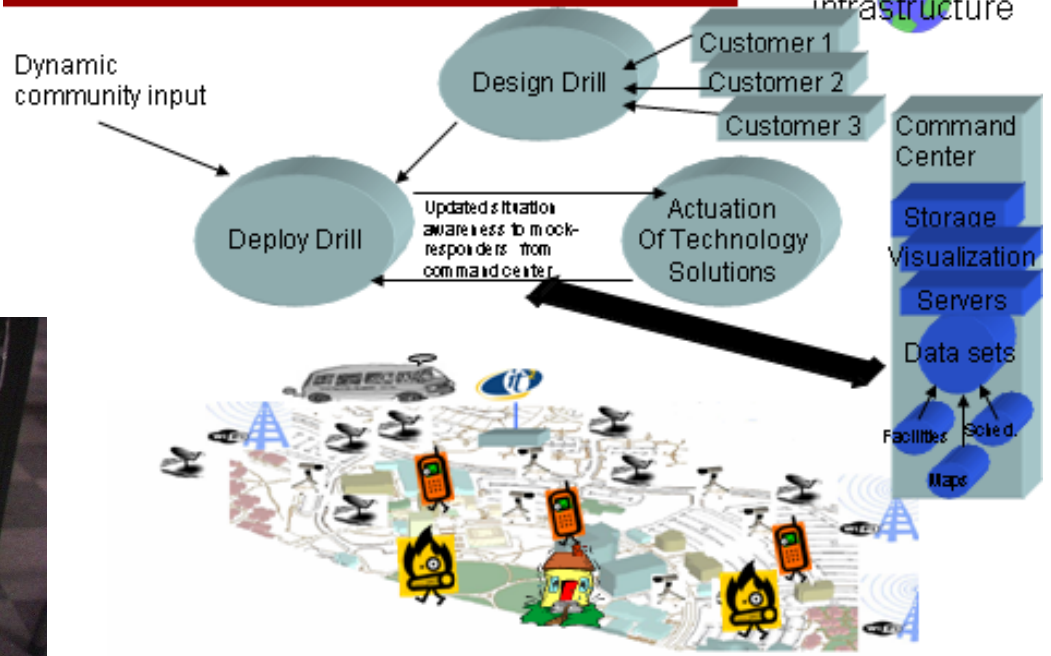


- RAID storage server
- multi-tile visualization display
- 8-32-bit-processor IBM server
- 8-64-bit-processor Sun server
- Datasets: drill, 911 calls, 9-11, CAD, GIS, people counter logs, LDC TDT4, disasters, KDD, UCI facilities

http://rescue-ibm.calit2.uci.edu/datasets/

The Paradigm

responsphere infrastructure



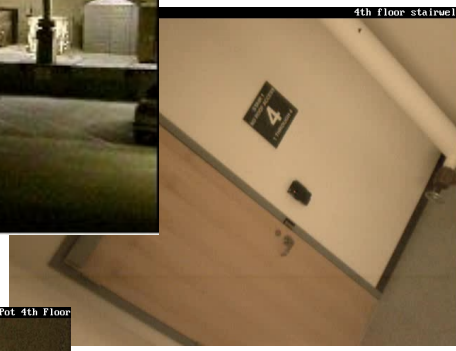
APPLICATIONS

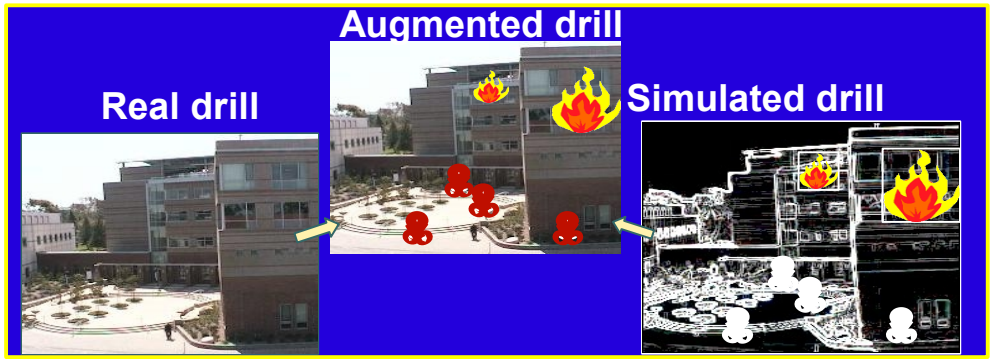
The infrastructure will allow to perform:

- People counting
- People tracking
- Event detection (hazard, security policy violation, etc)
- etc

Which enables applications such as:

- Testing IT solutions for emergency response in the context of a drill
- Surveillance (e.g. Video and/or RFID surveillance)
  - Coffee room control
- Augmenting a drill simulation
- SAMI
- Quasar
- Media broker (SAMI, VIEWS)
- Aut. sensing platform
- etc





PROBLEM

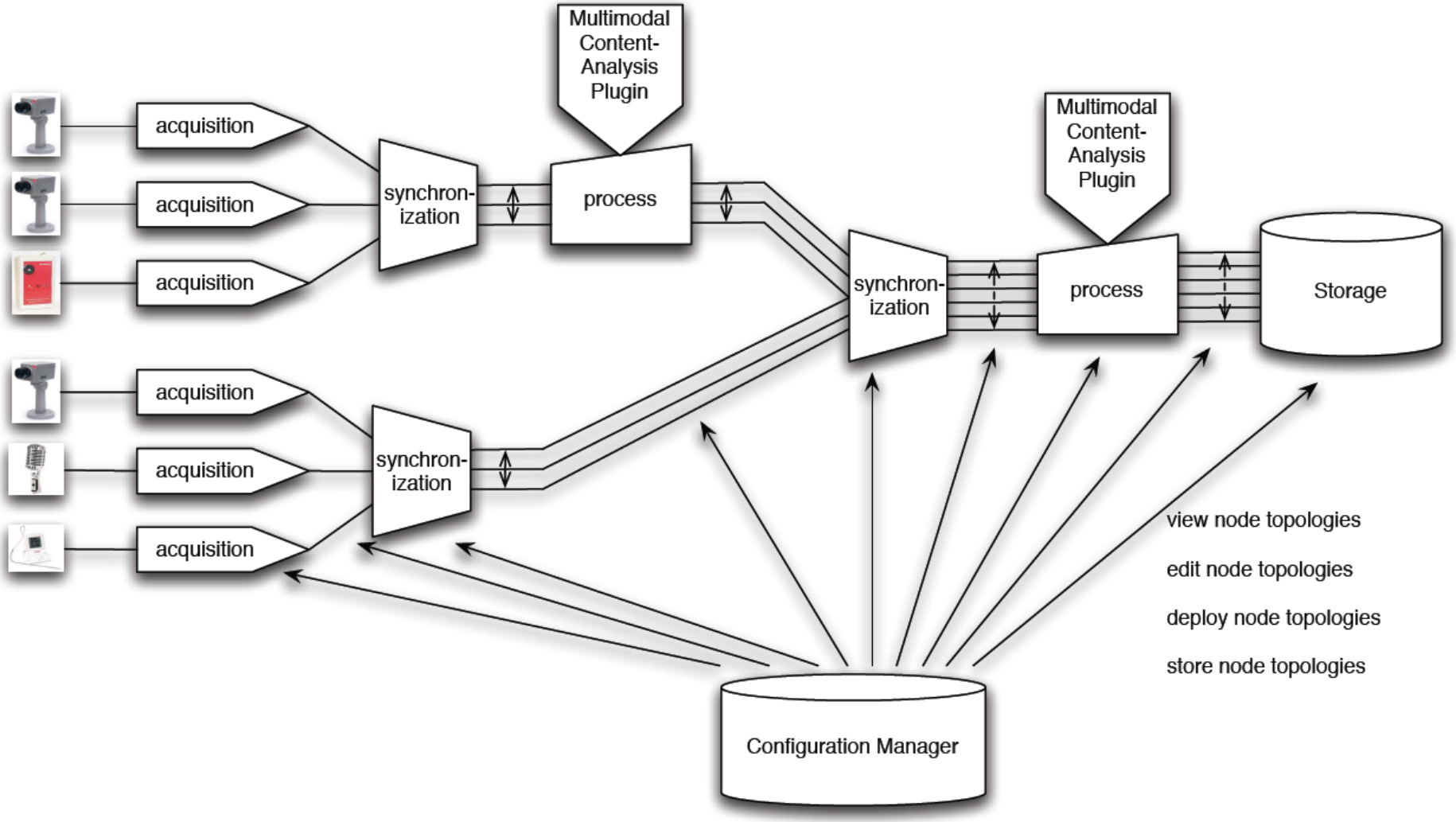
Collect data from sensing infrastructure

Data is unsynchronized

Data is multi-modal

Event detection by multi-modal processing

A DISTRIBUTED INFRASTRUCTURE FOR THE SYNCHRONIZED ACQUISITION OF SENSOR DATA



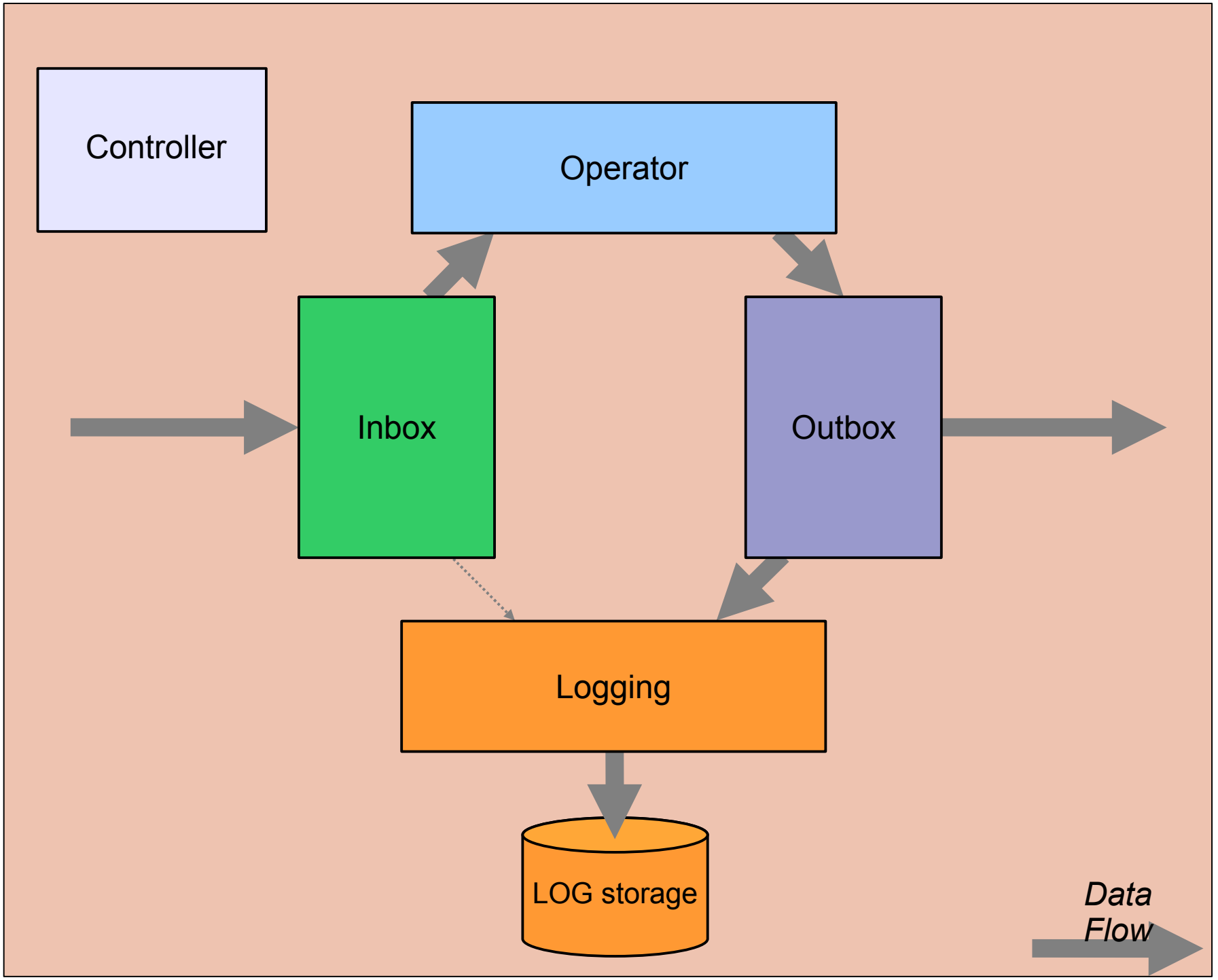
TECHNICAL GOALS

Multi-model sensing

Flexibility to support different applications

Simple real-time content analysis:
- reliability (nodes failing)
- synchronization
- abstraction from physical nuisances

A NODE


Controller
Operator
Inbox
Outbox
Logging
LOG storage
Data Flow

CONTROLLER

- Controls the node's functionality
- Function:
  - Creating different modules
  - Connecting them to each other
  - Starting up the node
- Configuration
  - File
  - Network

CONTROLLER

- Sample configuration file content:

Server Config

Port
8089
Server Channel
channel_1
Server Channel
channel_2

Client Config

addChannel
channelName
channel_1
serverName
localhost
serverPort
8089
addChannel
channelName
channel_2
serverName
127.0.0.1
serverPort
8089

COMMUNICATION BETWEEN NODES

Data wrapping

Reason for wrapping:

- Network traffic modes: data stream, data diagram…
- Data formats: mpeg, asf…

By wrapping we provide a generic data sending/receiving interface for different kinds of usage.

Also we hide the networking details of transferring data from the above layer.

COMMUNICATION BETWEEN NODES

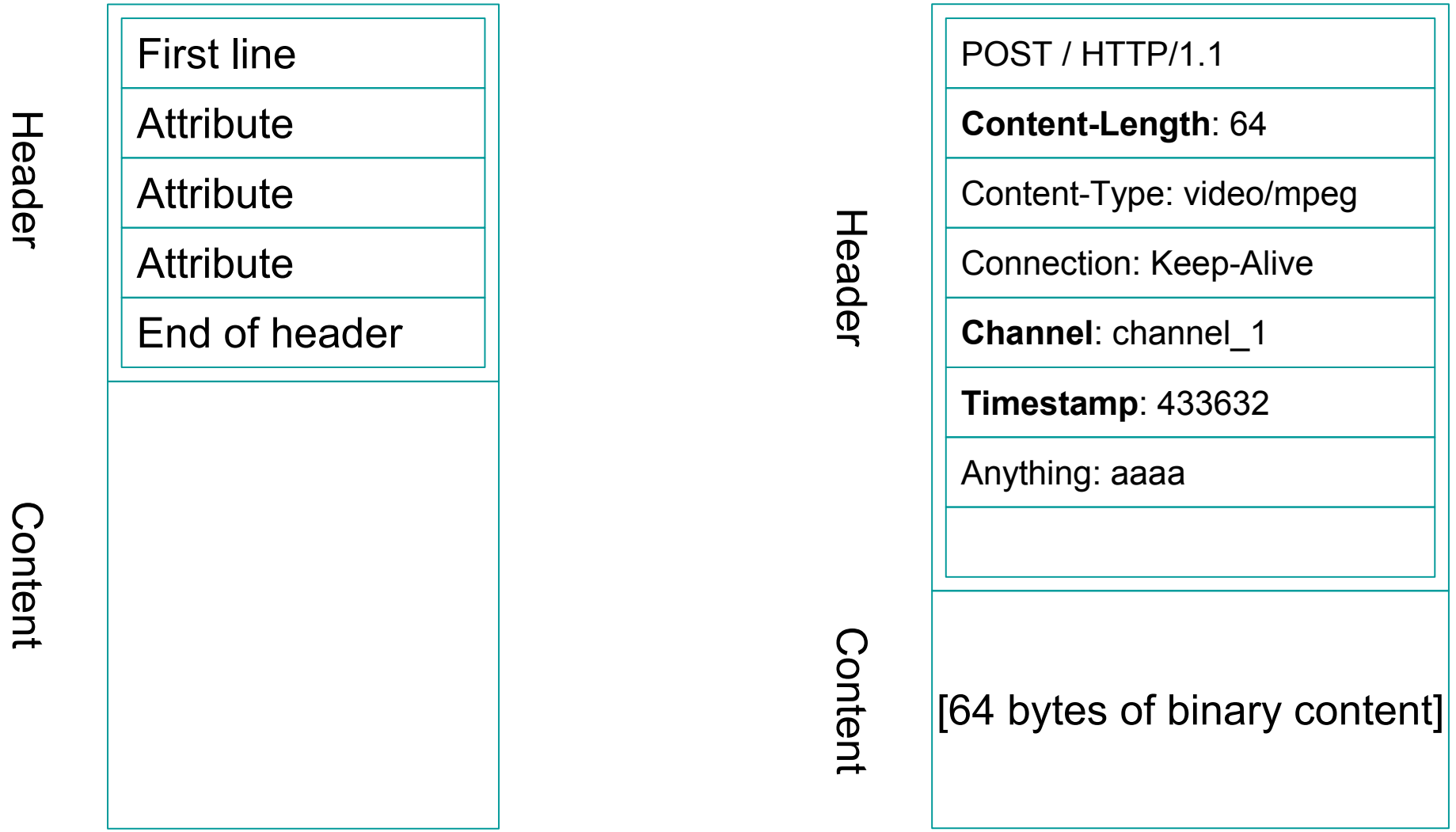
Data wrapping

- We are using:
  - TCP protocol (to ignore data lost in network)
  - Data packet mode (to provide support for data diagram, also compatible with data stream)

    *this also enables the controlling of traffic loads.
  - HTTP protocol (to take advantage of existing protocol)

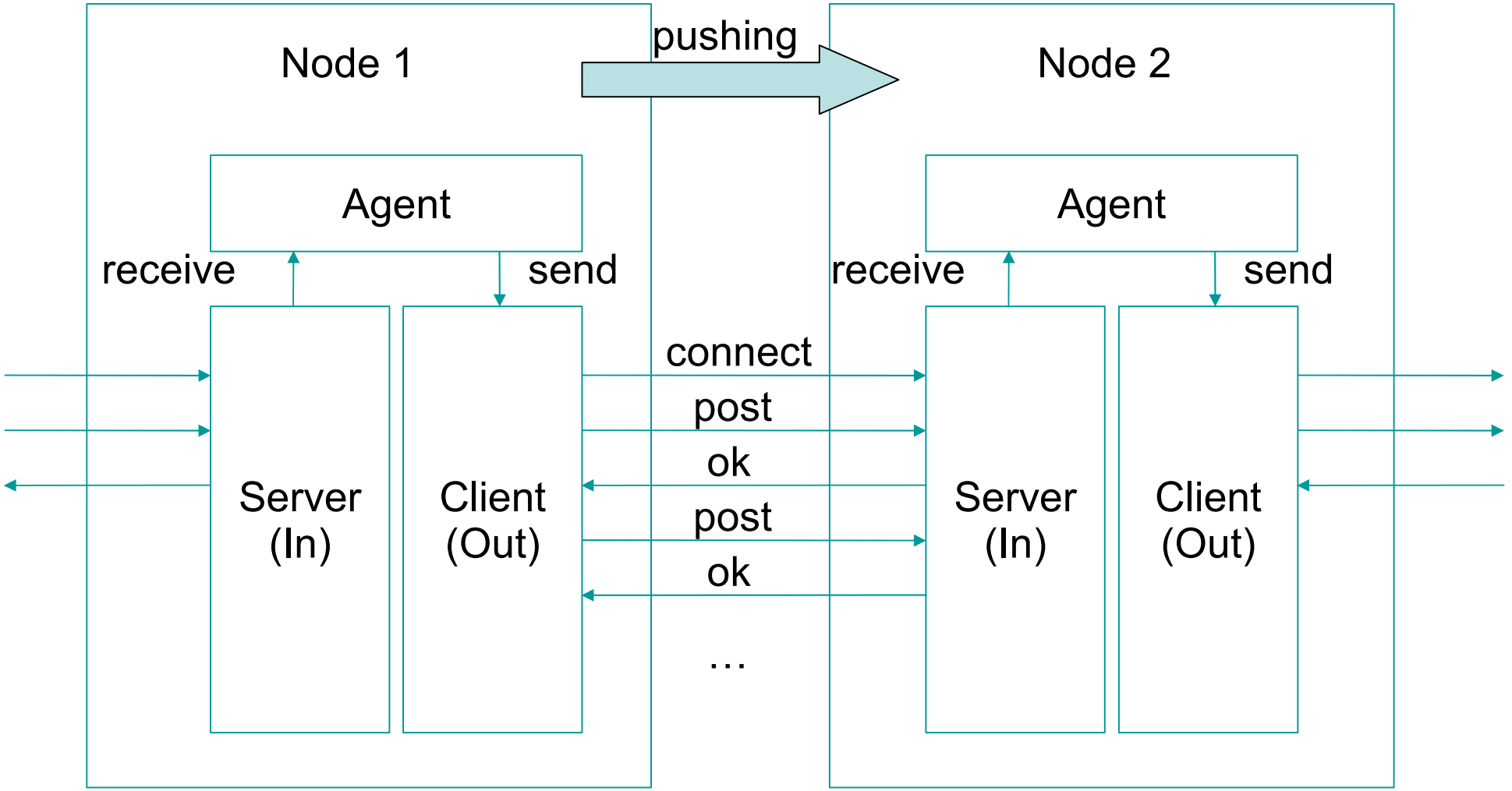
COMMUNICATION BETWEEN NODES

Data wrapping



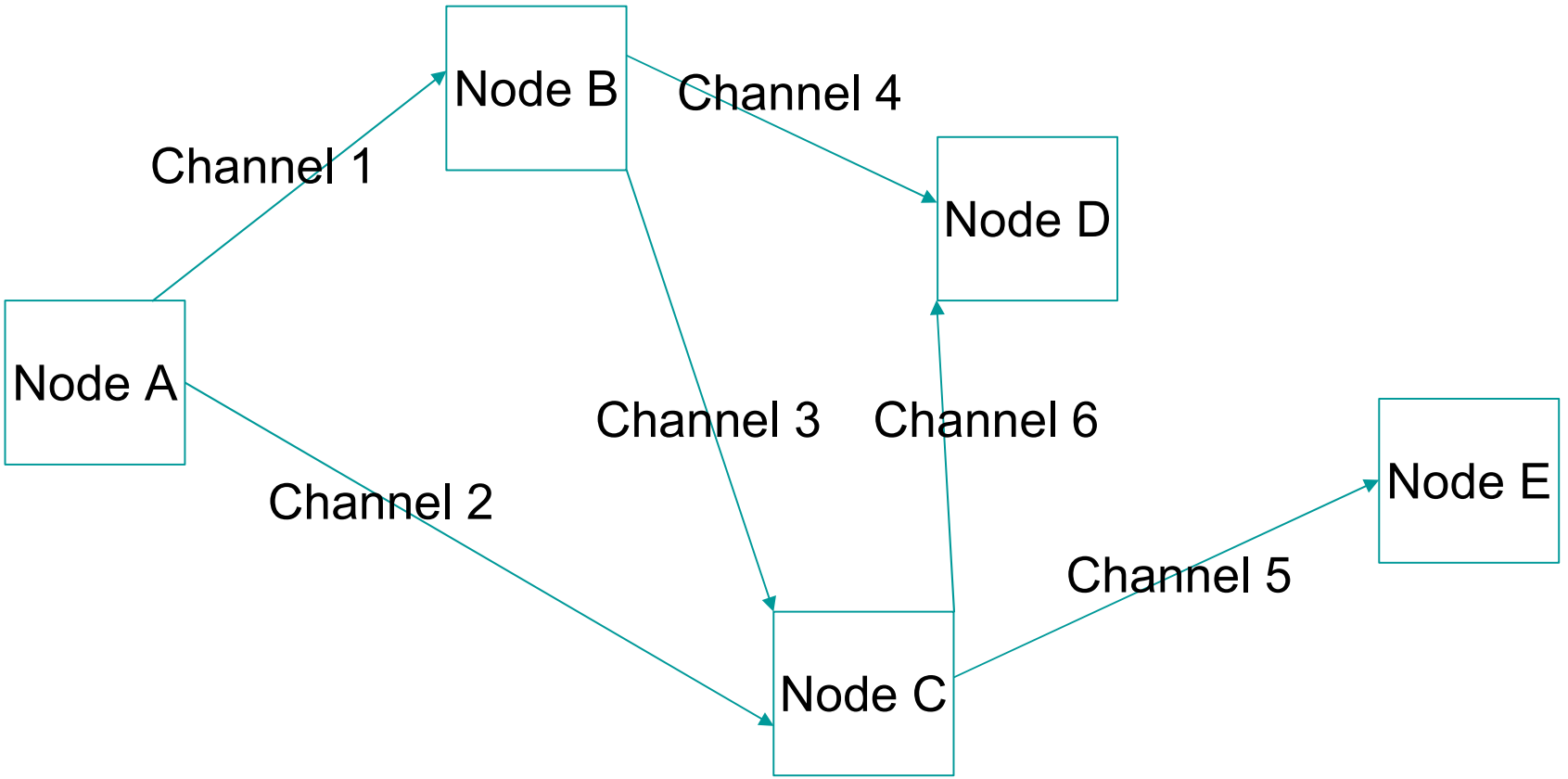
COMMUNICATION BETWEEN NODES

Communication method



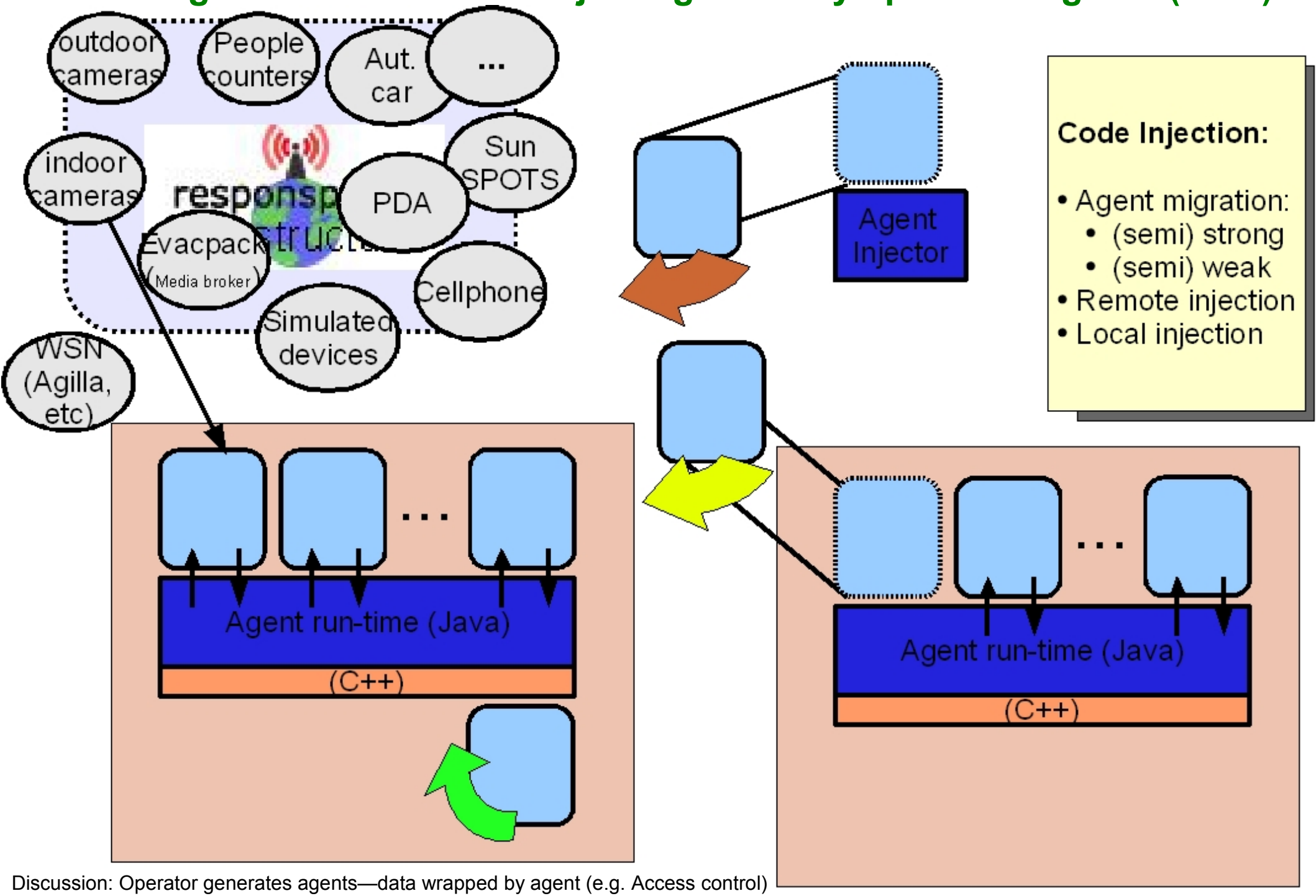
COMMUNICATION BETWEEN NODES

Channels in the network



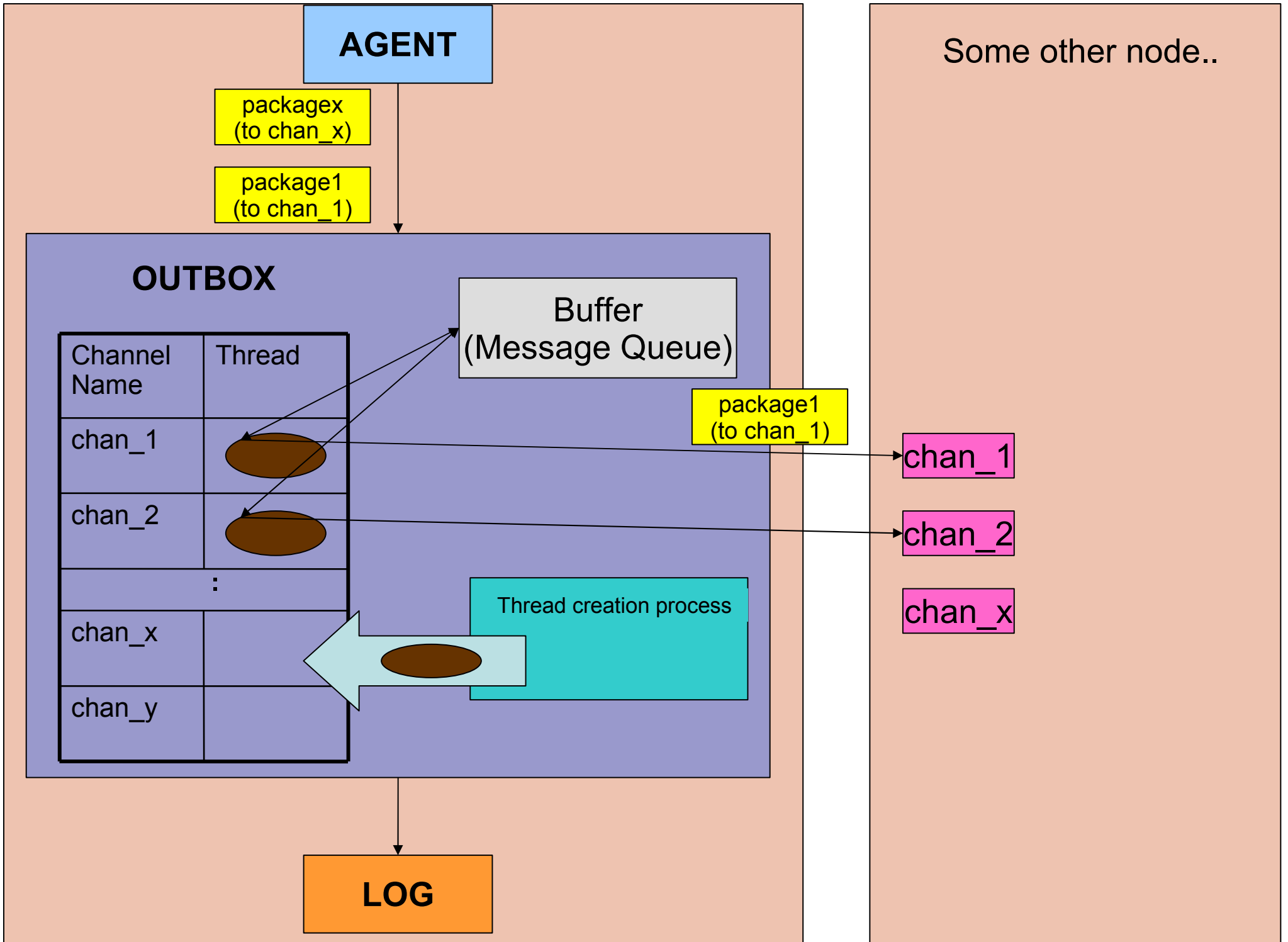
OPERATOR

Mobile agent middleware for injecting arbitrary operators/agents (Java)



Discussion: Operator generates agents—data wrapped by agent (e.g. Access control)

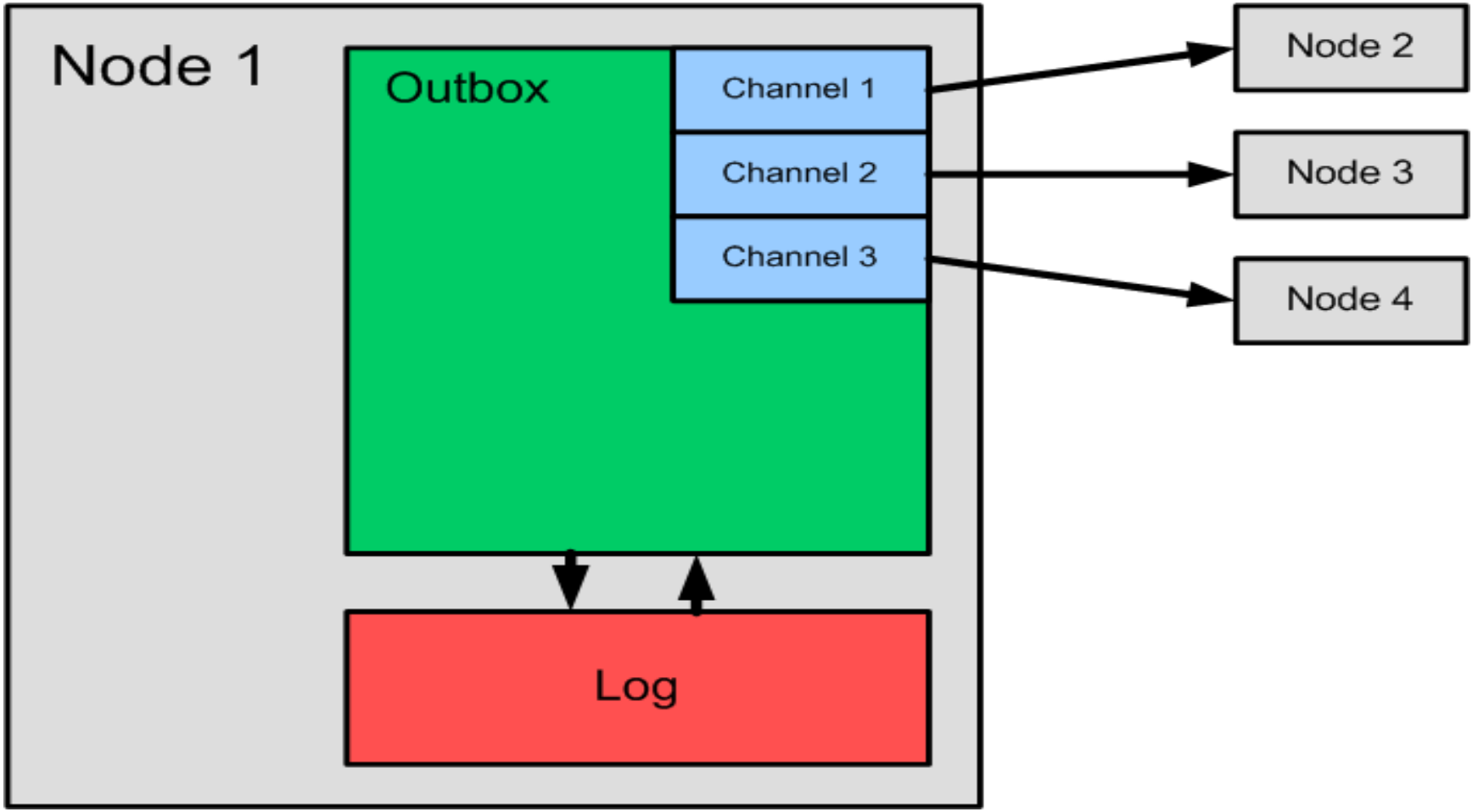
OUTBOX


AGENT
packagex
(to chan_x)
package1
(to chan_1)
OUTBOX
Buffer
(Message Queue)
Channel
Name
Thread
chan_1
chan_2
:
chan_x
chan_y
Thread creation process
package1
(to chan_1)
chan_1
chan_2
chan_x
LOG
Some other node..

OUTBOX
Functions provided in the outbox

- outbox(client*);
- int startUp(char*, char*, int);
- int send(packages*);
- int getStatus();
- static void *run(void *arg);
- void getMessage();
- void printChannel();
- messageQueue
  - packagesInQueue

LOG MODULE


Node 1
Outbox
Channel 1
Channel 2
Channel 3
Log
Node 2
Node 3
Node 4

LOG MODULE
Functionality

- Stores all the messages send by Outbox on external storage
- Retrieves messages from the external storage (during node recovery)
- Can be adapted for Inbox

LOG MODULE
Features

- One log file for all the *channels*
- Each message has a *timestamp* (unique and always increasing)
- *Index* on timestamp – efficient retrieval of messages based on timestamp
- Allows retrieval of:
  - Message with Timestamp greater or equal to a specified Timestamp
  - Next Message
  - Newest Message

DEMO