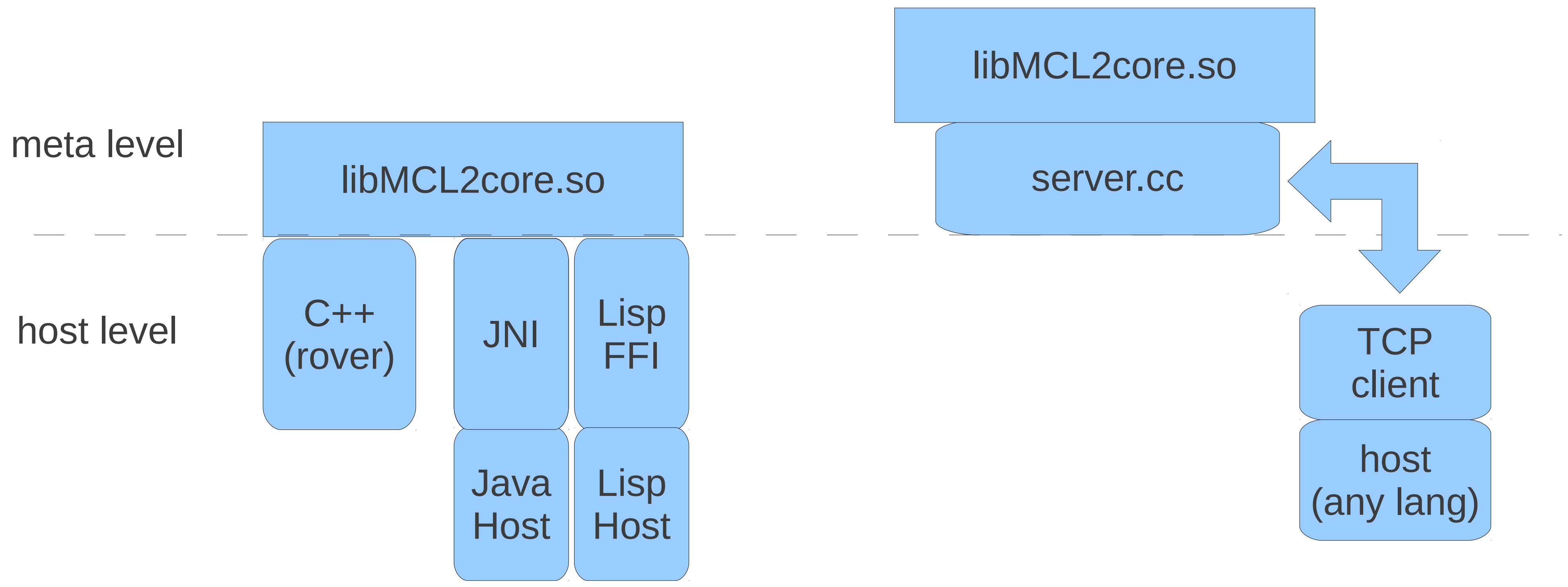
# MCL Crash Course

- APIs & Protocols
- Overview & Operation
- Details
  - Ontologies
  - Frames
  - Re-Entrancy
  - Exceptions & Meta*

# Ways of Integrating

- As a library
  - compiled into a C++ project
  - Using FFI/JNI
- Over TCP
  - build & run server



# Linking MCL

- C++ - just link libMCL2core
- JNI & FFI – you're on your own
  - the API functionality is in
    - /usr/local/include/mcl
    - mcl_multiagent_api.h
    - API functions are documented in api_overview.pdf

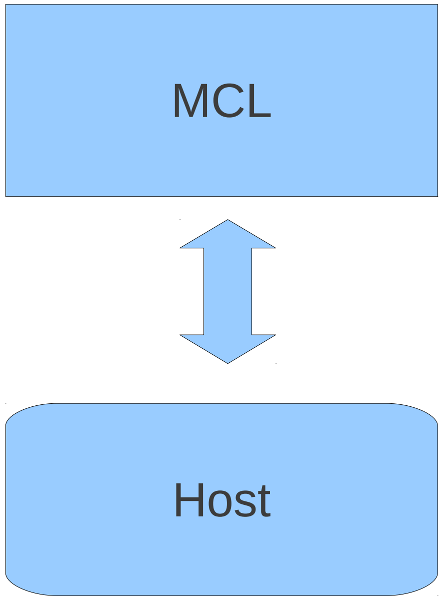
## MCL Over TCP/IP

- All API functions are implemented on a server
- Protocol is clear text
  - > *initializeMCL(“rover”,0)*
  - < *ok(initialized 'rover'.)*
- Implement a client or use java client package (under development)
- text protocol in api_overview.pdf

## Symbols

- Symbols are used to refer to
  - expectation types, responses, etc.
- built automatically during install
  - mcl/utils/symbols.def – can be extended
  - described in api-overview.pdf appendix
- compile down to enum/constants
- can be referred to in TCP/IP where appropriate
  - (use lowercase)

# Overview & Operation: Configuration

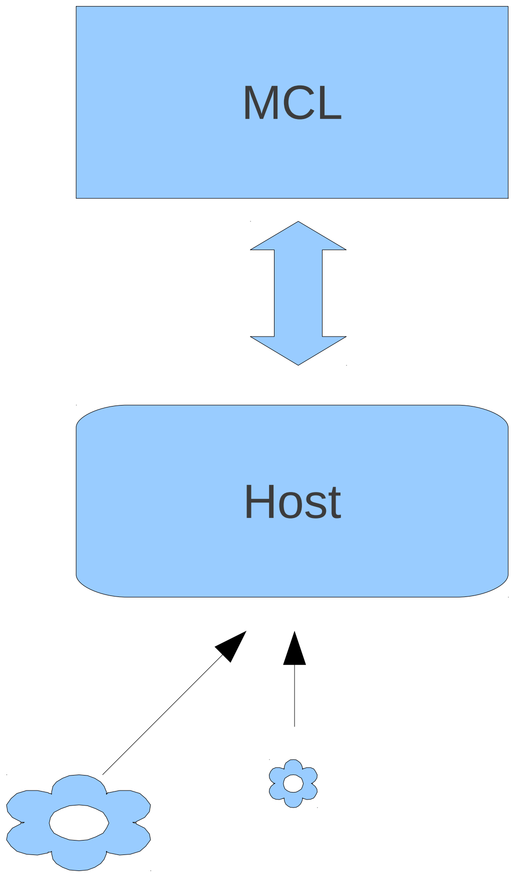


- initializeMCL(k,s)
  - assigns a key and sampling rate to the host
- configureMCL(k,d,a,c)
  - tells MCL where to get config files
- ontology(k,o)
  - tells MCL which ontology to use

## Configuration Files

- CPT Tables, Response Costs
  - $MCL_CONFIG_PATH/config/*domain/agent/controller/*
  - $MCL_CONFIG_PATH/config/default/
  - proprietary, text file formats
- Ontology Definition files
  - $MCL_CONFIG_PATH/netdefs/
  - in ODL format (proprietary, text)
  - see odl-overview.pdf

# Overview & Operation: Host Definition



- host properties
  - go into property vector
  - vector is on a stack
- declare observables
  - set their properties
  - self observables
  - object observables
    - define the type, observables
    - notice & unnotice objects

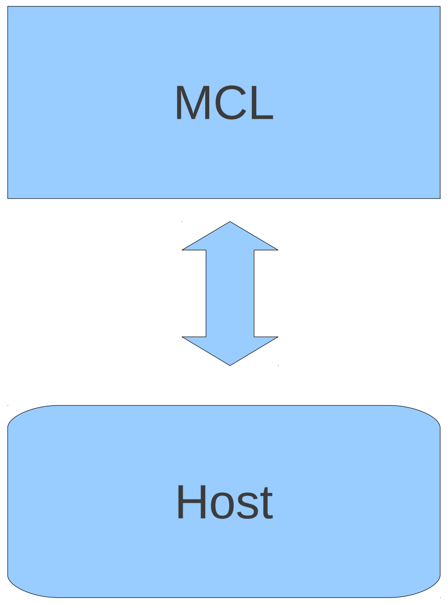
# Sensors & Properties

```
prefix PCI
size MIN_INDEX PCI
psym INTENTIONAL PCI
psym EFFECTORS_CAN_FAIL PCI
psym SENSORS_CAN_FAIL PCI
psym PARAMETERIZED PCI
psym DECLARATIVE PCI
psym RETRAINABLE PCI
psym HLC_CONTROLLING PCI
psym HTN_IN_PLAY PCI
psym PLAN_IN_PLAY PCI
psym ACTION_IN_PLAY PCI
size MAX PCI
def  PCI_COUNT PCI_MAX
```

```
/* data types     */
prefix DT
psym INTEGER DT
psym RATIONAL DT
psym BINARY DT
psym BITFIELD DT
psym SYMBOL DT

/* sensor codes   */
prefix SC
psym STATE    SC
psym CONTROL  SC
psym SPATIAL  SC
psym TEMPORAL SC
psym RESOURCE SC
psym REWARD   SC
psym AMBIENT  SC
psym OBJECTPROP SC
psym MESSAGE  SC
psym COUNTER  SC
psym UNSPEC   SC
size NUMCODES_LEGAL SC
```

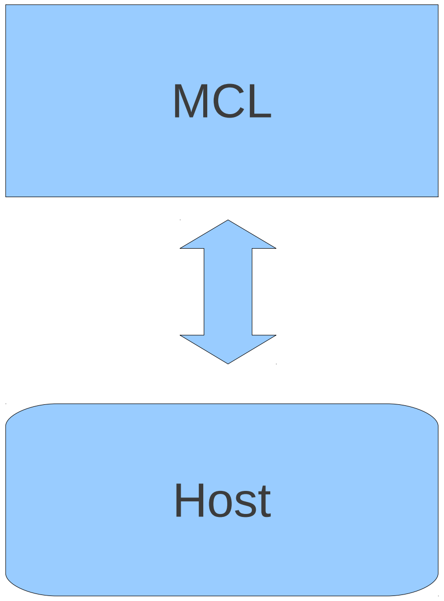
```
/* noise profiles */
prefix MCL_NP
psym NO_PROFILE MCL_NP
psym PERFECT    MCL_NP
psym UNIFORM    MCL_NP
def  MCL_NP_AUTOMATIC 0xFF
def  MCL_NP_DEFAULT MCL_NP_PERFECT

prefix PROP
/* property code indexes */
psym DT PROP
psym SCLASS PROP
psym NOISEPROFILE PROP
size COUNT PROP
def  NUMBER_OF_SENSOR_PROPS PROP_COUN
```

# Overview & Operation: Expectations & Groups



- declareGroup(k,g)
  - Expectation Group organizes expectations together
- declareExpectation(k,g,...)
  - many kinds types of expectations
  - refer to *symbols.def*

# Overview & Operation: Expectations & Groups



when an activity or a process begins, create an expectation group keyed off the activity or process

ground-level expectation groups, hierarchical structure possible

expectations are alive while expectation groups are alive.

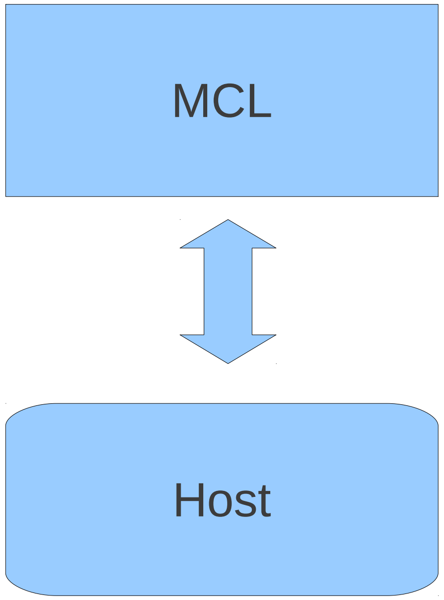
maintenance expectations
    checked continuously

effects expectations
    checked on group complete

# Expectation Types

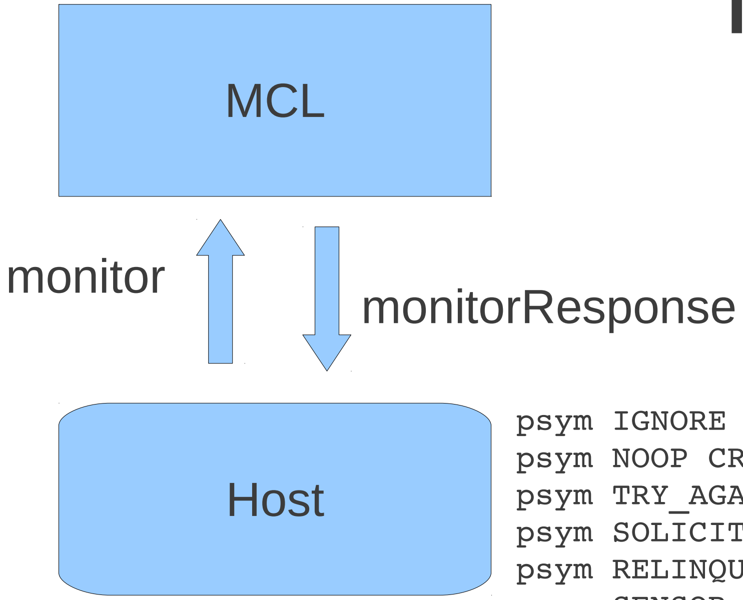
```
prefix EC
/* expectation codes */
def EC_ILLEGAL 0x00    //!< UNUSED expectation code

/* maintenance */
psym STAYUNDER EC      //!< expectation: stay under spec value
psym STAYOVER EC       //!< expectation: stay over spec value
psym MAINTAINVALUE EC  //!< expectation: maintain spec value
psym WITHINNORMAL EC   //!< expectation: stay within range of spec value

psym REALTIME EC       //!< expectation: compl. before realtime deadline
psym TICKTIME EC       //!< expectation: compl. before tick deadline

/* effects     */
psym GO_UP EC          //!< effect: go up from current value
psym GO_DOWN EC        //!< effect: go down from current value
psym NET_ZERO EC       //!< effect: no net change at conclusion
psym ANY_CHANGE EC     //!< effect: any change at all on conclusion
psym NET_RANGE EC      //!< effect: conclude within some range
psym TAKE_VALUE EC     //!< effect: change to specified value

psym DONT_CARE EC      //!< effect: never violate -- don't care expectation
psym BE_LEGAL EC       //!< expectation: always remain legal

psym DONT_CYCLE EC     //!< expectation: do not cycle endlessly
```

# Overview & Operation: monitoring



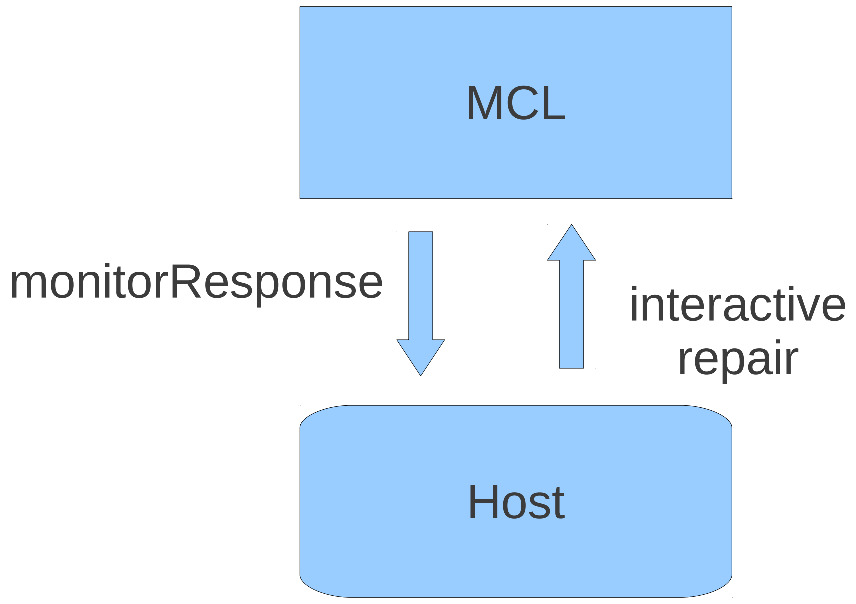
- monitor(k,u)
  - sends an update of sensor values and returns any recommendations
- mclMonitorResponse
- observables::update

# Overview & Operation: monitoring



```
psym IGNORE_CRC                //!< response code: ignore the violation
psym NOOP_CRC                  //!< response code: MCL takes No Operation
psym TRY_AGAIN_CRC             //!< response code: Host try same control block
psym SOLICIT_HELP_CRC          //!< response code: ask for help
psym RELINQUISH_CONTROL_CRC    //!< response code: let user control system
psym SENSOR_DIAG_CRC           //!< response code: run sensor diagnostic
psym EFFECTOR_DIAG_CRC         //!< response code: run effector diagnostic
psym SENSOR_RESET_CRC          //!< response code: reset / repair sensor
psym EFFECTOR_RESET_CRC        //!< response code: reset / repair effector
psym ACTIVATE_LEARNING_CRC     //!< response code: activate/reactivate learning
psym ADJ_PARAMS_CRC            //!< response code: adjust/optimize parameters
psym REBUILD_MODELS_CRC        //!< response code: rebuild underlying models
psym REVISIT_ASSUMPTIONS_CRC //!< response code: revisit control assumptions
psym AMEND_CONTROLLER_CRC      //!< response code: modify/repair control structure
psym REVISE_EXPECTATIONS_CRC //!< response code: revise controller expectations
psym ALG_SWAP_CRC              //!< response code: swap out underlying algorithm
psym CHANGE_HLC_CRC            //!< response code: change high level control goals
psym RESCUE_CRC                //!< response code: engage 'rescue' protocol
psym GIVE_UP_CRC               //!< response code: give up whatever host is doing

psym EXTENDED_CODE_CRC         //!< response code: no code in symbols
```

# Overview & Operation: interaction



- referent allows the host to refer to past anomalies
- functions for informing MCL of how the host is implementing responses
  - suggestionImplemented
  - suggestionIgnored
  - suggestionFailed
  - provideFeedback

## Ontologies

- Defined in ODL files
- Can (must) be extended to better suit hosts
- odl-overview.pdf
- ODL is proprietary, text

## Ontology Def

```
# node types:
#
# hostProp - host properties
# genInd - general purpose indication node
# concInd - concrete (fringe) indication directly activatable by
MCL
# iCore - indication core node
# HII - Host Initiated Indication
# failure - general purpose failure node
# genResponse - general purpose response node
# interactive - boolean interactive response node
# concResponse - concrete (implementable) response node

# link types:
#
# > intraontological
# link abstraction(src=,dst=)
#       - from specific (src) node to more general (dst)
# link IFC(src=,dst=)
#       - Indication Fringe to Core
# link specification(src=,dst=)
#       - from abstract to specific (response ontology base type)
#
# > interontological
# link diagnostic(src=,dst=)
#       - link from indication to failure
# link inhibitory(src=,dst=)
#       - link from indication to response (inhibiting response)
# link support(src=,dst=)
#       - link from indication to response (supporting response)
```

## Frames

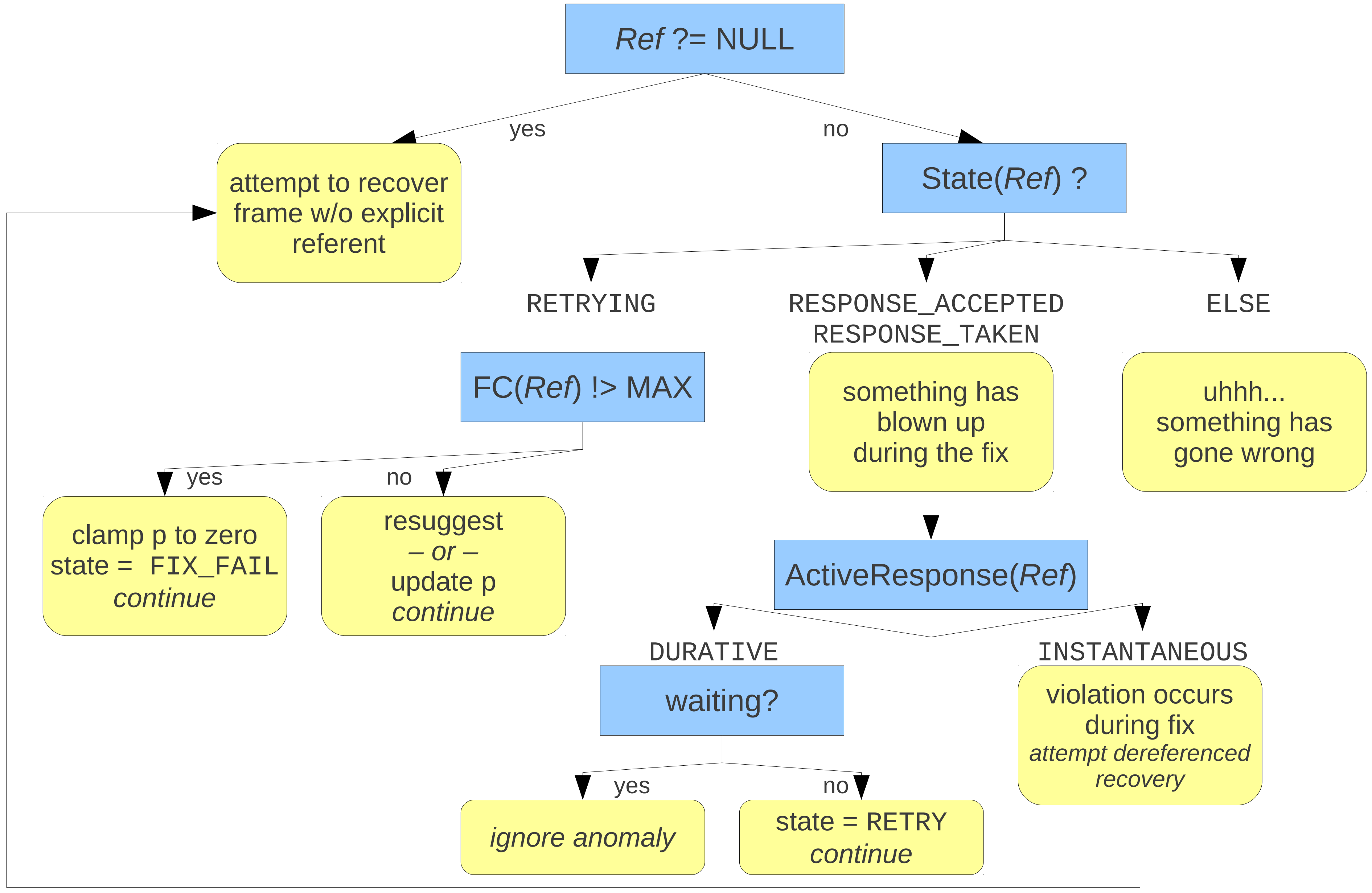
- MCL uses *Frames* to track anomaly processing
- contains
  - a copy of the Bayes Net
  - expectation, indication, Bayes activation traces
- *Re-Entrancy* is the process of comparing new anomalies against existing frames
  - unfinished anomaly
  - recurring anomaly
  - all new anomaly

# Re-Entrancy

- Decision of whether to resume old reasoning or start a new frame
- Complex decision
- Unsolved problem
- Dean's Dissertation

**New Anomaly**

$<EVS,IIS,Eg,\overline{Eg},Ref>$



*Ref* ?= NULL
yes
no
attempt to recover frame w/o explicit referent
State(*Ref*) ?
RETRYING
RESPONSE_ACCEPTED
RESPONSE_TAKEN
ELSE
FC(*Ref*) !> MAX
something has blown up during the fix
uhhh... something has gone wrong
yes
no
clamp p to zero
state = FIX_FAIL
*continue*
resuggest
*– or –*
update p
*continue*
ActiveResponse(*Ref*)
DURATIVE
INSTANTANEOUS
waiting?
violation occurs during fix
*attempt dereferenced recovery*
yes
no
*ignore anomaly*
state = RETRY
*continue*

## More Documents

- deployment.pdf
  - how to integrate with an existing host
- memos in mcl-papers

## Exceptions and Meta*

- MCL is being converted to generate C++ exceptions
- Exceptions can be used to trigger reasoning about MCL+Host
    - example: a host is failing to follow protocol
    - what can MCL do on its own when a host is not utilizing MCL properly? (Meta*)