T-REX usage:

From “simple” user to “advanced” programmer

Frédéric Py fpy@mbari.org

General Presentation and principles

A quick overview on autonomous agents/robots

- Trying to have an entity that can act autonomously (ie with as few human intervention as possible)
- One tension :
  - robots are real time systems (should react fast enough)
  - in general AI programs such as planner are dealing with NP problems (=> too slow to react)
- Two main schools :
  - Reactive approach (~ *try to reproduce a bug brain*) :
    - do not look ahead or look back just react correctly
    - all behaviors are competing to propose action and a policy (generally priority based compose and select the actions)
    - *How to deal with multiple objective ? how to reconfigure dynamically ? how to know if I can do it ? What is the impact of a new behavior in the system ? ...*
  - Layered approaches (~ plan at a high level while react at a low level):
    - To deal planner complexity just abstract the domain and put it on top of a (more) reactive architecture
    - Generally 3 layers :
      - Functional (RT control loops connected to HW),
      - Executive (fast system that refines high level directive into commands),
      - Planner (determine a sequence of directives that *should* allow the system to satisfy a set of objectives/goals)
    - *What if the plan fails to execute ? How to have all these systems interacting seamlessly ? Can I still react without breaking the plan ? If I break the plan will the planner be able to recover ? ...*

A quick overview on autonomous agents/robots

- Trying to have an entity that can act autonomously (ie with as few human intervention as possible)
- One tension :
  - robots are real time systems (should react fast enough)
  - in general AI programs such as planner are dealing with NP problems (=> too slow to react)
- Two main schools :
  - Reactive approach (~ *try to reproduce a bug brain*) :
    - do not look ahead or look back just react correctly
    - all behaviors are competing to provide a solution (priority based compose and select the actions)

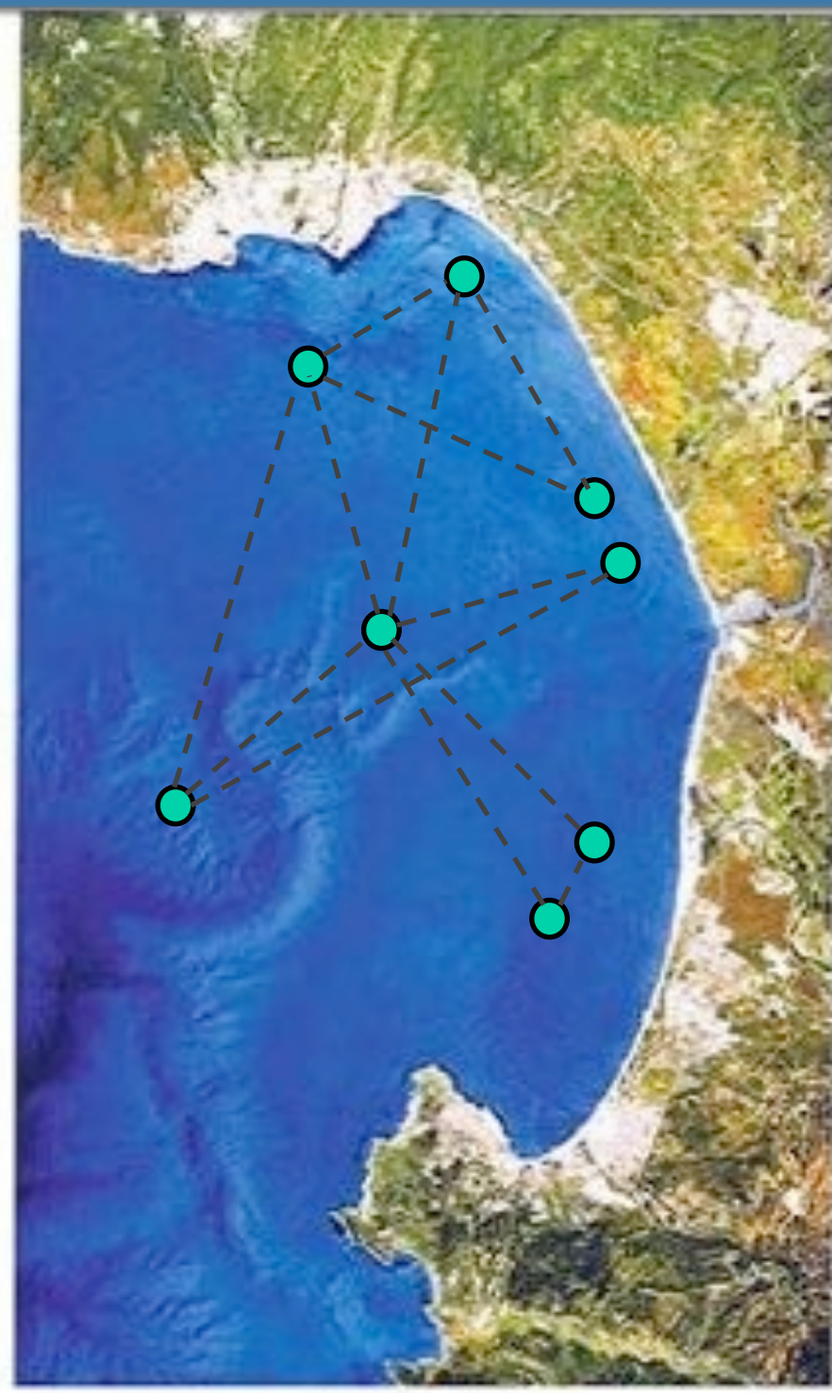
Highly dynamic/unknown environment,
Few simple objective(s)

    - *How to deal with multiple objective ? how to reconfigure dynamically ? how to know if I can do it ? What is the impact of a new behavior in the system ? ...*
  - Layered approaches (~ plan at a high level while react at a low level):
    - To deal planner complexity just abstract the domain and put it on top of a (more) reactive architecture
    - Generally 3 layers :
      - Functional (RT control loops connected to HW),
      - Executive (fast system that refines high level directive into commands),
      - Planner (determine a sequence of directives that *should* allow the system to satisfy a set of objectives/goals)

Relatively static/known environment (or no “real” RT constraints)
Multiple concurrent/complex objectives

    - *What if the plan fails to execute ? How to have all these systems interacting seamlessly ? Can I still react without breaking the plan ? If I break the plan will the planner be able to recover ? ...*

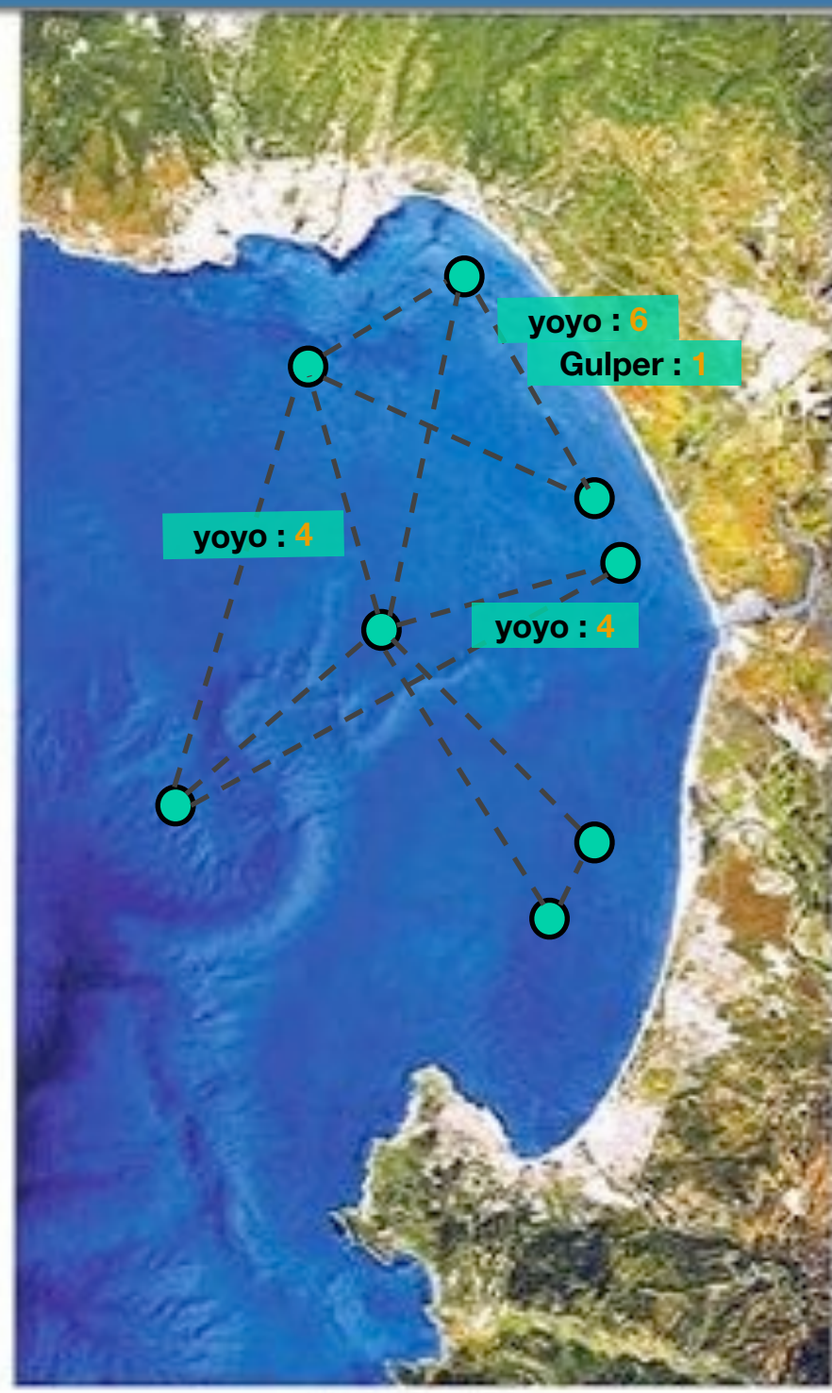
- Science Goal based Mission Planning
- A graph describes all the possible paths for the AUV
- An initial plan is produced onboard to execute the maximum number of goals taking into account operational constraints and science priorities
- Low priority goals may be removed in case of anomalies
- Or when a serendipitous event allows execution of a more critical goal
- Require to be both deliberative and reactive while robust and dependable



An AUV mission

What we had in mind for AUV operation

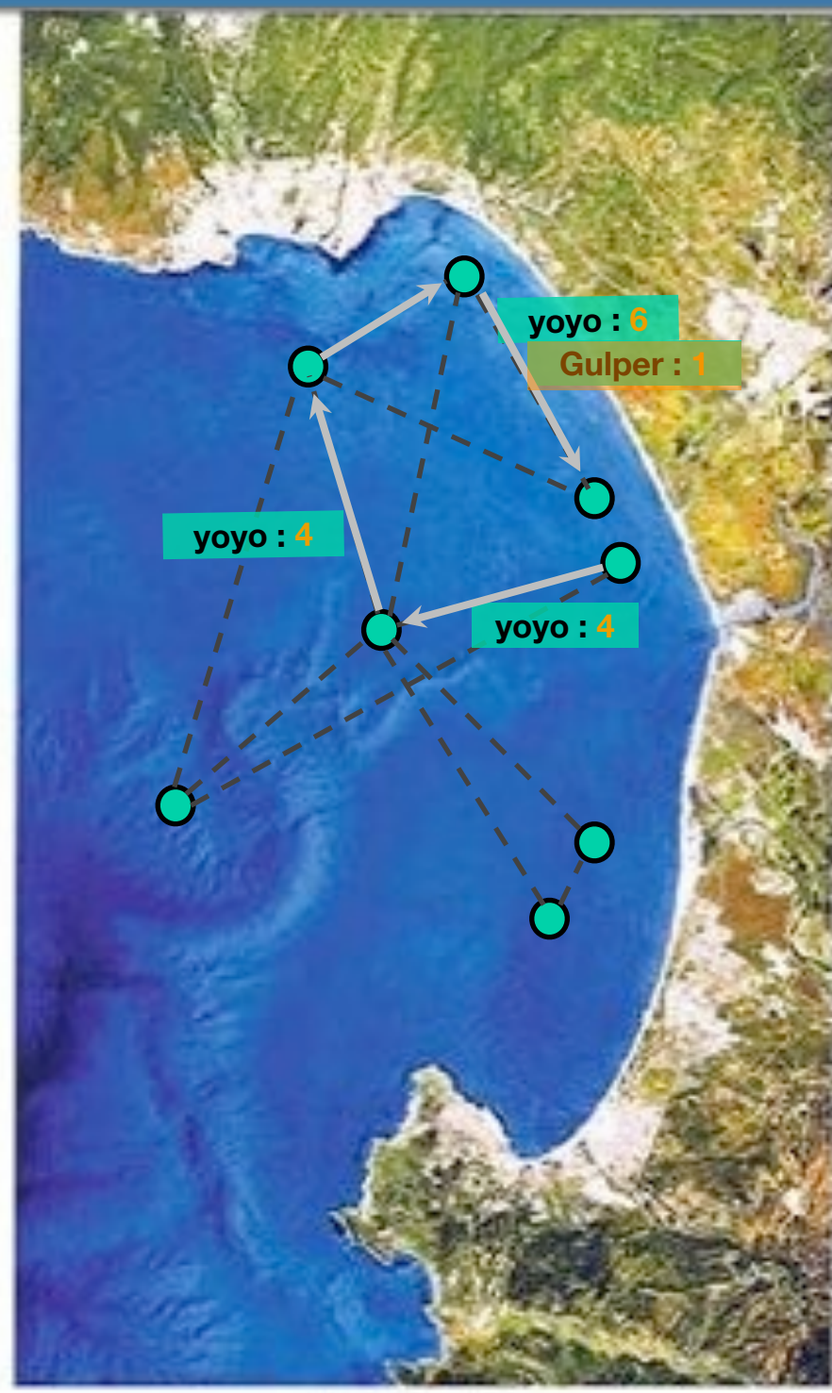
- Science Goal based Mission Planning
- A graph describes all the possible paths for the AUV
- An initial plan is produced onboard to execute the maximum number of goals taking into account operational constraints and science priorities
- Low priority goals may be removed in case of anomalies
- Or when a serendipitous event allows execution of a more critical goal
- Require to be both deliberative and reactive while robust and dependable



An AUV mission

What we had in mind for AUV operation

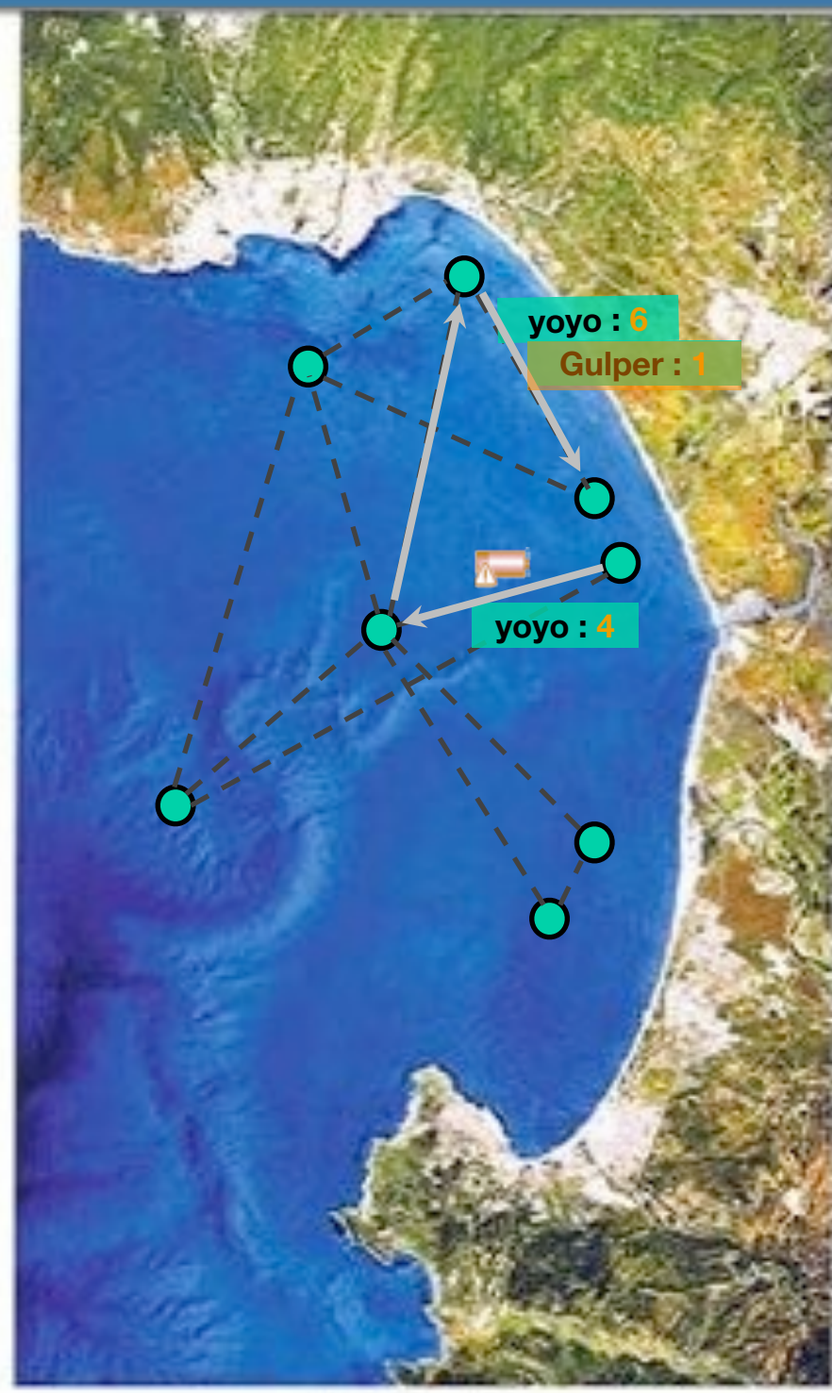
- Science Goal based Mission Planning
- A graph describes all the possible paths for the AUV
- An initial plan is produced onboard to execute the maximum number of goals taking into account operational constraints and science priorities
- Low priority goals may be removed in case of anomalies
- Or when a serendipitous event allows execution of a more critical goal
- Require to be both deliberative and reactive while robust and dependable



An AUV mission

What we had in mind for AUV operation

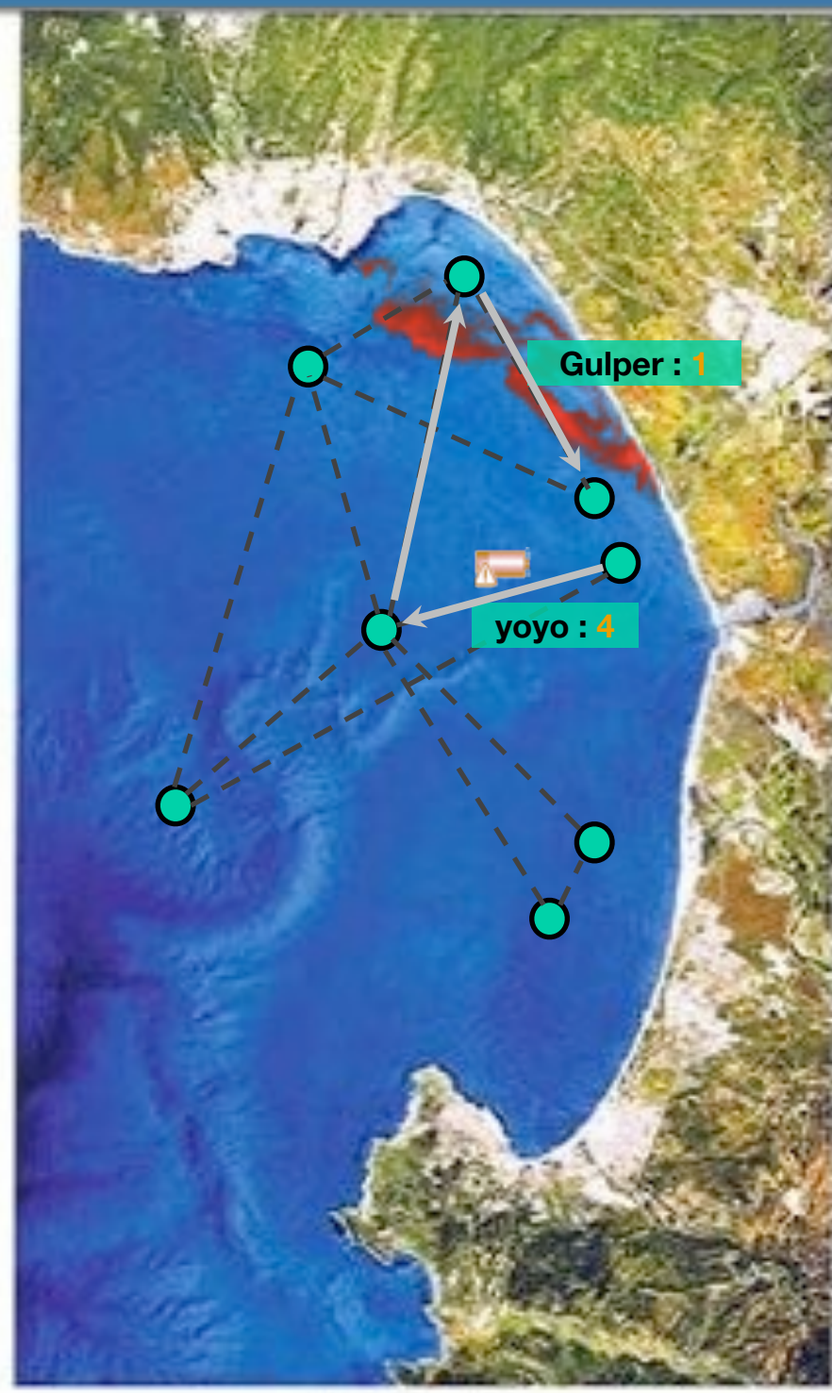
- Science Goal based Mission Planning
- A graph describes all the possible paths for the AUV
- An initial plan is produced onboard to execute the maximum number of goals taking into account operational constraints and science priorities
- Low priority goals may be removed in case of anomalies
- Or when a serendipitous event allows execution of a more critical goal
- Require to be both deliberative and reactive while robust and dependable



An AUV mission

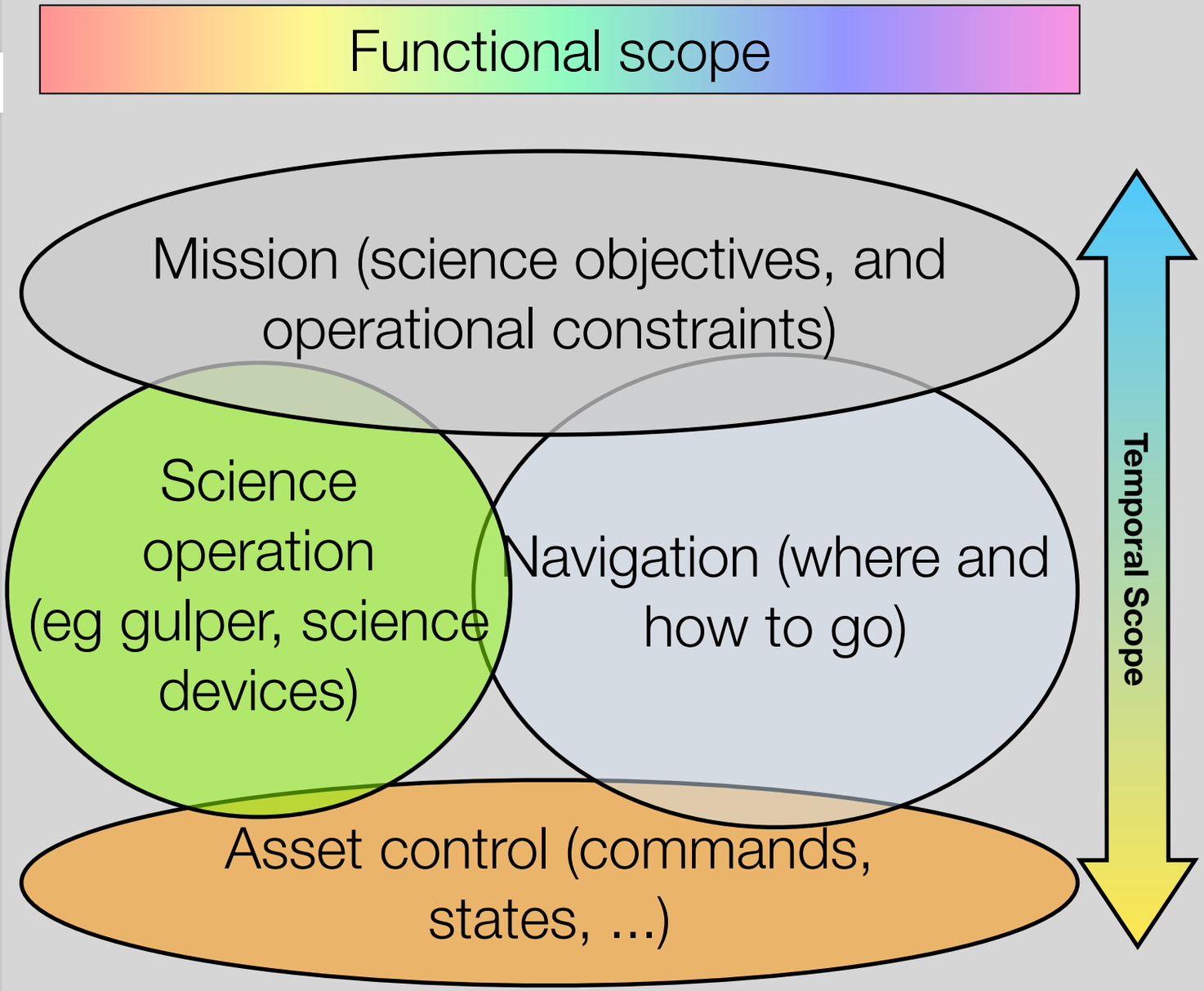
What we had in mind for AUV operation

- Science Goal based Mission Planning
- A graph describes all the possible paths for the AUV
- An initial plan is produced onboard to execute the maximum number of goals taking into account operational constraints and science priorities
- Low priority goals may be removed in case of anomalies
- Or when a serendipitous event allows execution of a more critical goal
- Require to be both deliberative and reactive while robust and dependable



An AUV mission

What we had in mind for AUV operation

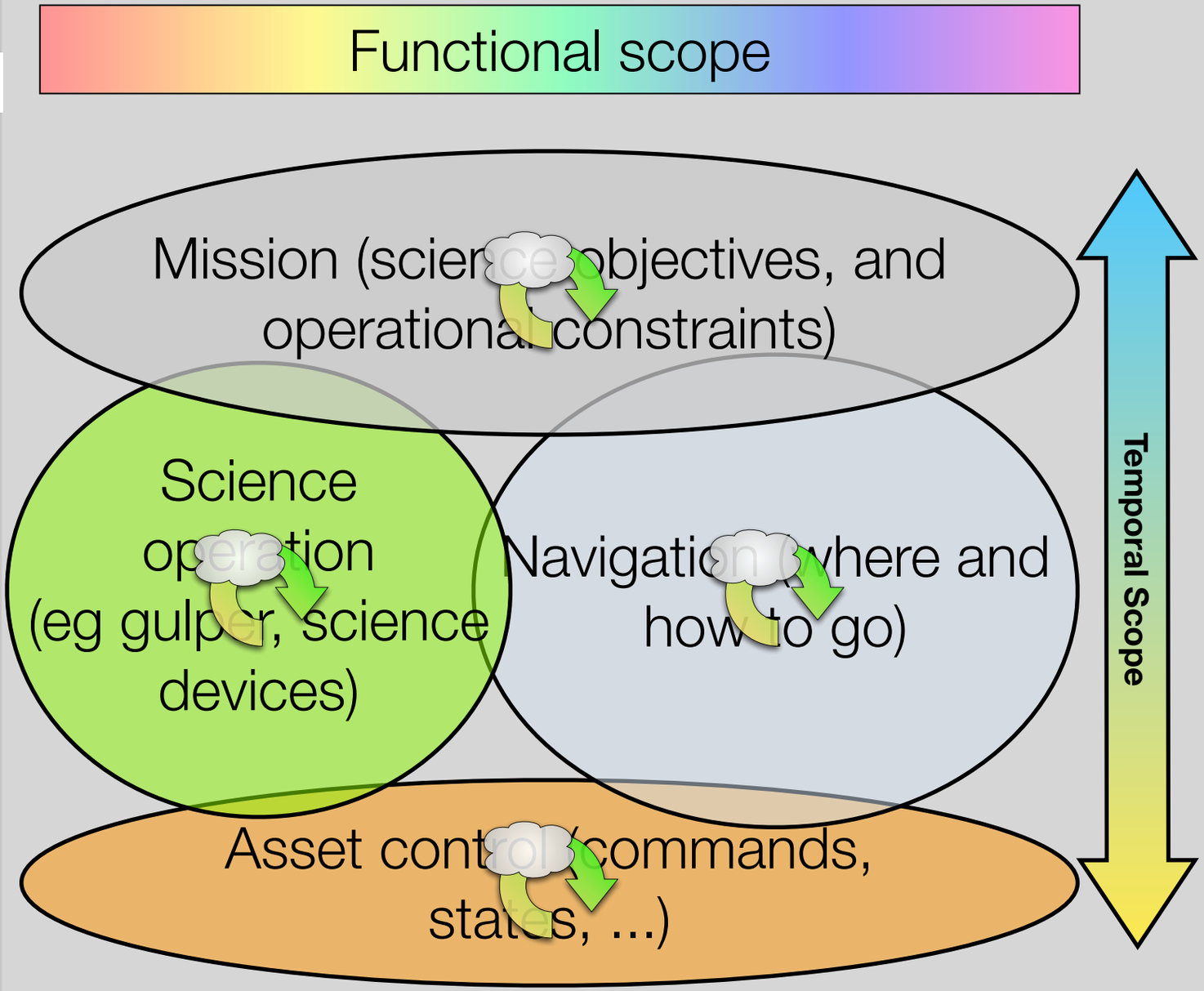


Each entity :

- has a level of decision responsibility based on its functional and temporal scope.
- do not deliberate at the same level
- overlaps with the others reflecting an interaction of decisions of both
- will have requirement on how fast it needs to deliberate
- possible analogy: ROV operation

Mission operation

An illustrative example on how different levels of control overlap



Each entity :

- has a level of decision responsibility based on its functional and temporal scope.
- do not deliberate at the same level
- overlaps with the others reflecting an interaction of decisions of both
- will have requirement on how fast it needs to deliberate
- possible analogy: ROV operation

All of them are sense/plan/act loops interacting

Mission operation

An illustrative example on how different levels of control overlap

What is TREX purpose

- Provide a Framework that :
  - Manage the execution of these control loops (reactor/adapter/...)
    - Try to ensure their look ahead and latency for reaction
  - Provide a unified interface to exchange goals and observations between reactors as desired and observed states
    - compliant with Europa timeline representation of state variables
    - strict ownership (on *timeline* is controlled by one and only one reactor)
    - hierarchy (there's no cyclic dependency between reactors)
  - Support a model based design where most reactors rely on a subpart of a global model (DeliberativeReactor based on Europa)
  - Is open to new reactor implementation
  - Control the granularity of time (tick) which is common to all reactors.
  - Ensure that all reactors are up to data at the tick rate.
- There are some assumption made to reduce cost :
  - Past is monotonic: observation made at a given time won't be changed
  - Inertial value: if no observation is given the last one still holds

TREX usage at MBARI

Set your environment properly

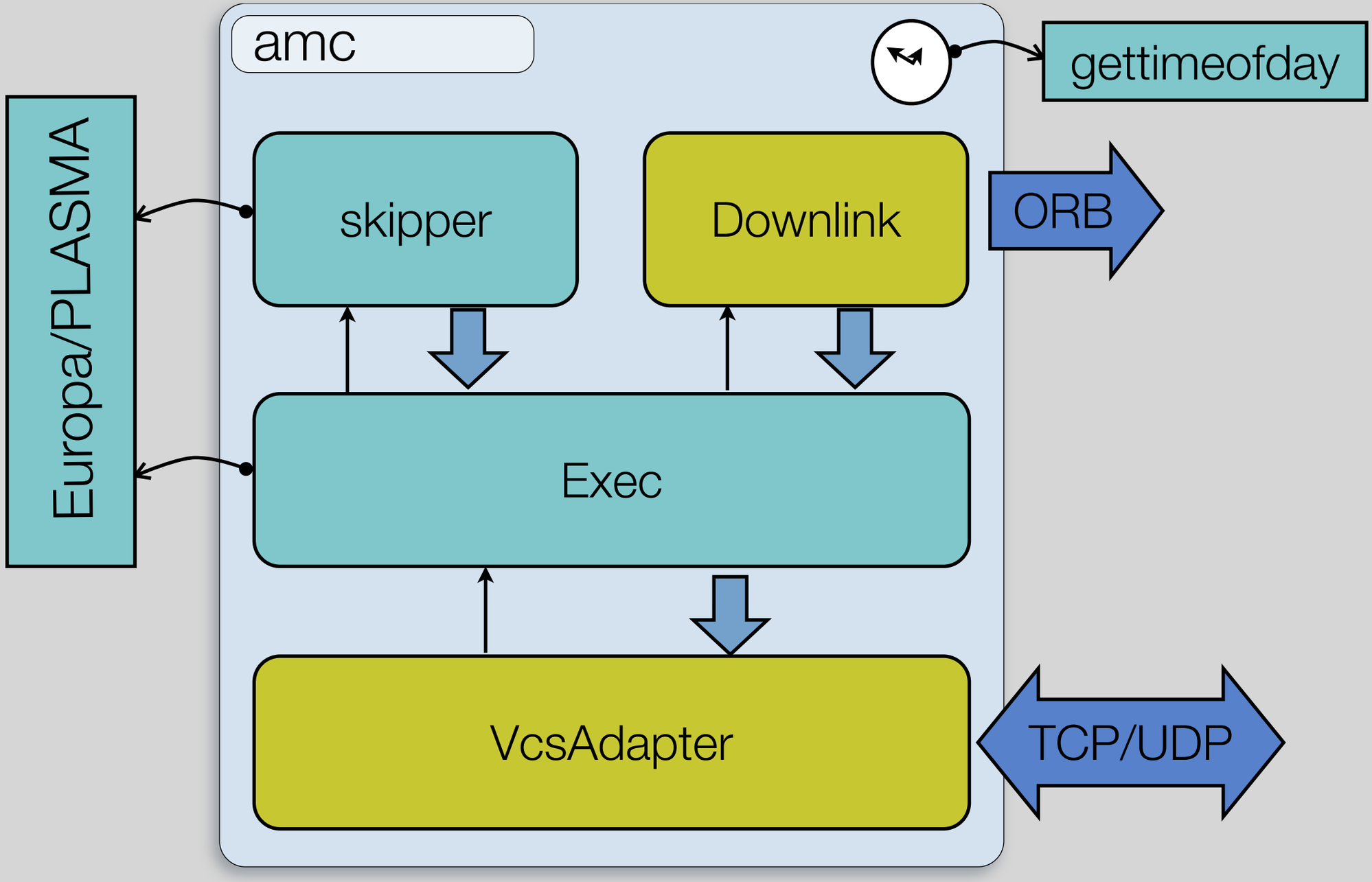
- Detailed on the wiki (https://oceana.mbari.org/confluence/display/AUV/TREX+-+installation)
  - Environment scripts are `bash`
  - to do it punctually :
    - `cd <path-to-TREX>/TREX`
    - `source devConfig`
  - or on your `~/.bashrc` :

```
ROOT_DIR=/home/dorado1/coding/TREX

if [ -f $ROOT_DIR/nightly/config/$HOSTNAME.env ]; then

   source $ROOT_DIR/nightly/config/$HOSTNAME.env

else

   source $ROOT_DIR/nightly/config/default.env

fi
```

- define `$HOSTNAME.env` based on the specifics of this machine (see `threadfish.shore.mbari.org` for example)
- `$ROOT_DIR` is an important variable and should be used to define paths dependent to it (`auv`, `auv-linux`,...) on the `$HOSTNAME.env` file. It is for example used for nightly builds on threadfish

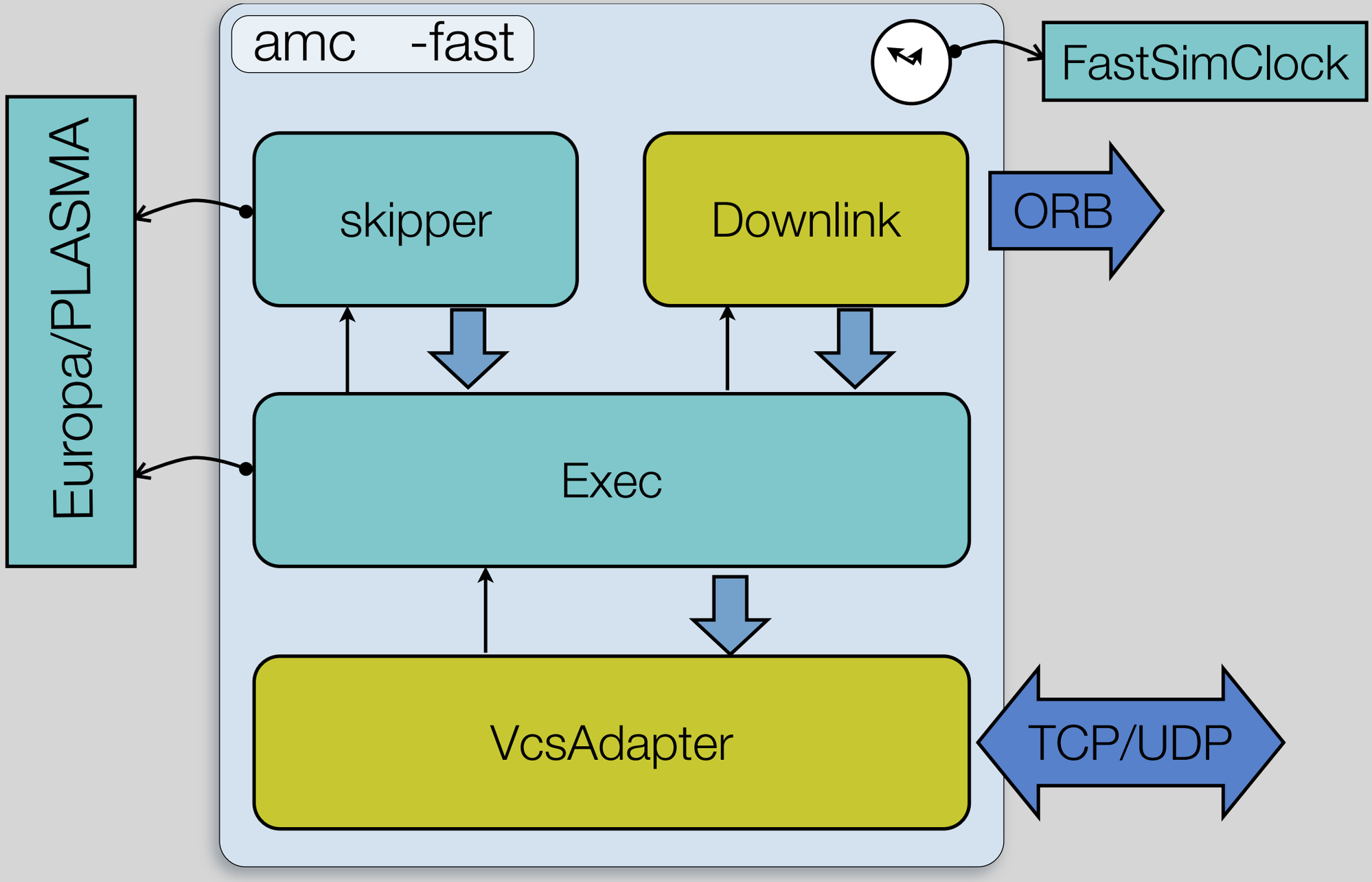
The set of commands (ctd2007)

- We have so far 3 different commands for running TREX serving different purpose :
  - amc : batch runner used mostly for operation run the mission until the agent “dies” (hopefully because it reached his end tick)
    - `amc_o_rt <mission>.cfg [-sim|-fast] [<nsteps>]`
  - sim : simple debugging interface that allows to execute the agent until a specific tick, step to next tick, activate/deactivate extra log messages
    - `sim_o_rt <mission> [-fast] [<nsteps>]`
  - batcher/samp : used to exercise the system with multliple random alteration of the execution (eg change the temperature, the error rate in position while underwater, ...)
    - `batcher_o_rt <config_file>`
- We have also few post-processing (`offboard`) tools to ease the log analysis :
  - scripts : `volGet.sh [<path>]`, `planHist.sh [<path>]`, `cleanLogs.sh`
  - matlab : `plotNav('<path>')`



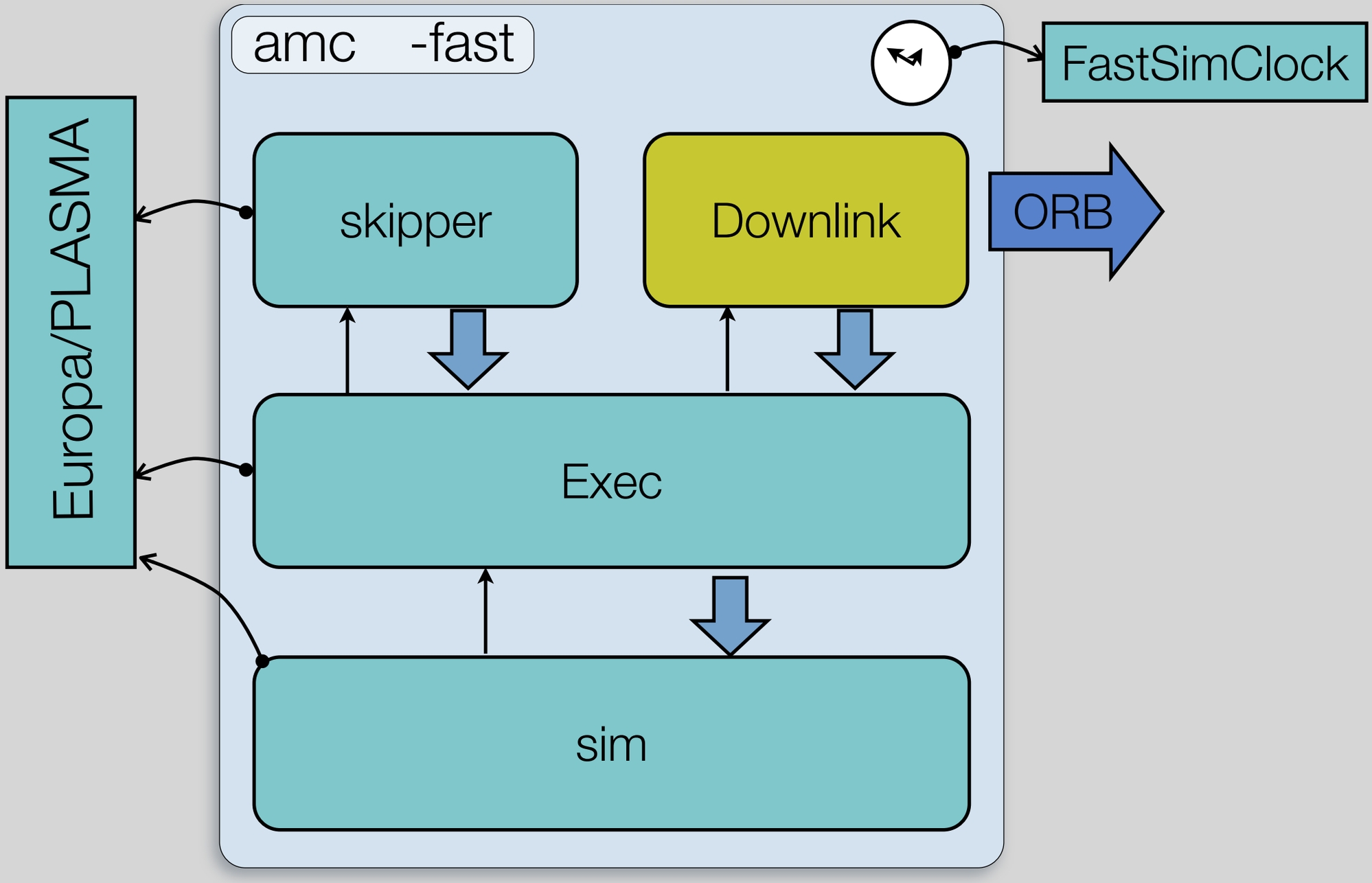
amc.vcs.dl.cfg

TREX “ctd2007” agent



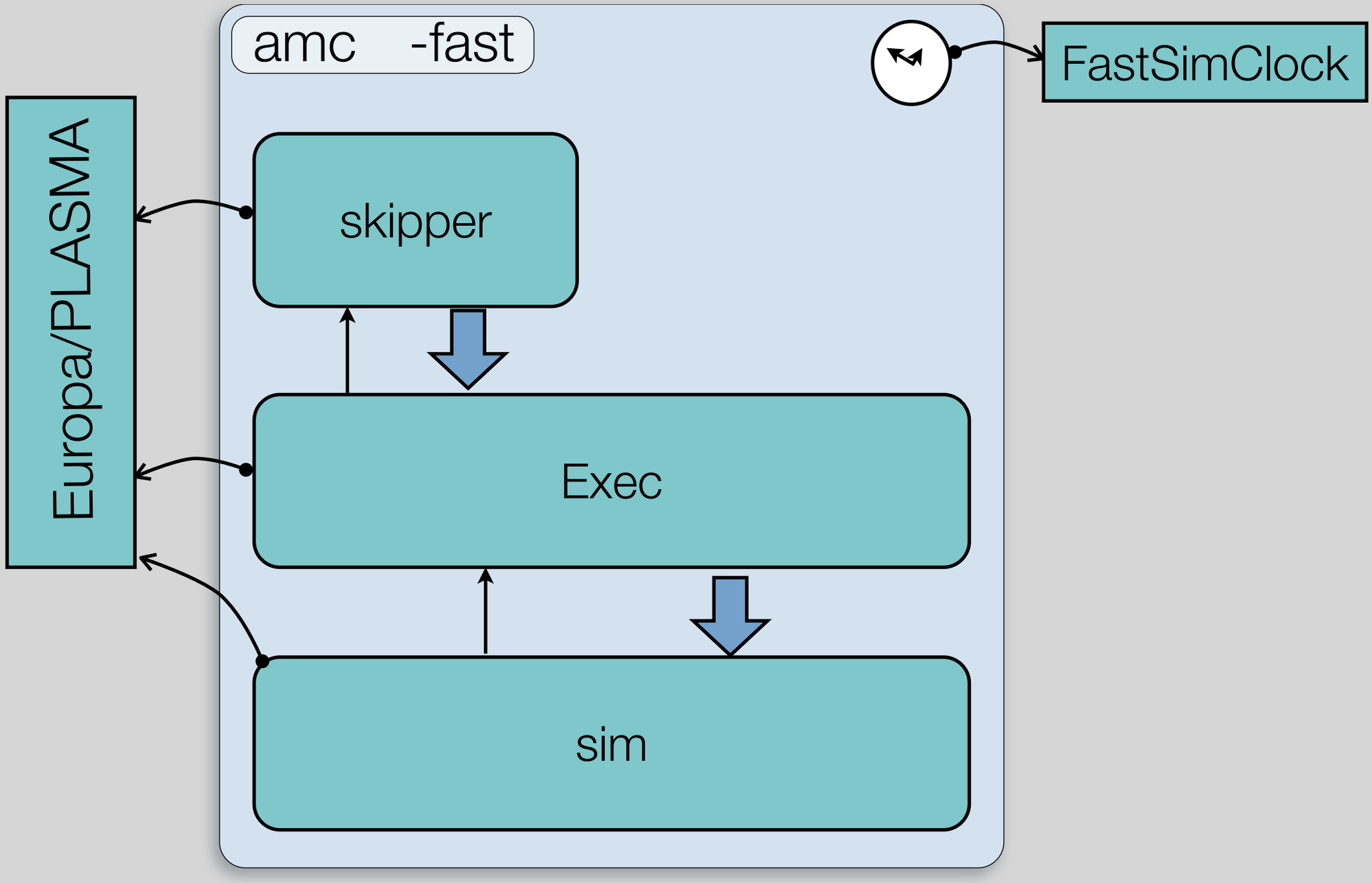
amc.vcs.dl.cfg

TREX “ctd2007” agent



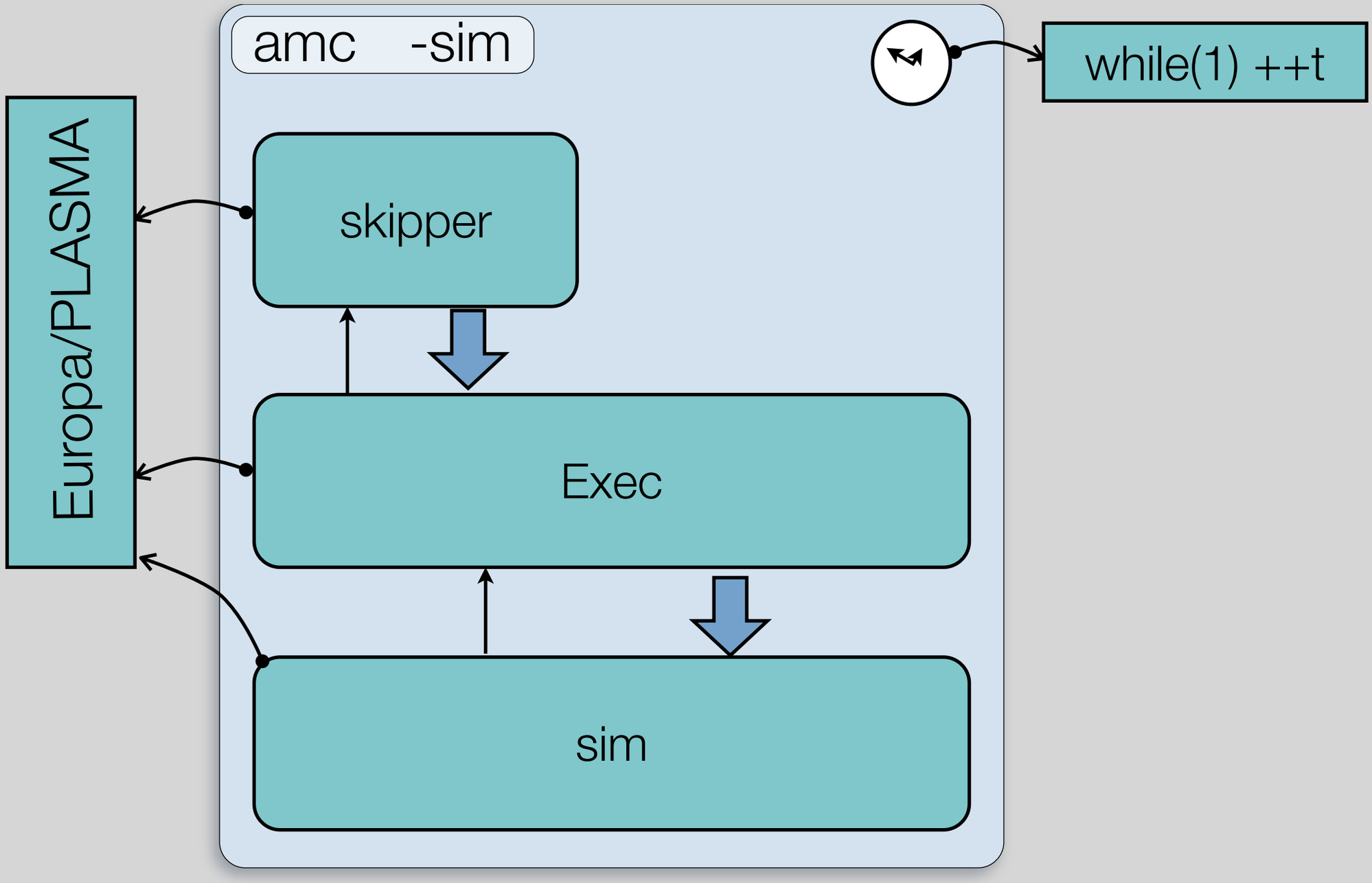
amc.sim.dl.cfg

TREX “ctd2007” agent



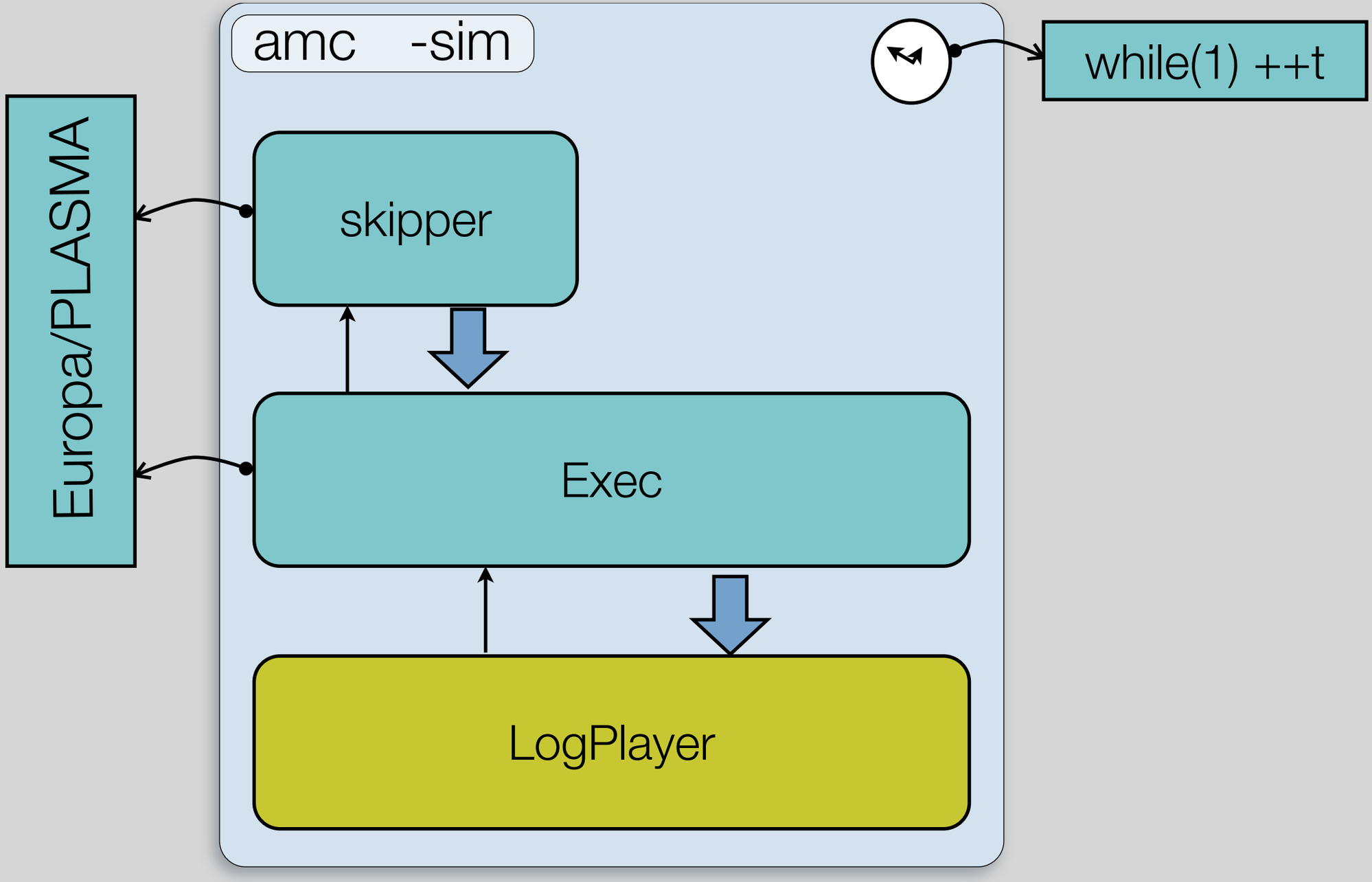
amc.sim.cfg

TREX “ctd2007” agent



amc.sim.cfg

TREX “ctd2007” agent



amc.play.cfg

TREX “ctd2007” agent

TREX logs (`$TREX_LOG_DIR/latest`)

Many logs are produced by TREX :

- `TREX.log` : general log file indicating overall reactors activity. You'll find here different message produced by reactors or over TREX components (such as the clock in some cases). Examples are
  - observations produced by reactors
  - goal requested from a reactor to another
  - Inconsistent/database relax/failed to plan : bad most of the time

    In general it is good to keep an eye on this file just to check quickly agent activity. (`tail -F latest/TREX.log`)
- `Debug.log` : debug messages produced by TREX and filtered by `Debug.cfg` they are generally activated while debugging a mission execution to have more hints on the cause of the failure.
- `<mission>.log` : logs event produced by some reactors to be able to replay them so you can replay exactly the mission for debugging
- `<mission>.<reactor>.plan` : log the plans produced by the reactor.
  - `sim` produce more detailed plan than `amc` and `batcher` do not produce these files
- `cfg/` : will eventually contain all the config files used for this run
- `cpuStat.log` : log some data about cpu time spent per tick but not accurate right now

The configuration files (ctd2007)

- Debug.cfg : what debug messages to display on Debug.log

  ./dbg+ and ./dbg- set two default configurations described on Debug.On.cfg and Debug.Off.cfg

  Generally used with sim_o_rt as debug messages can be *very* verbose

- <mission>.cfg : The mission configuration root file for example front.cfg

```
<Agent name="front" finalTick="25200"  config="amc.cfg"/>
```

- amc*.cfg : The description of the reactors to load. default is amc.cfg that uses some sort of home cooked AUV simulator inside TREX

```
<Config>
    <!-- Use this component when running the pseudo-sim -->
    <TeleoReactor log="1" name="sim" component="DeliberativeReactor"
                  lookAhead="1" latency="0" solverConfig="sim.solver.cfg"/>

    <TeleoReactor name="exec" component="DeliberativeReactor"
                  lookAhead="5" latency="1" solverConfig="exec.solver.cfg"/>

    <!-- Skipper has 10 Hour lookahead. -->
    <TeleoReactor name="skipper" component="DeliberativeReactor"
                  lookAhead="36000" latency="5"
                  solverConfig="skipper.solver.cfg"/>
</Config>
```

- From here it is reactor dependent but for DeliberativeReactor you have usually a <mission>.<reactor>.nddl that gives the model and initial objectives of this reactor

The configuration files (ctd2007)

- Debug.cfg : what debug messages to display on Debug.log
  ./dbg+ and ./dbg- set two default configurations described on Debug.On.cfg and Debug.Off.cfg
  Generally used with sim_o_rt as debug messages can be *very* verbose
- <mission>.cfg : The mission configuration root file for example front.cfg

Name of the mission

lifetime

config used

```
<Agent name="front" finalTick="25200"  config="amc.cfg"/>
```

- amc*.cfg : The description of the reactors to load. default is amc.cfg that uses some sort of home cooked AUV simulator inside TREX

```
<Config>
    <!-- Use this component when running the pseudo-sim -->
    <TeleoReactor log="1" name="sim" component="DeliberativeReactor"
                  lookAhead="1" latency="0" solverConfig="sim.solver.cfg"/>

    <TeleoReactor name="exec" component="DeliberativeReactor"
                  lookAhead="5" latency="1" solverConfig="exec.solver.cfg"/>

    <!-- Skipper has 10 Hour lookahead. -->
    <TeleoReactor name="skipper" component="DeliberativeReactor"
                  lookAhead="36000" latency="5"
                  solverConfig="skipper.solver.cfg"/>
</Config>
```

- From here it is reactor dependent but for DeliberativeReactor you have usually a <mission>.<reactor>.nddl that gives the model and initial objectives of this reactor

The configuration files (ctd2007)

- `Debug.cfg` : what debug messages to display on `Debug.log`

  `./dbg+` and `./dbg-` set two default configurations described on `Debug.On.cfg` and `Debug.Off.cfg`

  Generally used with sim_o_rt as debug messages can be *very* verbose

- `<mission>.cfg` : The mission configuration root file for example front.cfg

```
<Agent name="front" finalTick="25200"  config="amc.cfg"/>
```

- `amc*.cfg` : The description of the reactors to load, default is amc.cfg that uses some sort of home cooked AUV simulator inside TREX

log for replay ?

name

reactor type

look ahead for plans

latency for planning

```
<Config>
    <!-- Use this component when running the pseudo-sim -->
    <TeleoReactor log="1" name="sim" component="DeliberativeReactor"
                  lookAhead="1" latency="0" solverConfig="sim.solver.cfg"/>

    <TeleoReactor name="exec" component="DeliberativeReactor"
                  lookAhead="5" latency="1" solverConfig="exec.solver.cfg"/>

    <!-- Skipper has 10 Hour lookahead. -->
    <TeleoReactor name="skipper" component="DeliberativeReactor"
                  lookAhead="36000" latency="5"
                  solverConfig="skipper.solver.cfg"/>
</Config>
```

- From here it is reactor dependent but for `DeliberativeReactor` you have usually a `<mission>.<reactor>.nddl` that gives the model and initial objectives of this reactor

The configuration files (ctd2007)

- `Debug.cfg` : what debug messages to display on `Debug.log`

  `./dbg+` and `./dbg-` set two default configurations described on `Debug.On.cfg` and `Debug.Off.cfg`

  Generally used with sim_o_rt as debug messages can be *very* verbose

- `<mission>.cfg` : The mission configuration root file for example front.cfg

```
<Agent name="front" finalTick="25200"  config="amc.cfg"/>
```

- `amc*.cfg` : The description of the reactors to load, default is amc.cfg that uses some sort of home cooked AUV simulator inside TREX

log for replay ?

name

reactor type

```
<Config>
    <!-- Use this component when running the pseudo-sim -->
    <TeleoReactor log="1" name="sim" component="DeliberativeReactor"
                  lookAhead="1" latency="0" solverConfig="sim.solver.cfg"/>

    <TeleoReactor name="exec" component="DeliberativeReactor"
                  lookAhead="5" latency="1" solverConfig="exec.solver.cfg"/>

    <!-- Skipper has 10 Hour lookahead. -->
    <TeleoReactor name="skipper" component="DeliberativeReactor"
                  lookAhead="36000" latency="5"
                  solverConfig="skipper.solver.cfg"/>
</Config>
```

look ahead for plans

latency for planning

Europa specific cfg

- From here it is reactor dependent but for `DeliberativeReactor` you have usually a `<mission>.<reactor>.nddl` that gives the model and initial objectives of this reactor

The configuration files (ctd2007)

User generally do not mess with all of that and generally just assumes that `<mission>.cfg` is correctly set. For example :

- to follow a front I will use `front.cfg`
- to do nothing I will use `nothing.cfg`
- etc

Can also set `amc.cfg` to the correct cfg file :

```
cp amc.vcs.cfg amc.cfg          # AUV
cp amc.vcs.dl.cfg amc.cfg       # AUV with iridium connection
cp amc.sim.cfg amc.cfg          # do not connect to the AUV
```

Can even look at the top reactor nddl file (I have no responsibilities starting from there). For example `front.skipper.nddl` :

```
#include "front.nddl"
#include "ctd.skipper.nddl"

close();

rejectable(missionGoals.FrontFollow goal_0);
goal_0.priority = 0;
goal_0.a = node_a;              // lat/lon defined in front.nddl
goal_0.b = node_b;              // lat/lon defined  in front.nddl
goal_0.minDepth = 4.0;          // min depth of the yoyo
goal_0.maxDepth = 35.0;         // max depth of the yoyo
goal_0.minAltitude = 15.0;      // min altitude of the yoyo
goal_0.speed = 1.75;            // speed in m/s
goal_0.halfLength = 1000.0;     // map 1km before and after front centroid
goal_0.samplingRate = 5.0;      // collect temperature each 5m depth
```

How to deploy TREX on the host

- Example for `trex2` (on AUV CTD) from `threadfish` :

```
[dorado1@threadfish ~]$ cd $TREX_HOME
[dorado1@threadfish TREX]$ jam amc # we just need the batch runner on the AUV
   [...]
[dorado1@threadfish TREX]$ export AMC_TARGET=trex2 # indicate target for scp
[dorado1@threadfish TREX]$ ./deploy/deploy.exec    # deploy binary files
   [...]
gzip: exec.amc.tar.gz already exists; do you wish to overwrite (y or n)? y
dorado1@trex2's password:
Deployed exec.amc.tar.gz to trex2:./amc
[dorado1@threadfish TREX]$ ./deploy/deploy.mission # deploy configuration files
   [...]
gzip: missions.amc.tar.gz already exists; do you wish to overwrite (y or n)? y
Deployed missions.amc.tar.gz to trex2:./amc
[dorado1@threadfish TREX]
```

- Then from `trex2` :

```
[dorado1@trex2 ~]$ cd amc
[dorado1@trex2 amc]$ source config.amc # load environment variables
[dorado1@trex2 amc]$ ./install.amc all # install deployed archives
   [...]
[dorado1@trex2 amc]$ cd missions       # where the configuration files are
[dorado1@trex2 missions]$ cp amc.vcs.new.cfg amc.cfg # when connecting to the AUV
[dorado1@trex2 missions]$ nohup ../exec/amc_o_rt front.cfg & # run “front” mission
```

Terminal — bash — 167×20

```
mbari1224:~ fpy$ cd $TREX_LOG_DIR
mbari1224:log fpy$ tail -F latest/TREX.log
```

Keep an eye on the logs

Terminal — bash — 102×10

```
mbari1224:~ fpy$ cd $TREX_HOME
mbari1224:TREX fpy$ cd ctd2007/
mbari1224:ctd2007 fpy$ amc_o_rt front.cfg -sim &
```

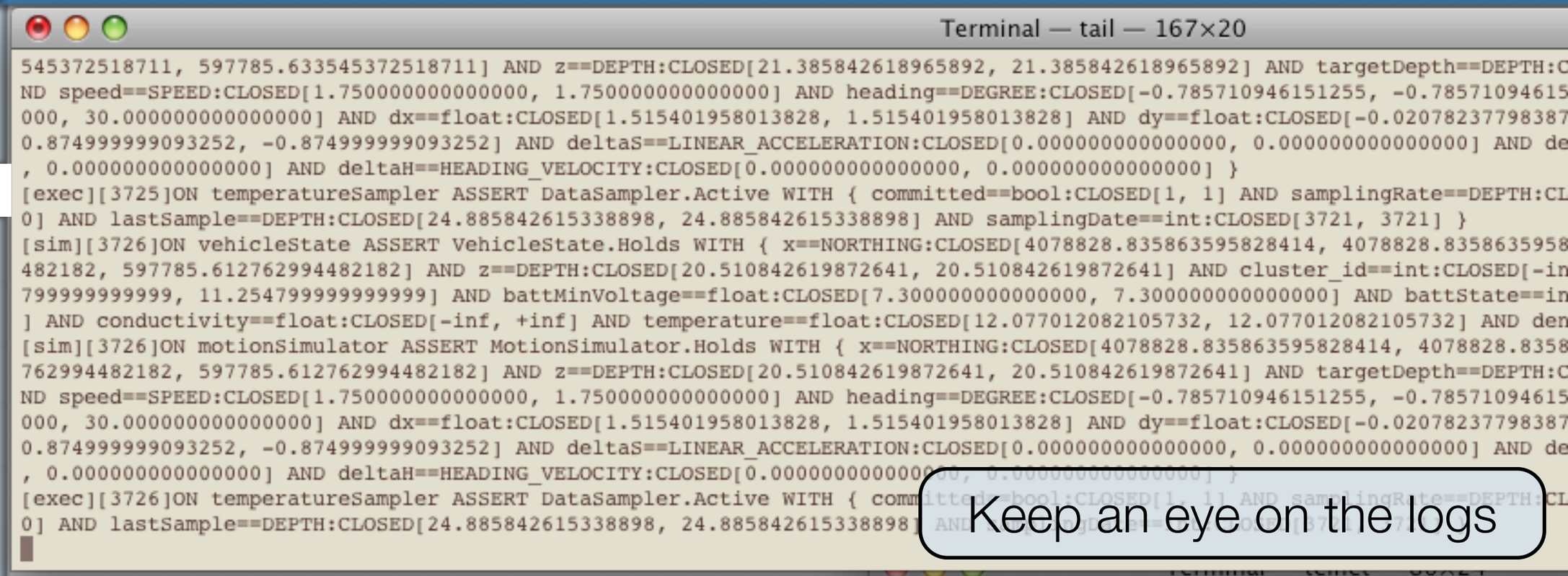
TREX commands

```
mbari1224:~ fpy$ telnet saumon
Trying 134.89.12.153...
Connected to saumon.shore.mbari.org.
Escape character is '^]'.
Welcome to QNX 4.25
Copyright (c) QNX Software Systems Ltd. 1982,1998
login: dorado1
password:
Last login: Fri Mar 06 01:25:30 2009 on //1/dev/ttyp0
Fri Mar 06 05:16:40 2009
$ cd auv_autonomy/
$ source configSrc
source: not found
$ . configSrc
$ cd $AUV/bin
$ nohup ./vcsServer -v &
```

AUV commands

Running TREX

My personal and recommended desktop configuration when I am running TREX



Running TREX

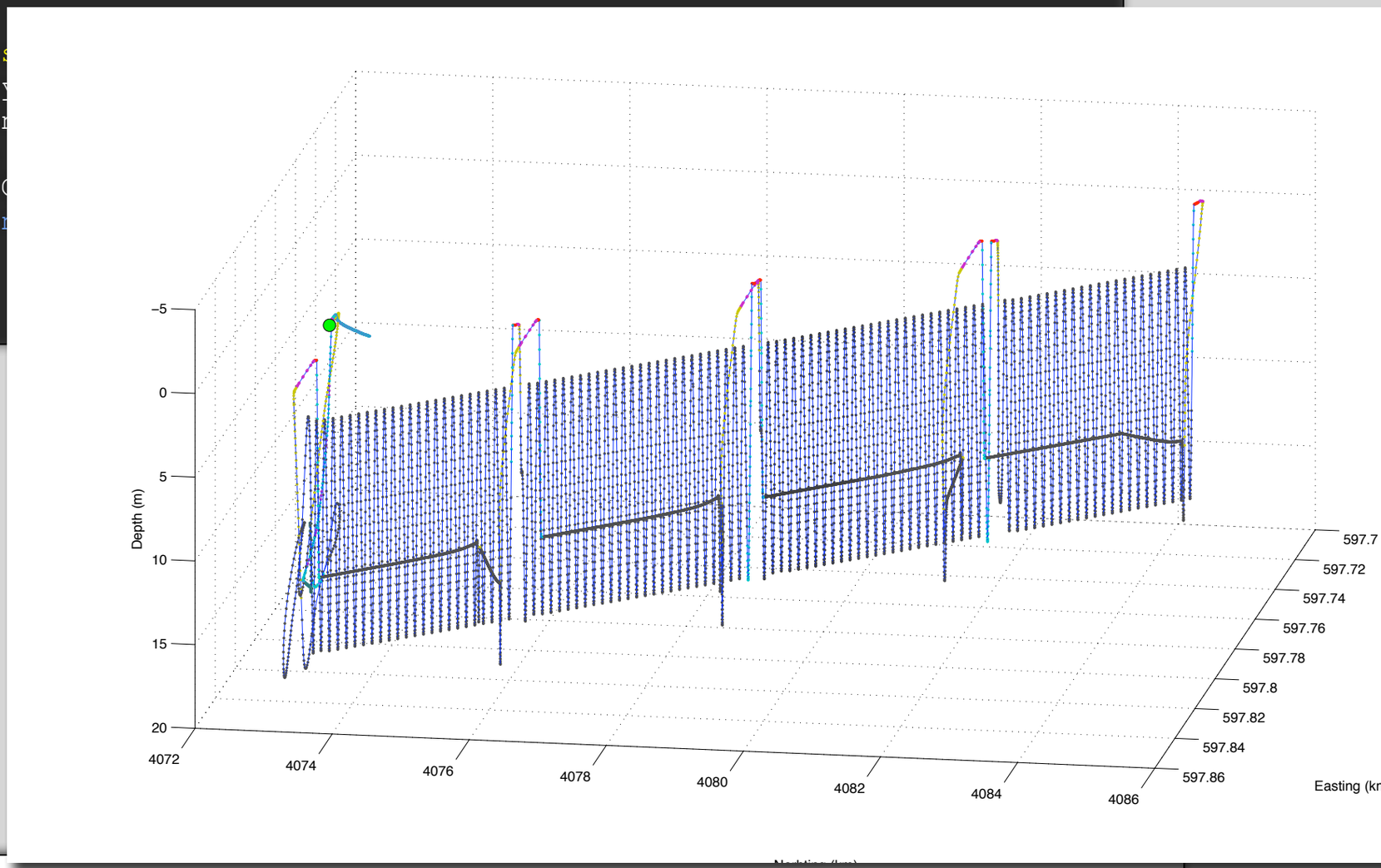
My personal and recommended desktop configuration when I am running TREX

```
[mbari1224 ~]$ volGet.sh # extract data from $TREX_LOG_DIR/latest
   or
[mbari1224 ~]$ volGet.sh ./2009.054.1 # extract data from the given path
Extracting VolumeSurvey high level actions :
   - Northing stored in /Users/fpy/Coding/TREX/log/latest/northing.survey
   - Easting stored in /Users/fpy/Coding/TREX/log/latest/northing.survey
Processing /Users/fpy/Coding/TREX/log/latest/front.log
generating path.kml # not working yet
/Users/fpy
done
```

```
matlab% cd ~/coding/TREX/log
matlab% plotNav('2009.054.1') % argument is the path to the processed log
%%%% returned structure describe when each behavior have been active
```

basic log processing

```
[mbari1224 ~]$ volGet.sh # extract data from $TREX_LOG_DIR/latest
   or
[mbari1224 ~]$ volGet.s
Extracting VolumeSurvey
   - Northing stored in
   - Easting stored in
Processing /Users/fpy/C
generating path.kml # n
/Users/fpy
done
```




```
matlab% cd ~/coding/TREX/log
matlab% plotNav('2009.054.1') % argument is the path to the processed log
%%%% returned structure describe when each behavior have been active
```

basic log processing

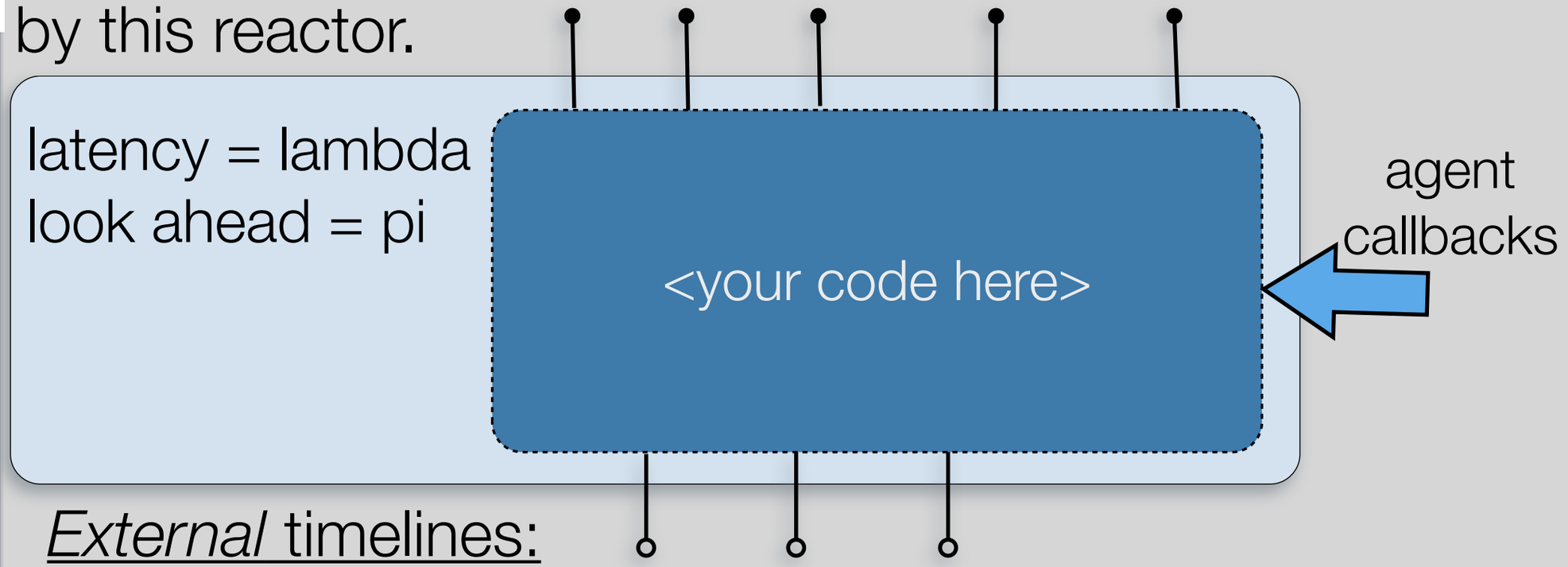
TREX internals :

How it works and how to extend it

Agent life in short : (`agent/base/Agent.hh`)

- amc/sim/... are programs that :
  - load a set of reactor (at least 5 different types existing now) `Agent::initialize`
  - check that they are correctly set `Agent::initialize`
  - connect to a clock (3 existing now) `Agent::initialize`
  - At each clock tick : `Agent::run` / `Agent::doNext`
    - propagate observations (eg position, sensors, ...) through the reactors
    - allow dispatching of commands between reactors
    - let deliberate the reactor that needs to
    - wait next tick
- `amc` do it in batch mode
- `sim` in a gdb like interface allowing to step through the clock ticks
- `batcher` spawn multiple instances of `samp` that do it in batch mode

Internal timelines:
state variables managed
by this reactor.



External timelines:
connection to *Internal* timeline
of another reactor. I can :

- have state updates (observation)
- suggest a value in the future (goal)

A reactor in general

Initialization of reactor (agent/base/Teleoreactor.hh)

1. Constructor `TeleoReactor::TeleoReactor(LabelsStr name, TiXMLElement const &config)`
   - `name` : A symbolic identifier for this instance
   - `config` : an XML structure for configuration purpose
   - This the point where you can collect information in the configuration files
2. Collect information `queryTimelineMode(list<> &internals, list<> &externals)`
   - `internals` : place to put the name of the timelines I own
   - `externals` : place to put the name of the timelines I want to observe/use
   - The agent collect these informations so it can create connection between reactors and check for inconsistencies, or loops and order the reactors for better management.
3. Initialization `handleInit(Tick init, map<> const &servers, ObserverId &observer)`
   - `init` : initial date for the mission (usually 0)
   - `servers` : list of the connection point to external timelines
   - `observer` : where to post my observations
   - Ask reactors to complete their initialization so we can start the mission

Callbacks during mission execution

- `handleTickStart()` : the clock has advanced. If I have new goals for others it is time to post them
- `handleRequest(GoalId goal)` : A new goal have been posted for you
- `notify(ObservationID obs)` : A new observation have been posted
- `bool synchronize()` : time to :
  - check if the observations collected do not impact my “plan”
  - provide my own observations if required
- `bool hasWork()` : do you need to deliberate ?
- `resume()` : do one step (preferably short) of deliberation
- An example can be seen on `ctd2007/DownlinkReactor.[hh|cc]`
- “Documented” header on the wiki :

  https://oceana.mbari.org/confluence/display/AUV/Example+of+A+TREX+reactor

How to produce an observation

- During `synchronize` just call `sendNotify(Observation const &obs)`
- An observation is an encapsulation of a data structure corresponding to a state of a timeline with attributes that can be intervals (`agent/base/Observer.hh`)
- Generally create an `ObservationByValue` class that is basically some kind of associative array (attribute name -> domain)
- See `ctd2007/VCSAdapter.cc` on the method `synchronize()` for an example :

```
m_stateObs = new ObservationByValue("vehicleState", "VehicleState.Holds");
// Position
m_stateObs->push_back("x", IntervalDomain(ROUND(sp.position.x, 2)).copy());
m_stateObs->push_back("y", IntervalDomain(ROUND(sp.position.y, 2)).copy());
m_stateObs->push_back("z", IntervalDomain(ROUND(sp.position.z, 2)).copy());
   [...]
sendNotify(*m_stateObs);
```

How to produce an observation

- During `synchronize` just call `sendNotify(Observation const &obs)`
- An observation is an encapsulation of a data structure corresponding to a state of a timeline with attributes that can be intervals (`agent/base/Observer.hh`)
- Generally create an `ObservationByValue` class that is basically some kind of associative array (attribute name -> domain)
- See `ctd2007/VCSAdapter.cc` on the method `synchronize()` for an example :

```
m_stateObs = new ObservationByValue("vehicleState", "VehicleState.Holds");
// Position
m_stateObs->push_back("x", IntervalDomain(ROUND(sp.position.x, 2)).copy());
m_stateObs->push_back("y", IntervalDomain(ROUND(sp.position.y, 2)).copy());
m_stateObs->push_back("z", IntervalDomain(ROUND(sp.position.z, 2)).copy());
   [...]
sendNotify(*m_stateObs);
```

timeline name

state name

attribute name

value domain

How to produce a goal ?

- To be done correctly it can be tricky right now (simplification in the coding queue with chunk of code already in `ctd2007/DownlinkReactor.cc`
- But during `handleTickStart` a call of `sendRequest(Goal const &goal)` send the goal to the corresponding reactor
- `Goal` class (`agent/base/Server.hh`) is similar to `Observation` class except that it also includes 3 special attributes :
  - `start` : domain of possible start dates
  - `duration` : domain of possible durations
  - `end` : domain of possibles end dates
- If for any reason you do not want a goal to be executed anymore just call `sendRecall(goal)`

Already existing reactors

- DeliberativeReactor (`agent/base/DbCore.hh`)

  embed europa planner to produce flexible plan based on a nddl model

- LogPlayer (`agent/base/LogPlayer.hh`)

  able to replay a xml log file to reproduce observations and goals request made by a reactor during a past mission

- VCSAdapter (`ctd2007/VCSAdapter.hh`)

  connection to AUV LayeredControl architecture through sockets

- Downlink (`ctd2007/DownlinkReactor.hh`)

  connect to the iridium modem to send message and receive goals from the shore

- Vitre (`agent/base/VitreReactor.hh`)

  broken code. Used to feed a java interface to see what's going on.

Debugging a mission using sim

How to replay a mission

```
[mbari1224 ~]$ cd $TREX_LOG_DIR/latest # go to the directory of this mission logs
[mbari1224 2009.062.3]$ cd cfg          # go to the config directory
# add missing configuration files
[mbari1224 cfg]$ cp $TREX_HOME/ctd2007/amc.play.cfg amc.cfg # replace VCSAdapter by a LogPlayer
[mbari1224 cfg]$ cp $TREX_HOME/ctd2007/Debug.On.cfg Debug.cfg
[mbari1224 cfg]$ cp $TREX_HOME/ctd2007/vcs.cfg .
[mbari1224 cfg]$ ls
Debug.cfg        front.exec.xml        sim.solver.cfg
amc.cfg          front.log             skipper.solver.cfg
exec.solver.cfg   front.sim.xml        vcs.cfg
front.cfg        front.skipper.xml
[mbari1224 cfg] sim_o_rt front
TREX Agent initalized
Options:
 Q :- Quit
 N :- Next
 G :- Goto <tick> e.g. g100
 R :- Reload Debug.cfg
 W :- Wrap the agent to tick 0.
 + :- enable pattern e.g. '+Agent'
 - :- disable pattern e.g. '-Agent'
 ! :- disable all debug messages
[0]> !                    # no debug message for now
[0]> g3000                # go at the tick where the bug occurred
[3000]> r                 # reactivate debug messages
[3000]> n                 # advance to the step of the bug
```

How to replay a mission

```
[mbari1224 ~]$ cd $TREX_LOG_DIR/latest # go to the directory of this mission logs
[mbari1224 2009.062.3]$ cd cfg          # go to the config directory
# add missing configuration files
[mbari1224 cfg]$ cp $TREX_HOME/ctd2007/amc.play.cfg amc.cfg # replace VCSAdapter by a LogPlayer
[mbari1224 cfg]$ cp $TREX_HOME/ctd2007/Debug.On.cfg Debug.cfg
[mbari1224 cfg]$ cp $TREX_HOME/ctd2007/vcs.cfg .
[mbari1224 cfg]$ ls
Debug.cfg         front.exec.xml        sim.solver.cfg
amc.cfg           front.log             skipper.solver.cfg
exec.solver.cfg    front.sim.xml        vcs.cfg
front.cfg         front.skipper.xml
[mbari1224 cfg] sim_o_rt front
TREX Agent initalized
Options:
 Q :- Quit
 N :- Next
 G :- Goto <tick> e.g. g100
 R :- Reload Debug.cfg
 W :- Wrap the agent to tick 0.
 + :- enable pattern e.g. '+Agent'
 - :- disable pattern e.g. '-Agent'
 ! :- disable all debug messages
[0]> !                          # no debug message for now
[0]> g3000                      # go at the tick where the bug occurred
[3000]> r                       # reactivate debug messages
[3000]> n                       # advance to the step of the bug
```

Kind of work too with AUV linux fast sim

```
$ cp amc.vcs.cfg amc.cfg
$ sim_o_rt front -fast 30
```

but still some strange behavior

More to come later ...

- The Debug.log file contains all the messages produced by the function debugMsg("type", "message") provided that "type" is listed in Debug.cfg
- Debug.On.cfg produces a lot of messages from the planner that are tricky to interpret even after next section
- Luckily you have over logs that can displayed graphically the plan (sim and amc produce these log but sim is more complete)
- For now we will stay at this level and it is up to the debugger to activate the correct messages on his Debug.cfg (or even run sim under gdb)
- Lets move on to The DeliberativeReactor and Europa planner

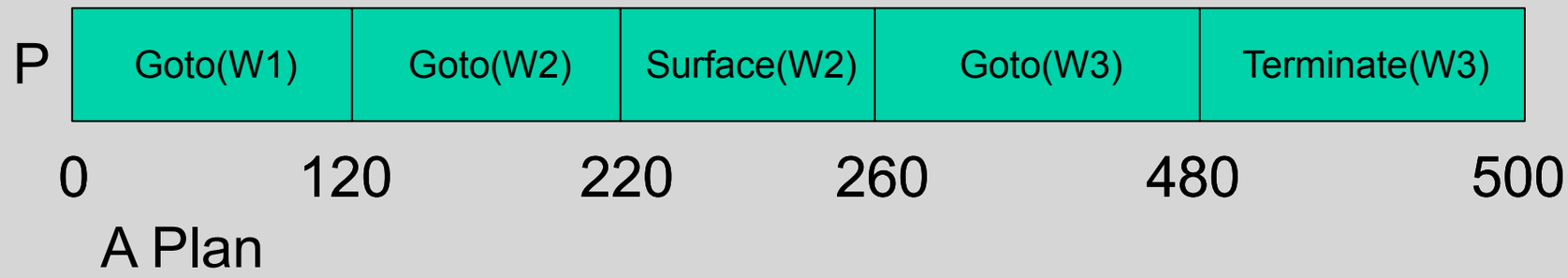
The deliberative Reactor

The deliberative reactor

- TREX is mostly an architecture that simplify the interaction and execution of interleaved control loops
- Nothing really enforce you to use Europa planner instead of another existing planner otr to even use this kind of things
- That being said the whole design of TREX was based on having a planner as the core part of the control loop
- Moreover it takes its roots in lot of the representation used specifically by Europa (observations and goals are *tokens* in *timelines*)
- Finally a reactor is already implemented so lets look at this more in details

plan as timed program | grounded sequence plan

Timed sequence of values
Values are *predicates* (logic)
***Grounded* since *variables* bound**
Functional Semantics:
- P(0) == Goto(W1)
- P(240) == Surface(W2)

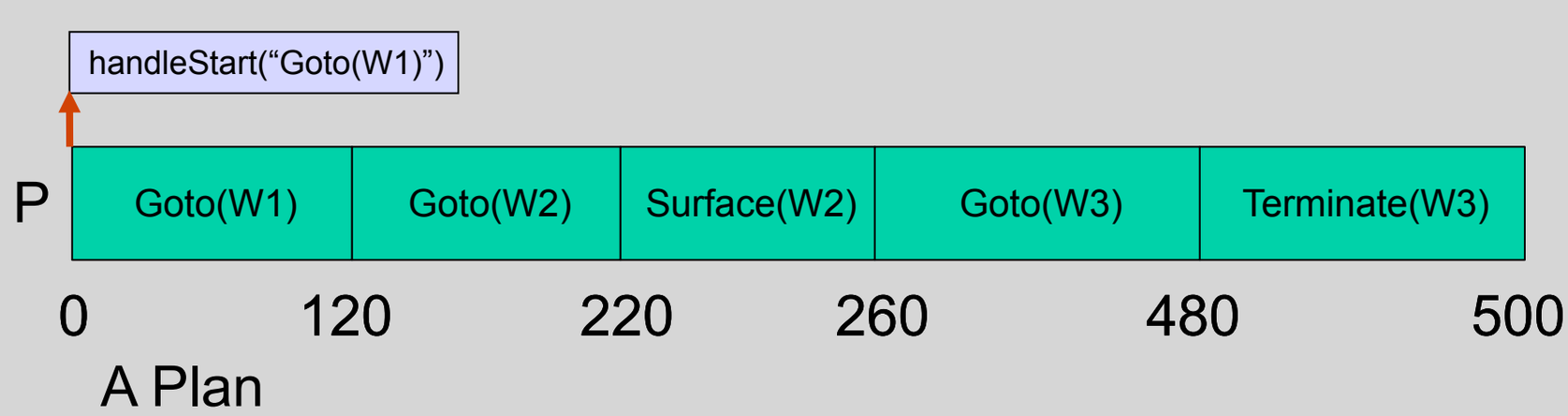


plan as timed program | grounded sequence plan

Plan Execution with A Clock

plan as timed program | grounded sequence plan

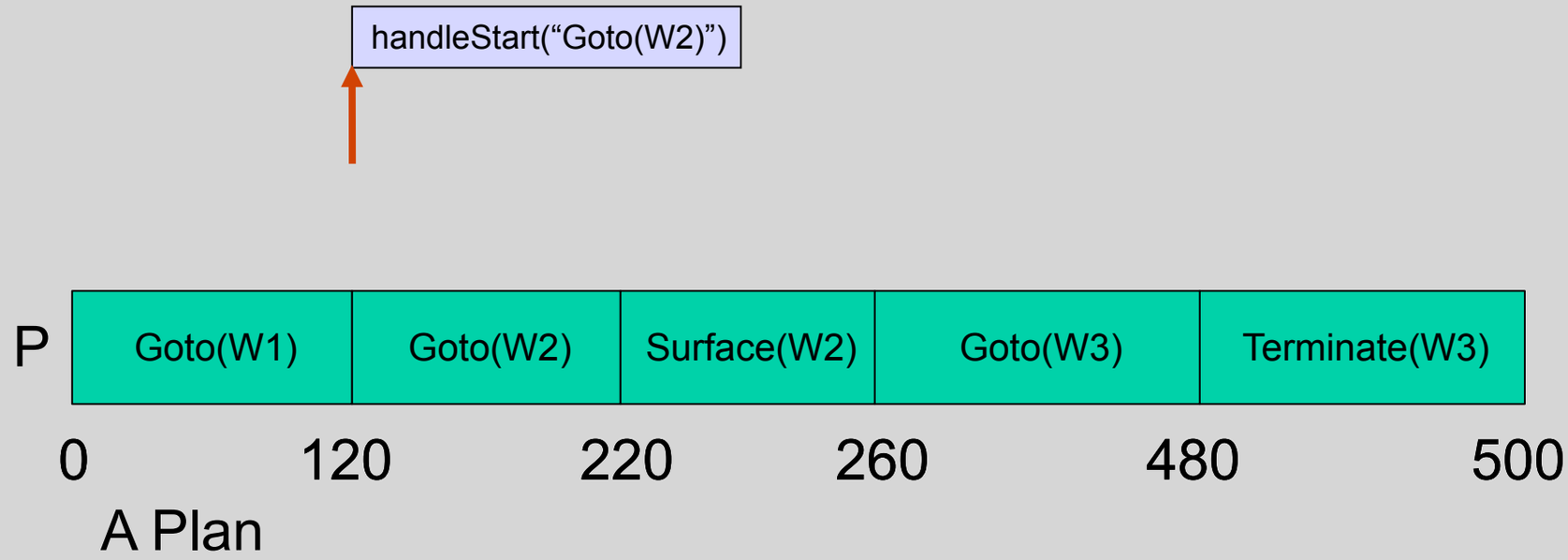
Plan Execution with A Clock



plan as timed program

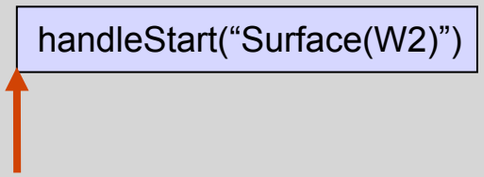
grounded sequence plan

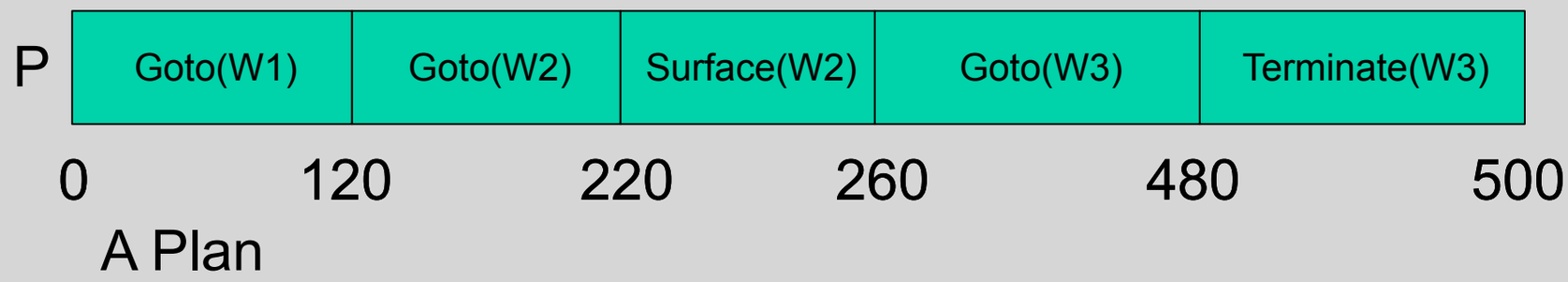
Plan Execution with A Clock



plan as timed program | grounded sequence plan

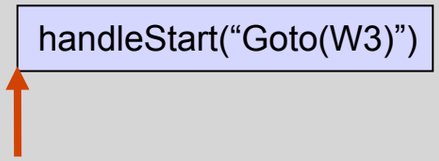
Plan Execution with A Clock

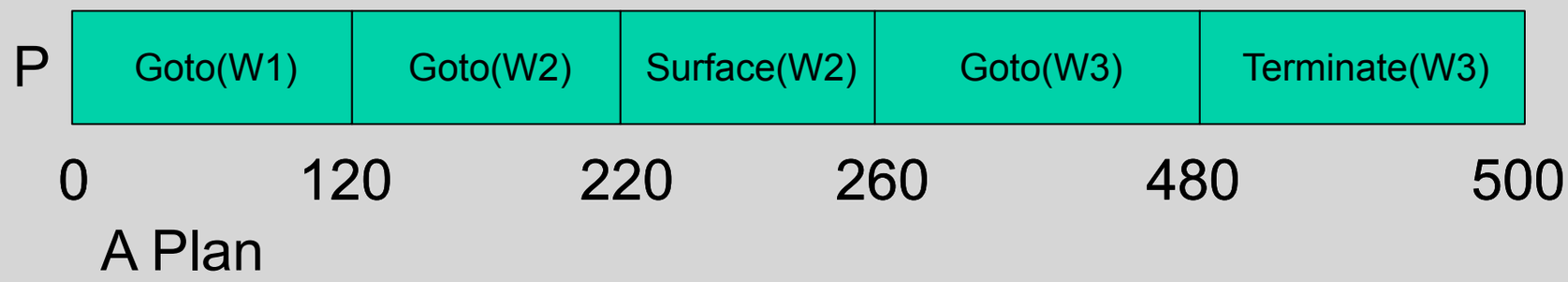




plan as timed program | grounded sequence plan

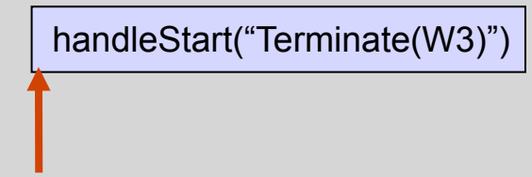
Plan Execution with A Clock

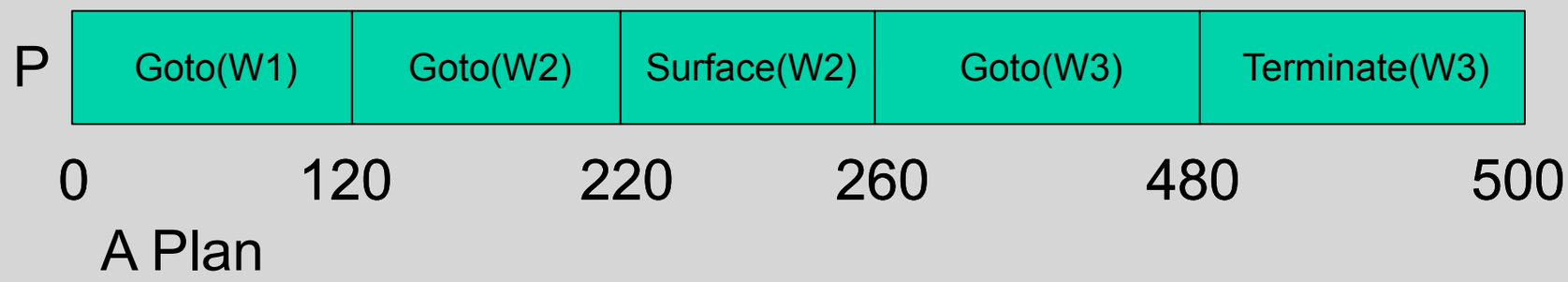




plan as timed program | grounded sequence plan

Plan Execution with A Clock





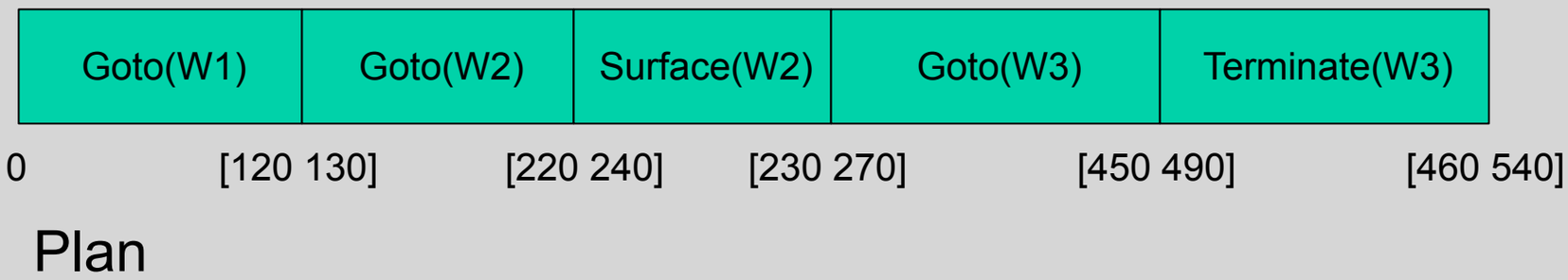
plan as timed program | grounded sequence plan

handleStart(“Terminate(W3)”)

Single Threaded
Grounded
Brittle

plan as timed program | grounded sequence plan

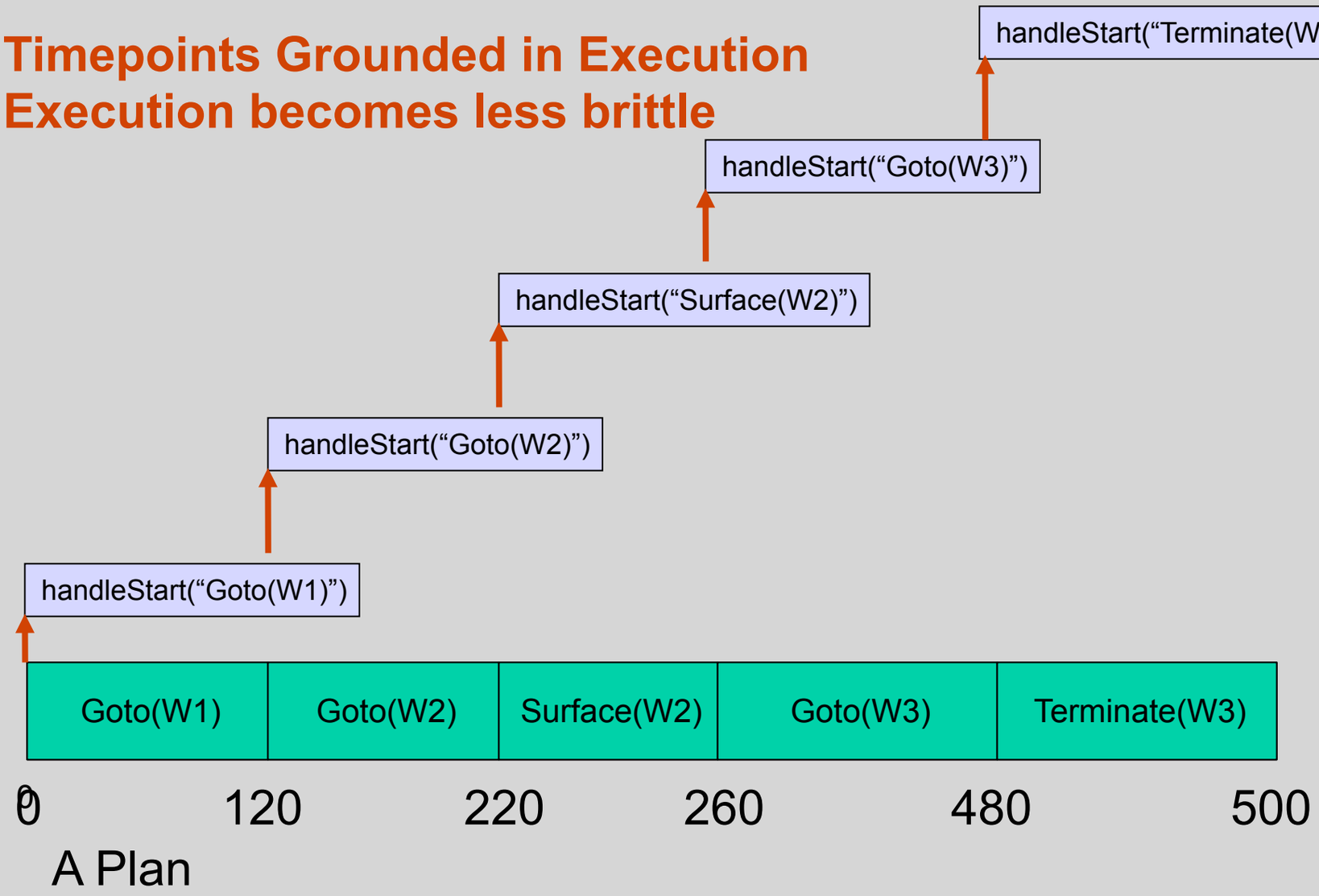
Timepoints Grounded in Execution
Execution becomes less brittle



plan as timed program

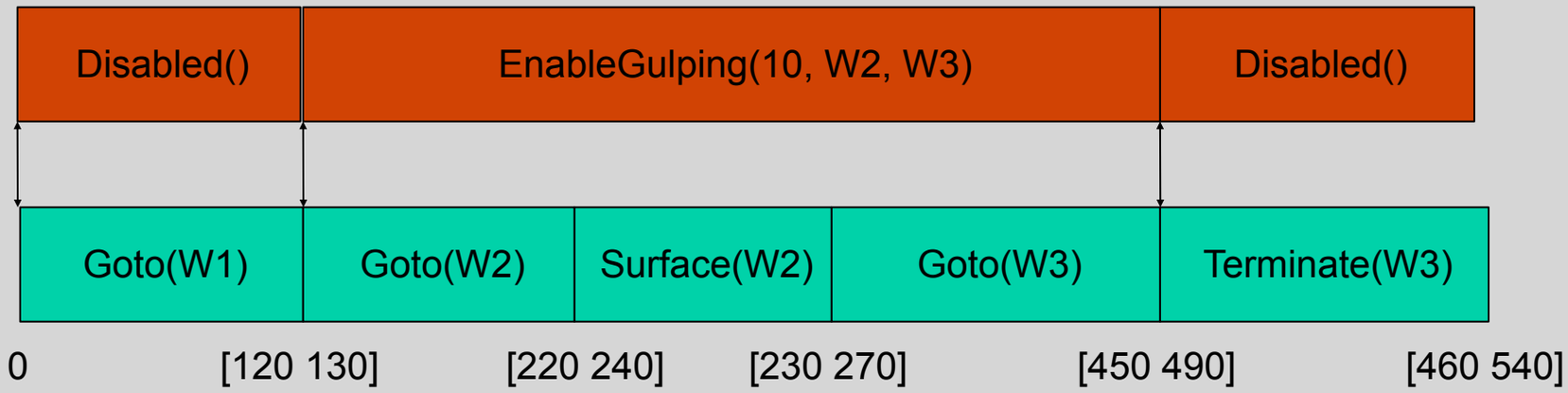
flexible sequence plan

Timepoints Grounded in Execution
Execution becomes less brittle



plan as timed program

flexible sequence plan



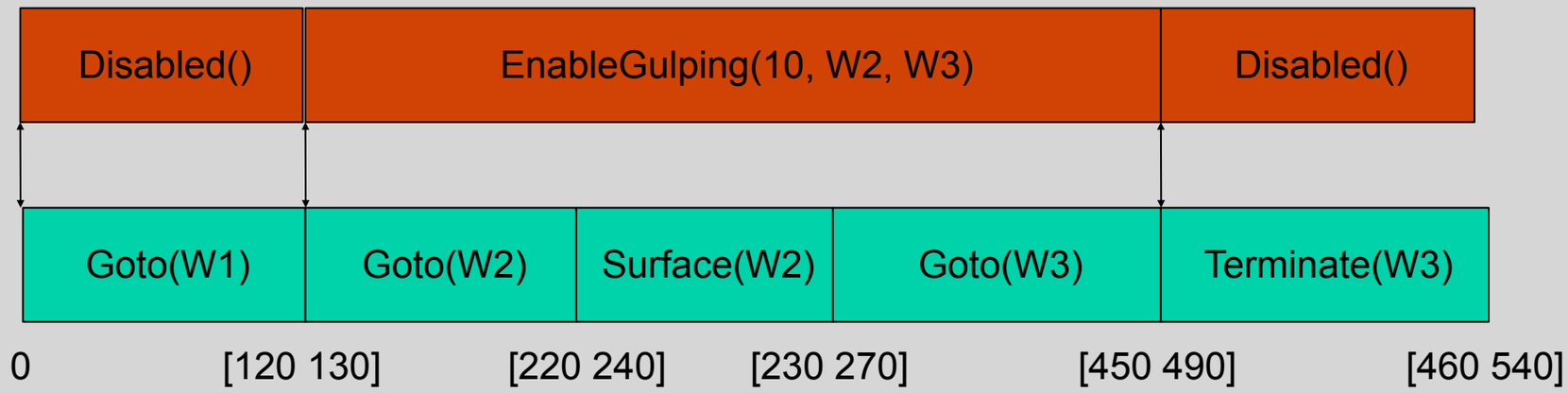
Plan

2 threads of execution

plan as timed program | flexible concurrent plan

Timeline
Gulper Behavior Status

Active Navigator



Plan

2 threads of execution

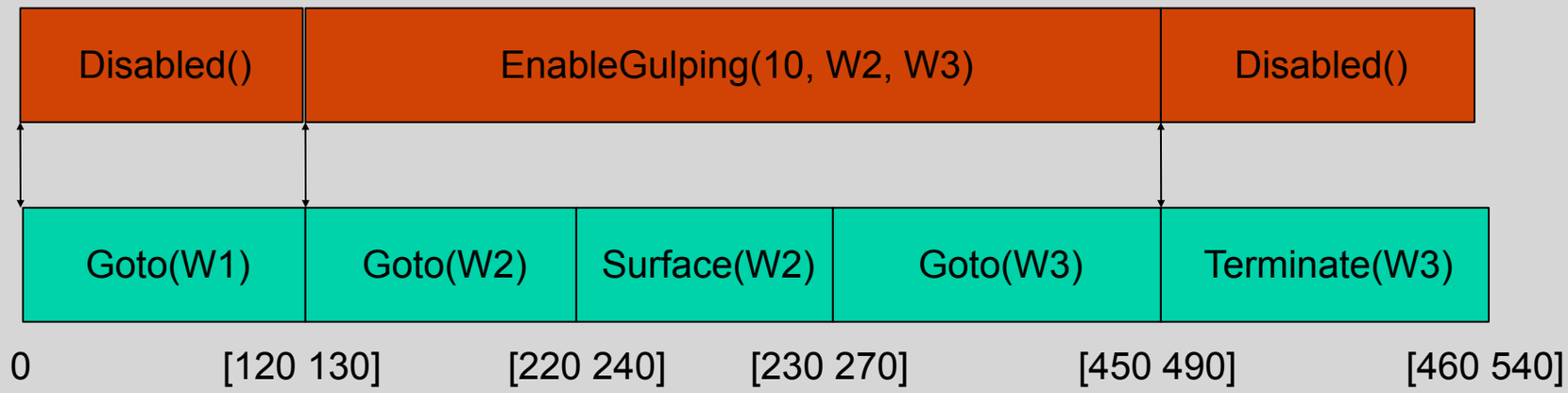
plan as timed program | flexible concurrent plan

Timeline
Gulper Behavior Status

Active Navigator

Timeline Values (Tokens)
{Disabled, EnableGulping(MaxGulps, From, To)}

{Goto(Loc), Surface(Loc), Terminate(Loc)|



Plan

2 threads of execution

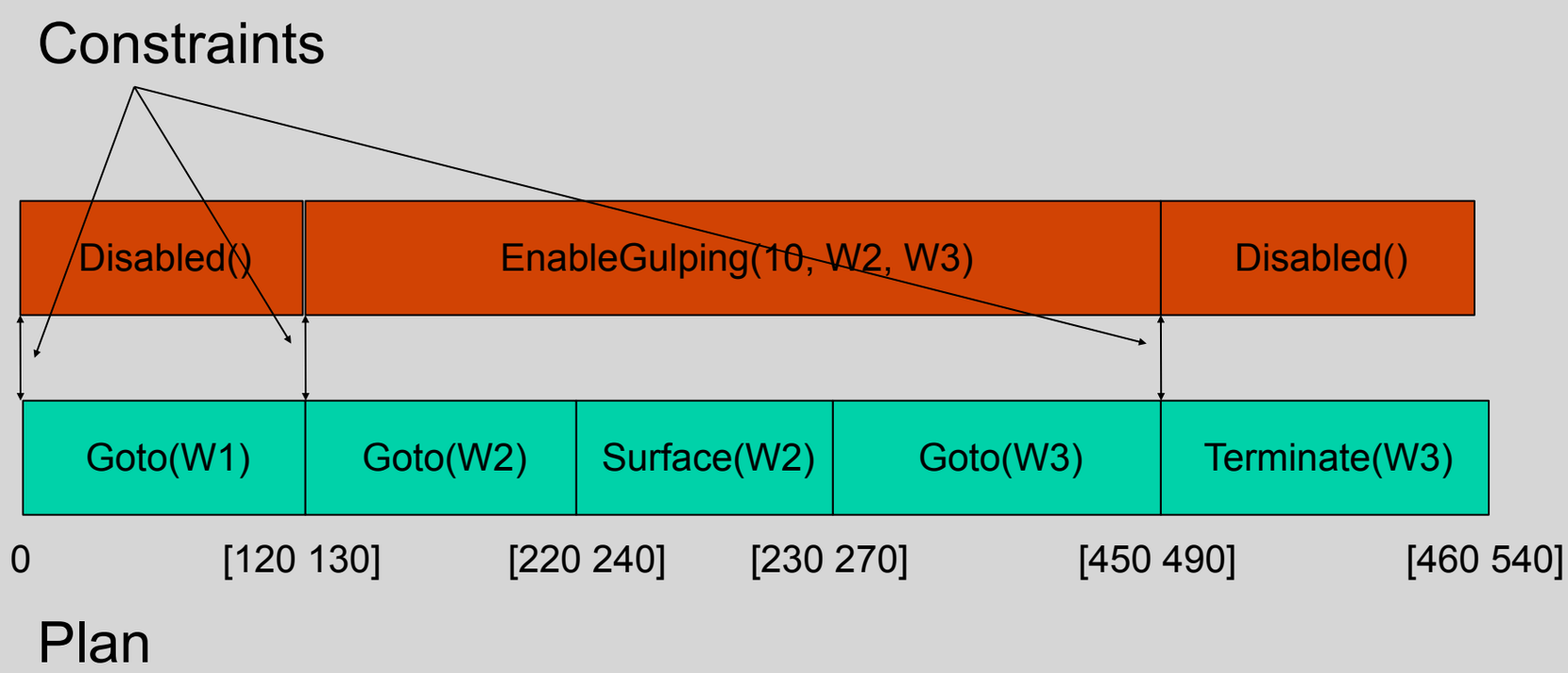
plan as timed program | flexible concurrent plan

Timeline
Gulper Behavior Status

Active Navigator

Timeline Values (Tokens)
{Disabled, EnableGulping(MaxGulps, From, To)}

{Goto(Loc), Surface(Loc), Terminate(Loc)|



2 threads of execution

plan as timed program | flexible concurrent plan

Initial State
At 0 : At(W0)
At 0 : GulperStatus=Disabled()

Goals
Over [130 450] GulperStatus=EnableGulping(10, W2, W3)

Model
Vehicle Dynamics
Terrain
Behavior Model

Automated planning

Automated plans generations

Defines Temporal Scope

Initial State

At 0 : At(W0)

At 0 : GulperStatus=Disabled()

Goals

Over [130 450] GulperStatus=EnableGulping(10, W2, W3)

Model

Vehicle Dynamics
Terrain
Behavior Model

Automated planning

Automated plans generations

Timeline Value Restriction

Initial State

At 0 : At(W0)

At 0 : GulperStatus=Disabled()

Goals

Over [130 450] GulperStatus=EnableGulping(10, W2, W3)

Model

Vehicle Dynamics
Terrain
Behavior Model

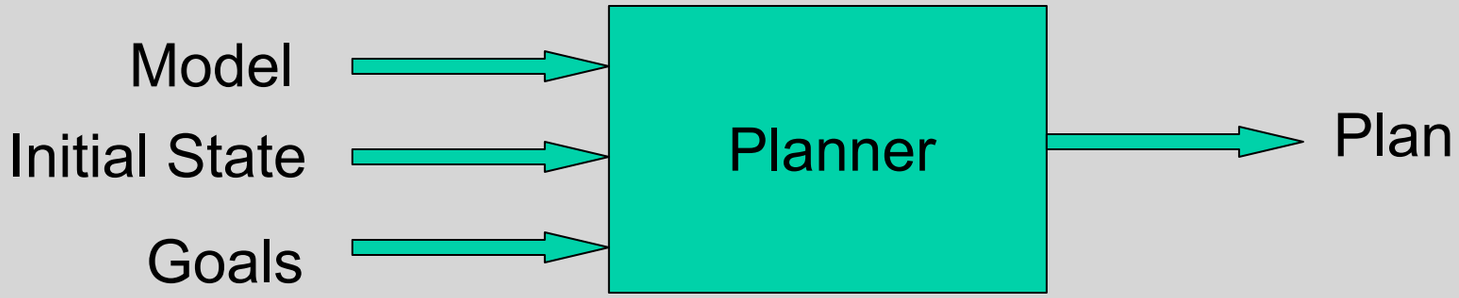
Automated planning

Automated plans generations

Initial State
At 0 : At(W0)
At 0 : GulperStatus=Disabled()

Goals
Over [130 450] GulperStatus=EnableGulping(10, W2, W3)

Model
Vehicle Dynamics
Terrain
Behavior Model



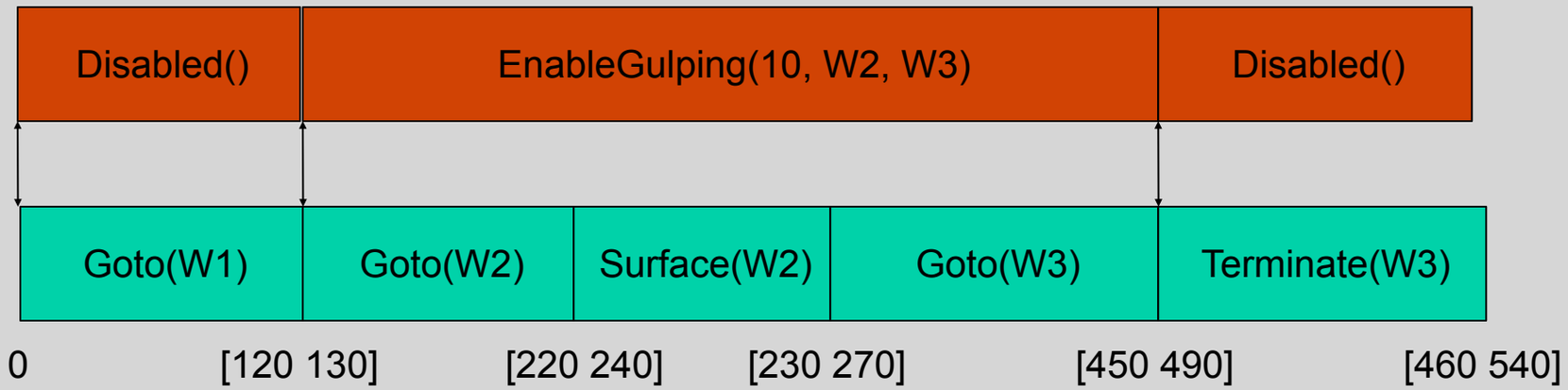
Automated planning

Automated plans generations

Initial State
At 0 : At(W0)
At 0 : GulperStatus=Disabled()

Goals
Over [130 450] GulperStatus=EnableGulping(10, W2, W3)

Model
Vehicle Dynamics
Terrain
Behavior Model



Plan

Automated planning

Automated plans generations

CSP based planning (Europa)

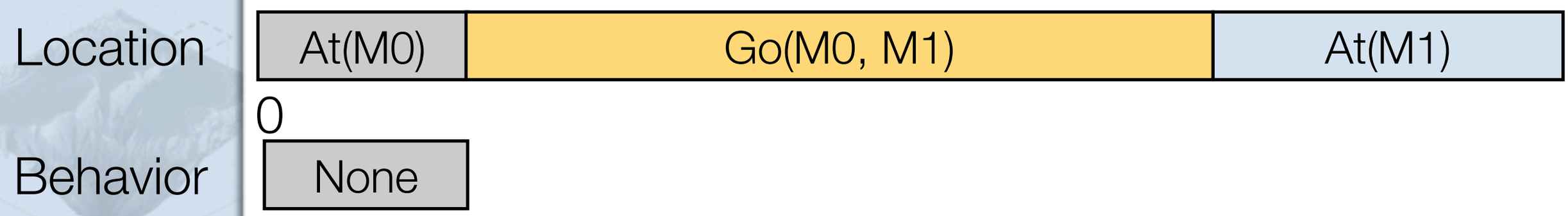
- Describe the plan model as a constraint satisfaction problem (CSP)
  - Just give constraints between the different states variables of the domain
  - state variables are described as timelines
  - A timeline is a sequence of states
  - states are described as token
  - A token is predicate that is true between its start and it end for a duration end
  - All attributes of the token are domains (intervals or set of values)
- Constraints can be :
  - numerical computations (eg duration = distance*speed)
  - or temporal relations (eg Go(from, to) meets At(from))
- The role of the planner is to find a (set of) solutions that satisfies the goals knowing an initial solution

How it works ? (quickly)

- For the CSP planner every steps between (initial situation + goal) toward a complete sequence of action that link these is a plan
- Its role is to find a plan that is
  - complete (no more unexplained situation)
  - and consistent (no contradictions in the plan with the model)
- Least commitment : do not constraint more than required
  - eg if something can happen between 2 and 3pm why should I enforce it when we can decide that during this interval (at execution)
  - similarly if a robot has two free arms why force one when it can be decide when we will use it ? (improved robustness to failure)

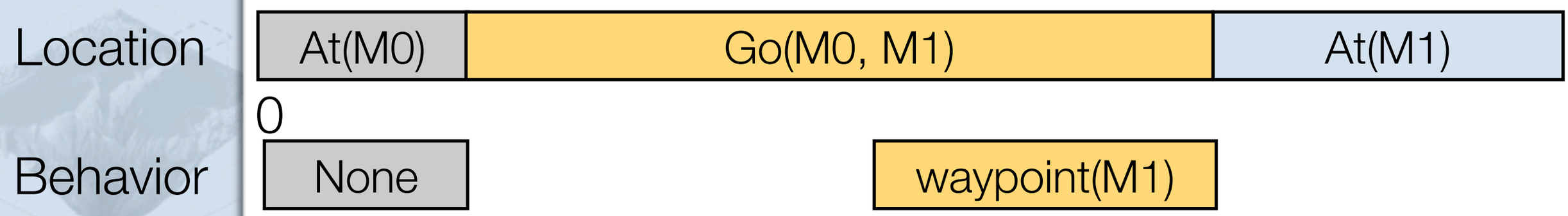
How it works ? (quickly)

- For the CSP planner every steps between (initial situation + goal) toward a complete sequence of action that link these is a plan
- Its role is to find a plan that is
  - complete (no more unexplained situation)
  - and consistent (no contradictions in the plan with the model)
- Least commitment : do not constraint more than required
  - eg if something can happen between 2 and 3pm why should I enforce it when we can decide that during this interval (at execution)
  - similarly if a robot has two free arms why force one when it can be decide when we will use it ? (improved robustness to failure)



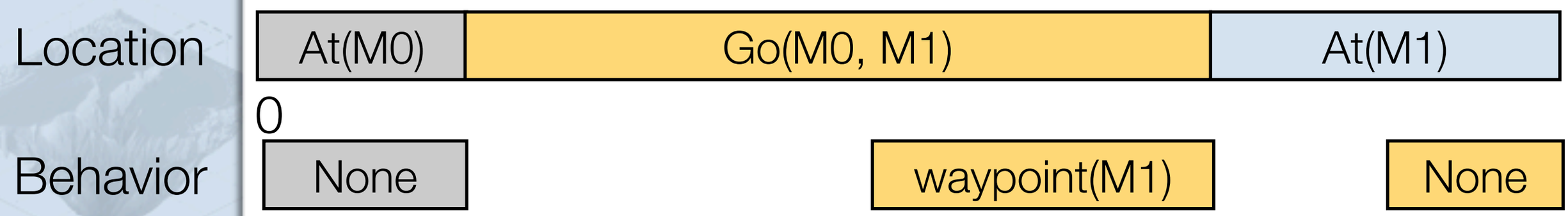
How it works ? (quickly)

- For the CSP planner every steps between (initial situation + goal) toward a complete sequence of action that link these is a plan
- Its role is to find a plan that is
  - complete (no more unexplained situation)
  - and consistent (no contradictions in the plan with the model)
- Least commitment : do not constraint more than required
  - eg if something can happen between 2 and 3pm why should I enforce it when we can decide that during this interval (at execution)
  - similarly if a robot has two free arms why force one when it can be decide when we will use it ? (improved robustness to failure)



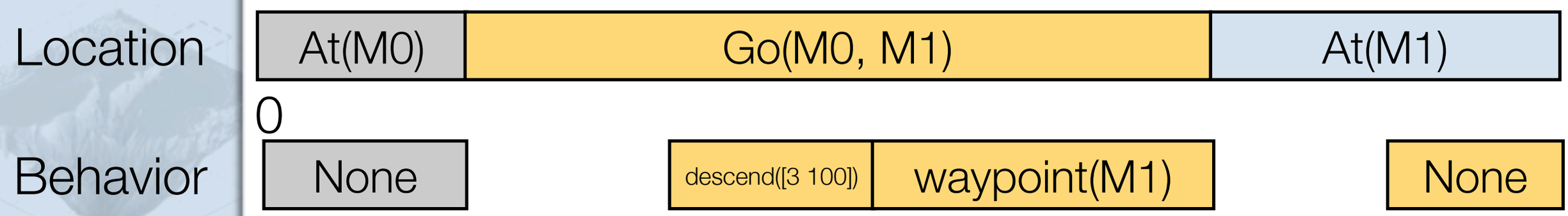
How it works ? (quickly)

- For the CSP planner every steps between (initial situation + goal) toward a complete sequence of action that link these is a plan
- Its role is to find a plan that is
  - complete (no more unexplained situation)
  - and consistent (no contradictions in the plan with the model)
- Least commitment : do not constraint more than required
  - eg if something can happen between 2 and 3pm why should I enforce it when we can decide that during this interval (at execution)
  - similarly if a robot has two free arms why force one when it can be decide when we will use it ? (improved robustness to failure)



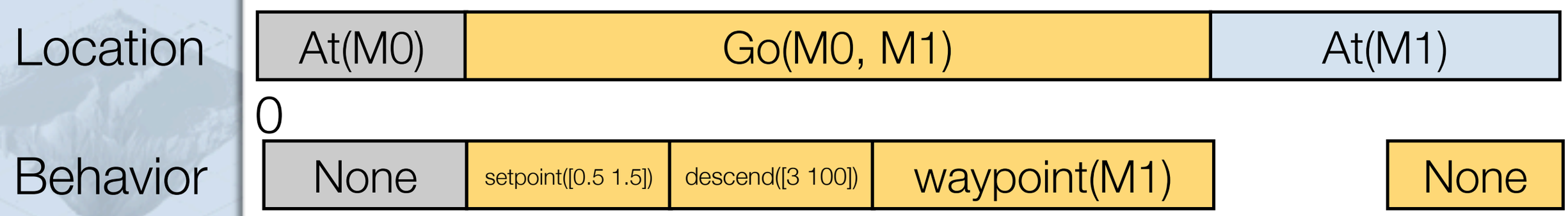
How it works ? (quickly)

- For the CSP planner every steps between (initial situation + goal) toward a complete sequence of action that link these is a plan
- Its role is to find a plan that is
  - complete (no more unexplained situation)
  - and consistent (no contradictions in the plan with the model)
- Least commitment : do not constraint more than required
  - eg if something can happen between 2 and 3pm why should I enforce it when we can decide that during this interval (at execution)
  - similarly if a robot has two free arms why force one when it can be decide when we will use it ? (improved robustness to failure)



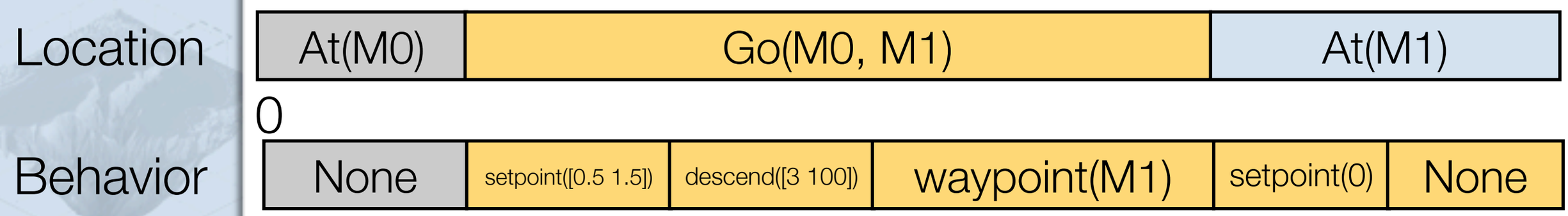
How it works ? (quickly)

- For the CSP planner every steps between (initial situation + goal) toward a complete sequence of action that link these is a plan
- Its role is to find a plan that is
  - complete (no more unexplained situation)
  - and consistent (no contradictions in the plan with the model)
- Least commitment : do not constraint more than required
  - eg if something can happen between 2 and 3pm why should I enforce it when we can decide that during this interval (at execution)
  - similarly if a robot has two free arms why force one when it can be decide when we will use it ? (improved robustness to failure)



How it works ? (quickly)

- For the CSP planner every steps between (initial situation + goal) toward a complete sequence of action that link these is a plan
- Its role is to find a plan that is
  - complete (no more unexplained situation)
  - and consistent (no contradictions in the plan with the model)
- Least commitment : do not constraint more than required
  - eg if something can happen between 2 and 3pm why should I enforce it when we can decide that during this interval (at execution)
  - similarly if a robot has two free arms why force one when it can be decide when we will use it ? (improved robustness to failure)



Modeling for TREX in nddl

- NDDL (noodle) is the domain description language used by Europa
- you can find more details and some doc here :

  http://babelfish.arc.nasa.gov/trac/europa/wiki/EuropaDocs

- It is an object oriented language (some java inspiration but well ... you'll see)
- TREX defines 2 main classes :
  - AgentTimeline : timelines shared between reactors
  - Actions : an internal construct used to enforce thing to happen as soon as possible (but sometimes it is tricky to use)
- TREX also specifies constraints that helps for execution
- Here we will just look at the AgentTimeline

A toy example : the Navigation timeline

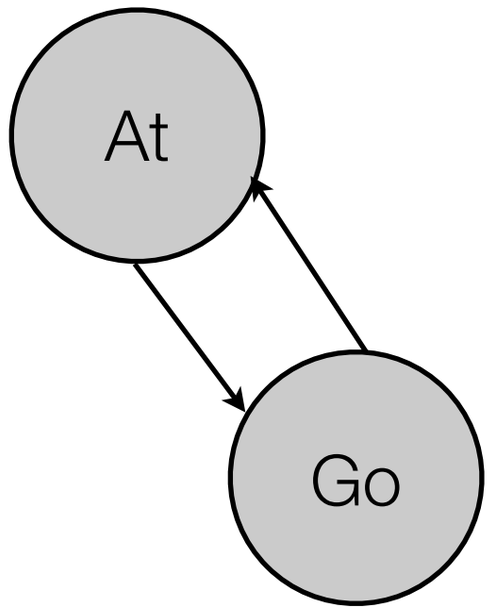
- decalaration :

```
#include “TREX.nddl”

class Navigation extends AgentTimeline{
  predicate At {
     float latitude, longitude;
     DEPTH depth;
  }
  predicate Go {
     float lat_from, lon_from, lat_to, lon_to;
     DEPTH depth_from, depth_to;
  }
  Navigation(Mode mode) {
     super(mode, “At”);
   }
}
```

A toy example : the Navigation timeline

- simple rules :

```
#include “Navigation.nddl”

Navigation::Go {
  met_by(At startPos);  // temporal relation
  eq(startPos.latitude, lat_from); // equality
  eq(startPos.longitude, lon_from);
  eq(startPos.depth, depth_from);

  meets(At endPos);   // temporal relation
  eq(endPos.latitude, lat_to);
  eq(endPos.longitude, lon_to);
  eq(endPos.depth, depth_to);
}
Navigation::At {
  met_by(Go);
  meets(Go);
}
```



Temporal relations

- describe how one token relates to another. Most of them are based on Allen relations for intervals.

- Some of them :
  - Y before X : Y ends before X starts (after)
  - Y met_by X : Y starts right when X terminates (meets)
  - Y equals X : X and Y start and end concurrently
  - Y contains X : Y starts and ends during X (contained_by)
  - Y starts X : X and Y start at the same time (ends)

- Others are just temporal relations between time points
  - precedes(x, y) : x is a date less or equal to y. eg precedes(start, end)
  - conccurrent(x,y): x and y are the same time eg concurrent(start, start)
  - temporalDistance(x, d, y) : x+d=y . eg temporalDistance(start, [1 10], end)

- All these constraints will be used by the planner to know temporal constraints between the tokens and instants.

Other constraints

- Usually these are constraints applied to the variables of the tokens and domain to relate them.
- Many already exist in Europa. Keep in mind that the planner can evaluate in any order (z=x+y == x=z-y == y=z-x) depending on how the constraints propagate :
  - planning : goal -> commands
  - synchronization : commands -> goal
- Some of them already in Europa :
  - addEq(a, b, c) : a+b=c or c-b=a ...
  - mulEq(a, b, c) : a*b=c or c/b=a ...
  - testEQ(x, a, b) : x = (a==b) with x a boolean (testLEQ, testLT)
  - eq(a,b), leq(a,b), lt(a, b) : a==b, a<=b, a<b
  - and more ...
- This can be even extended by implementing new ones in C++

Dynamic CSP : the `if` in Europa

- Problem with a constraint : when you put a constraint it becomes a statement
  - for example met_by(Go) in At rule imposes that for any At I have in the plan they **must** be met by a Go

    (impossible from there to have At preceded by another Ay without a Go between them)
- Luckily Europa offer a if control structure :
  - if(variable) { body } : constraints in body will be activated only when variable is a singleton (variable has one single value)
  - if(variable==constant) { body } : constraints in body activated on when variable is the singleton constant.
- Example for Go:

```
float dist;                                   // dist = [-inf +inf]
bool tooClose;                                // tooClose = {true, false}
calcDistance(dist, from, to);                 // dist = [-inf +inf] inter distance(from, to)
testLEQ(tooClose, dist, 1.5);                 // toClose = {t,f} inter dist<=1.5
if( tooClose==true ) {
   eq(duration ,1);                           // too close : set the Go duration to 1
}
if( tooClose==false ) {
  float travelTime;
  mulEq(speed, travelTime, dist);             // travelTime = dist/speed
  leq(travelTime, duration);                  // Go duration is at least travelTime
}
```

Constraints added by TREX

- TREX add some extra constraints that ease the execution management
  - default(x, y), bind(x, y) : if y is a singleton then set x to its closest value to y
    - example bind(start, 0) : start as soon as possible. If start=[100, 1000] set start at 100
  - bindMax(x, y) : set x to the closest value of y maximum possible value
    - bindMax(start, setpoint.end) : set start as close as possible to the maximum possible value of setpoint.end.
    - if setpoint.end=[150 200] equivalent to bind(start, 200)
  - defaultOnCommit(x, y) : equivalent to bind(x,y) except that it will be evaluated if and only if the token has started to be executed (ie start<=execution frontier)
    - defaultOnCommit(speed, 1.5) : if speed has not been set I will try to set it at 1.5m/s (or as close as I can)
- TREX re-implements also for now mulEq and addEq so they are less float precision sensitive for more robustness

Debugging the plan model

How to identify a plan inconsistency ?

- Follow the steps described previously
- Open Debug.log and search for “empty” string.
- If you find it it indicates which variable was generating the plan inconsistency
- This variable was over constrained and its resulting domain of values due to the intersection of the constraints resulted on no possible values
- Then you can see how you can correct it (may be tricky)
- That can be even tricker if it is a temporal inconsistency (eg a start time became empty)
  - compare it to access to a deleted memory cell in gdb : the effect can be far after the real cause

How to see plan produced

- To ease the debug TREX produces for each deliberative reactors a .plan file the embed code that can be parsed by graphviz tools
- This display the plan and the temporal relations between the token of the plan each time a reactor produces a new plan
- batch do not log it, amc has a lightweight version of it and sim try to extract as much data as possible

- To use it you need to install graphviz (http://www.graphviz.org/).
- Post process the file : planHist.sh [logpath] (default is latest)
- open the plan you want in logpath/plan.<reactor>
  - the plan files are named p<tick>.dot
  - in unix dotty p<tick>.dot

How to replay a mission

```
[mbari1224 ~]$ cd $TREX_LOG_DIR/latest # go to the directory of this mission logs
[mbari1224 2009.062.3]$ cd cfg          # go to the config directory
# add missing configuration files
[mbari1224 cfg]$ cp $TREX_HOME/ctd2007/amc.play.cfg amc.cfg # replace VCSAdapter by a LogPlayer
[mbari1224 cfg]$ cp $TREX_HOME/ctd2007/Debug.On.cfg Debug.cfg
[mbari1224 cfg]$ cp $TREX_HOME/ctd2007/vcs.cfg .
[mbari1224 cfg]$ ls
Debug.cfg        front.exec.xml        sim.solver.cfg
amc.cfg          front.log             skipper.solver.cfg
exec.solver.cfg   front.sim.xml        vcs.cfg
front.cfg        front.skipper.xml
[mbari1224 cfg] sim_o_rt front
TREX Agent initalized
Options:
 Q :- Quit
 N :- Next
 G :- Goto <tick> e.g. g100
 R :- Reload Debug.cfg
 W :- Wrap the agent to tick 0.
 + :- enable pattern e.g. '+Agent'
 - :- disable pattern e.g. '-Agent'
 ! :- disable all debug messages
[0]> !                      # no debug message for now
[0]> g3000                  # go at the tick where the bug occurred
[3000]> r                   # reactivate debug messages
[3000]> n                   # advance to the step of the bug
```

How to replay a mission

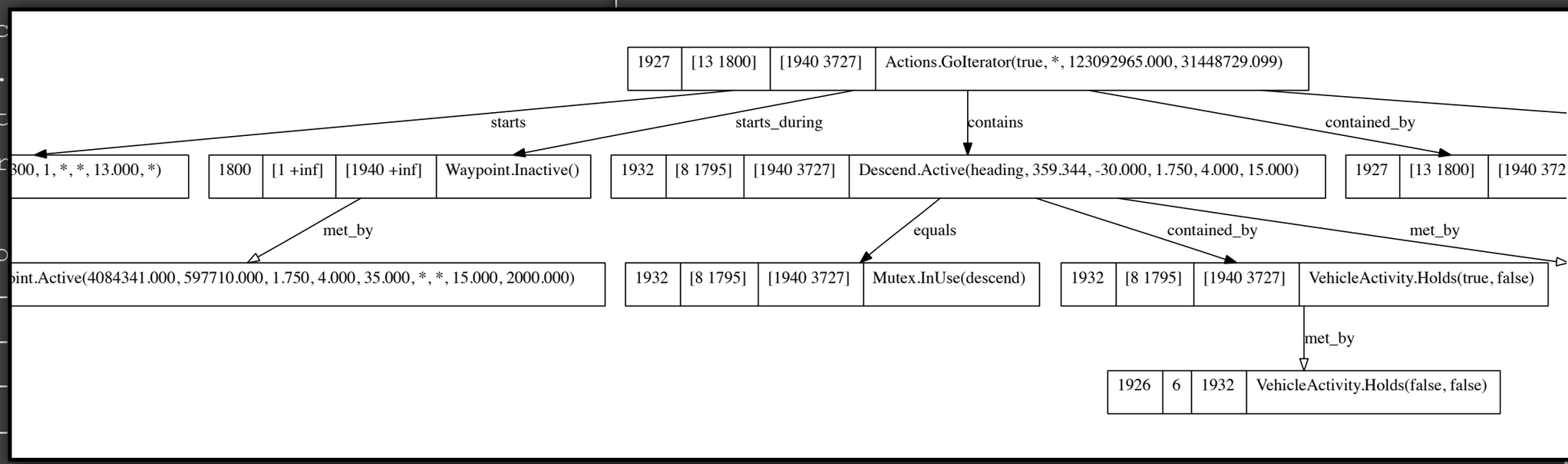
```
[mbari1224 ~]$ cd $TREX_LOG_DIR/latest # go to the directory of this mission logs
[mbari1224 2009.062.3]$ cd cfg                 # c
# add missing configuration files
[mbari1224 cfg]$ cp $TREX_HOME/ctd2007/amc
[mbari1224 cfg]$ cp $TREX_HOME/ctd2007/Debu
[mbari1224 cfg]$ cp $TREX_HOME/ctd2007/vcs
[mbari1224 cfg]$ ls
Debug.cfg          front.exec.xml          sim.
amc.cfg            front.log               skipp
exec.solver.cfg     front.sim.xml          vcs.
front.cfg          front.skipper.xml
[mbari1224 cfg] sim_o_rt front
TREX Agent initalized
Options:
 Q :- Quit
 N :- Next
 G :- Goto <tick> e.g. g100
 R :- Reload Debug.cfg
 W :- Wrap the agent to tick 0.
 + :- enable pattern e.g. '+Agent'
 - :- disable pattern e.g. '-Agent'
 ! :- disable all debug messages
[0]> !                          # no debug message for now
[0]> g3000                      # go at the tick where the bug occurred
[3000]> r                       # reactivate debug messages
[3000]> n                       # advance to the step of the bug
```

```
[mbari1224 ~]$ cd $TREX_LOG_DIR
[mbari1224 ~] planHist.sh              # process latest run
Extracting plans from exec
Extracting plans from skipper
[mbari1224 ~] cd latest/plan.exec # plans of exec
[mbari1224 ~] dotty 1939.dot      # plan produced at 1939
```

How to replay a mission

```
[mbari1224 ~]$ cd $TREX_LOG_DIR/latest # go to the directory of this mission logs
[mbari1224 2009.062.3]$ cd cfg         # (
# add missing configuration files
[mbari1224 cfg]$ cp $TREX_HOME/ctd2007/amc
[mbari1224 cfg]$ cp $TREX_HOME/ctd2007/Debu
[mbari1224 cfg]$ cp $TREX_HOME/ctd2007/vcs
[mbari1224 cfg]$ ls
Debug.cfg        front.exec.xml        sim.
amc.c
exec.
front
[mbar
TREX
Optio
 Q :-
 N :-
 G :-
 R :-
 W :- Wrap the agent to tick 0.
 + :- enable pattern e.g. '+Agent'
 - :- disable pattern e.g. '-Agent'
 ! :- disable all debug messages
[0]> !                      # no debug message for now
[0]> g3000                  # go at the tick where the bug occurred
[3000]> r                   # reactivate debug messages
[3000]> n                   # advance to the step of the bug
```

```
[mbari1224 ~]$ cd $TREX_LOG_DIR
[mbari1224 ~] planHist.sh          # process latest run
Extracting plans from exec
Extracting plans from skipper
[mbari1224 ~] cd latest/plan.exec # plans of exec
[mbari1224 ~] dotty 1939.dot      # plan produced at 1939
```



How to replay a mission

```
[mbari1224 ~]$ cd $TREX_LOG_DIR/latest # go to the directory of this mission logs
[mbari1224 2009.062.3]$ cd cfg         # c
# add missing configuration files
[mbari1224 cfg]$ cp $TREX_HOME/ctd2007/amc
[mbari1224 cfg]$ cp $TREX_HOME/ctd2007/Debu
[mbari1224 cfg]$ cp $TREX_HOME/ctd2007/vcs
[mbari1224 cfg]$ ls
Debug.cfg       front.exec.xml          sim.
amc.c
exec.
front
[mbar
TREX
Optio
 Q :-
 N :-
 G :-
 R :-
 W :- Wrap the agent to tick 0.
 + :- enable pattern e.g. '+Agent'
 - :- disable pattern e.g. '-Agent'
 ! :- disable all debug messages
[0]> !                      # no debug message for now
[0]> g3000                  # go at the tick where the bug occurred
[3000]> r                   # reactivate debug messages
[3000]> n                   # advance to the step of the bug
```

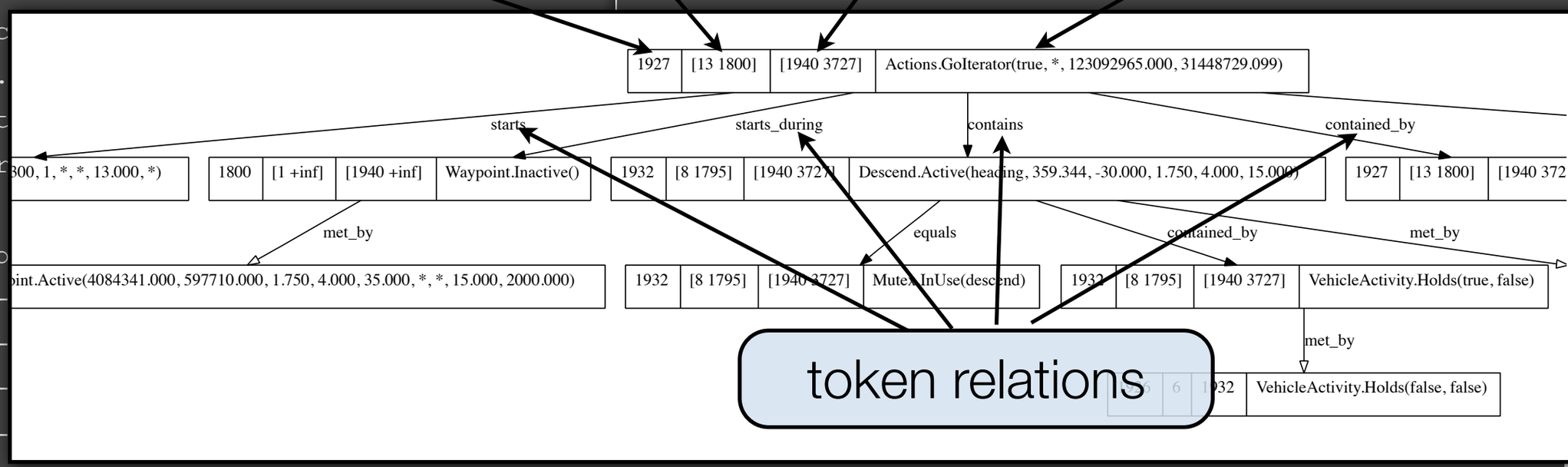
```
[mbari1224 ~]$ cd $TREX_LOG_DIR
[mbari1224 ~] planHist.sh          # process latest run
Extracting plans from exec
Extracting plans from skipper
[mbari1224 ~] cd latest/pla.exec # plans of exec
[mbari1224 ~] ddty 1939.dot      # produced at 1939
```

start

duration

end

token value



token relations

Example of a run

Thank you

TREX wiki : https://oceana.mbari.org/confluence/display/AUV/Autonomy
TREX web page : http://www.mbari.org/autonomy/TREX/
TREX open source : http://code.google.com/p/trex-autonomy/
Europa : http://babelfish.arc.nasa.gov/trac/europa