# Plan for Today

- Task 3.x
  - 3.x due nightly
  - final submission Friday: 3.20
- Project Part 2 comments
- System testing example

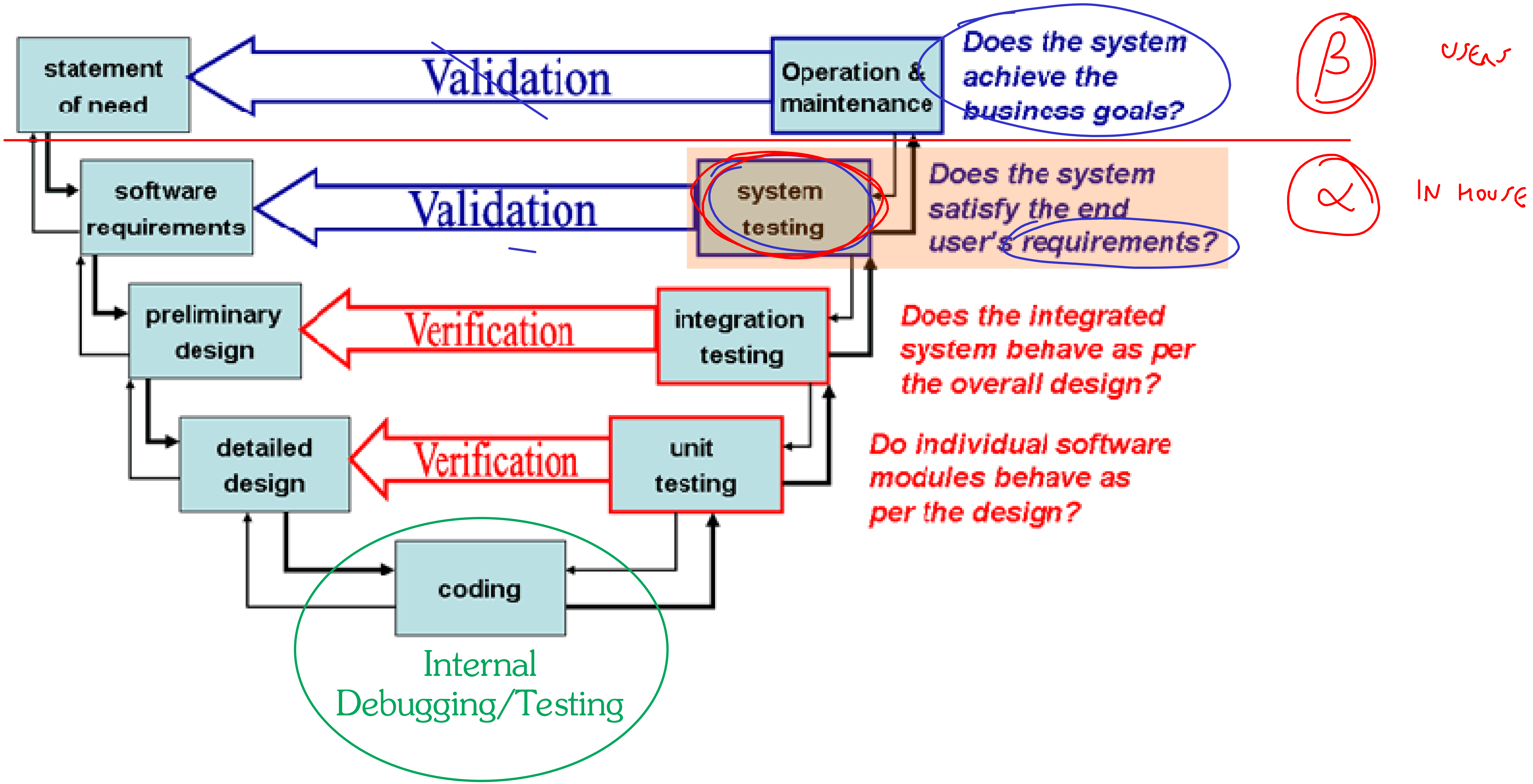
- Lab Tuesday, Thursday, Friday

Lecture 49 – 30 November

# Project Part 1

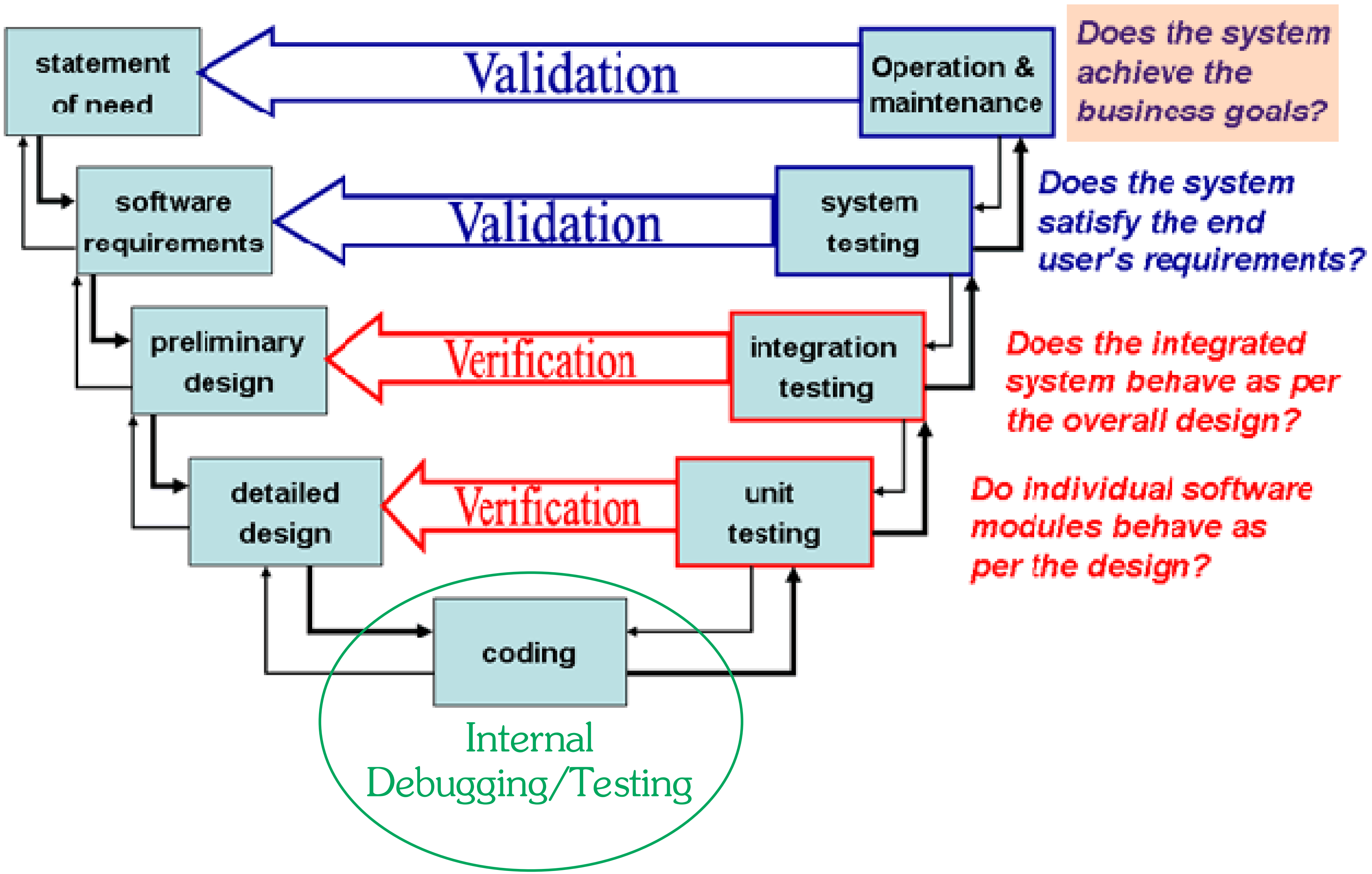
# Project Part 2

# Dynamic Testing



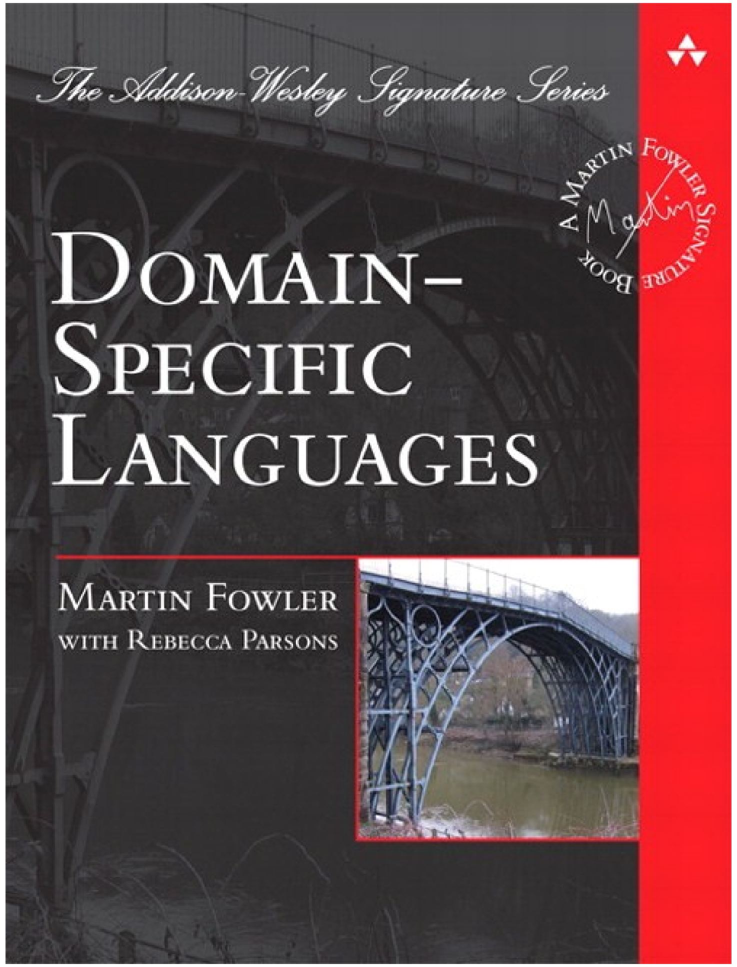

read ➔ understand ➔ plan ➔ execute ➔ verify ➔ reflect

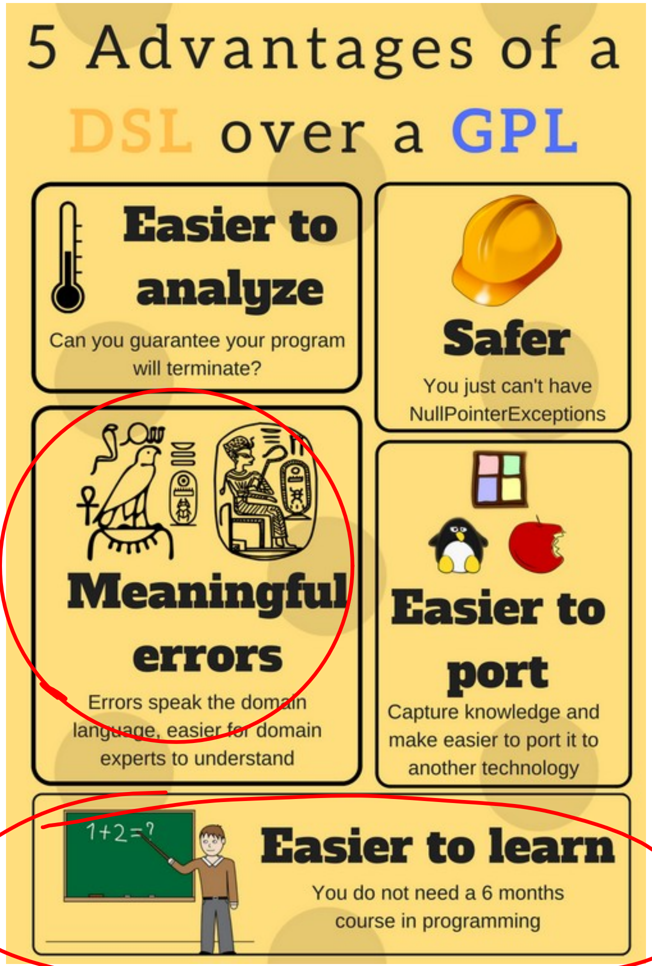
## Dynamic Testing



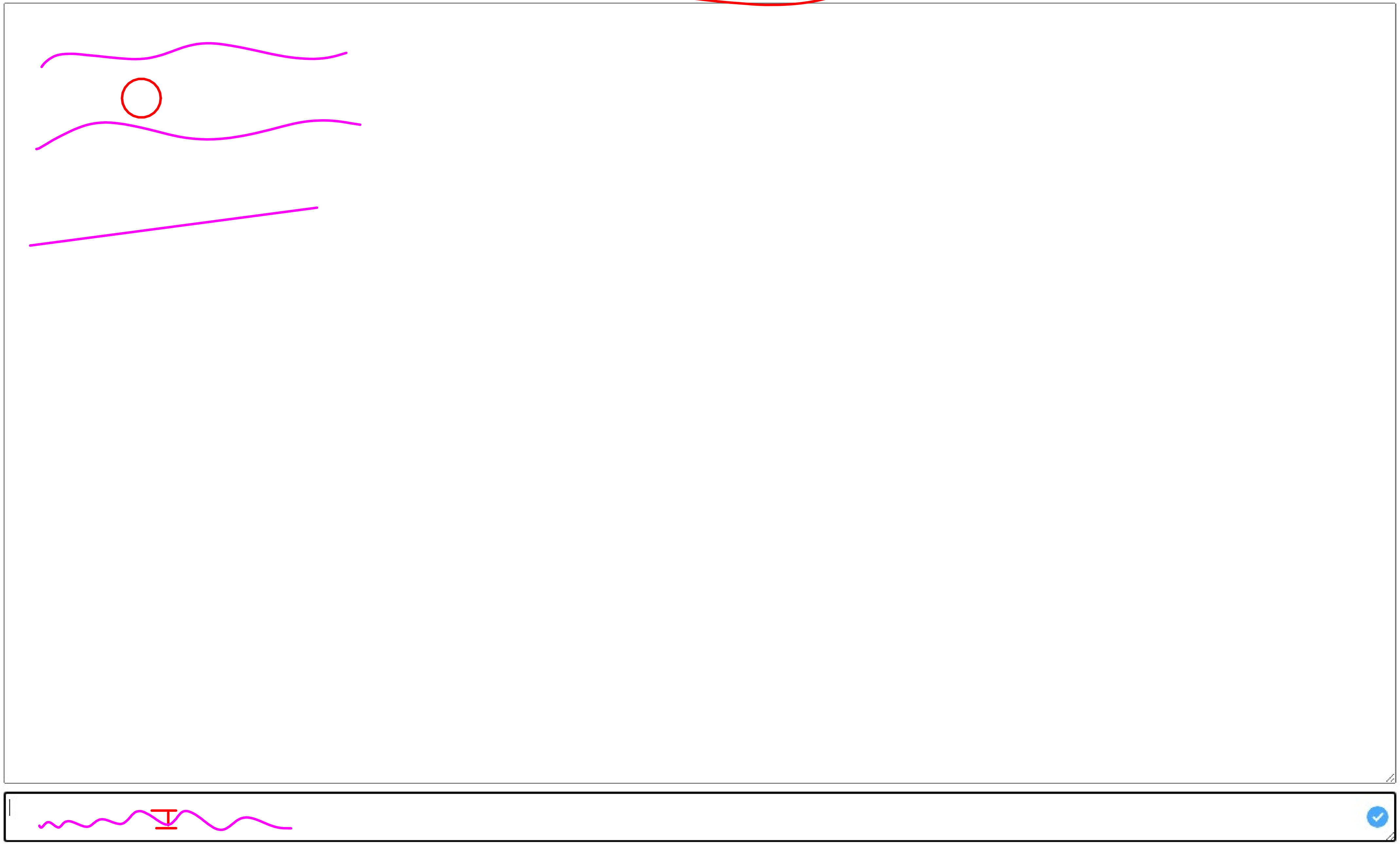

read ➔ understand ➔ plan ➔ execute ➔ verify ➔ reflect

# Domain-Specific Language



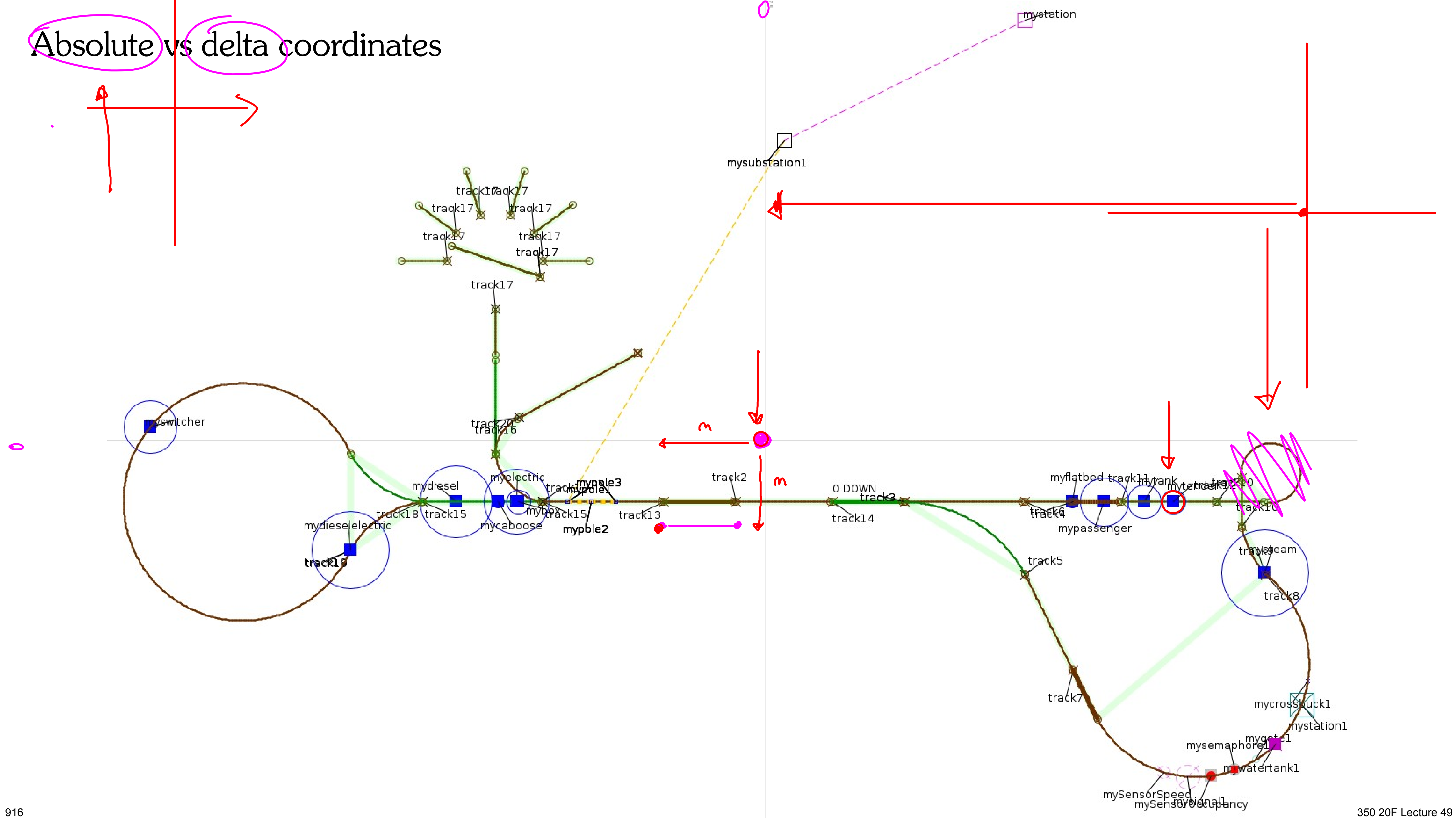
# Console



```
# site
ENABLE LOGIN
DISABLE LOGIN

# maintenance
UPDATE MAINTENANCE ANNUAL DATE date
UPDATE MAINTENANCE ELT DATE date
UPDATE MAINTENANCE GPS [DATE date]
UPDATE MAINTENANCE VOR DATE date
UPDATE MAINTENANCE MAGNETO HOBBS hobbs
UPDATE MAINTENANCE OIL HOBBS hobbs

# messages
LIST MESSAGES
SHOW MESSAGE id
DELETE MESSAGE id
UPDATE MESSAGE id SUBJECT subject
UPDATE MESSAGE id BODY body
UPDATE MESSAGE id PRIORITY priority
UPDATE MESSAGE id UNREAD [USER username]

# applications
LIST APPLICATIONS
SHOW APPLICATION id
DELETE APPLICATION id
APPROVE APPLICATION id
DENY APPLICATION id

# scheduling
LIST BOOKINGS
LIST BOOKINGS DATE dateYM
SHOW BOOKING id
DELETE BOOKING id

# postflights
LIST POSTFLIGHTS
LIST POSTFLIGHTS DATE dateYM
LIST POSTFLIGHTS PENDING [USER username]
SHOW POSTFLIGHT id
DELETE POSTFLIGHT id
UPDATE POSTFLIGHT id COMMENTS comments
UPDATE POSTFLIGHT id HOBBS START hobbs
UPDATE POSTFLIGHT id HOBBS END hobbs
UPDATE POSTFLIGHT id FUEL BEFORE fuel
UPDATE POSTFLIGHT id FUEL AFTER after

# documents
LIST DOCUMENTS
SHOW DOCUMENT id
ADD DOCUMENT TITLE title CATEGORY id [DESCRIPT
DELETE DOCUMENT id
UPDATE DOCUMENT id TITLE title
UPDATE DOCUMENT id DESCRIPTION description
UPDATE DOCUMENT id POSITION position
UPDATE DOCUMENT id DATE date
UPDATE DOCUMENT id REMOVE DATE
SHOW DOCUMENT CATEGORY docid

LIST DOCUMENT CATEGORIES
ADD DOCUMENT CATEGORY name POSITION position
DELETE DOCUMENT CATEGORY id
UPDATE DOCUMENT CATEGORY id POSITION position
UPDATE DOCUMENT CATEGORY id TO CATEGORY name

# users
LIST USERS
SHOW USER username
LOCK USER username
UNLOCK USER username
RETIRE USER username
RESET USER username PASSWORD password
ADD USER username ROLE role
DELETE USER username ROLE role
SWITCH USER username
WELCOME USER username
ENFORCE SCHEDULING POLICY FOR USER username
WAIVE SCHEDULING POLICY FOR USER username
LIST LOGINS

# splash
LIST SPLASHES
SHOW SPLASH id
ADD SPLASH FILENAME filename [DESCRIPTION desc
DELETE SPLASH id
UPDATE SPLASH id FILENAME filename
UPDATE SPLASH id DESCRIPTION description
UPDATE SPLASH id POSITION position

# database
LIST TABLES
SHOW TABLE tablename [ORDER fieldname]
EXECUTE SQL QUERY query
EXECUTE SQL STATEMENT statement

# events
ADD EVENT TITLE string DESCRIPTION string DATE date FROM time TO time
LIST EVENTS
SHOW EVENT id
DELETE EVENT id
UPDATE EVENT id TITLE string
UPDATE EVENT id DESCRIPTION string
UPDATE EVENT id DATE date FROM time TO time

# misc
REBOOT
REFRESH
BACKUP DATABASE
RECOVER DATABASE
LIST PROPERTIES
HELP
```

# Coordinates

# Absolute vs delta coordinates



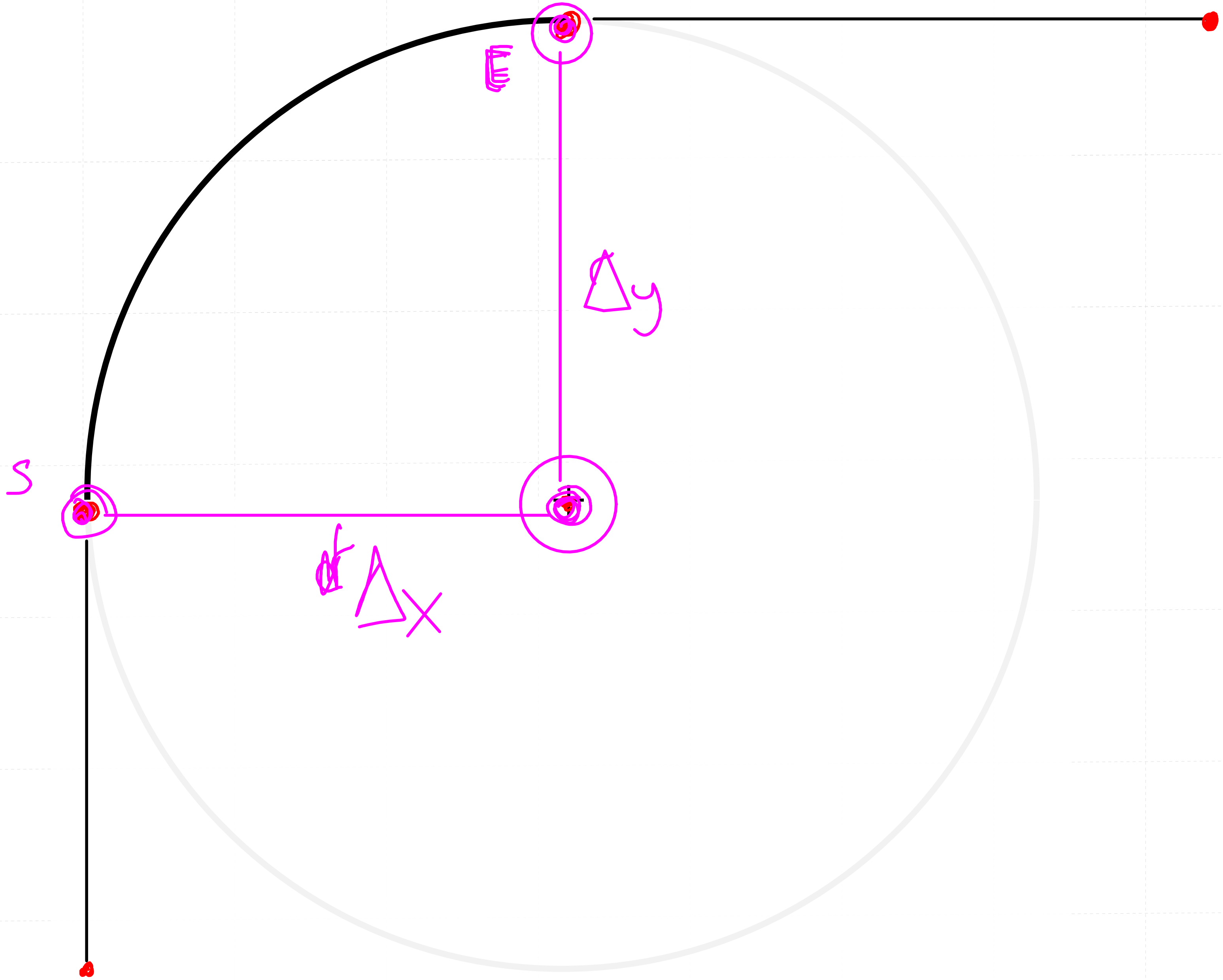
# Architecture

## Architecture is about boundaries

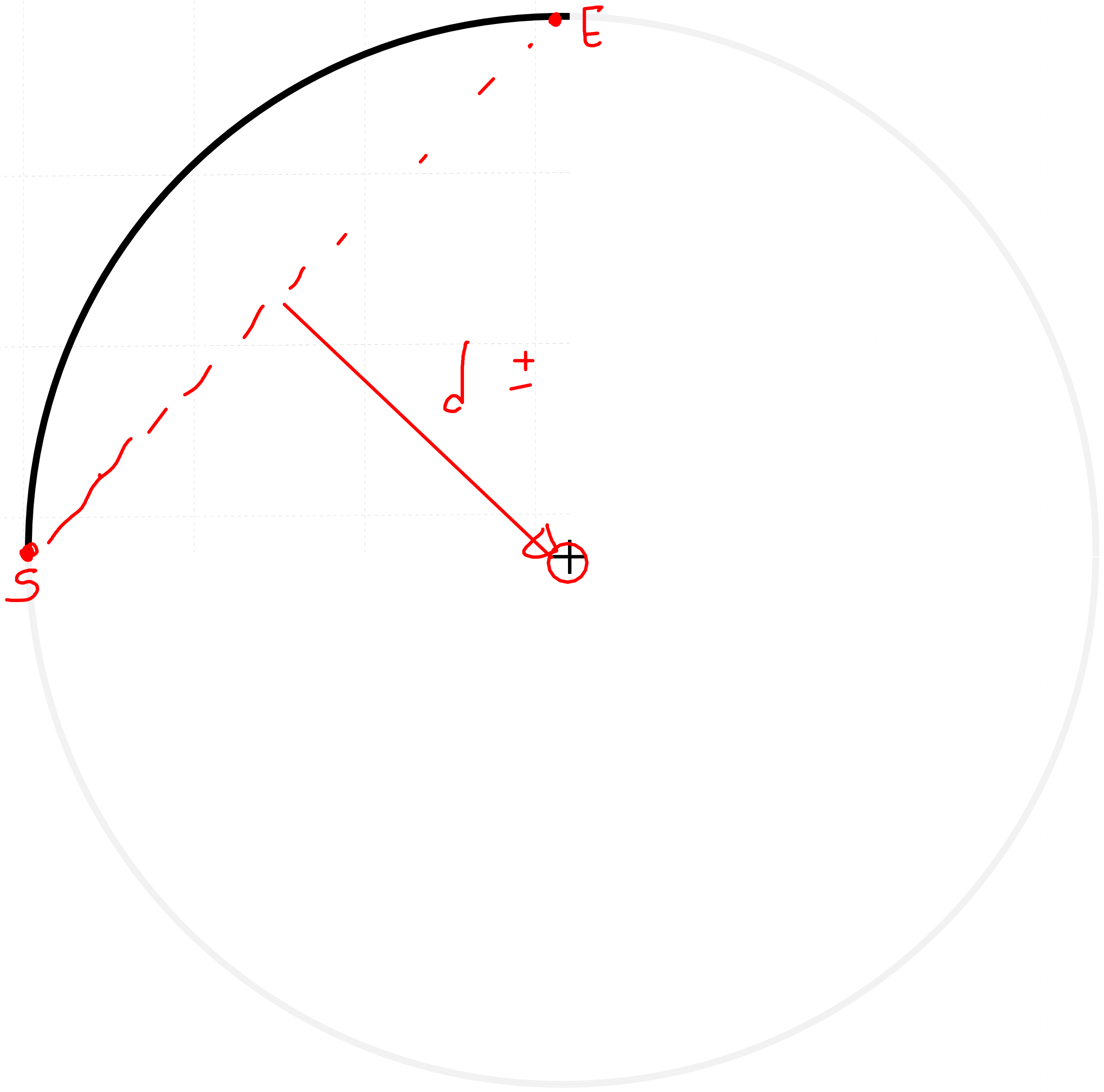
In essence, a system's architecture is what defines the shape of the system. More specifically, a system's architecture defines how the system is divided into components, how those components are arranged, what of kinds boundaries exist between different components and how the components communicate across those boundaries. Basically, it's all about the way we are using boundaries to separate parts of the system that shouldn't know too much about each other.

The purpose of this kind of separation is to make it easier to develop, deploy and maintain the system. Especially the maintenance part is critical, because this is typically the most risky and expensive part. Often, the first version of a system making it to production is only the start, and most of the work will happen after that. Additional requirements will be added, existing functionality will need to be changed, etc. Adequate boundaries will provide the necessary flexibility to make this kind of maintenance possible, allowing the system to grow without exponentially increasing the work needed to add or adjust a piece of functionality.

https://convincedcoder.com/2019/04/27/Software-architecture-boundaries/




```java
 * @param reference - the reference coordinates
 * @param deltaStart - the delta position of the start of the path
 * @param deltaEnd - the delta position of the end of the path
 * @param deltaOrigin - the origin of the arc if it were projected as a complete circle
 */
public ShapeArc(final CoordinatesWorld reference, final CoordinatesDelta deltaStart, final CoordinatesDelta deltaEnd, final CoordinatesDelta deltaOrigin)
```



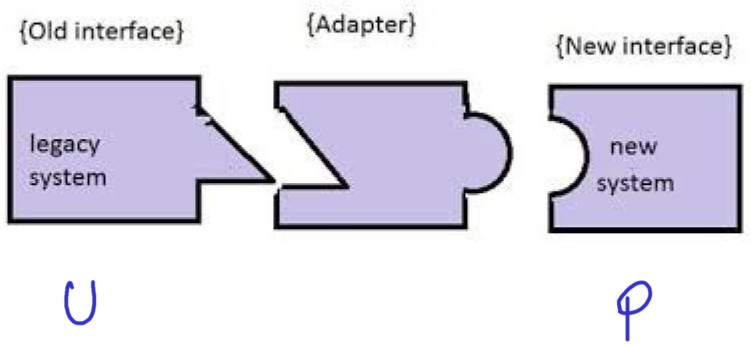

```
 * @param distanceOrigin - the distance orthogonal to line between start and end, may be negative
 */
public ShapeArc(final CoordinatesWorld reference, final CoordinatesDelta deltaStart, final CoordinatesDelta deltaEnd, final double distanceOrigin)
```

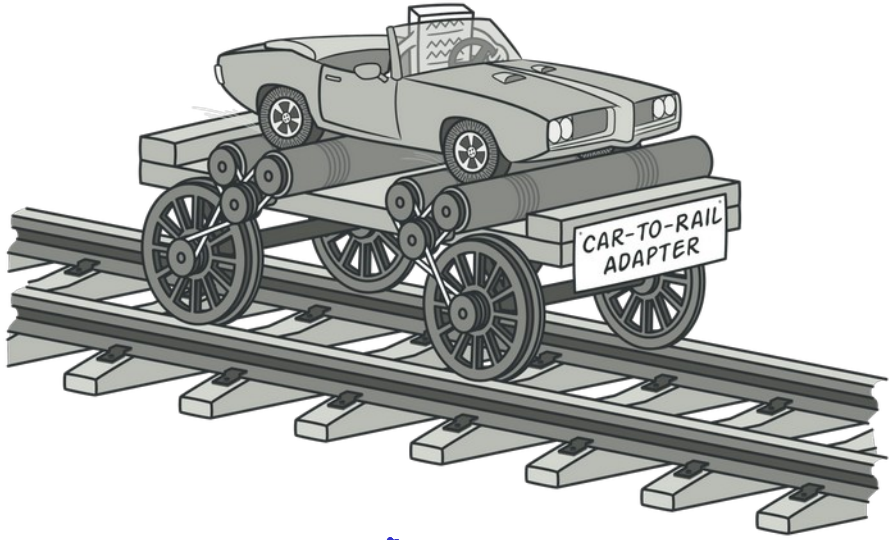
| | | |
|---|---|---|
| 48 | CREATE TRACK SWITCH TURNOUT id1 REFERENCE ( coordinates_world \| ( '$' id2 ) ) STRAIGHT DELTA START coordinates_delta1 END coordinates_delta2 CURVE DELTA START coordinates_delta3 END coordinates_delta4 DISTANCE ORIGIN number | CommandCreateTrackSwitchTurnout |
| | Creates turnout switch track **id1** with the straight segment starting at **coordinates_delta1** meters and ending at **coordinates_delta2** meters from **coordinates_world** or **id2** and the curved segment starting at **coordinates_delta3** meters and ending at **coordinates_delta4** meters with the orthogonal to the line connecting the start and end of the curved segment **number** meters long. | |

```
public CommandCreateTrackSwitchTurnout(final String id,
                                       final CoordinatesWorld reference,
                                       final CoordinatesDelta delta1Start,
                                       final CoordinatesDelta delta1End,
                                       final CoordinatesDelta delta2Start,
                                       final CoordinatesDelta delta2End,
                                       final CoordinatesDelta delta2Origin)
```

| | |
|---|---|
| 48 | CREATE TRACK SWITCH TURNOUT id1 REFERENCE ( coordinates_world \| ( '$' id2 ) ) STRAIGHT DELTA START coordinates_delta1 END coordinates_delta2 CURVE DELTA START coordinates_delta3 END coordinates_delta4 DISTANCE ORIGIN number — CommandCreateTrackSwitchTurnout |
| | Creates turnout switch track **id1** with the straight segment starting at **coordinates_delta1** meters and ending at **coordinates_delta2** meters from **coordinates_world** or **id2** and the curved segment starting at **coordinates_delta3** meters and ending at **coordinates_delta4** meters with the orthogonal to the line connecting the start and end of the curved segment **number** meters long. |

# impedance mismatch

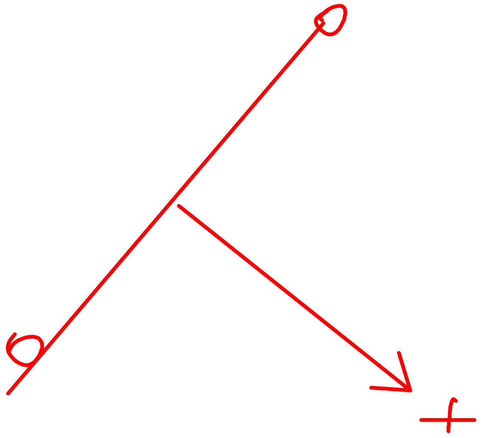


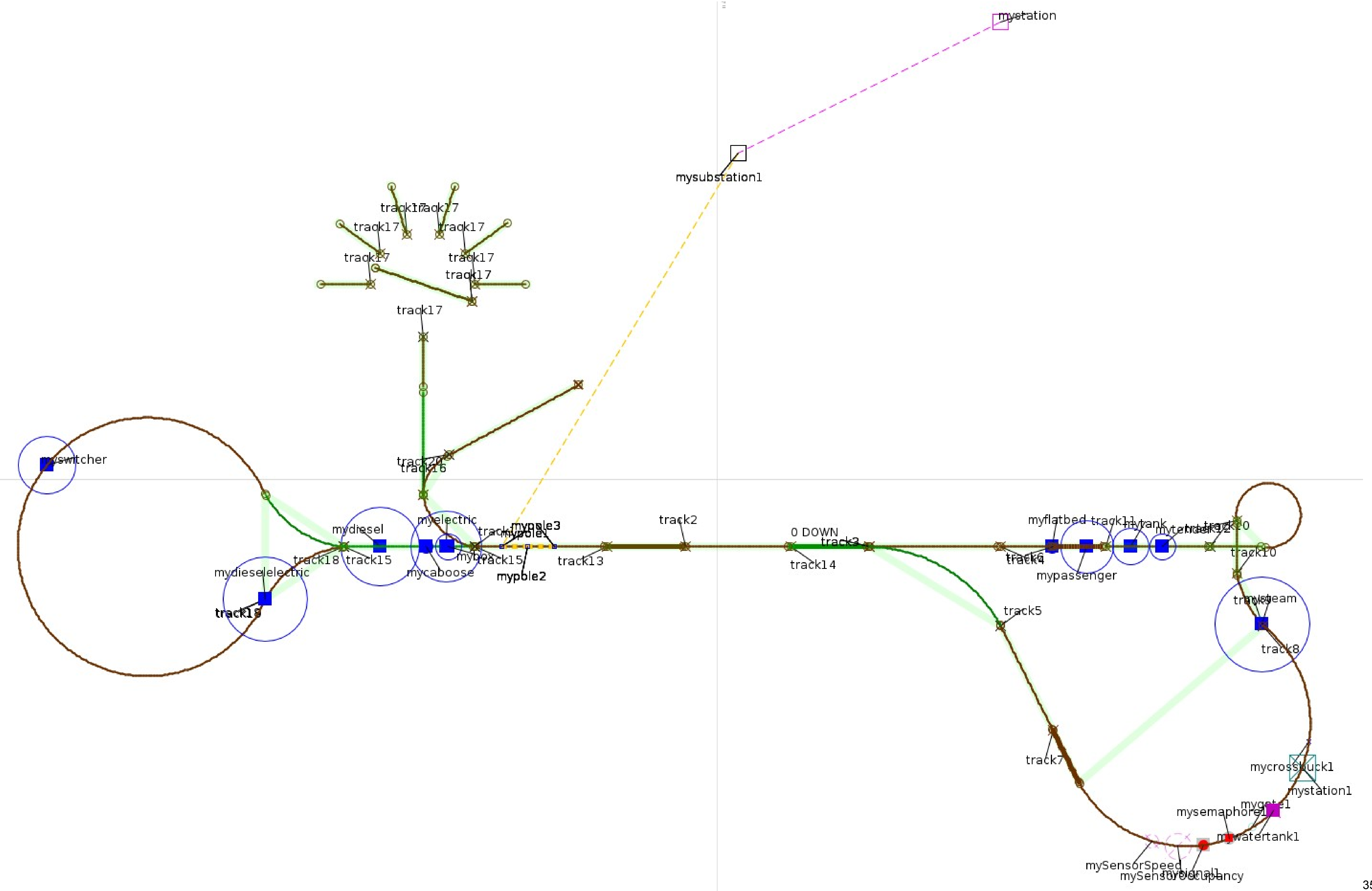

```
public CommandCreateTrackSwitchTurnout(final String id,
                                       final CoordinatesWorld reference,
                                       final CoordinatesDelta delta1Start,
                                       final CoordinatesDelta delta1End,
                                       final CoordinatesDelta delta2Start,
                                       final CoordinatesDelta delta2End,
                                       final CoordinatesDelta delta2Origin)
```

| 48 | `CREATE TRACK SWITCH TURNOUT id1 REFERENCE ( coordinates_world \| ( '$' id2 ) ) STRAIGHT DELTA START coordinates_delta1 END coordinates_delta2 CURVE DELTA START coordinates_delta3 END coordinates_delta4 DISTANCE ORIGIN number` | `CommandCreateTrackSwitchTurnout` |
|---|---|---|
| | Creates turnout switch track `id1` with the straight segment starting at `coordinates_delta1` meters and ending at `coordinates_delta2` meters from `coordinates_world` or `id2` and the curved segment starting at `coordinates_delta3` meters and ending at `coordinates_delta4` meters with the orthogonal to the line connecting the start and end of the curved segment `number` meters long. | |

```
ShapeArc
public static CoordinatesDelta calculateDeltaOrigin(final CoordinatesWorld reference,
                                                    final CoordinatesDelta deltaStart,
                                                    final CoordinatesDelta deltaEnd,
                                                    final double distanceOrigin)
{
   double halfdistance = (deltaStart.calculateDistance(deltaEnd) / 2);

   Angle angle = deltaStart.calculateBearing(deltaEnd);

   CoordinatesDelta deltaHalfway = deltaStart.calculateTarget(angle, halfdistance);

   Angle angleRight = (distanceOrigin >= 0 ? angle.add(Angle.ANGLE_090) : angle.subtract(Angle.ANGLE_090));

   double distanceOrthogonal = Math.abs(distanceOrigin);

   CoordinatesDelta deltaOrigin = deltaHalfway.calculateTarget(angleRight, distanceOrthogonal);

   return deltaOrigin;
}
```



```
public CommandCreateTrackSwitchTurnout(final String id,
                                       final CoordinatesWorld reference,
                                       final CoordinatesDelta delta1Start,
                                       final CoordinatesDelta delta1End,
                                       final CoordinatesDelta delta2Start,
                                       final CoordinatesDelta delta2End,
                                       final CoordinatesDelta delta2Origin)
```



mystation
mysubstation1
track17
track20
track16
myswitcher
mydiesel
myelectric
mypole3
track2
0 DOWN
track3
myflatbed
track11
mytank
track10
track18
track15
mycaboose
mybox
mypole2
track13
track14
track4
mypassenger
mydieselelectric
track5
mysteam
track8
track7
mycrossbuck1
mystation1
mysemaphore1
mywatertank1
mySensorSpeed
mySensorOccupancy

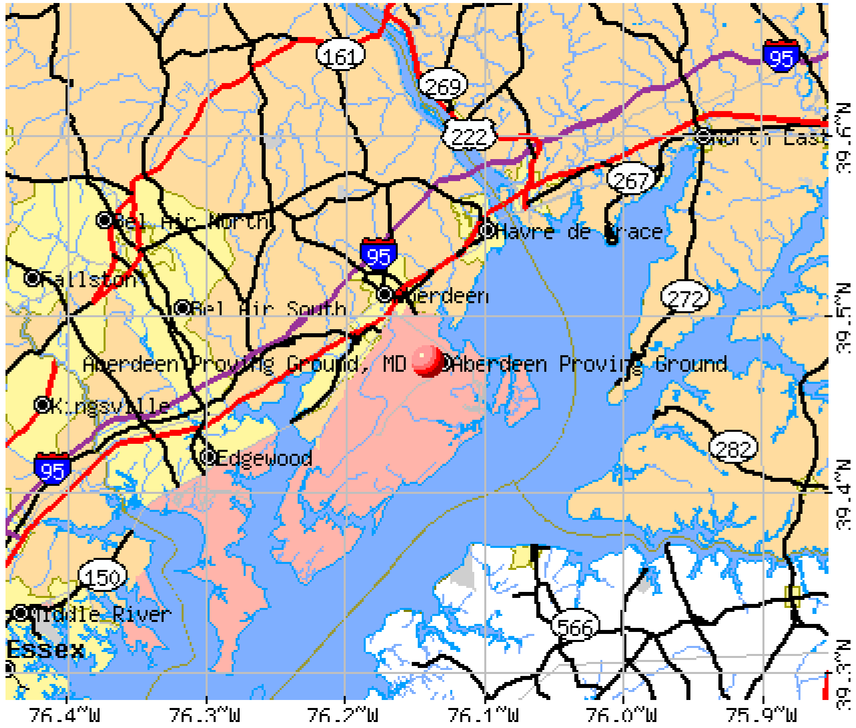
# Real-World Test Report

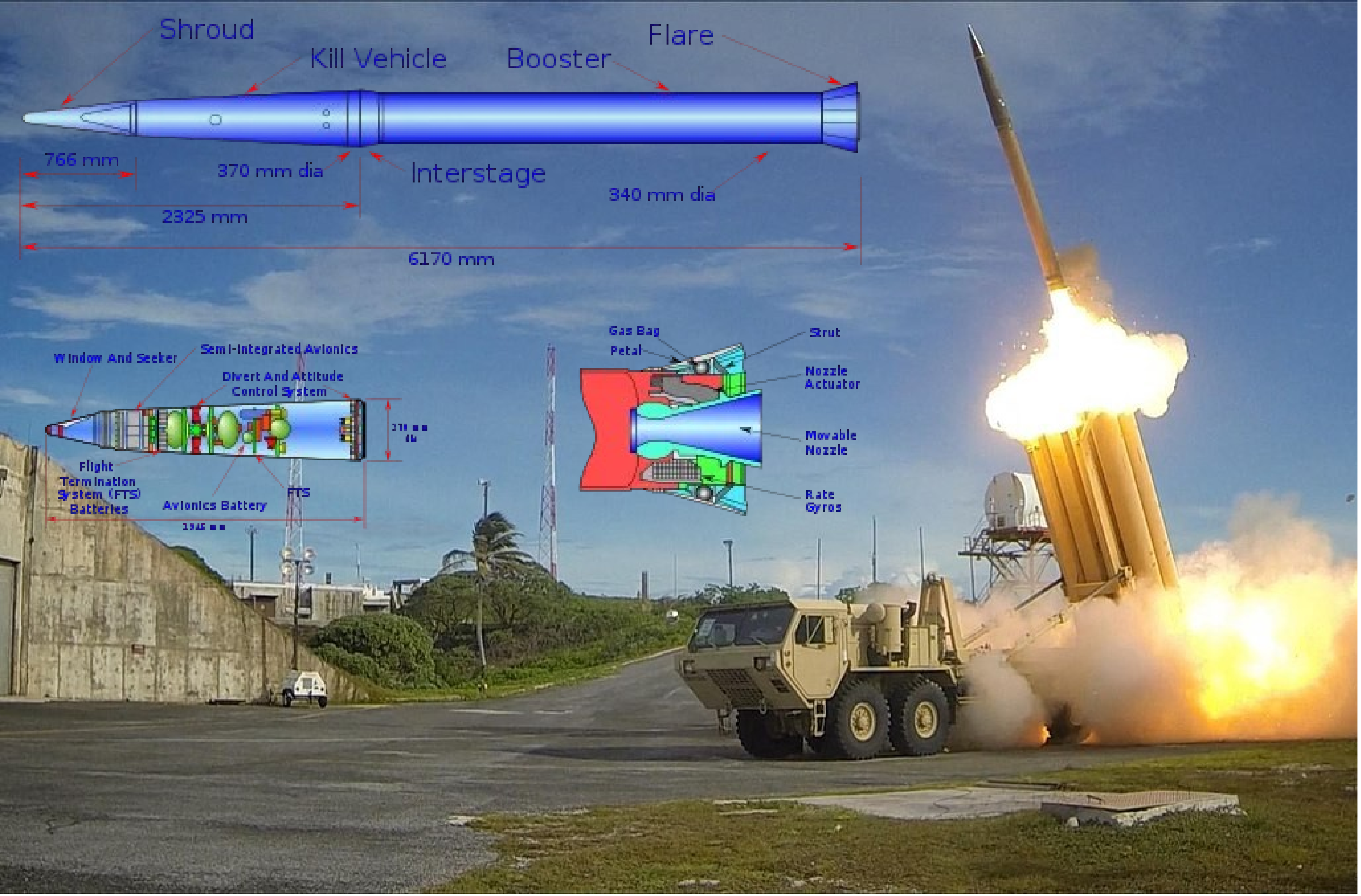

NEW MEXICO
WHITE SANDS
MISSILE RANGE


U.S.ARMY
WWW.EPG.ARMY.MIL
U.S. ARMY ELECTRONIC PROVING GROUND
1877
FORT
HUACHUCA

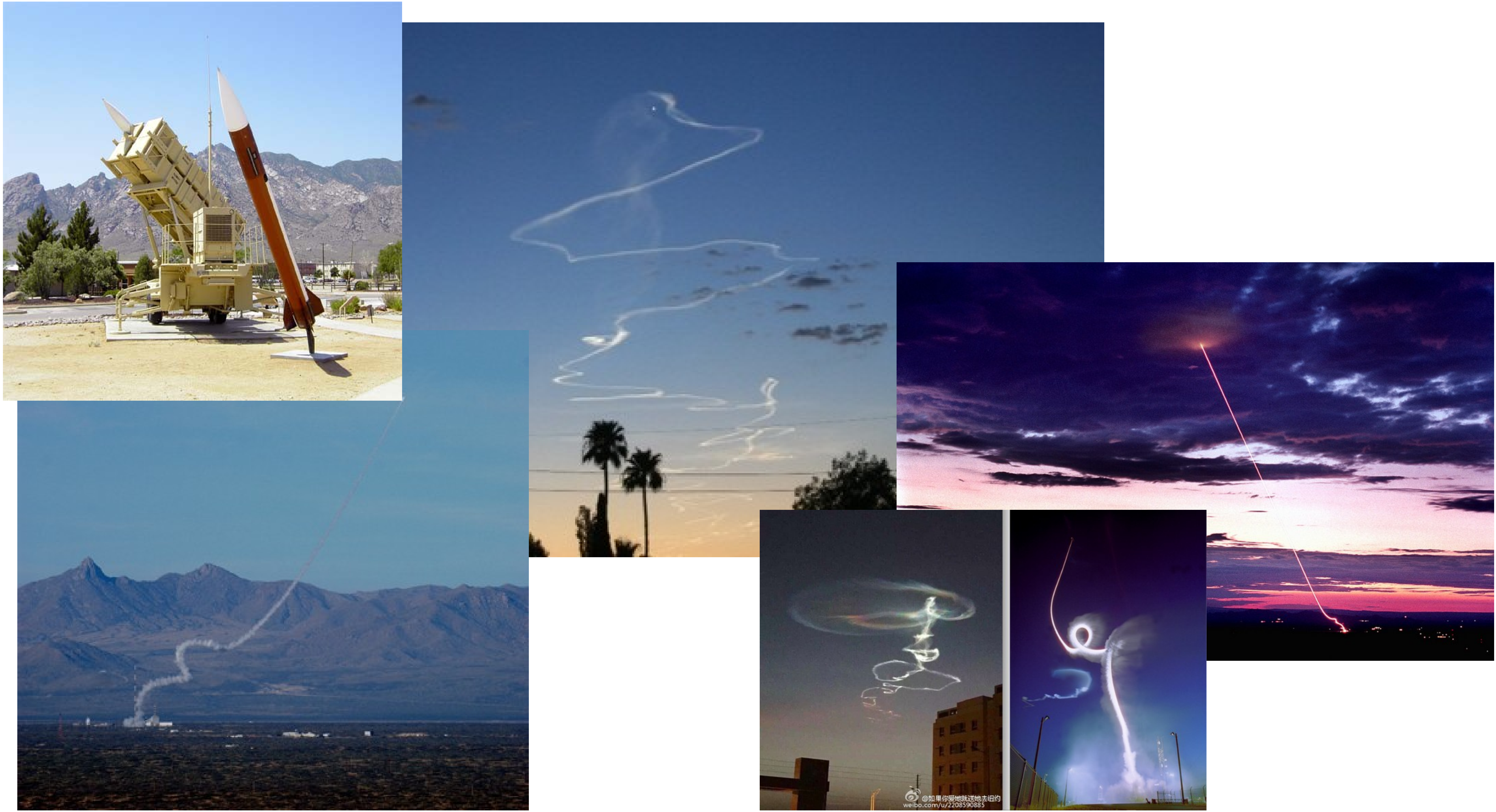
$100M


Maryland
Allentown
Philadelphia
Aberdeen Proving Ground
Baltimore
Washington
Richmond
Virginia Beach
© 2005 Sperling's BestPlaces

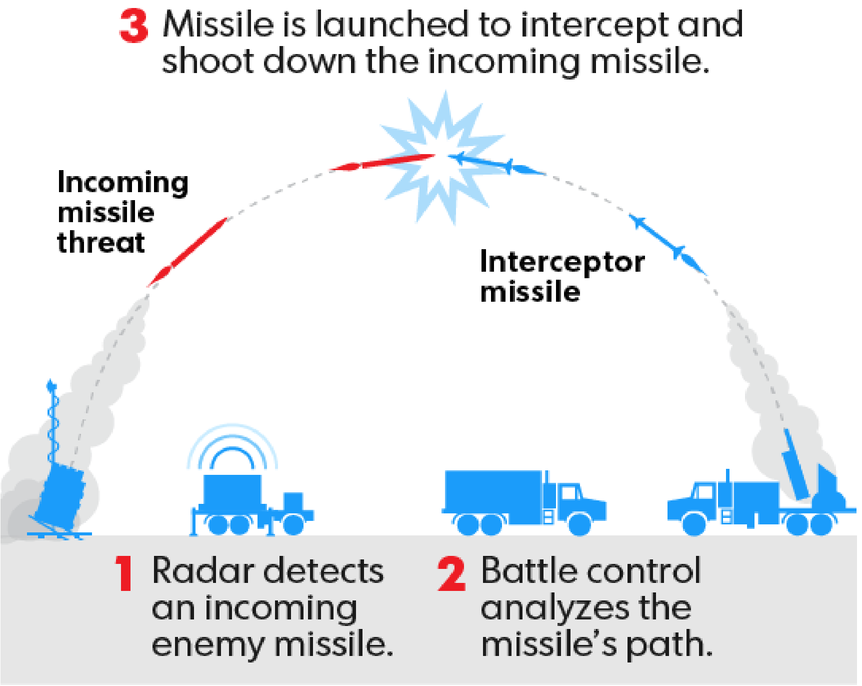

Bel Air North
Havre de Grace
Fallston
Bel Air South
Aberdeen
Aberdeen Proving Ground, MD
Aberdeen Proving Ground
Edgewood
Middle River
Essex
North East
76.4°W
76.3°W
76.2°W
76.1°W
76.0°W
75.9°W

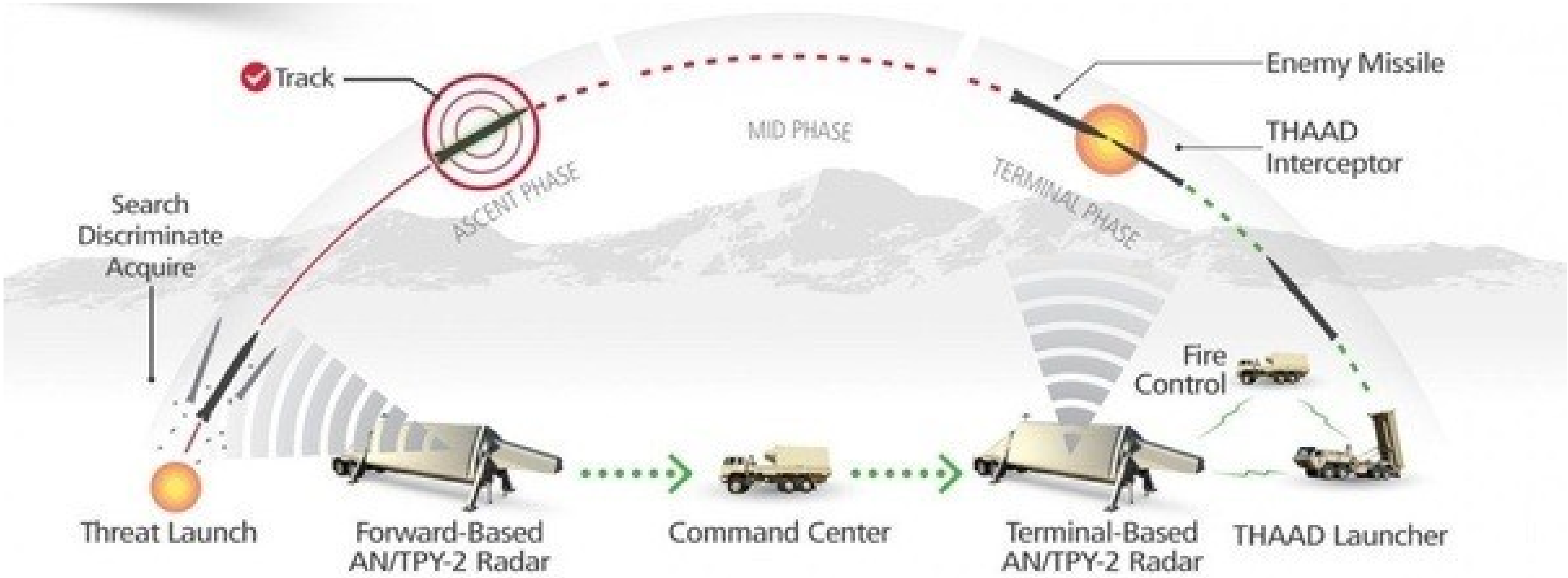


Shroud
Kill Vehicle
Booster
Flare
766 mm
370 mm dia
Interstage
340 mm dia
2325 mm
6170 mm
Window And Seeker
Semi-Integrated Avionics
Divert And Attitude
Control System
Flight
Termination
System (FTS)
Batteries
Avionics Battery
FTS
Gas Bag
Petal
Strut
Nozzle
Actuator
Movable
Nozzle
Rate
Gyros

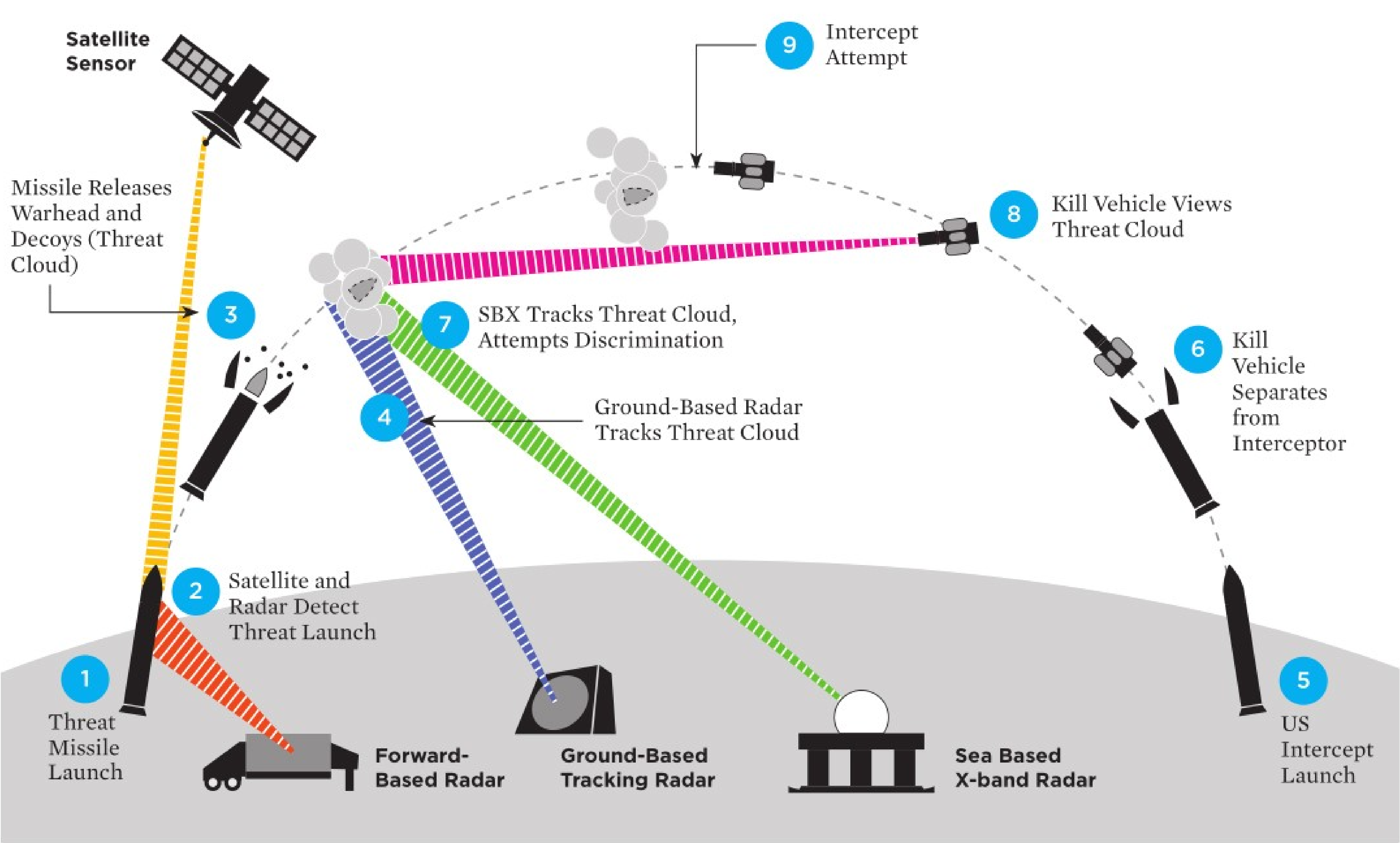


# HOW THE TERMINAL HIGH-ALTITUDE AREA DEFENSE SYSTEM (THAAD) WORKS



## Anatomy of an Intercept




*The GMD system involves a complex, global network of components. The launch of the threat missile (1) is detected by forward-based radars, if present, and satellite-based infrared sensors (2). The threat missile releases its warhead and decoys (in this example the decoys are balloons, and a balloon contains the warhead; together they are referred to as the "threat cloud") (3), and the ground-based radar begins tracking the threat cloud (4). Based on information from this radar, the GMD system launches one or more interceptors (5), each of which releases a kill vehicle (6). If a discrimination radar, such as the Sea Based X-band Radar, is in place it will observe the threat cloud to try to determine which object is the warhead (7) and pass this information to the kill vehicle. The kill vehicle also observes the threat cloud to attempt to determine which object is the warhead (8). It then steers itself into the path of the chosen object and attempts to destroy it with the force of impact (9).*

The final debris field is a composite of all the runs. When it is decomposed into individual runs, each should exhibit high accuracy and precision. Figure 4 illustrates the correct behavior for the target.

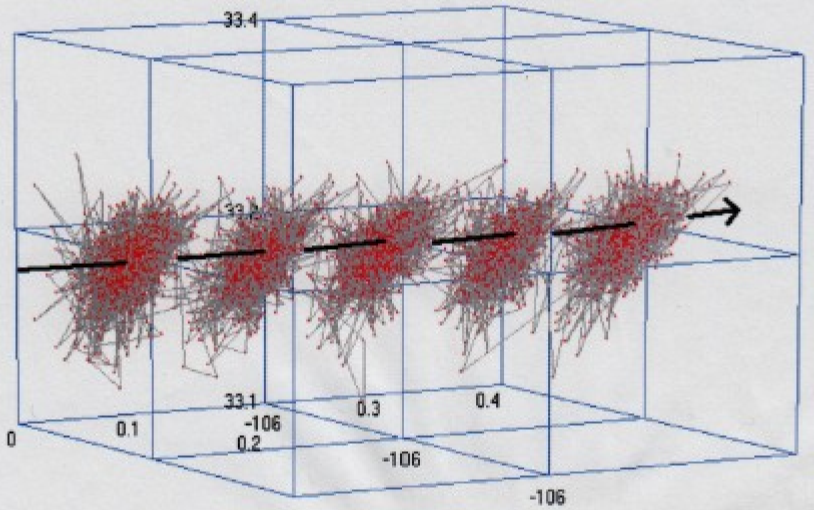
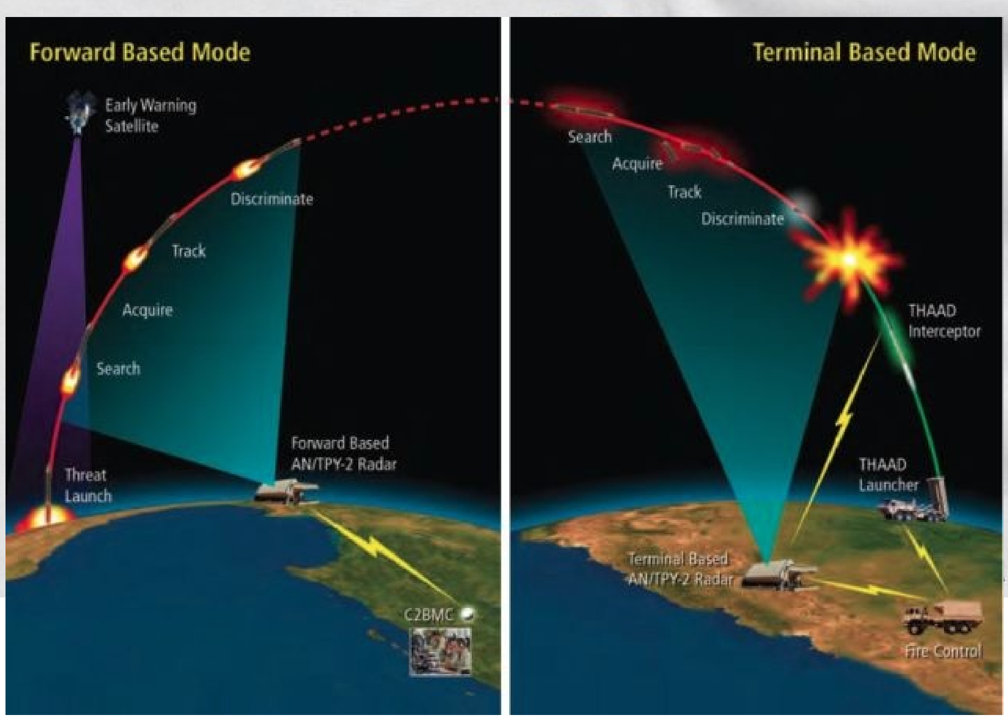
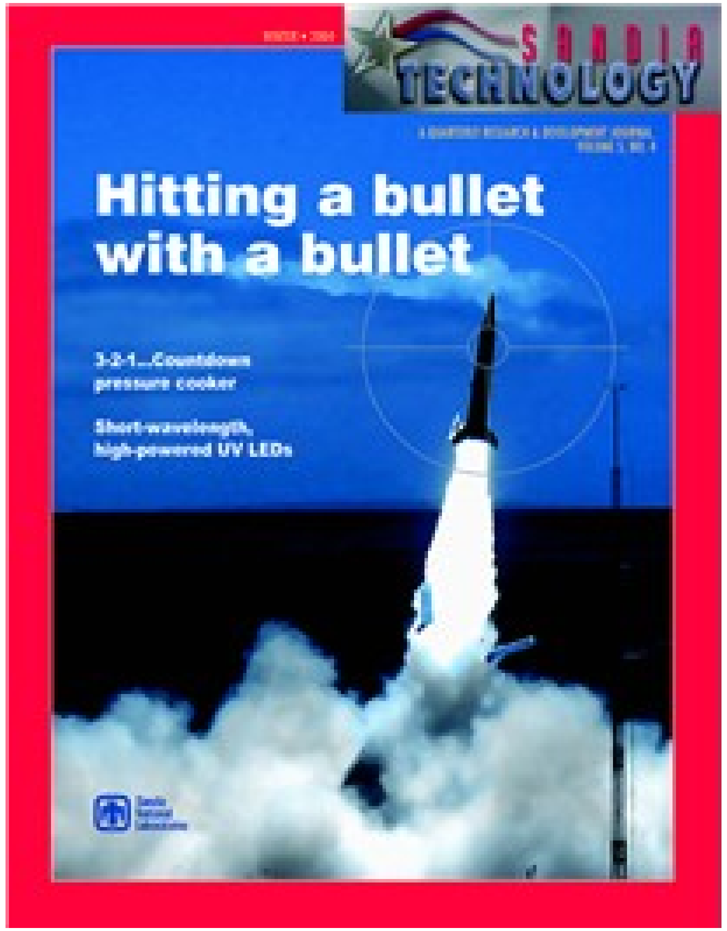



Figure 4: Decomposed Runs for Target Debris Field, Arrow Faces Ground

Figure 5 illustrates the incorrect behavior for the interceptor, where the clusters exhibit high variation along a single axis. It is important to note that the *distribution* of debris within each cluster is consistent with the initial conditions; only the *location* of each is incorrect.




# Analysis of Debris-Propagation Failure Attributed to KIDD 4.1

**First Revision, 18 March 2003**

## Problem Statement

KDAT uses the KIDD Kernel 4.1 impactor program to simulate an initial debris field created by the collision of an interceptor with a target vehicle. It relies on Monte Carlo analysis of many separate, independent runs that are later combined to generalize the aggregate results. This process appears to work correctly on the first run. On subsequent runs, however, pronounced cumulative drift of debris consistently away from the expected area of localization is observed. The combined analysis of all runs produces a skewed distribution over a large, relatively sparsely populated elliptical debris field. A correct analysis should produce a Gaussian distribution over a densely populated circularly clustered field. This behavior is observed only for the interceptor.

## Initial Conditions

As specified in the attached `kidd.inp` and `kidd.fil` files, a 100-kilogram interceptor lethally engages a boosterless 1,000-kilogram target at an altitude of 117 kilometers and roughly a 63-degree angle of convergence.

Unless otherwise specified, the following analysis is based on 5 Monte Carlo simulations with different seeds.

## Expected Results

The initial impact is calculated by KIDD. The resulting fragmentation cluster is then propagated to the ground to generate two debris fields, one for the remnants of each vehicle. Figure 1 illustrates presumably[1] correct debris fields that are expected from the initial conditions. Each cluster represents the distribution of 11-foot-pound debris.

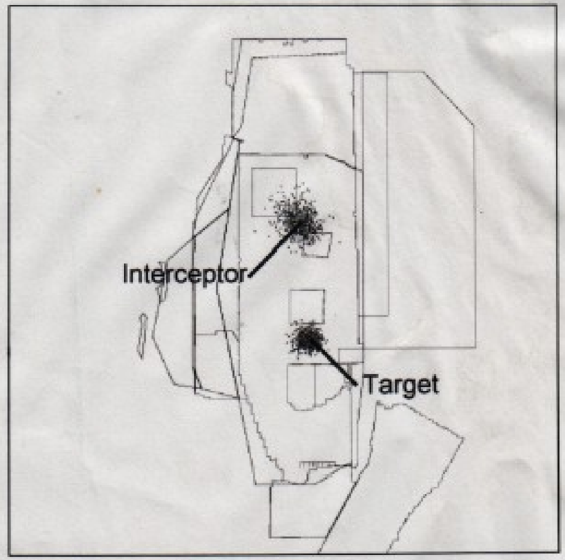



Figure 1: Correct Plot

[1] Flight-safety experts indicated that the fields are consistent with both the initial conditions and with output from another impactor program, IMPACT.



The final debris field is a composite of all the runs. When it is decomposed into individual runs, each should exhibit high accuracy and precision. Figure 4 illustrates the correct behavior for the target.

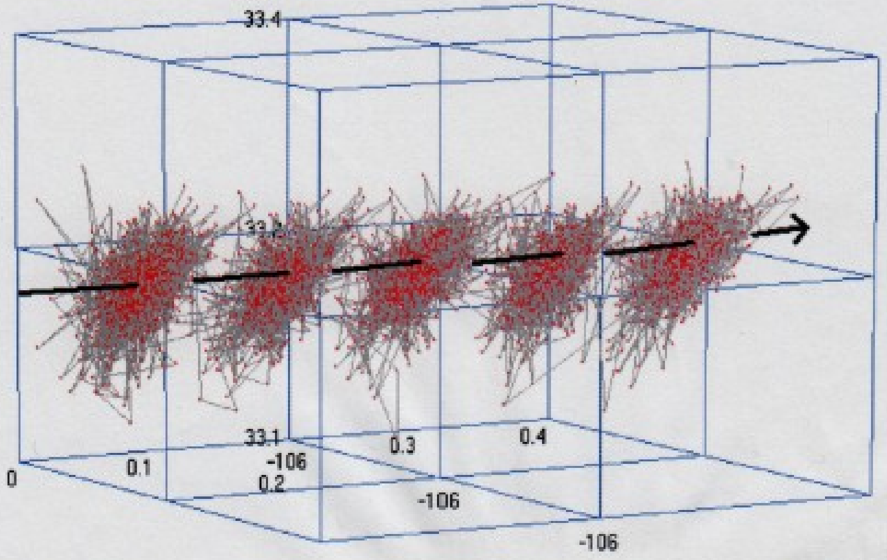



Figure 4: Decomposed Runs for Target Debris Field, Arrow Faces Ground

Figure 5 illustrates the incorrect behavior for the interceptor, where the clusters exhibit high variation along a single axis. It is important to note that the *distribution* of debris within each cluster is consistent with the initial conditions; only the *location* of each is incorrect.

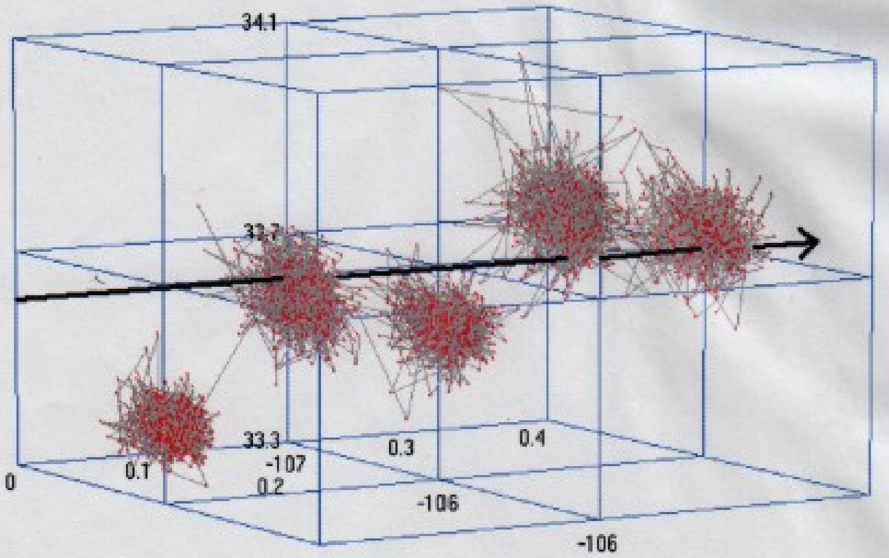



Figure 5: Decomposed Runs for Interceptor Debris Field, Arrow Faces Ground