MODELS TO CODE

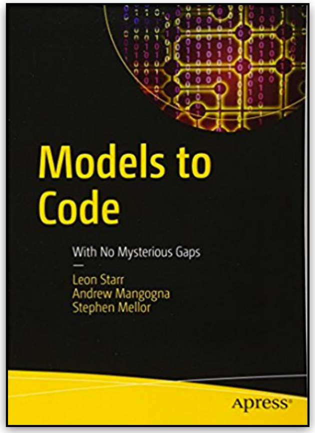
(with no mysterious gaps)

Leon Starr *leon_starr@modelint.com*

@leon_starr

modelint

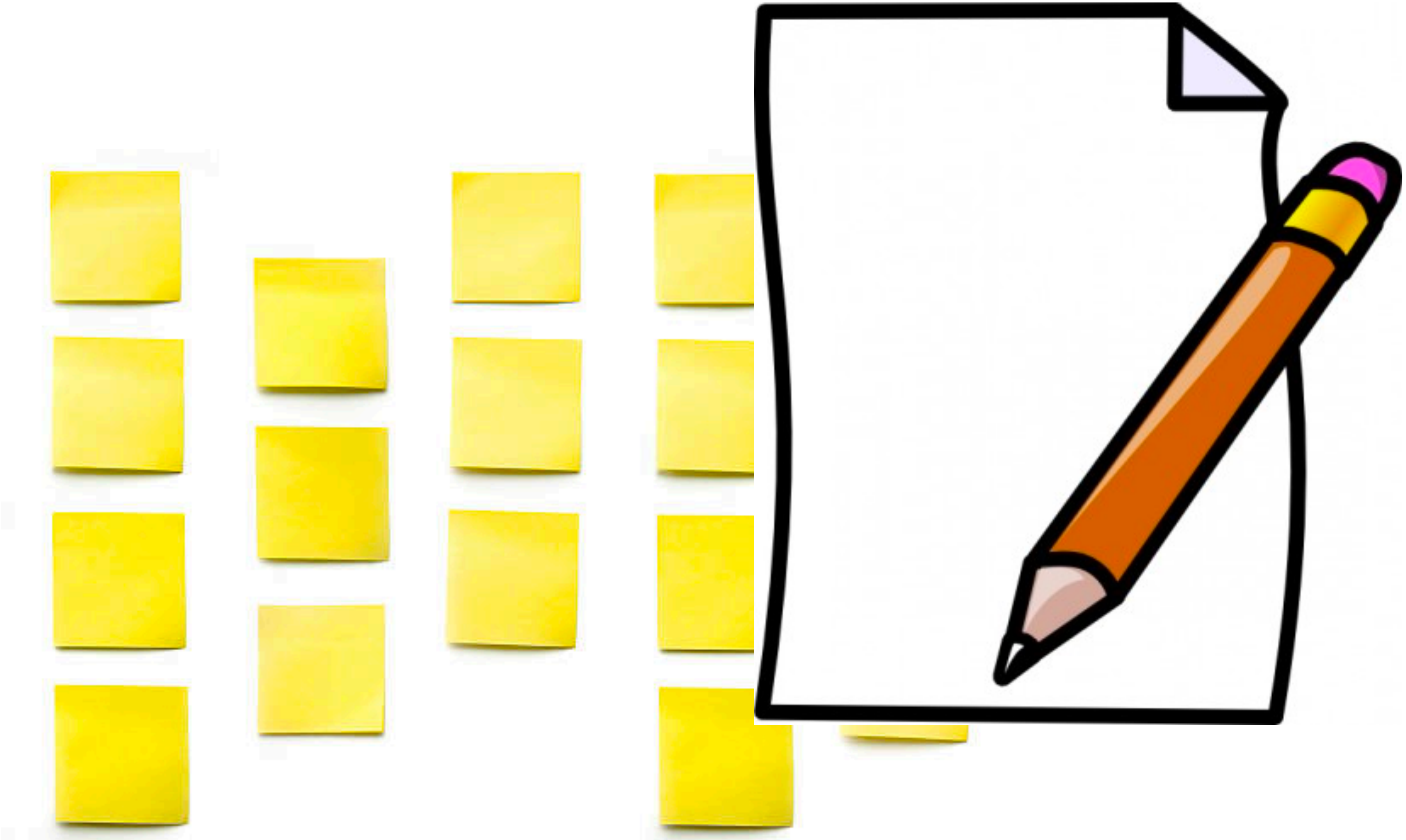
Model Integration LLC

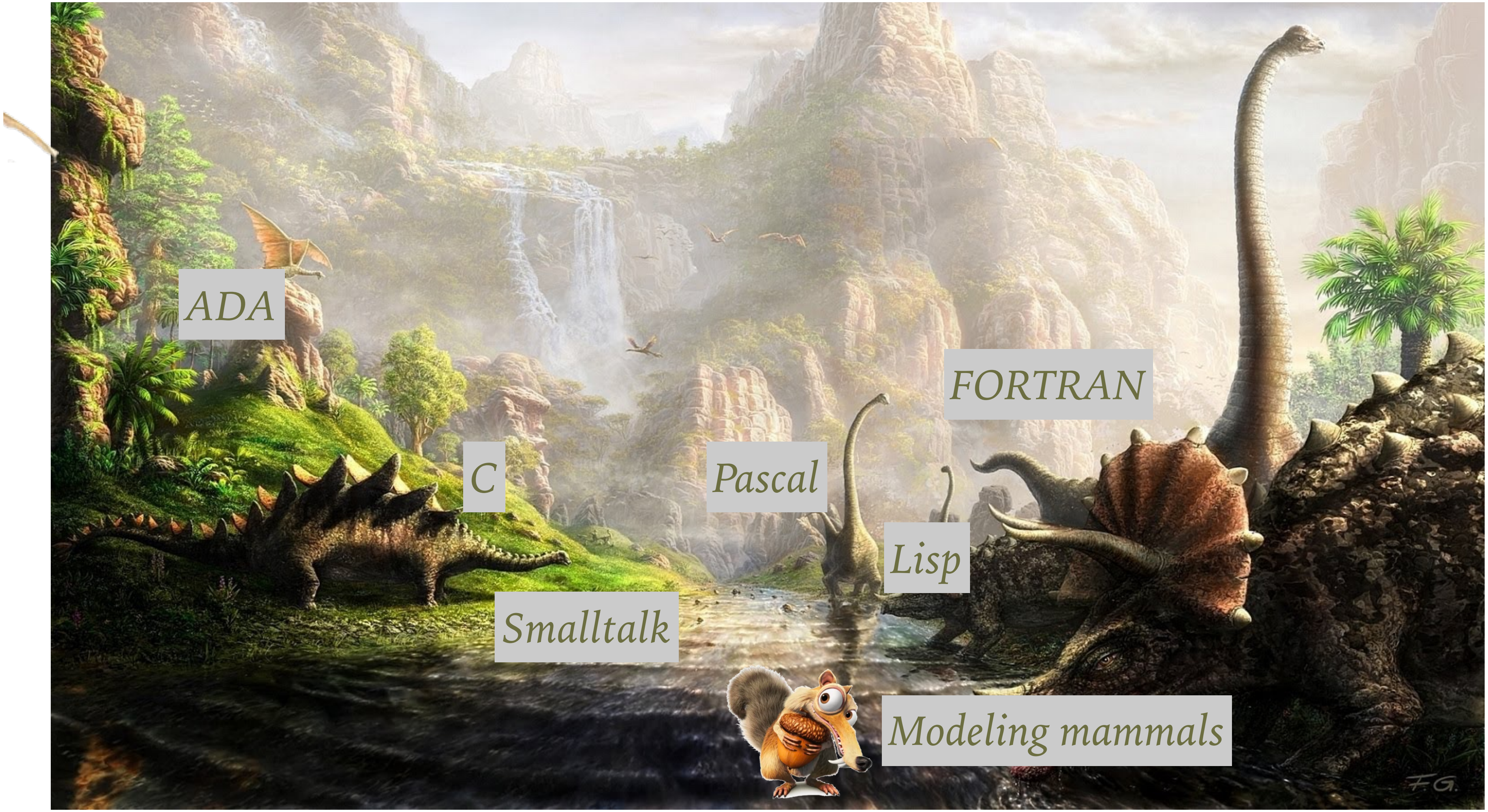


www.modelint.com

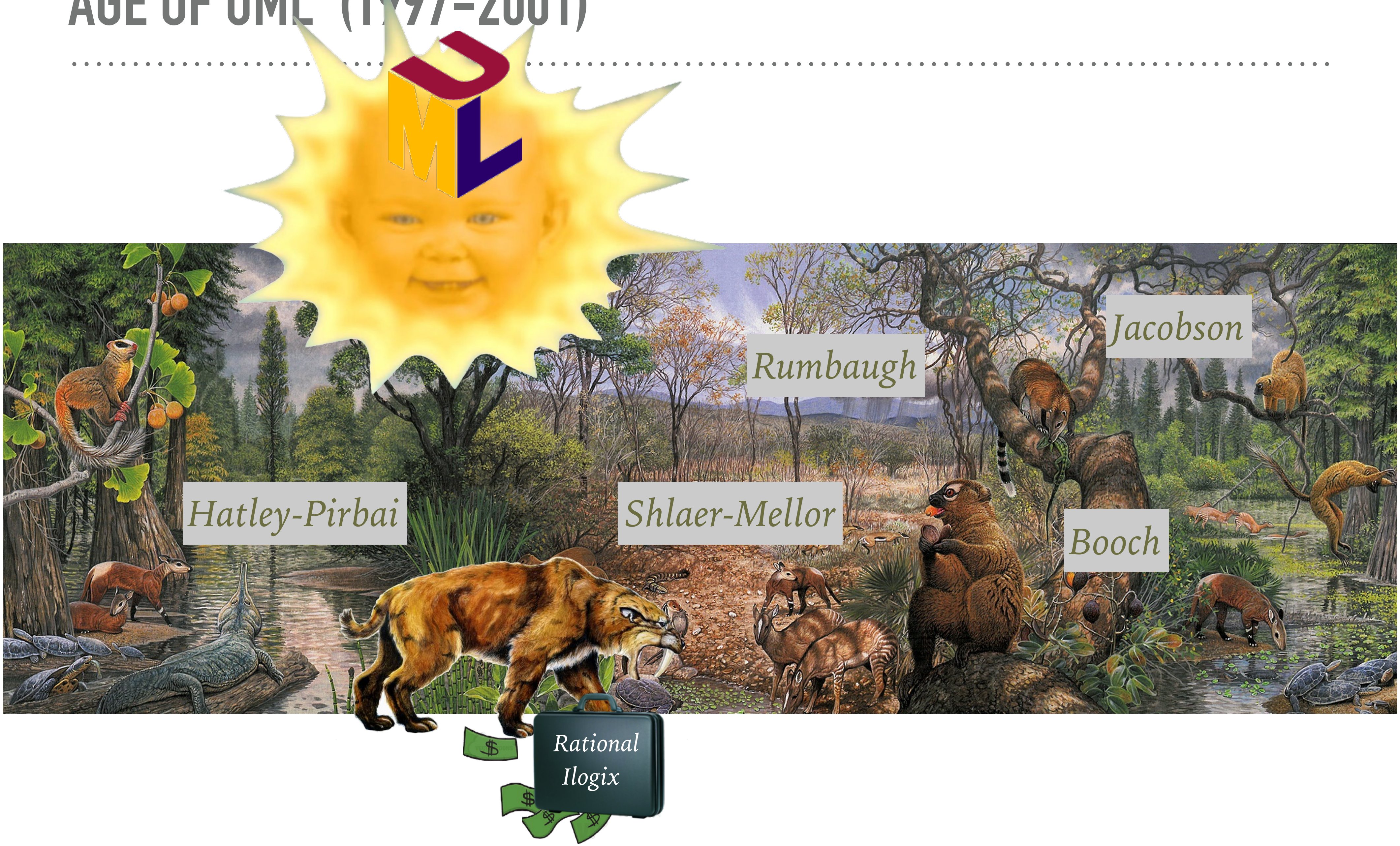
WHY ISN'T EVERYONE MODELING?

MY EARLY DAYS OF MODELING




ADA
FORTRAN
C
Pascal
Lisp
Smalltalk
Modeling mammals

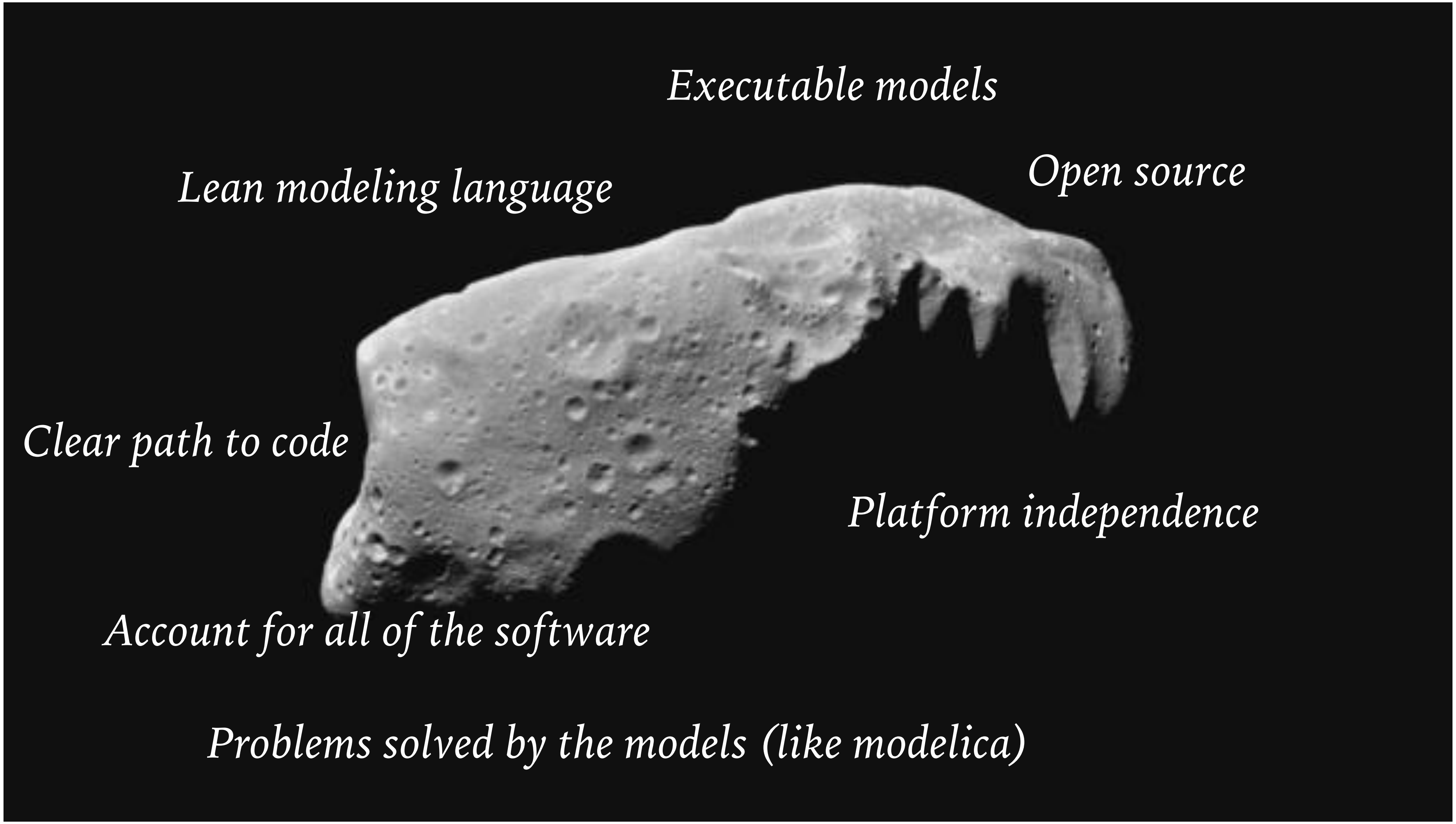
AGE OF UML (1997–2001)



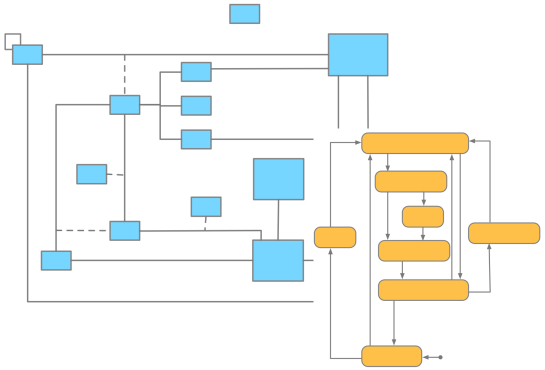
POST UML AGILE ICE AGE (2002–2013)

THINGS ARE WARMING UP AGAIN (2014–PRESENT DAY)

THE DINOSAUR KILLING ASTEROID IS ON ITS WAY



MONOLITHIC TOOL APPROACH



Executable UML

Code!

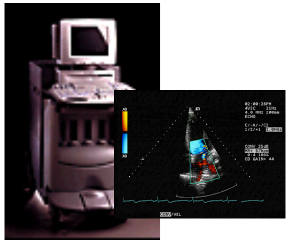
(hope you like it)

Unlikely to work for many real-world platforms

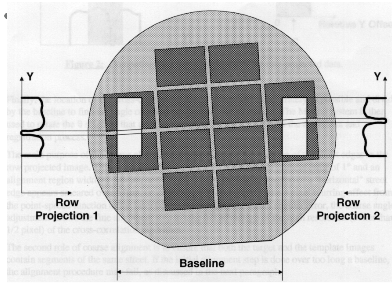
MODELING ON CHALLENGING PLATFORMS

Factory Automation

Video Effects



Diagnostic Ultrasound

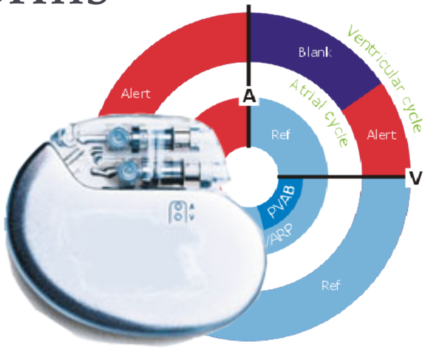


Semiconductor Inspection

Chromatography

Airborne platforms

Battle Simulation

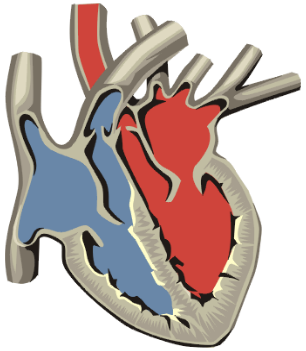


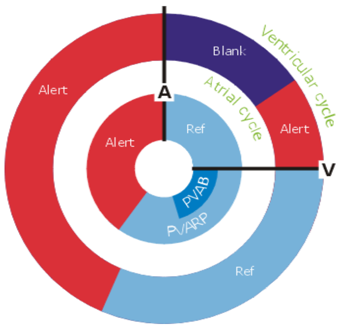
Implantable Medical

Vehicle Networks

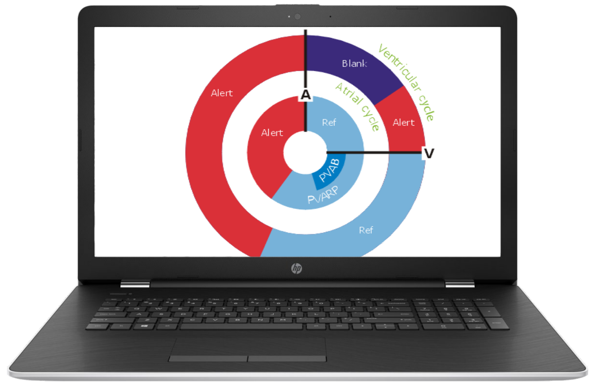
PRESERVE THE MODELS

Cardiac behavior models



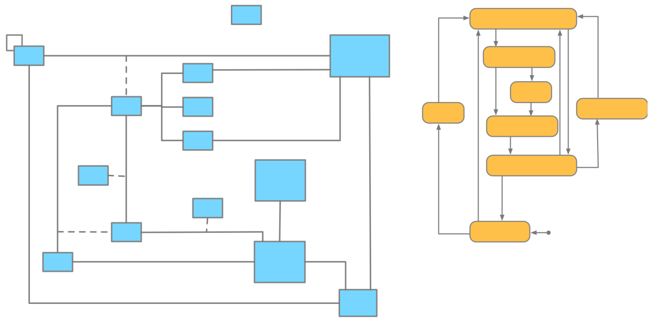


Pace - Inhibition
Scheduling models



PATHS TO CODE THAT DESTROY THE MODELS

Set of "high level" models



Destruction Scenario 1

Add platform specific detail

↓

Out of time and/or money

↓

Write code instead

↓

System

Destruction Scenario 2

Add platform specific detail

↓

Models morph into code diagrams

↓

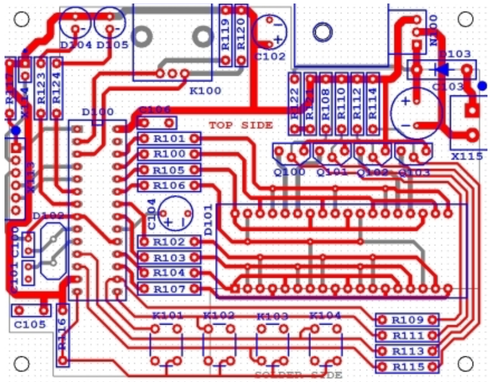
System

Source models are now irrelevant

MAPPING AS DONE BY HARDWARE ENGINEERS

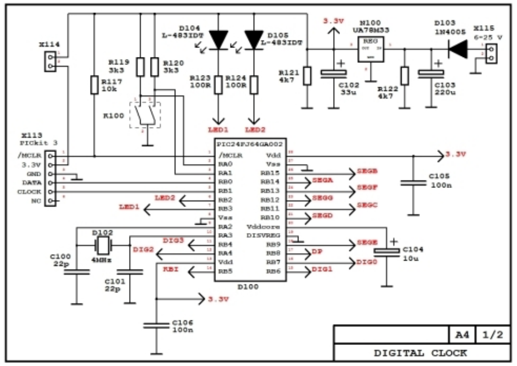
Just because a detail is systematically ignored, doesn't mean that it is not important

Layout diagram



Focuses on layout geometry

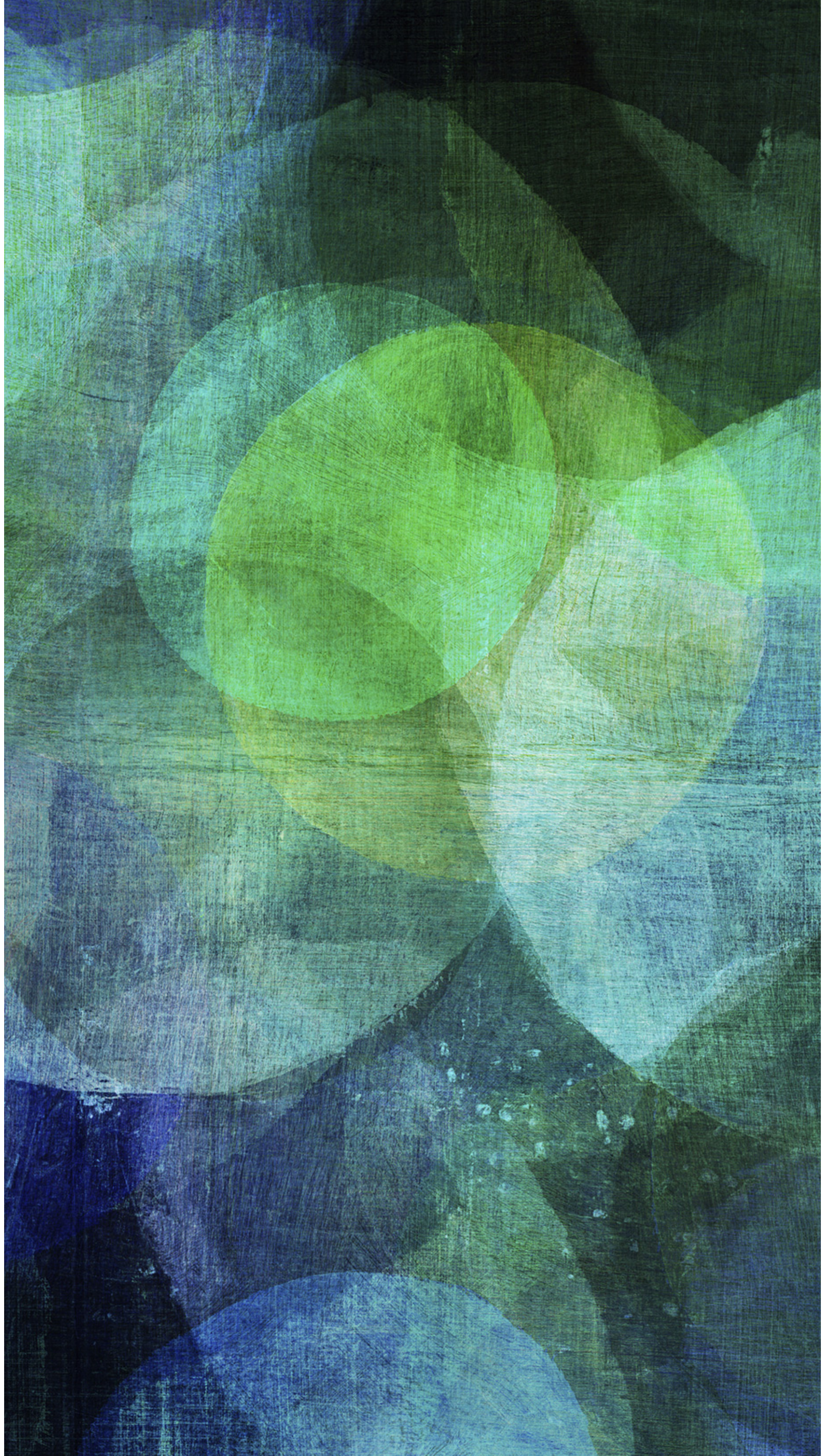
Schematic



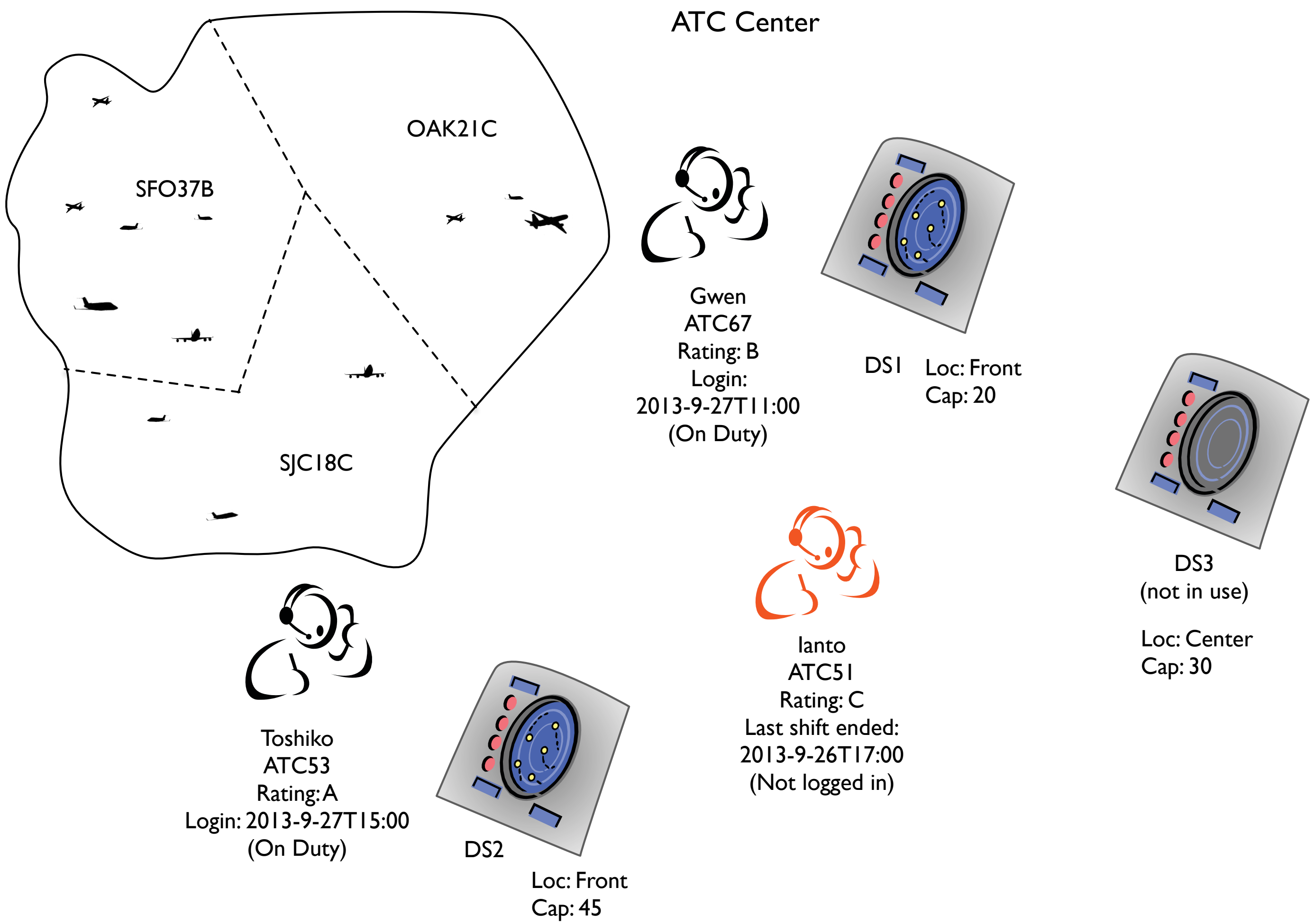
Focuses on component properties and connectivity

Excludes layout details

CASE STUDY



CASE STUDY: AIR TRAFFIC CONTROL



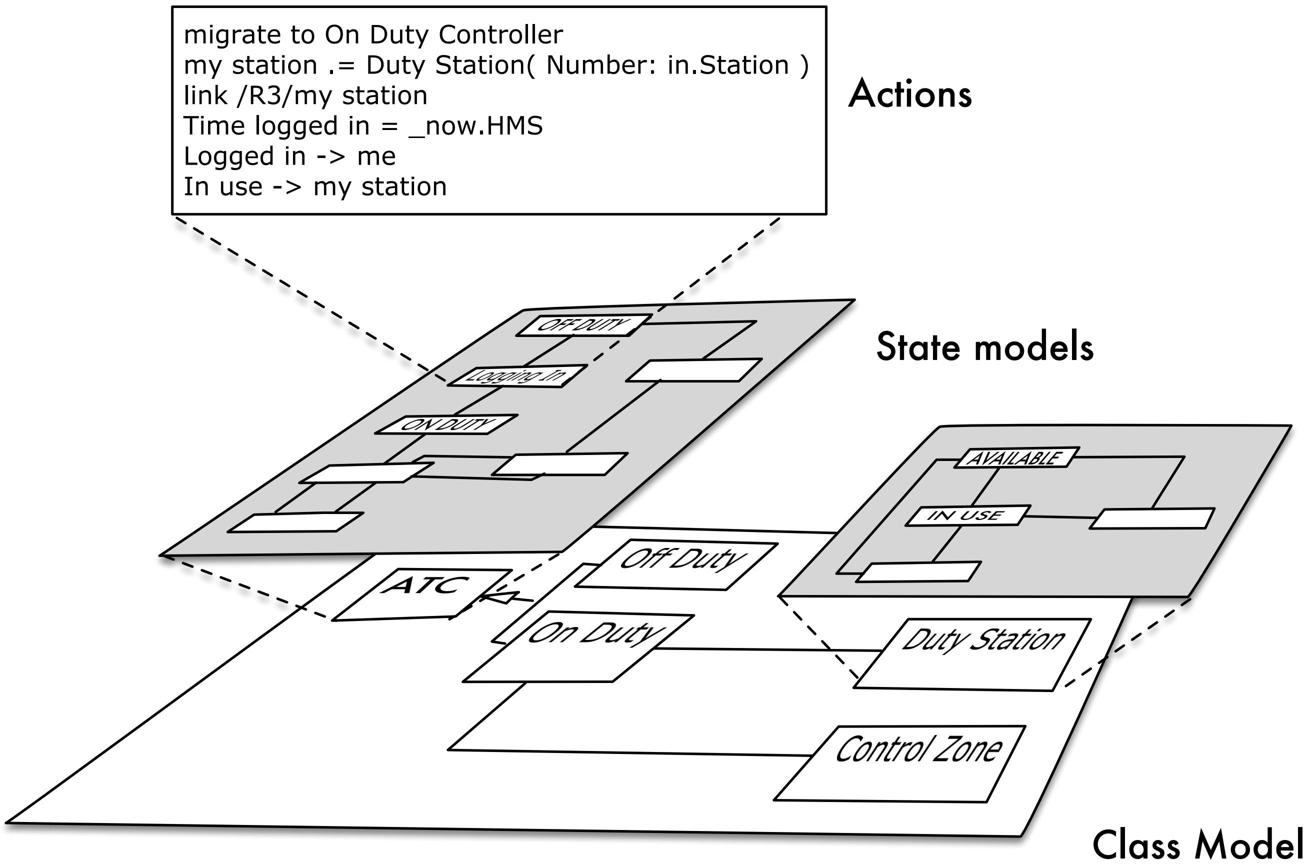
RULES / REQUIREMENTS

- A controller can not direct air traffic while off duty.
- An on duty controller must be logged into a duty station.
- A duty station may or may not be available.
- A control zone must have its traffic directed by one air traffic controller at all times.
- An air traffic controller may not work a shift longer than two hours and fifteen minutes.

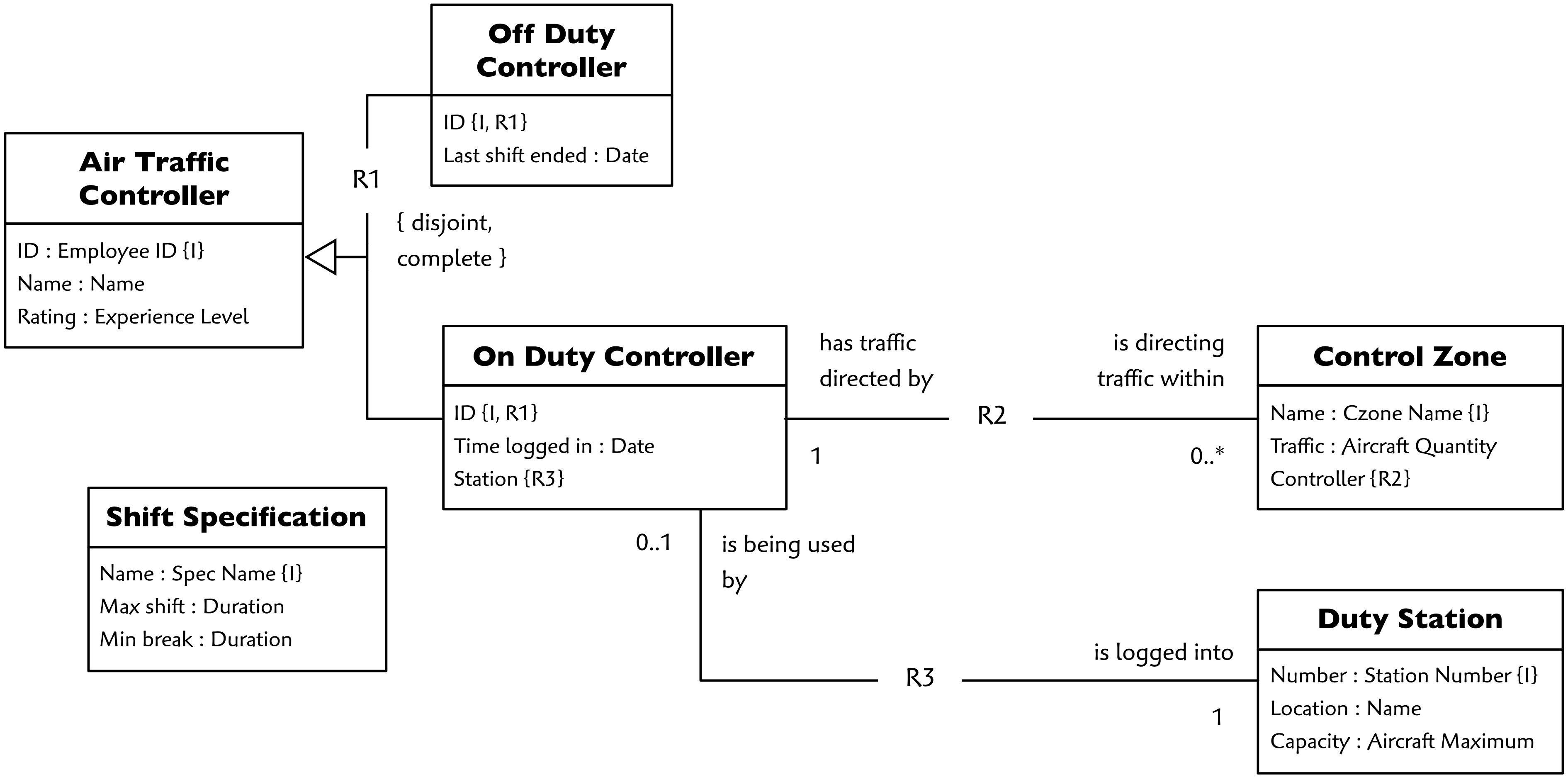
SOME LIMITATIONS OF WRITTEN REQUIREMENTS

- Ambiguous
- Contradictions
- Incomplete
- Same word for many things
- Different words for the same thing
- Over-specification
- Under-specification
- Inconsistency
- Arbitrary partitioning

MODELING BEHAVIOR WITH STATES AND ACTIONS



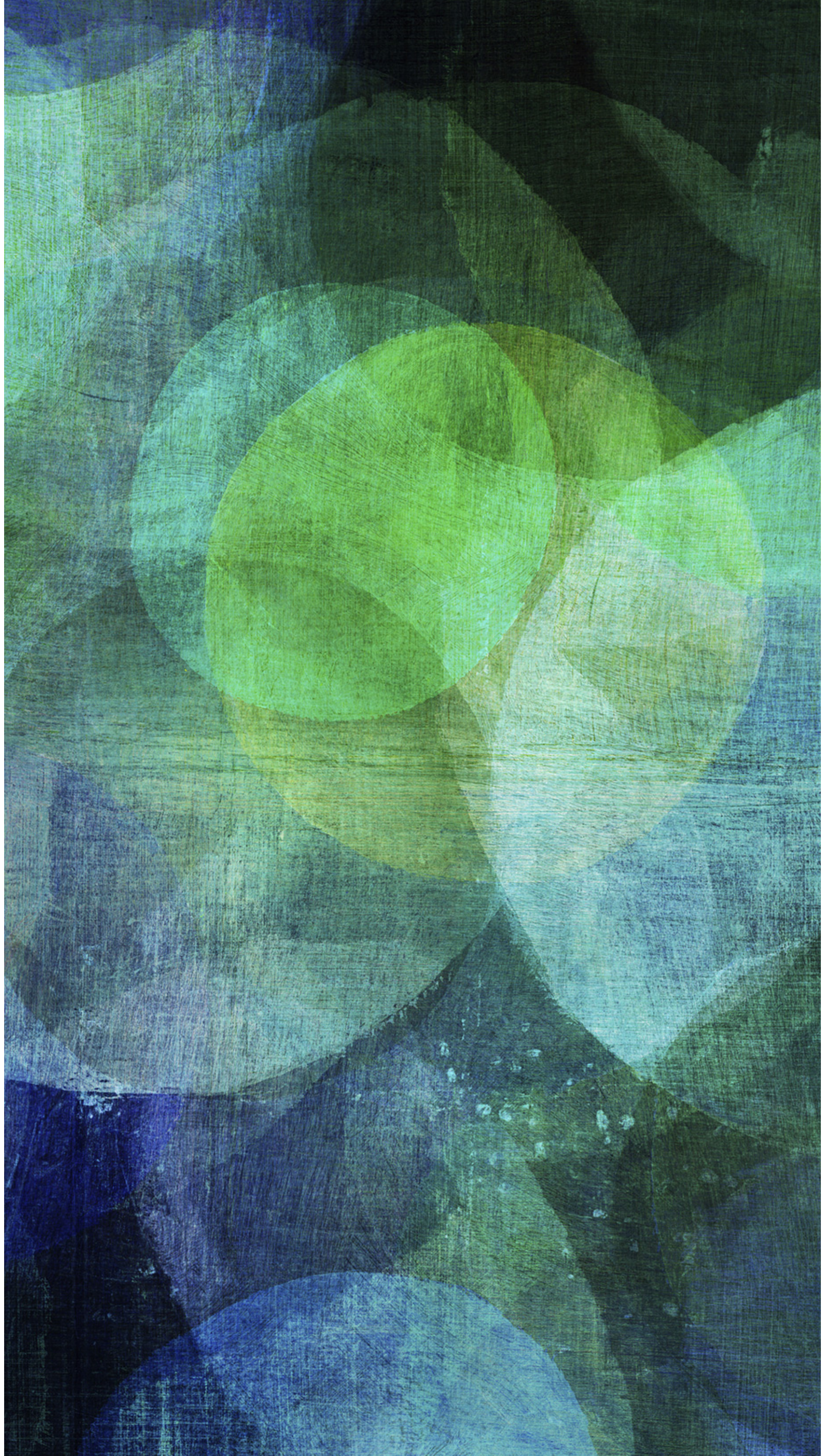
THE CLASS MODEL: DATA, RULES, CONSTRAINTS



SOURCE MODEL PROPERTIES

- Executable (class + state + action models)
- Platform independent (models requirements)
- Lean, mathematically based language (Executable UML)

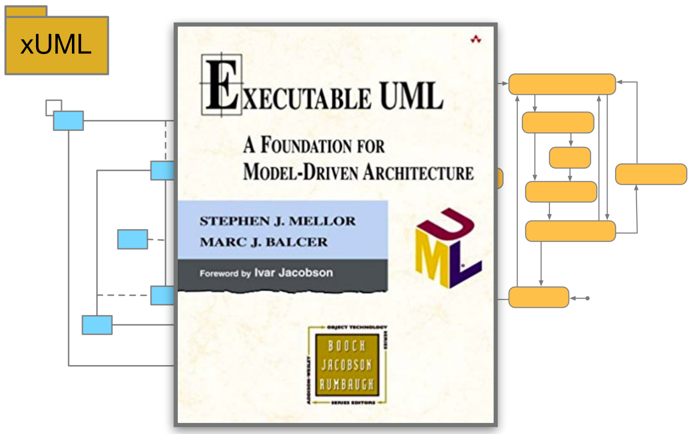
DESIGN



PLATFORM CHARACTERISTICS

- Limited memory
- Limited execution cycles
- Interrupts
- Timely response required

STEP 1: GET AN MX DOMAIN FOR YOUR CLASS OF PLATFORM



xUML rules & semantics

Platform constraints

Design Input

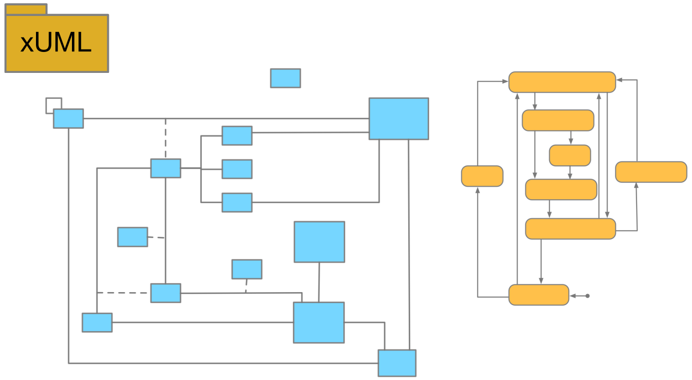
MX Domain

MX = Model Execution

It's realized as a C library

Provides runtime support for data storage, access, state machine logic, action execution

STEP 2: MARK UP YOUR MODELS WITH DESIGN DIRECTIVES



Domains

Data types

Classes, attributes, relationships

States, transitions, events

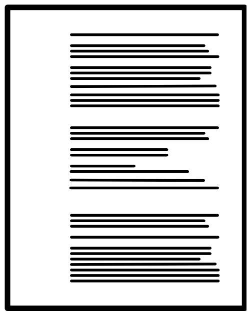
Activities / Actions

Design Input

Pattern selection

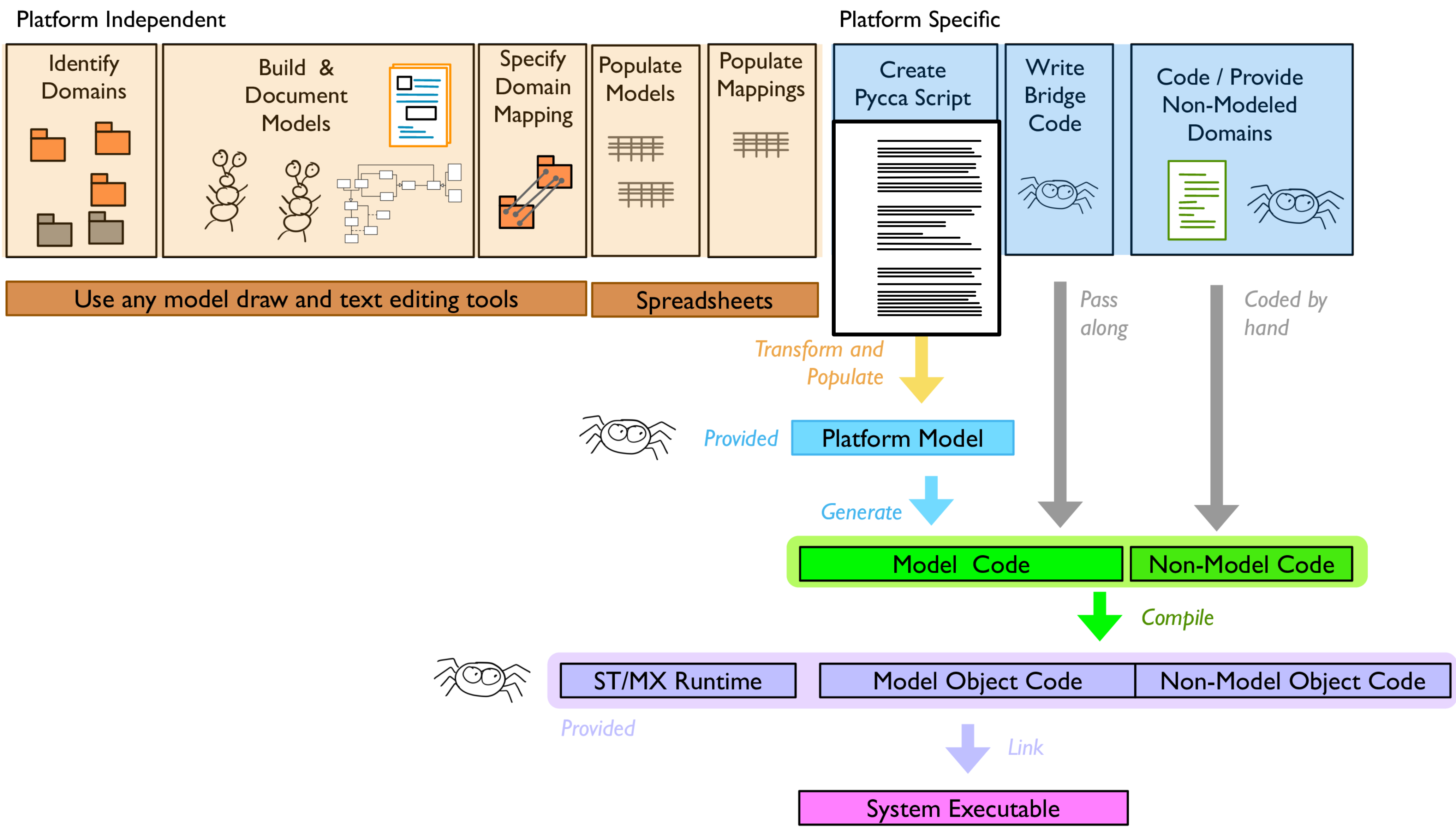
Choose imp data types

Code for some actions

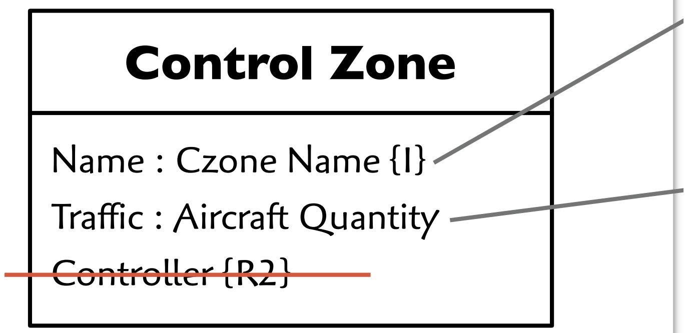


Model Design Script

STEP 3: COMPILE & LINK



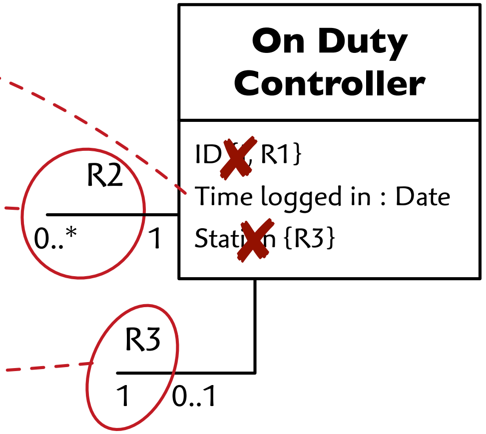
CHOOSE DATA TYPES



MODELED CLASS TO STRUCT

In model design script

```
class On_Duty_Controller
    attribute (Date_t Time_logged_in) default {0}
    reference R2 ->>l Control_Zone
    reference R3 -> Duty_Station
end
```



Implementation link types

Generated code

```
struct On_Duty_Controller {
    Date_T Time_logged_in ;
    rlink_t R2  ;
    struct Duty_Station *R3 ;
} ;
```

INITIAL INSTANCE POPULATION

Air Traffic Controllers

ID {I}	Name	Rating
ATC53	Toshiko	A
ATC67	Gwen	B
ATC51	Ianto	C

Same object

Superclass table

On Duty Controllers

ID {I, R1}	Time logged in	Duty Station {R2}
ATC53	9/27/13 15:00	DS2
ATC67	9/27/13 11:00	DS1

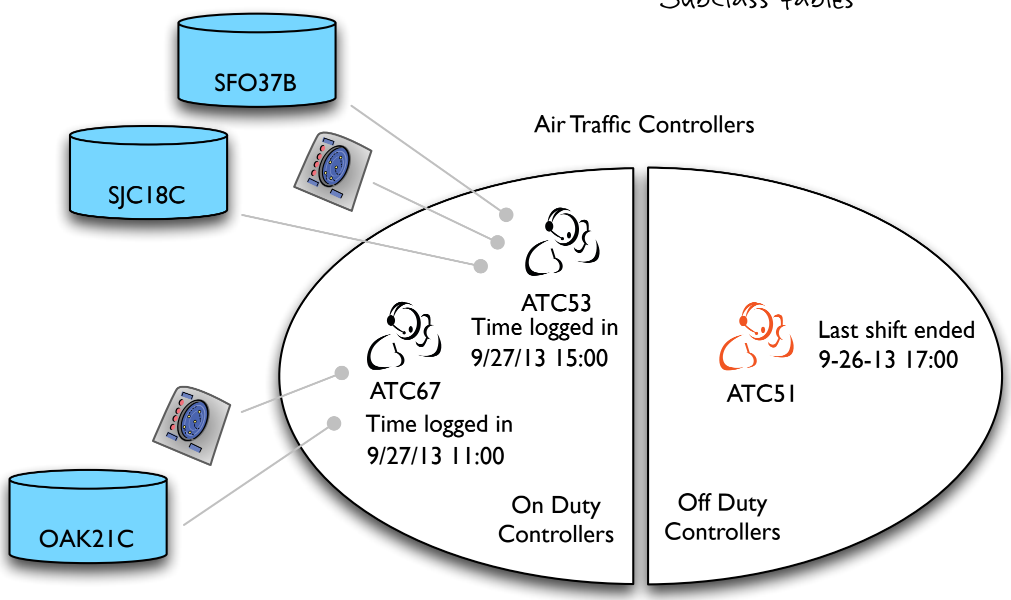
Off Duty Controllers

ID {I, R1}	Last shift ended
ATC51	9-26-13 17:00

Subclass tables

Duty Station

Number {I}	Location	Capacity
DS1	Front	20
DS3	Center	30
DS2	Front	45



Control Zones

Name	Traffic	Controller
SJC18C	30	ATC53
SFO37B	25	ATC53
OAK21C	15	ATC67

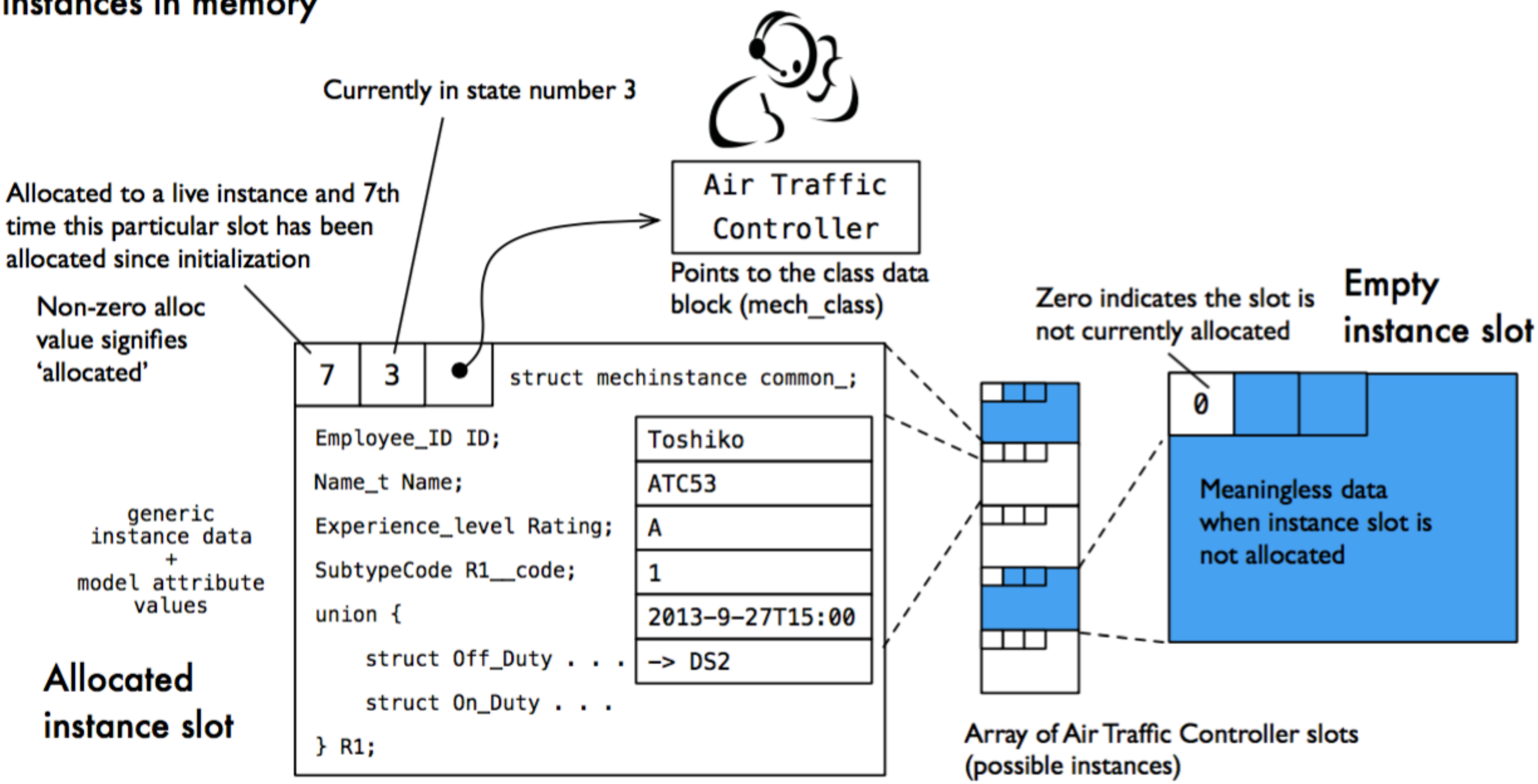
THE INSTANCE DATA

```
table Air_Traffic_Controller
        (Employee_ID ID)  (Name_T Name)  (Experience_Level Rating)  R1
@atc53  {"53"} ❶          {"Toshiko"}    {"A"}    -> On_Duty_Controller.atc53 ❷
@atc67  {"67"}            {"Gwen"}       {"B"}    -> On_Duty_Controller.atc67
@atc51  {"51"}            {"Ianto"}      {"C"}    -> On_Duty_Controller.atc51
end

table On_Duty_Controller
            R2                 R3
@atc53      ->> sfo end        -> s2 ❸
@atc67      ->> oak end        -> s1
@atc51      ->> sjc end        -> s3
end

table Control_Zone
        (Czone_Name Name)      (Aircraft_Quantity Traffic)     R2
@sfo    {"SFO37B"}             {27}                        ❹   -> atc53
@oak    {"OAK21C"}             {18}                            -> atc67
@sjc    {"SJC18C"}             {9}                             -> atc51
end

table Duty_Station
     (Station_Number Number) (Name_T Location) (Aircraft_Maximum Capacity)
@s1  {"S1"} ❺                {"Front"}         {20}
@s2  {"S2"}                  {"Center"}        {30}
@s3  {"S3"}                  {"Front"}         {45}
end
```

```
/*
 * Initial Instance Storage for, "Air_Traffic_Controller"
 */
static struct Air_Traffic_Controller Air_Traffic_Controller_storage[3] = { ❶
    {❷.common_ = {1, 0, &Air_Traffic_Controller_class}, "53", "Toshiko", "A", .R1__code = ↩
        1, .R1 = {.R1_On_Duty_Controller = {0, .R2 = {.next = &Control_Zone_storage[0]. ↩
        R2__links, .prev = &Control_Zone_storage[0].R2__links}, .R3 = &Duty_Station_storage ↩
        [1]}}},
    {.common_ = {2, 0, &Air_Traffic_Controller_class}, "67", "Gwen", "B", .R1__code = 1, . ↩
        R1 = {.R1_On_Duty_Controller = {0, .R2 = {.next = &Control_Zone_storage[1].R2__links ↩
        , .prev = &Control_Zone_storage[1].R2__links}, .R3 = &Duty_Station_storage[0]}}},
    {.common_ = {3, 0, &Air_Traffic_Controller_class}, "51", "Ianto", "C", .R1__code = 1, . ↩
        R1 = {.R1_On_Duty_Controller = {0, .R2 = {.next = &Control_Zone_storage[2].R2__links ↩
        , .prev = &Control_Zone_storage[2].R2__links}, .R3 = &Duty_Station_storage[2]}}}
} ;
/*
 * Initial Instance Storage for, "Control_Zone"
 */
static struct Control_Zone Control_Zone_storage[3] = { ❸
    {"SFO37B", 27, .R2 = &Air_Traffic_Controller_storage[0].R1.R1_On_Duty_Controller, . ↩
        R2__links = {.next = &Air_Traffic_Controller_storage[0].R1.R1_On_Duty_Controller.R2, ↩
         .prev = &Air_Traffic_Controller_storage[0].R1.R1_On_Duty_Controller.R2, }},
    {"OAK21C", 18, .R2 = &Air_Traffic_Controller_storage[1].R1.R1_On_Duty_Controller, . ↩
        R2__links = {.next = &Air_Traffic_Controller_storage[1].R1.R1_On_Duty_Controller.R2, ↩
         .prev = &Air_Traffic_Controller_storage[1].R1.R1_On_Duty_Controller.R2, }},
    {"SJC18C", 9, .R2 = &Air_Traffic_Controller_storage[2].R1.R1_On_Duty_Controller, . ↩
        R2__links = {.next = &Air_Traffic_Controller_storage[2].R1.R1_On_Duty_Controller.R2, ↩
         .prev = &Air_Traffic_Controller_storage[2].R1.R1_On_Duty_Controller.R2, }}
} ;
/*
 * Initial Instance Storage for, "Duty_Station"
 */
static struct Duty_Station Duty_Station_storage[3] = {
    {.common_ = {1, 0, &Duty_Station_class}, "S1", "Front", 20, .R3 = NULL},
    {.common_ = {2, 0, &Duty_Station_class}, "S2", "Center", 30, .R3 = NULL},
    {.common_ = {3, 0, &Duty_Station_class}, "S3", "Front", 45, .R3 = NULL}
} ;
```

RUNTIME DATA

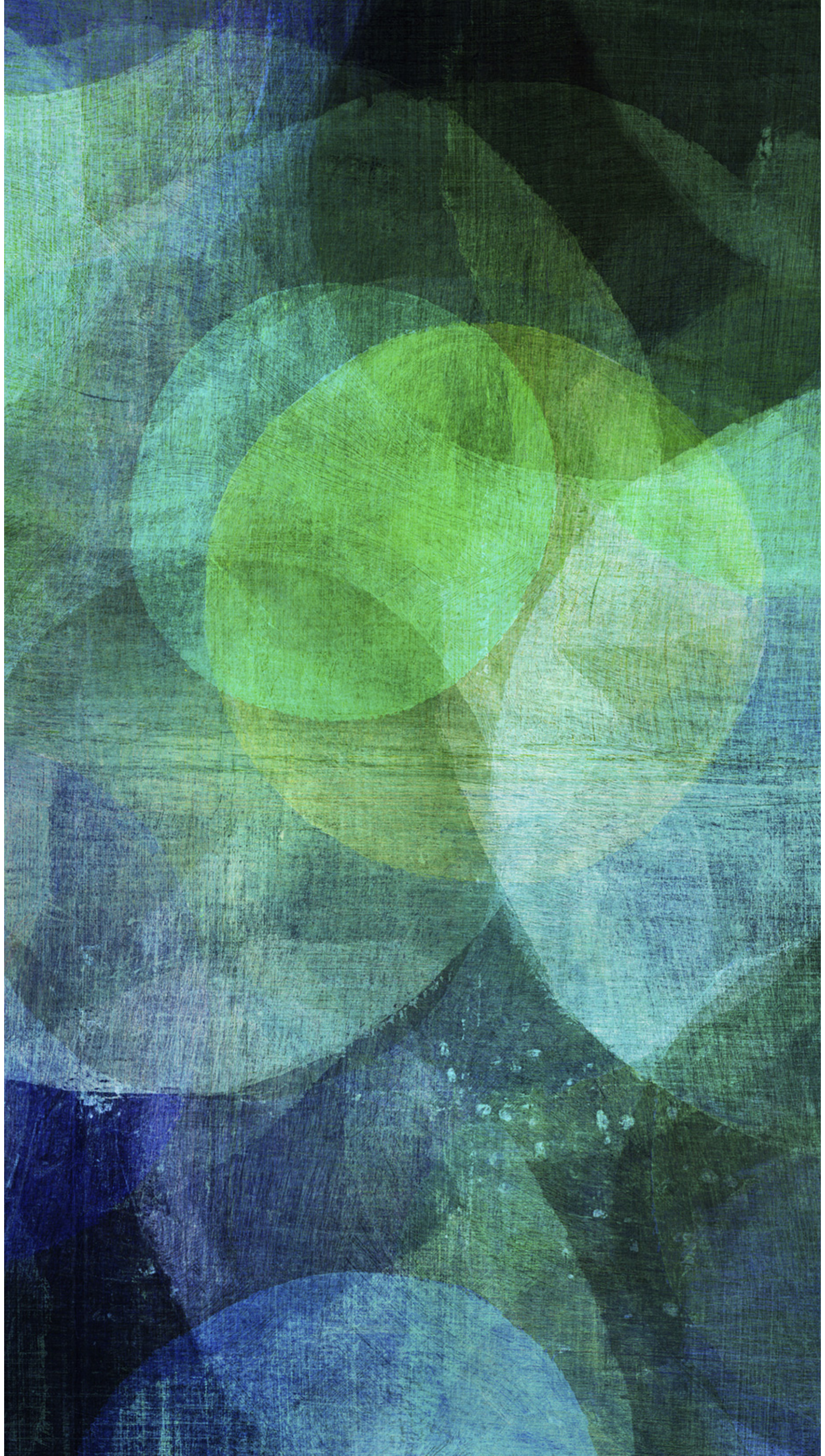
Example dynamic instances in memory



PERFORMANCE

- Code size
  - MX runtime: ~ 4K
  - 2 domain example: +12K
    - 20 classes, bridging +60 instances
- CPU 7MHz
  - 1491 signal dispatches / sec

SUMMARY



REQUIRED FOR MBSE TO BECOME MAINSTREAM

- Models must add real value
- Don’t destroy the models when implementing
- Map the models to code, don’t “mix in” detail
- It’s still necessary to write code and that’s okay!
- There must be a clear path from models to final code

WHAT'S NEXT FOR MBSE?