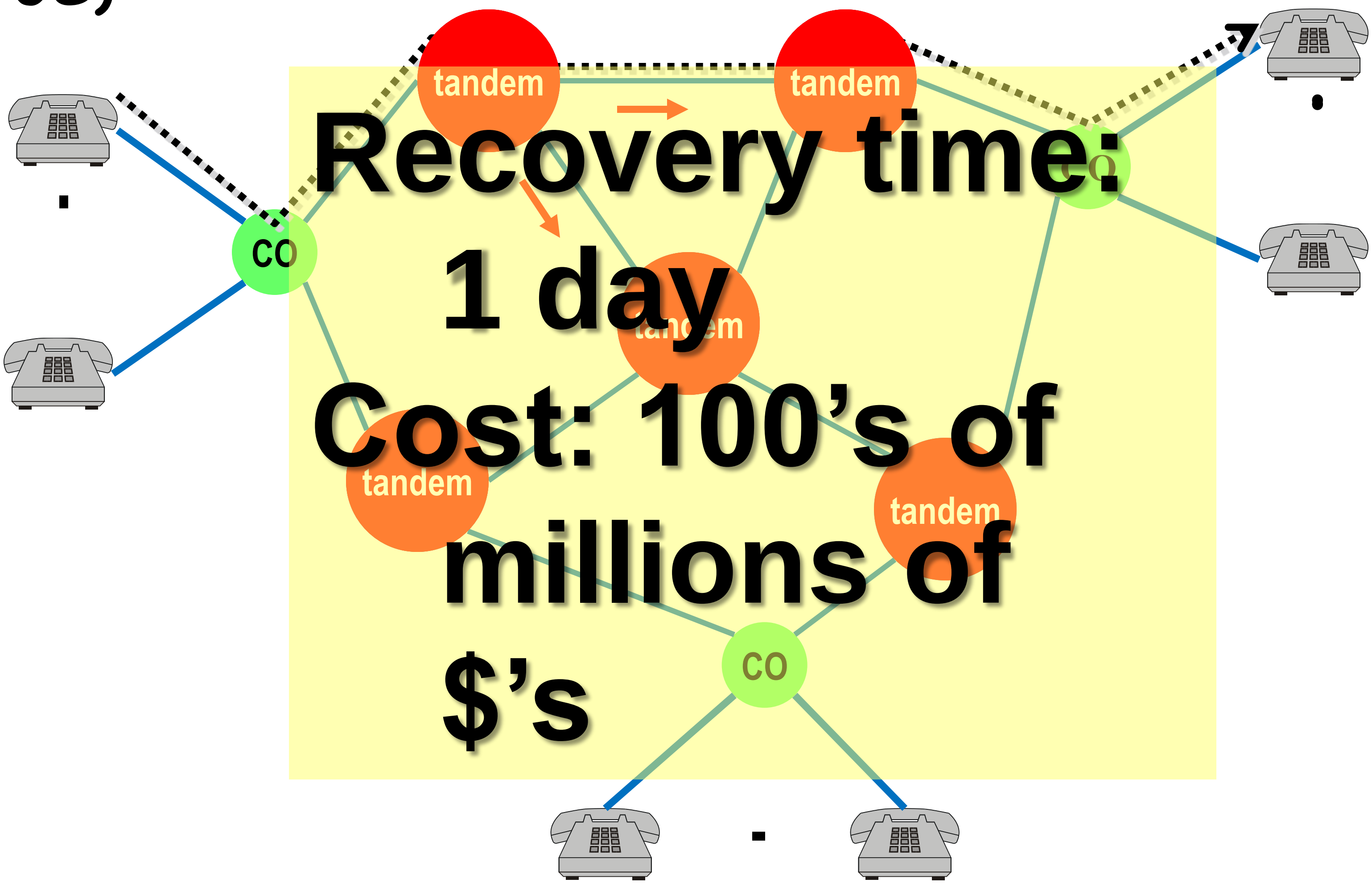
Model-Based Engineering of Real-Time and Embedded Systems

Bran Selic
Malina Software Corp., Canada
Adjunct Prof., U. Of Toronto and Carleton U., Canada
selic@acm.org

Overview

- On Model-Based Software Engineering
- Applying MBSE to Real-Time/Embedded Systems
- The Principal Research Challenges of MBSE

- 1990: AT&T Long Distance Network (Northeastern US)



The Culprit

- The (missing) "break" that broke it

...and, it's all HIS fault!

```
. . .;
switch (...) {
     case a : ...;
          break;
     case b :...;
          break;
          . . .
     case m : ...;
     case n : ...;
          . . .
};
```

Aaargh! Forgot the "break"...

The Enemy: Complexity

- Modern real-time software systems are very complex and getting more so
  - Complex behavior and structure
  - Increasing demands for greater dependability (availability, reliability, performance, etc.)
  - …while, at the same time, our current software projects success rate is dismal? (< 50%)
- Complexity: Essential vs Accidental
- Thesis: Far too much of this complexity is accidental and a consequence of inappropriate implementation technologies methods
  - i.e., complexity that is due to our technologies and methods

The Impact

- **Abstraction of software is extremely difficult and risky**
  - Any detail can be critical!
  - Eliminates our most effective means for managing complexity
- **Our ability to exploit formal mathematical methods is severely limited**
  - Mathematics is at the core of all successful modern engineering

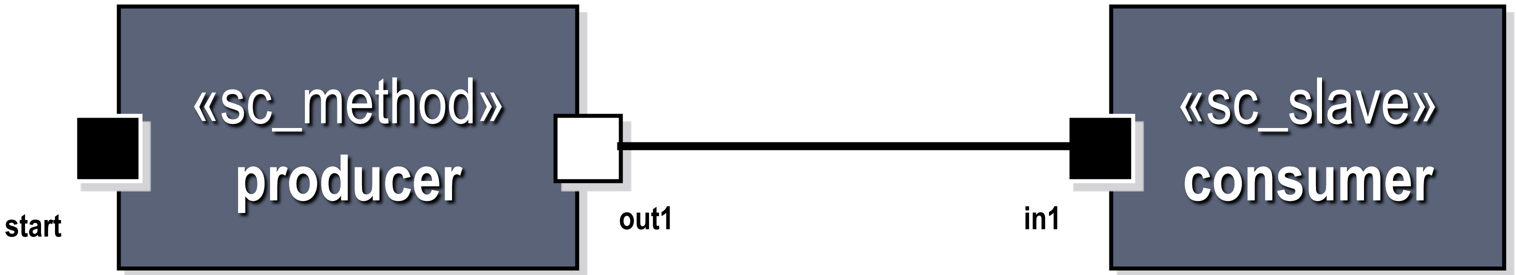
A Fragment of Modern Software…

Malina SOFTWARE CORP.

```
SC_MODULE(prod)
{
sc_outmaster<int> out1;
sc_in<bool> start; // kick-start
void generate_data ()
{
for(int i =0; i <10; i++) {
out1 = i ; //to invoke slave;}
}
SC_CTOR(prod)
{
SC_METHOD(generate_data);
sensitive << start;}};
SC_MODULE(con)
{
sc_inslave<int> in1;
int sum; // state variable
void accumulate (){
sum += in1;
}
SC_CTOR(con)
{
SC_SLAVE(accumulate, in1);
sum = 0; // initialize
};
SC_MODULE(top) // container
{
prod *A1;
con *B1;
sc_link_mp<int> link1;
SC_CTOR(top)
{
A1 = new prod("A1");
A1.out1(link1);
B1 = new con("B1");
B1.in1(link1);}};
```

Do you see the architecture of this system?

...and Its UML 2 Model

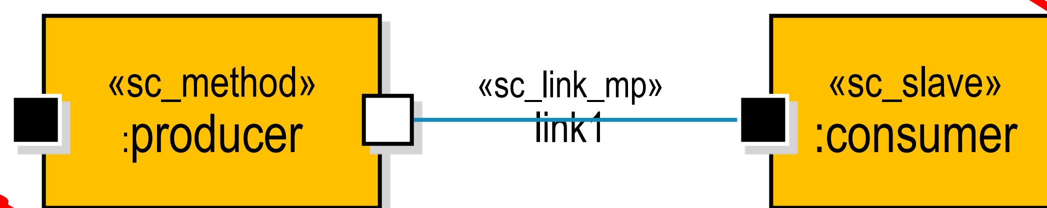


Can you see it now?

The Program and Its Model

```
SC_MODULE(prod)
{
sc_outmaster<int> out1;
sc_in<bool> start; // kick-start
void generate_data ()
{
for(int i =0; i <10; i++) {
out1 =i ; //to invoke slave;}
}
SC_CTOR(prod)
{
SC_METHOD(generate_data);
sensitive << start;}};
SC_MODULE(con)
{
sc_inslave<int> in1;
int sum; // state variable
void accumulate (){
sum += in1;
}
```

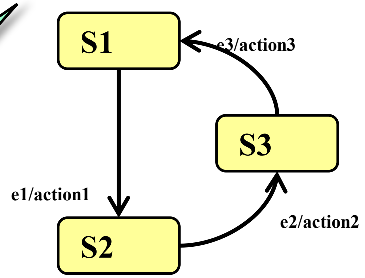
```
SC_CTOR(con)
{
SC_SLAVE(accumulate, in1);
sum = 0; // initialize
};
SC_MODULE(top) // container
{
prod *A1;
con *B1;
sc_link_mp<int> link1;
SC_CTOR(top)
{
A1 = new prod("A1");
A1.out1(link1);
B1 = new con("B1");
B1.in1(link1);}};
```



- An approach to software development in which software models play an indispensable role
- Based on two time-proven ideas:

(1) ABSTRACTION

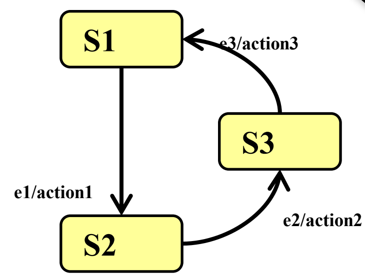
Realm of modeling languages




```
switch (state) {
   case‘1:action1;
          newState(‘2’);
          break;
   case‘2:action2;
          newState(‘3’);
          break;
   case’3:action3;
          newState(‘1’);
          break;}
```

(2) AUTOMATION

Realm of tools



```
switch (stat
   case‘1:ac
          newState(‘2’);
          break;
   case‘2:action2;
          newState(‘3’);
          break;
   case’3:action3;
          newState(‘1’);
          break;}
```

Why Build Models?

1. To <u>understand</u>

    the interesting characteristics of an existing or intended system

2. To <u>communicate</u>

    - understanding and design intent

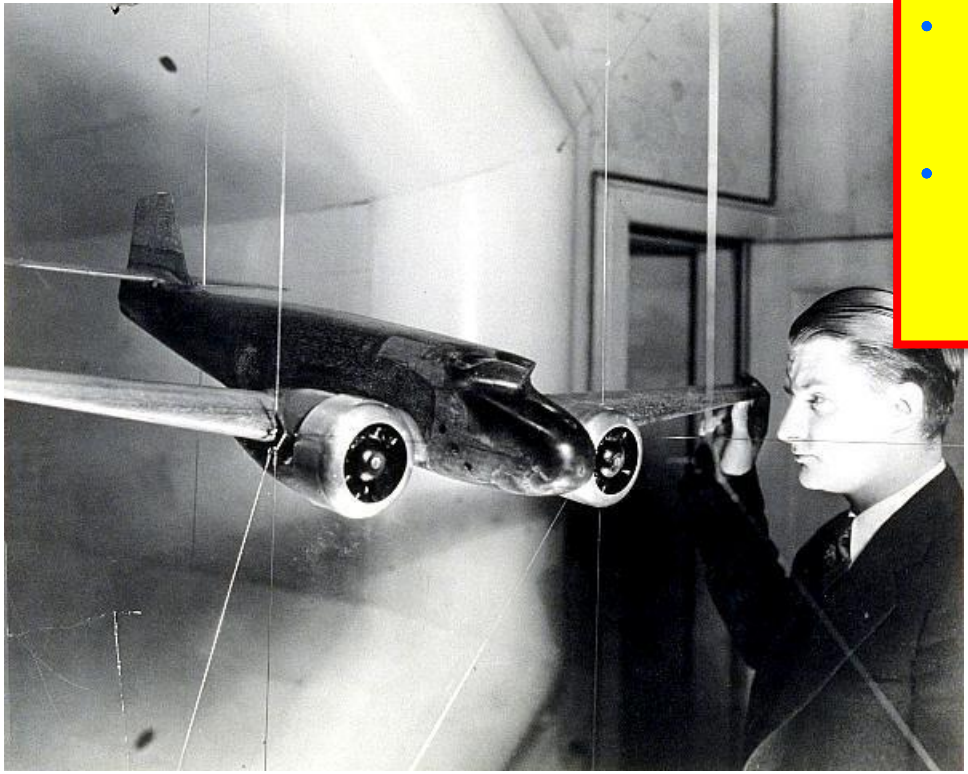
3. To <u>predict</u>

    - the characteristics of interest (by analysing models)

Engineering Models

- **Engineering model:**

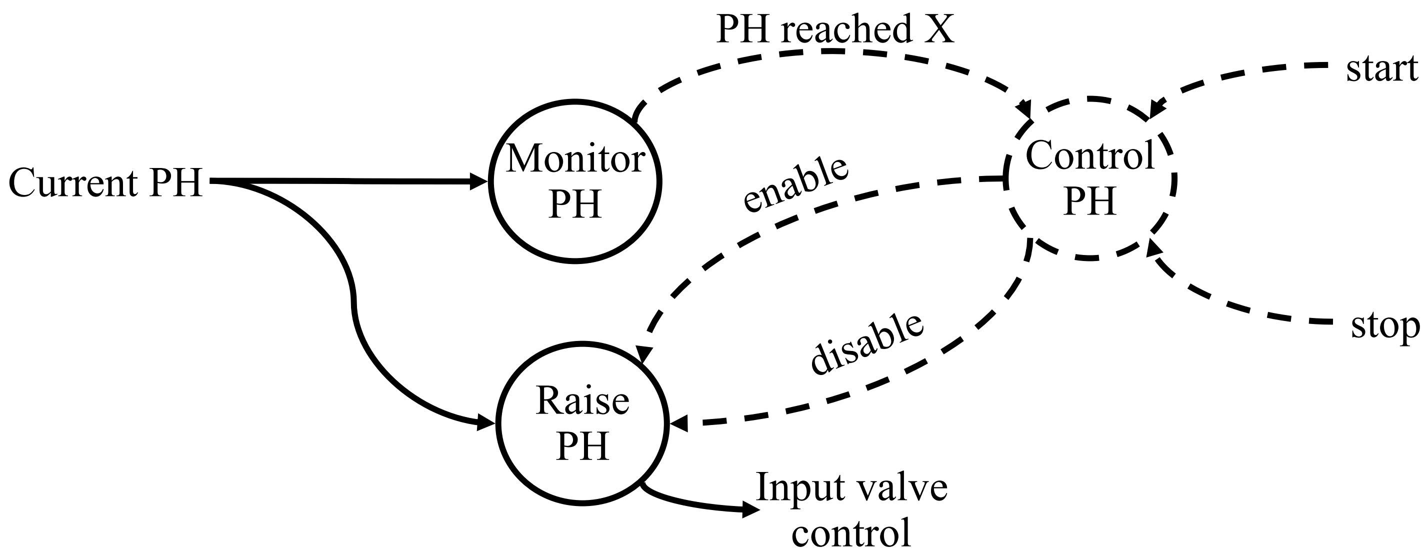
  A reduced representation of some system or process, which emphasizes properties that are of interest to a given set of concerns



- We don't see everything at once
- What we do see is adjusted to the model's purpose and to human understanding

What about models of software systems?

What About Models of Software?



"…bubbles and arrows, as opposed to programs, …never crash"

-- B. Meyer
"UML: The Positive Spin"
American Programmer, 1997

Key Characteristics of Useful Models

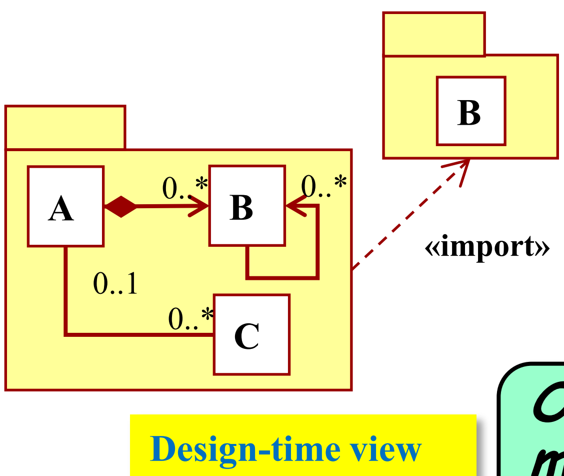
- **Clear purpose**
  - Known audience, perspective (viewpoints), and expected value
- **Minimal (abstract)**
  - Emphasizes what is relevant while removing/hiding what is not
- **Understandable**
  - Expressed in a form that is readily understood by its audience
- **Accurate**
  - Faithfully represents relevant aspects of the modeled system
- **Predictive**
  - Can help answer key questions about the modeled system
- **Cost-effective**
  - Much cheaper and faster to construct than actual system

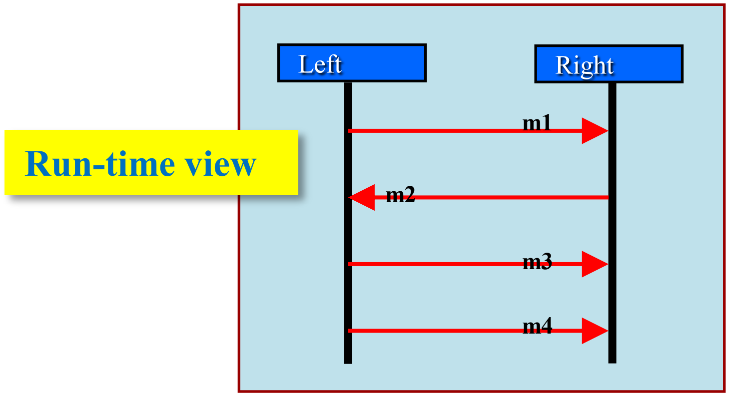
What's a Software Model?

- **Software model**: An engineering model (specified using a modeling language) of some software that represents:

1. The run-time view of the software: the structure and behavior of the *software in execution* and/or

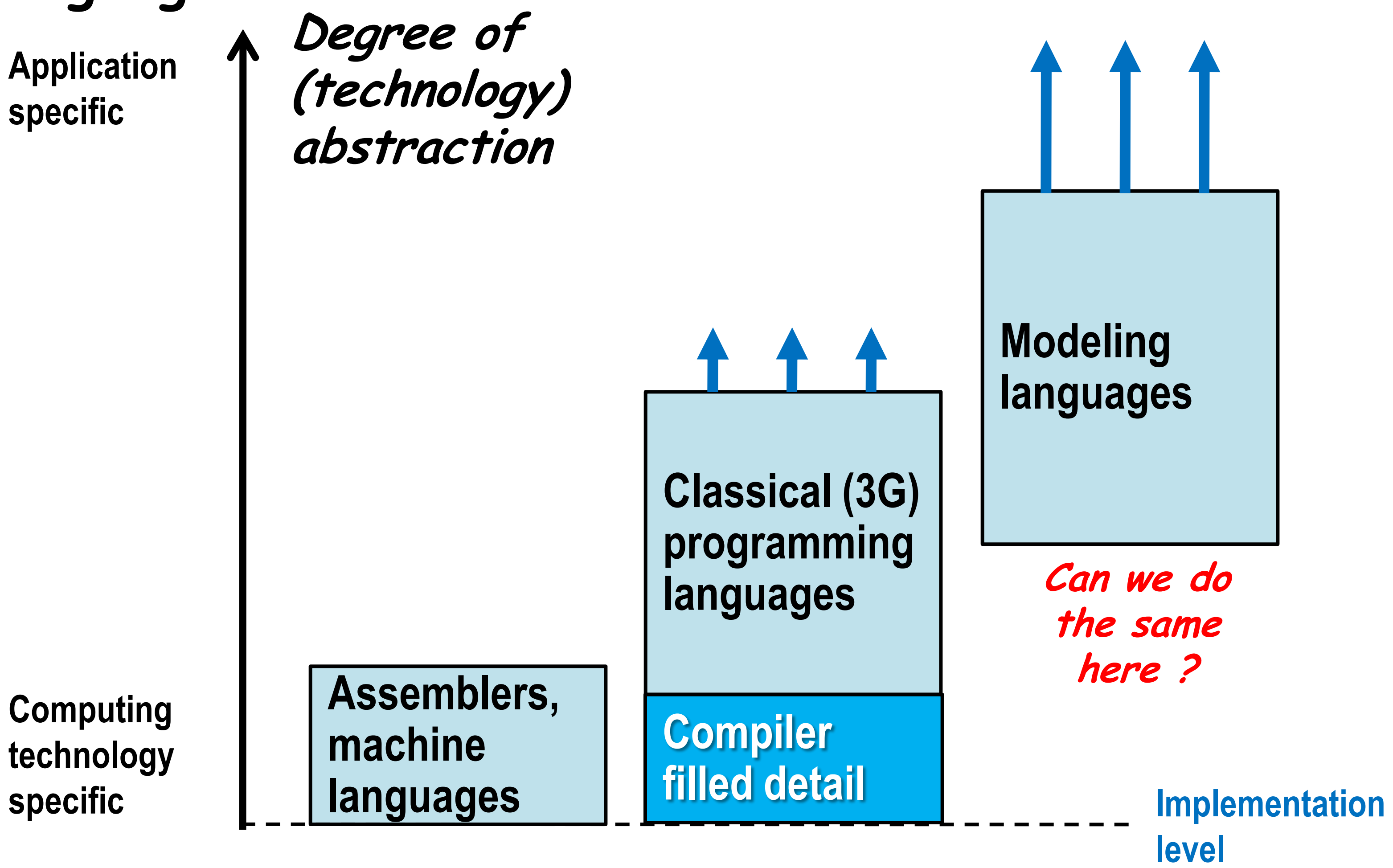
2. The design-time view of the software: The structure and content of the *software specification*





One of the primary motives for many modeling languages is the need to more clearly represent software in execution

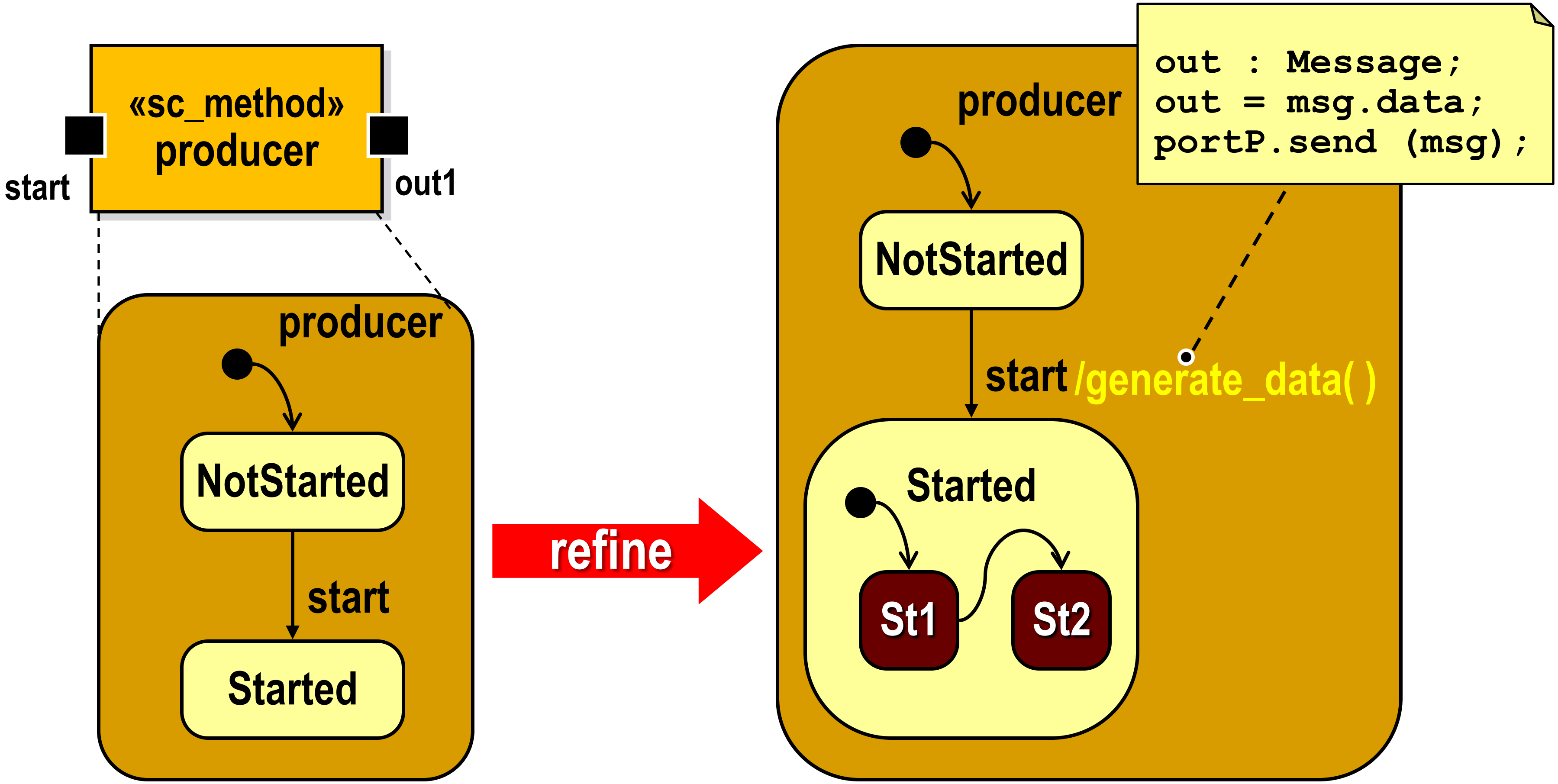
- The next phase of development in computer languages…



Categories of Modeling Languages

- Classical informal Design/Analysis/Documentation (DAD) modeling languages
  - Informal documentation-oriented languages
  - Have been used for decades and proven effective
  - However, they bring nothing new that will lead us to quantum leaps in productivity and quality
- Executable modeling languages
  - Based on precise (possibly formal) semantics
  - Value-add: Early and direct evaluation of design choices
  - Value-add: Potential for computer-based verification
  - Value-add: Potential for spanning the full development cycle from architectural design through implementation languages

Modeling with Executable Languages



Early architecture model

More refined model

- Models can be refined and verified continuously until ***the model becomes the system that it was modeling!***

A Unique Feature of Software

Malina SOFTWARE CORP.

- A software model and the software being modeled share the same medium—the computer

Software has the unique property that it allows us to directly evolve models into implementations <u>without fundamental discontinuities in the expertise, tools, or methods</u>!

⇒ High probability that key design decisions will be preserved in the implementation and that the results of prior analyses will be valid

MBSE: State of the Practice

- Example: Major Telecom Equipment Vendor
  - Adopted MBSE Tooling
  - Used commercial MBSE tools: Rose RealTime (with fully automated code generation), Test RealTime, RUP
- Product : Network Controller
  - 7.5 Million lines of auto-generated C++ code
  - 400+ developers working on a single UML model
- Performance (throughput, memory):
  - Within ± 15% of hand-crafted code
- Productivity improvements
  - 80% fewer bugs
  - Estimated productivity improvement = factor of 4
- There are many similar examples…

Sampling of Successful MBSE Products

Automated doors, Base Station, Billing (In Telephone Switches), Broadband Access, Gateway, Camera, Car Audio, Convertible roof controller, Control Systems, DSL, Elevators, Embedded Control, GPS, Engine Monitoring, Entertainment, Fault Management, Military Data/Voice Communications, Missile Systems, Executable Architecture (Simulation), DNA Sequencing, Industrial Laser Control, Karaoke, Media Gateway, Modeling Of Software Architectures, Medical Devices, Military And Aerospace, Mobile Phone (GSM/3G), Modem, Automated Concrete Mixing Factory, Private Branch Exchange (PBX), Operations And Maintenance, Optical Switching, Industrial Robot, Phone, Radio Network Controller, Routing, Operational Logic, Security and fire monitoring systems, Surgical Robot, Surveillance Systems, Testing And Instrumentation Equipment, Train Control, Train to Signal box Communications, Voice Over IP, Wafer Processing, Wireless Phone

Overview

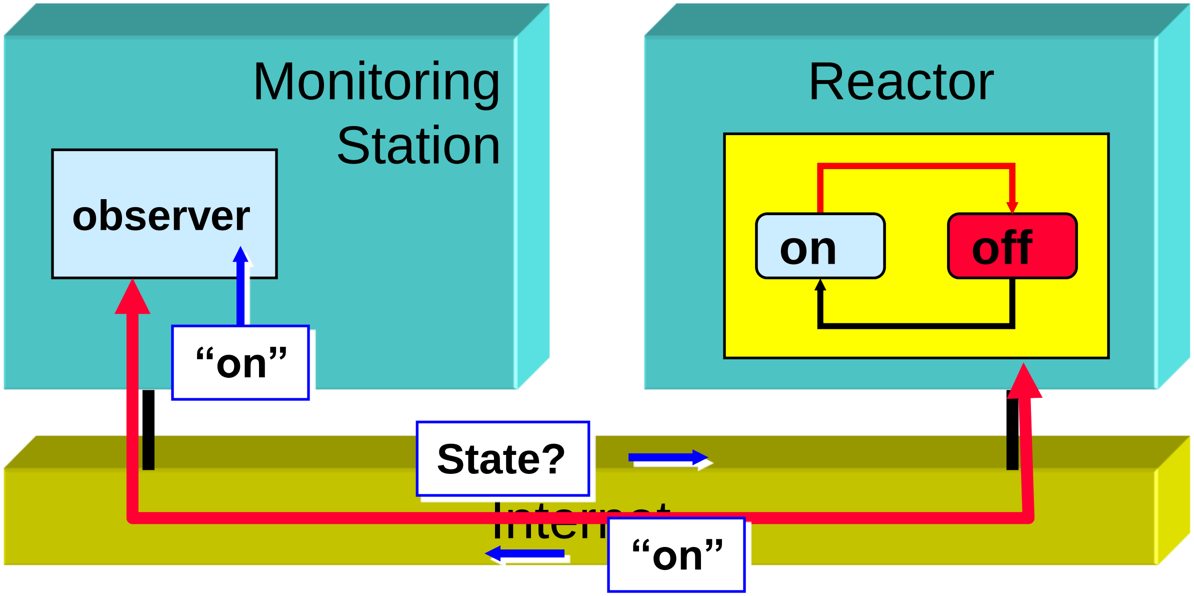
- On Model-Based Software Engineering
- Applying MBSE to Real-Time/Embedded Systems
- The Principal Research Challenges of MBSE

Real-Time and Embedded Systems

- **Systems whose software interacts with the physical world in a timely fashion**
- **Particularly challenging: *Must contend with the full complexity and unpredictability of the physical world***
  - Concurrency
  - Asynchrony and interruptions (e.g., failures)
  - Stringent quantitative constraints
    - Time constraints
    - Resource limitations
    - Availability requirements
    - Safety requirements
    - The laws of physics
- ***<u>RTE systems need to be engineered!</u>***

The Impact of Platforms on SW

- Example: The problem of out-of-date information



The software must operate correctly even if its status information may be out of date!

Not Just a Question of Quantity/Performance

"It is not possible to guarantee that agreement can be reached in finite time over an asynchronous communication medium, if the medium is lossy or one of the distributed sites can fail"

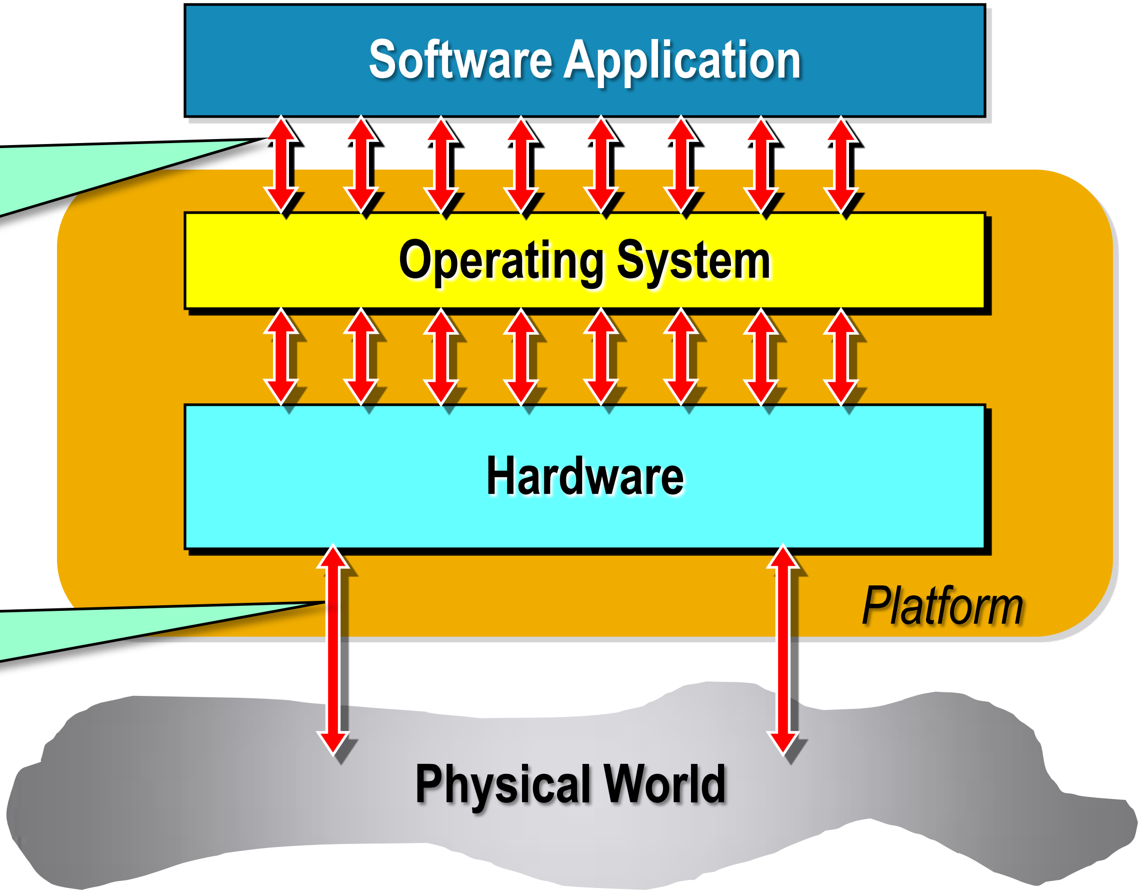
- Fischer, M., N. Lynch, and M. Paterson, "Impossibility of Distributed Consensus with One Faulty Process" Journal of the ACM, (32, 2) April 1985.

- ***In many real systems, the physical platform is a primary design constraint***

  <u>Computer system = software + hardware</u>

- ***Yet, many practitioners still believe that "platform concerns" are second-order issues***

Platfroms: The Raw Material of SW



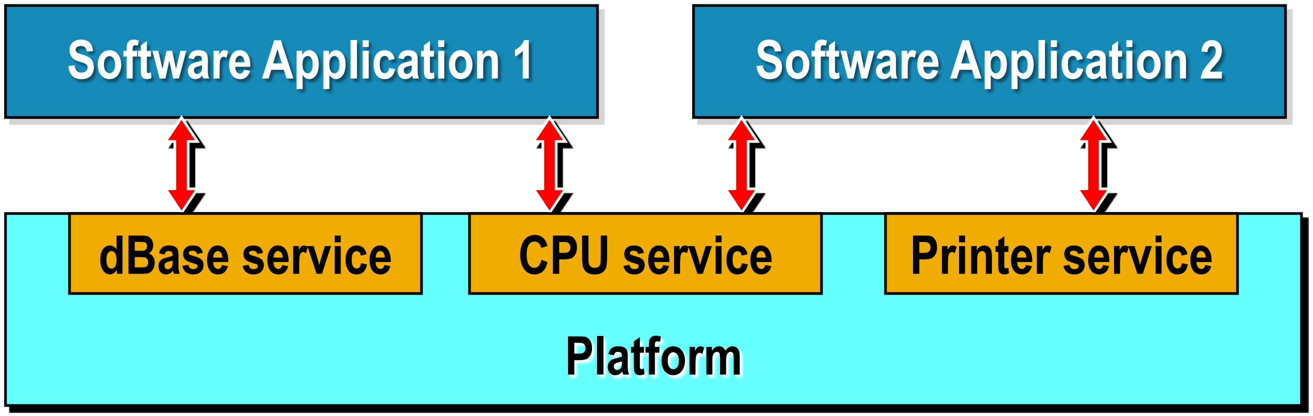
NB: Software engineering is very weak on methods for specifying platform requirements of software applications

A platform also acts as a gateway to the physical world

- ***Platform:***

  the full complement of software and hardware required for an application program to execute correctly

Platforms as Service Providers



- The relationship between applications and platforms can be represented as an instance of the client-server pattern
  - NB: Most platforms can support multiple independent applications

⇒ Services are often shared by multiple applications

Quality of Service

- ***Quality of Service*:**

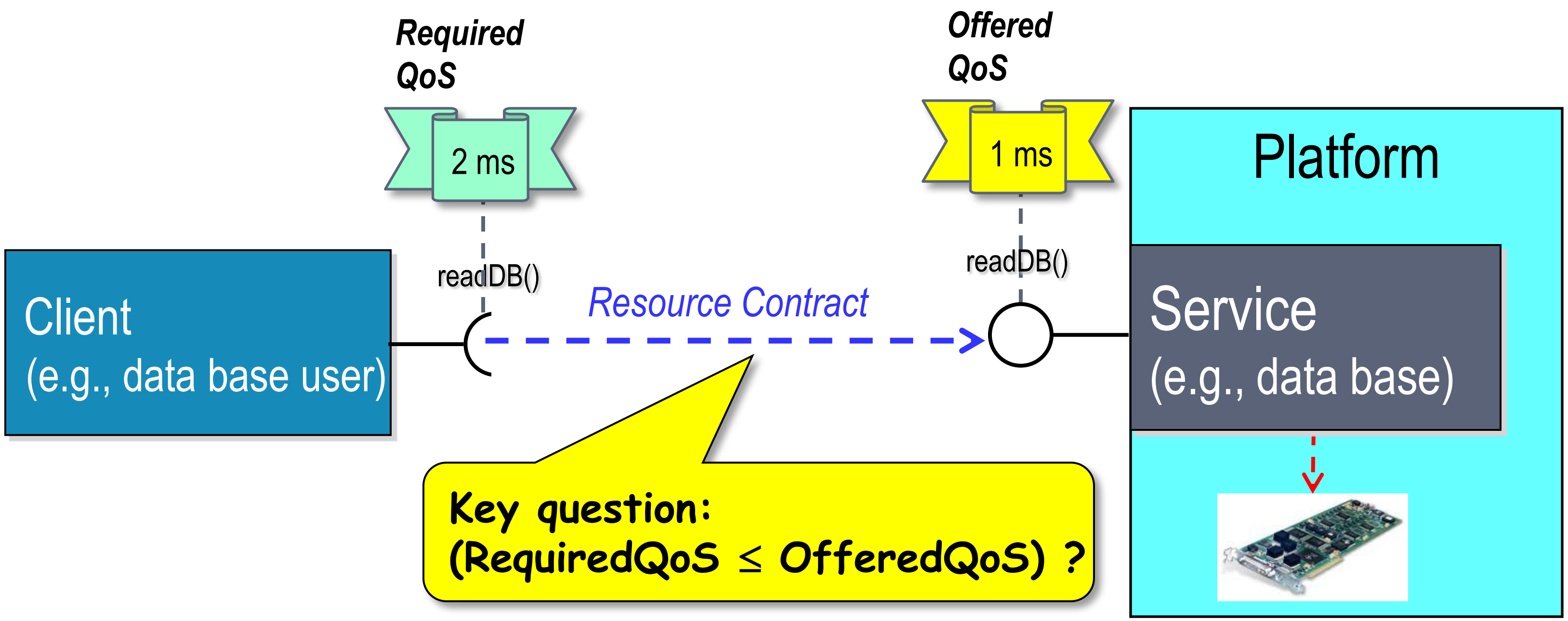
  the degree of effectiveness in the provision of a service

  - e.g. throughput, capacity, response time

- **The two sides of QoS:**
  - *offered QoS*: the QoS that is available (supply side)
  - *required QoS*: the QoS that is required (demand side)

QoS Analysis Example

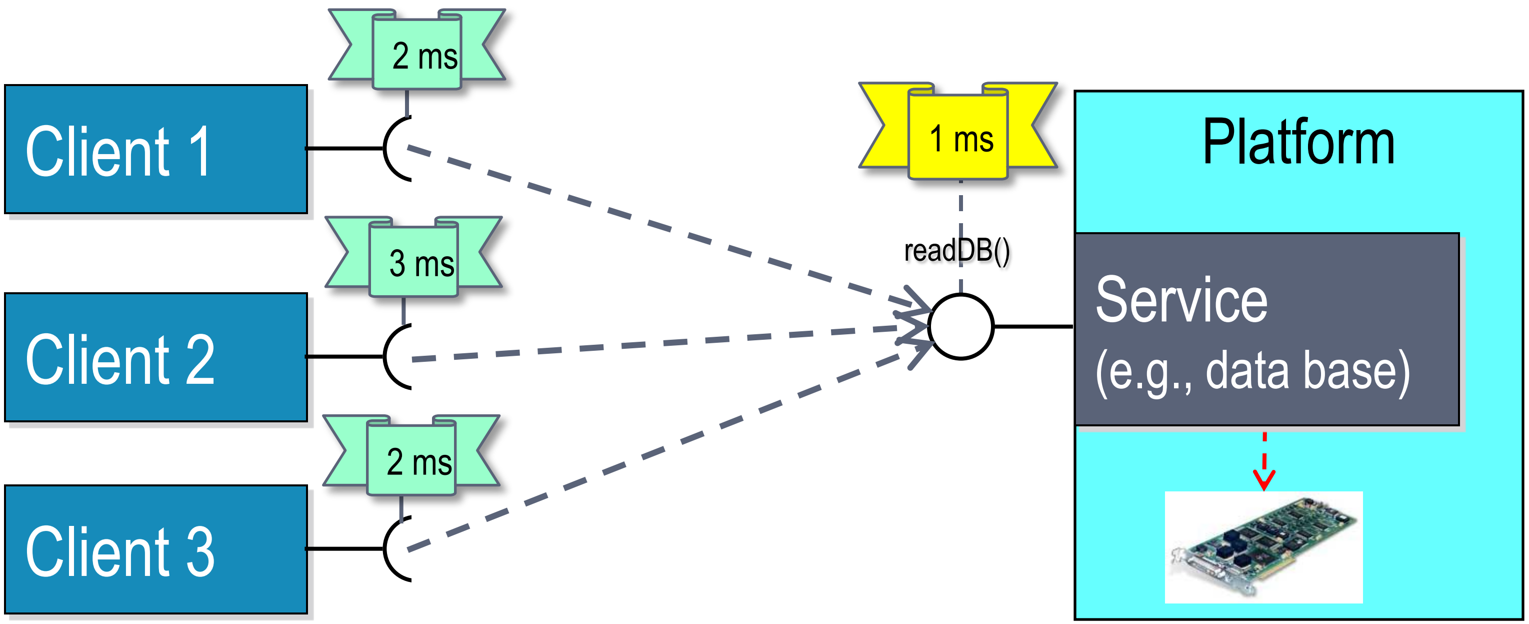
- Key analysis question: Does a service (platform) have the capacity to support its clients?
  - i.e., does supply meet demand?



The Platform Sharing Problem

Malina SOFTWARE CORP.

Multiple independent components (applications) can become implicitly coupled if they share platform resources



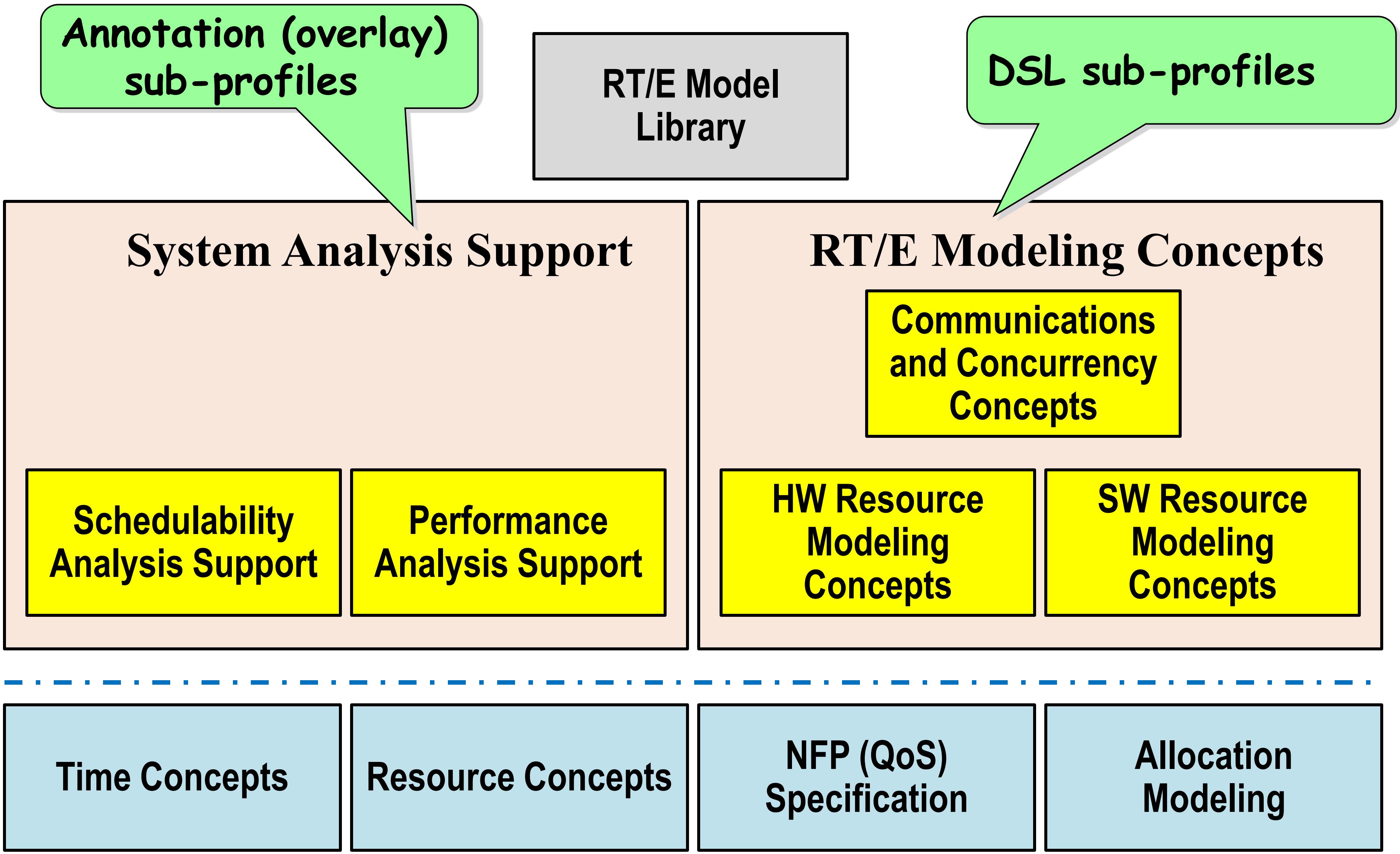
MBE Languages for RT/E Systems

- SysML: The Systems Modeling Language
  - A derivative (profile) of UML 2
  - Allows reuse of UML 2 tools and expertise
  - For high-level modeling of complete (hardware/software/wetware) systems, their surrounding contexts, and their requirements
  - Can be used in conjunction with UML 2 for smoother transition between system and software modeling
- UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE)
  - For precise modeling RT/E systems and their platforms
  - ...and, for analysis of RT/E system properties (schedulability, performance)
- Also, numerous custom domain-specific modeling languages

MARTE Capabilities

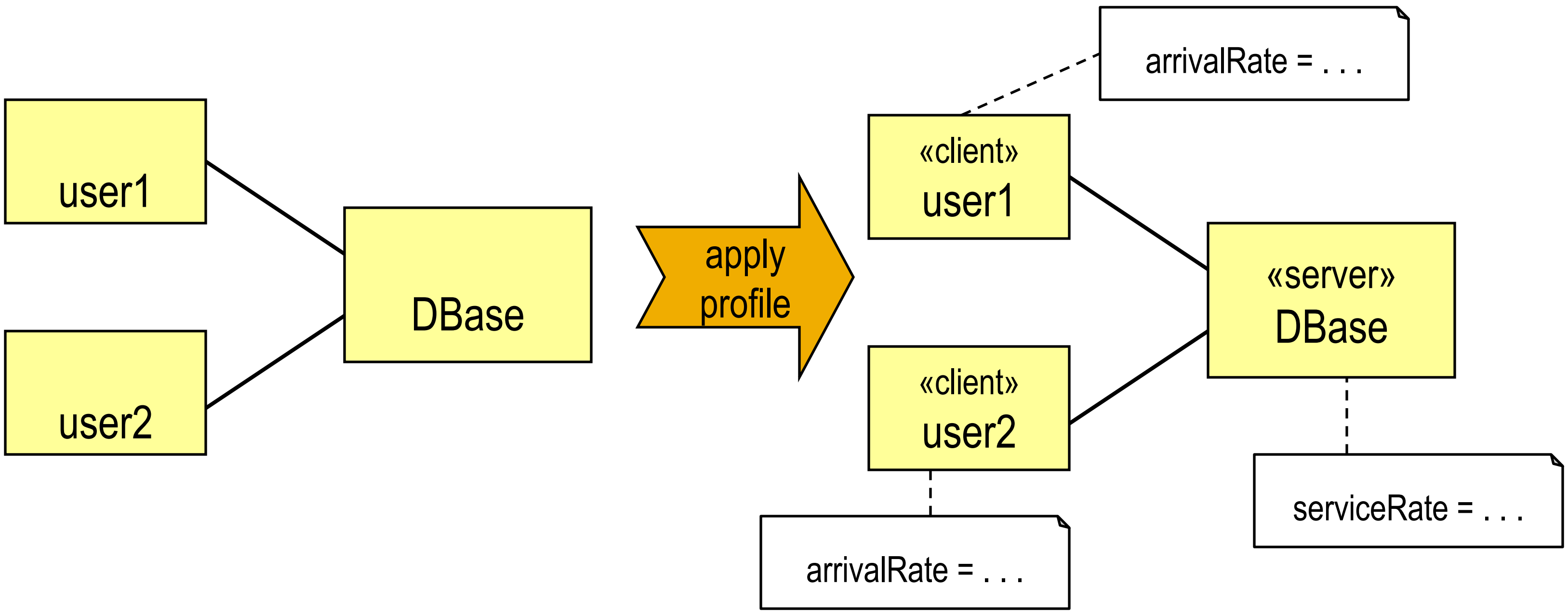
- An extensible collection of complementary domain-specific modeling languages
  - A language for modeling time
  - A language for modeling component-based real-time applications
  - A language for modeling platforms
  - A language for specifying the allocation of software to platforms
  - A language for defining QoS characteristics of software applications and platforms and defining their values
  - A model annotation language for analyzing system performance
  - A model annotation language for analyzing schedulability

MARTE Structure



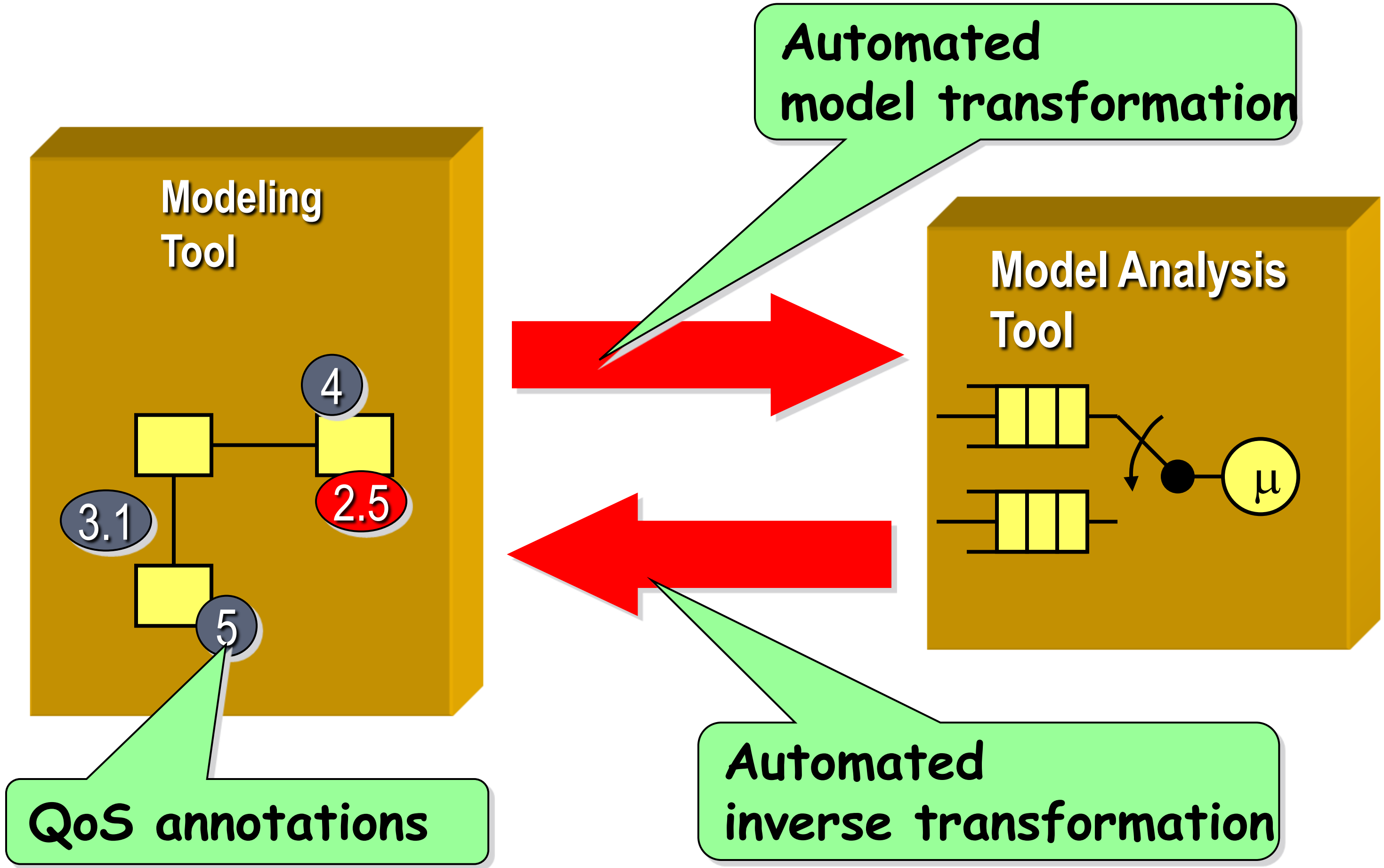
Annotation Profiles

- A profile can be used as an overlay mechanism that can be dynamically applied or "unapplied" to provide a desired view of an UML model
    - Allows a UML model to be interpreted from the perspective of the viewpoint definer
- NB: Applying or unapplying profiles has no effect on the underlying model



Automated Design Analyses with MARTE

- Analyze a design for desired or undesired properties
  - …using inter-formalism transformations and formal methods

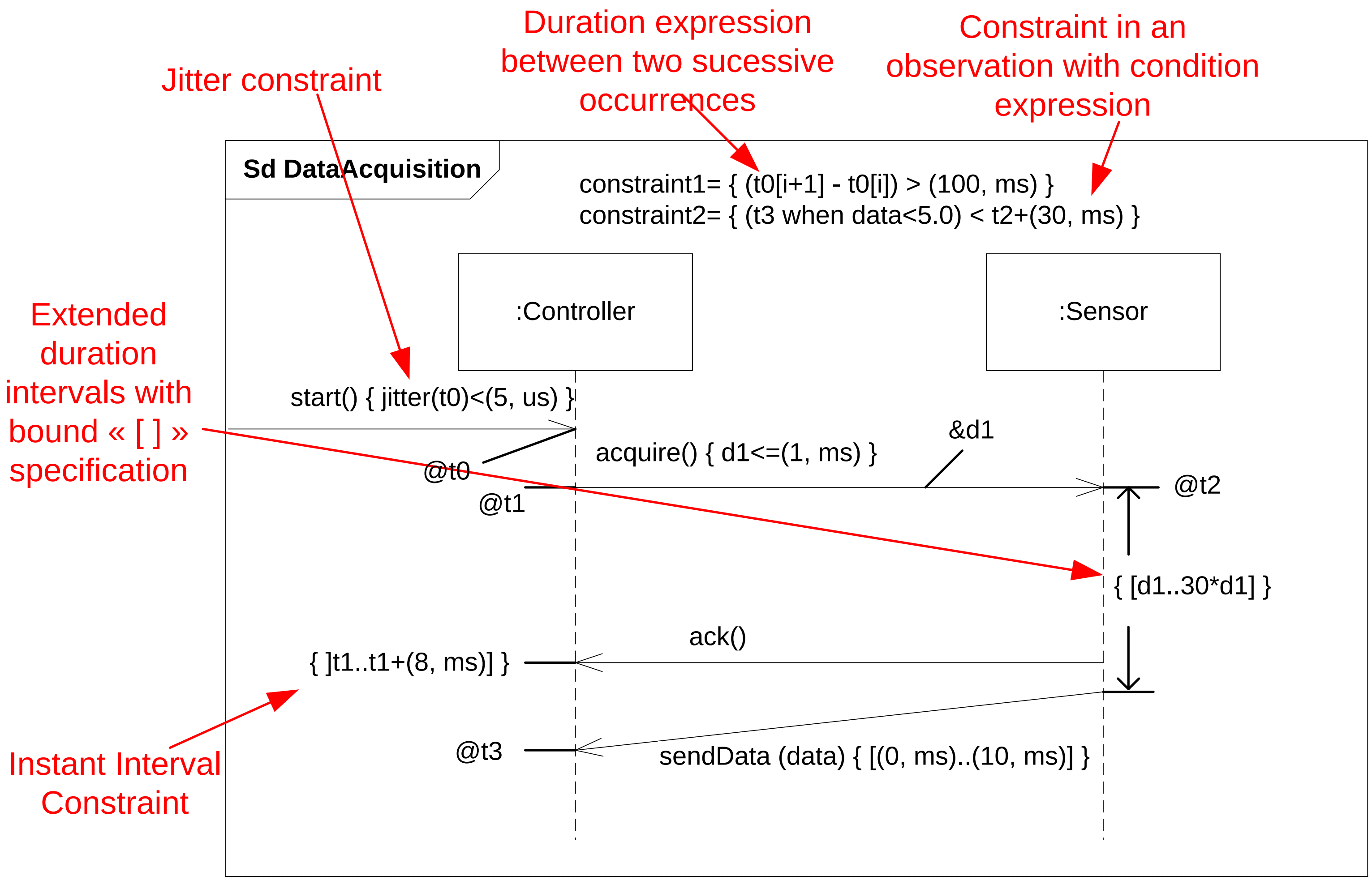


The MARTE Model of Time

Malina SOFTWARE CORP.

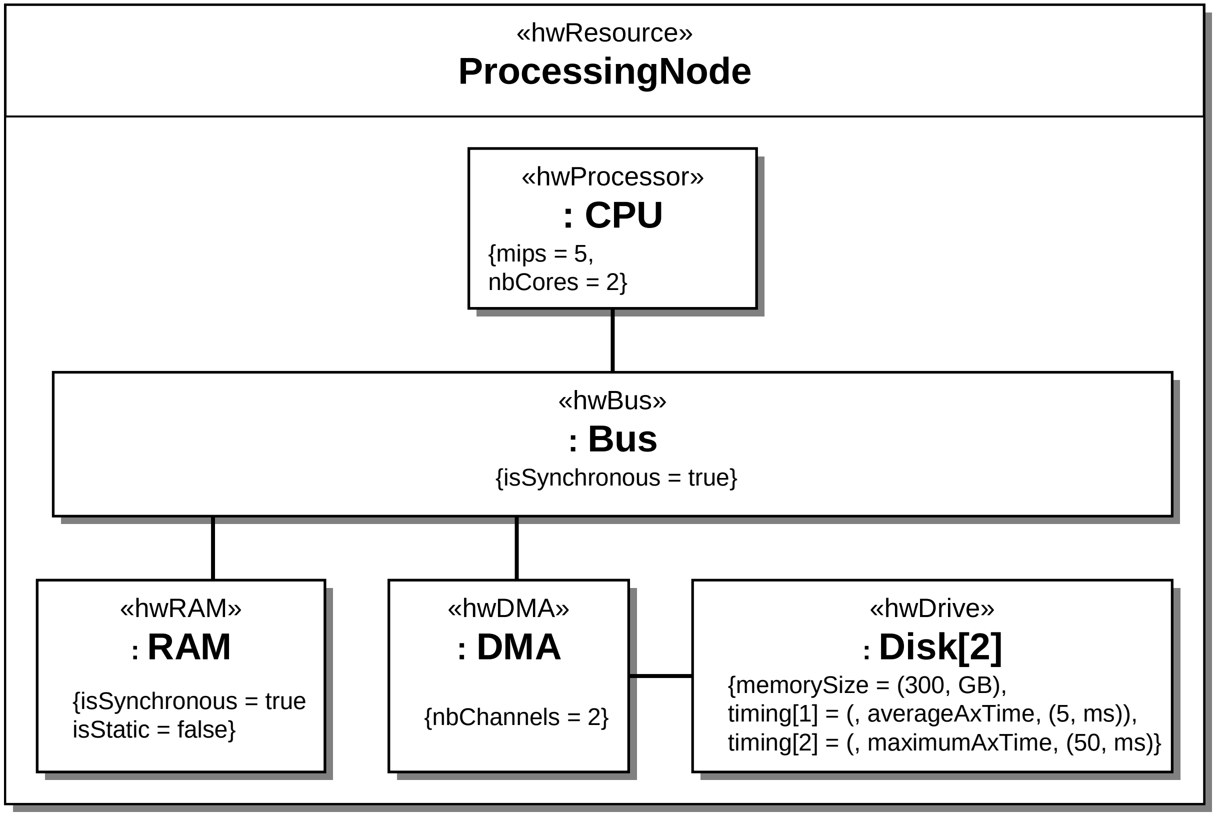
- Both discrete and continuous (dense) time models are supported
  - Time as a progression of instants
- Support for multiple concurrent time bases
  - ...and relationships between their instants (coincident, before, after)
- Timing mechanisms
  - Clocks, timers
- Time-related phenomena
  - Timed instances, timed events, durations, time constraints, etc.
- Used extensively in other parts of the profile

Example: Timing Annotations



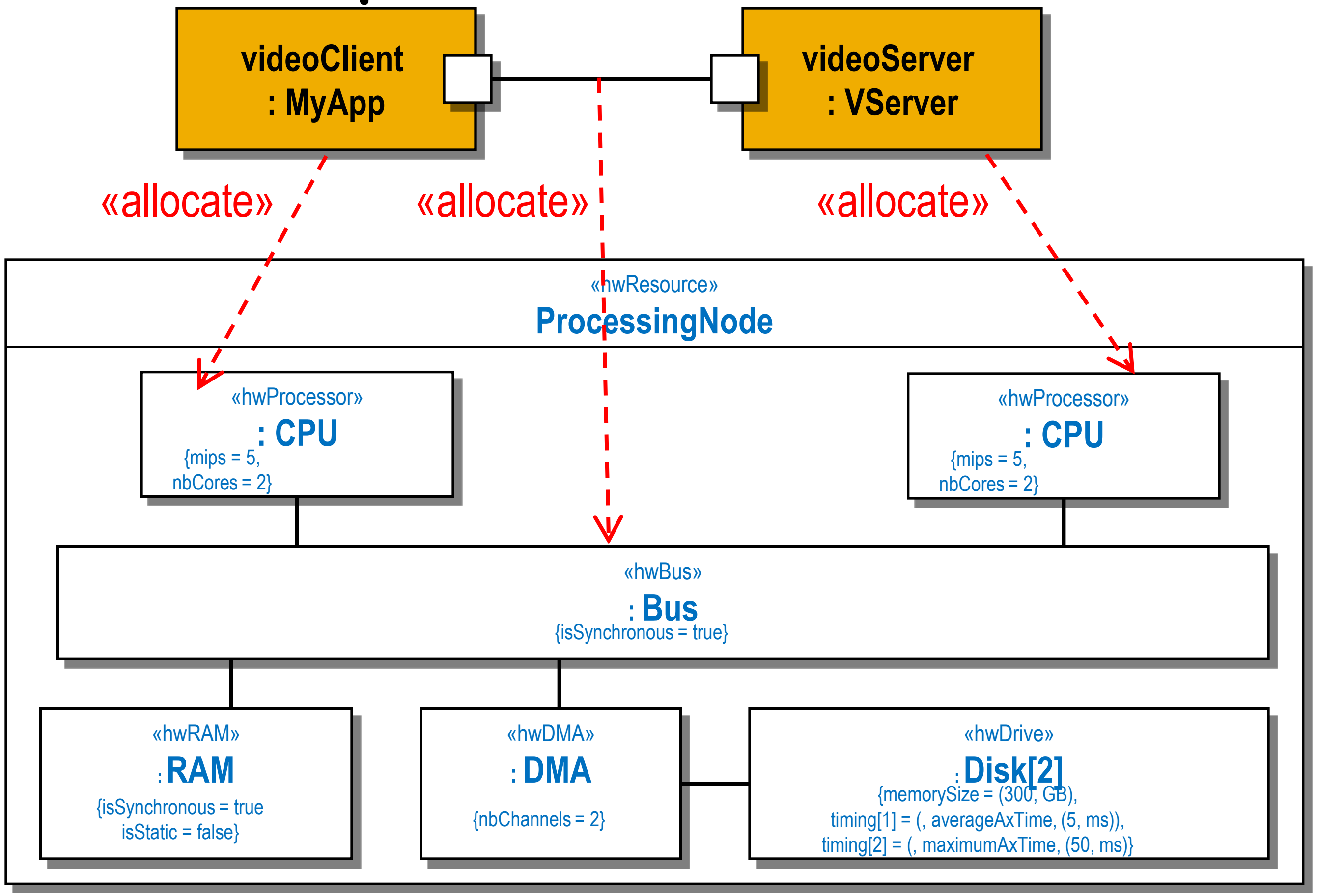
Slide courtesy of Sebastien Gerard, CEA

Example Hardware Platform Model



Example: Modeling Deployment

- Specifying the allocation of application elements to elements of the platform



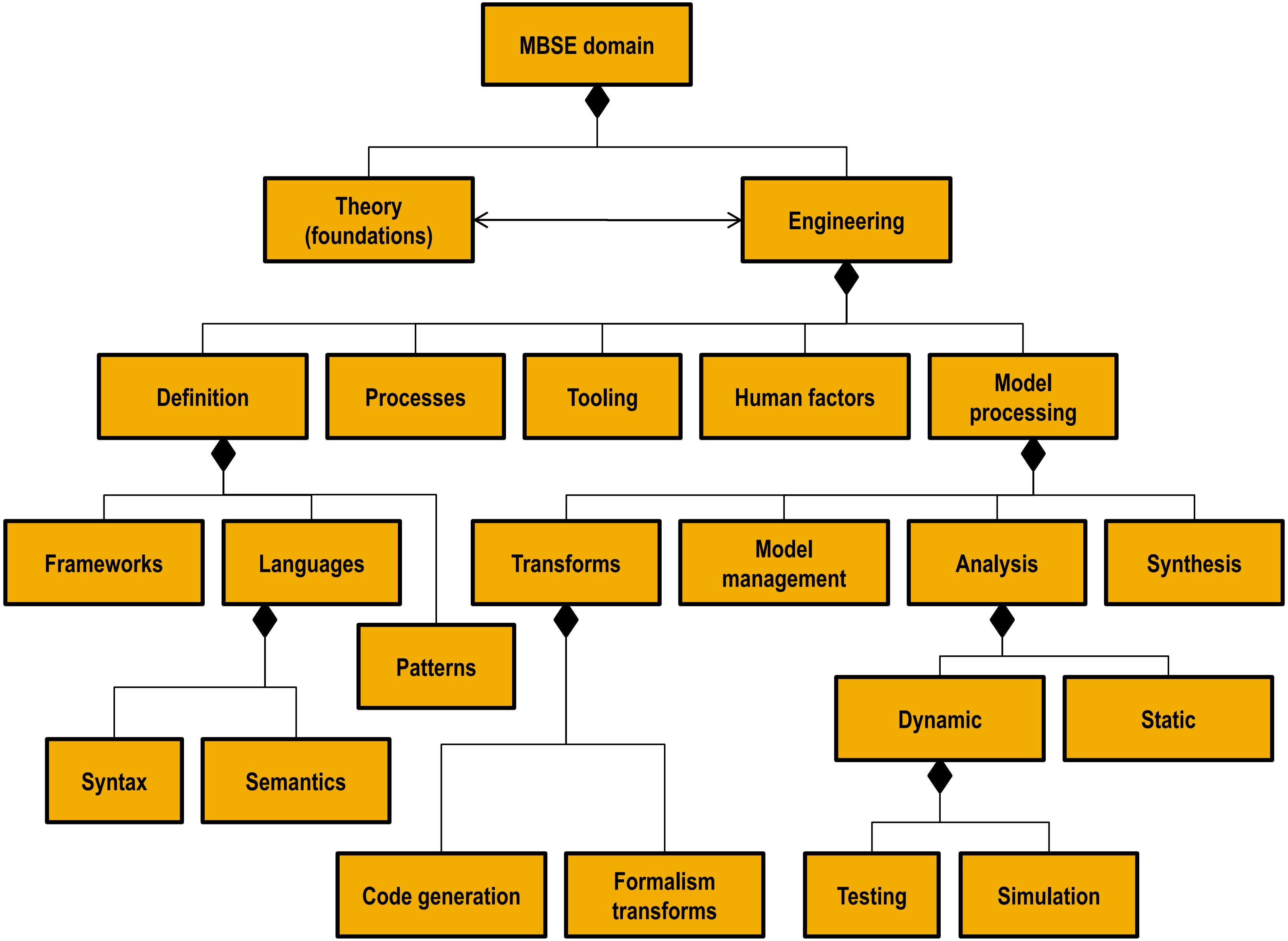
Overview

- On Model-Based Software Engineering
- Applying MBSE to Real-Time/Embedded Systems
- The Principal Research Challenges of MBSE

The State of the MBSE Domain

- At present, most MBSE technological advances are being made by industry
  - Usually by smaller specialized enterprises trying to solve a specific problem from the customer base
  - Typically technology- or vendor-specific localized solutions
- What is missing is a comprehensive theoretical underpinning for MBSE as a basis of a systematic, comprehensive, and reliable engineering discipline
  - A major set of research challenges

A Map of the MBSE Research Space

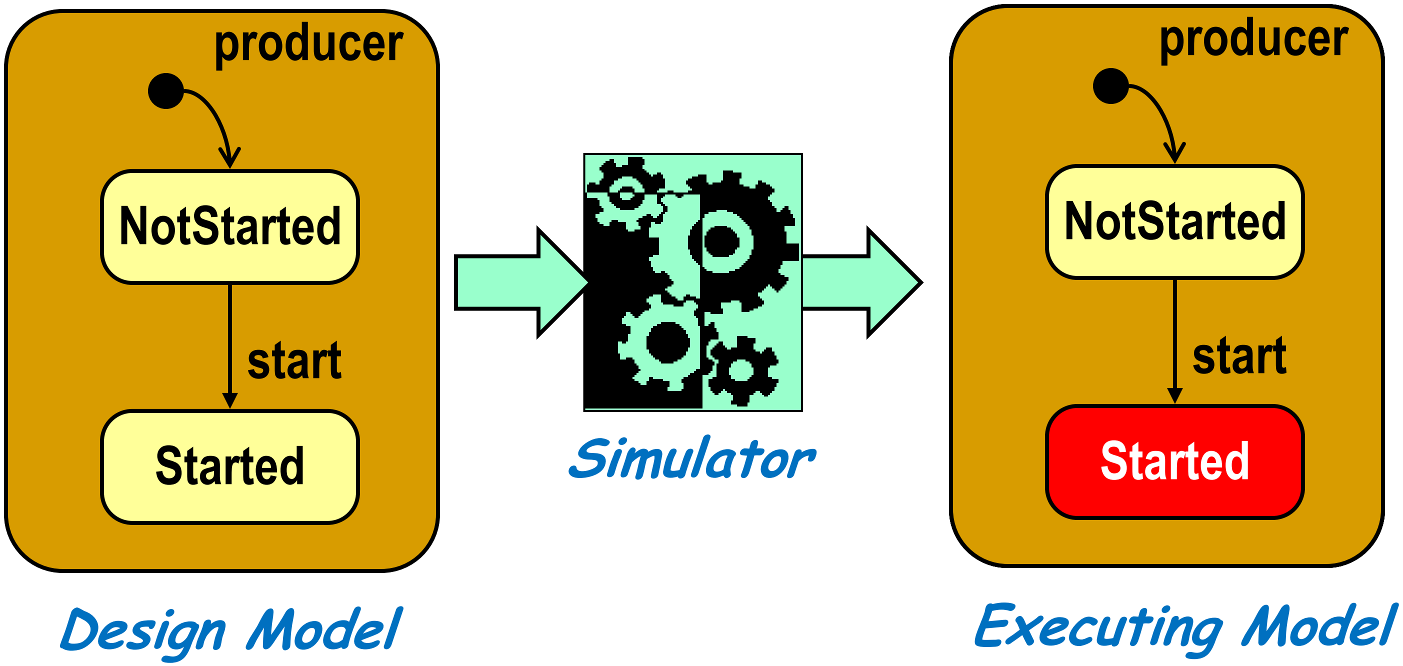


Opportunity: Predictable Computer Languages

- New generation of computer languages based on well-understood and stable formalisms
  - E.g., state machines, Petri nets, controlled structural dynamics
- Potential advantages:
  - Simpler semantics
  - More open to automated formal (mathematical) analyses methods
  - Greatly reduced likelihood of catastrophic errors
  - Can span the full range from early architectural design through implementation

Opportunity: Executable Models

- **Ability to execute and observe highly abstract and incomplete models**
  - To evaluate critical design choices as early as possible and mitigate risk
  - To gain confidence
  - To validate requirements with stakeholders



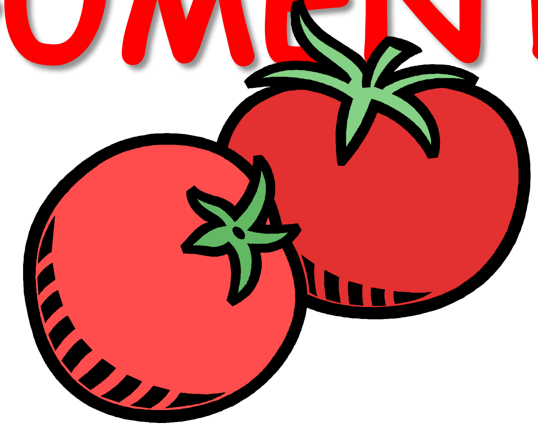
Conclusions

Malina SOFTWARE CORP.

- Traditional software technologies are incapable of adequately addressing the needs of today's RTE software
  - Too much accidental complexity
  - Insufficient automation
- Model-based software engineering methods have proven that they *can* provide significant enhancements to productivity and quality
  - Higher degrees of automation and abstraction
  - Use of domain-specific languages (MARTE) and tools (Papyrus)
- However, many research challenges still remain to be resolved before it can claim to be a mature engineering discipline

DANKE SCHÖN:

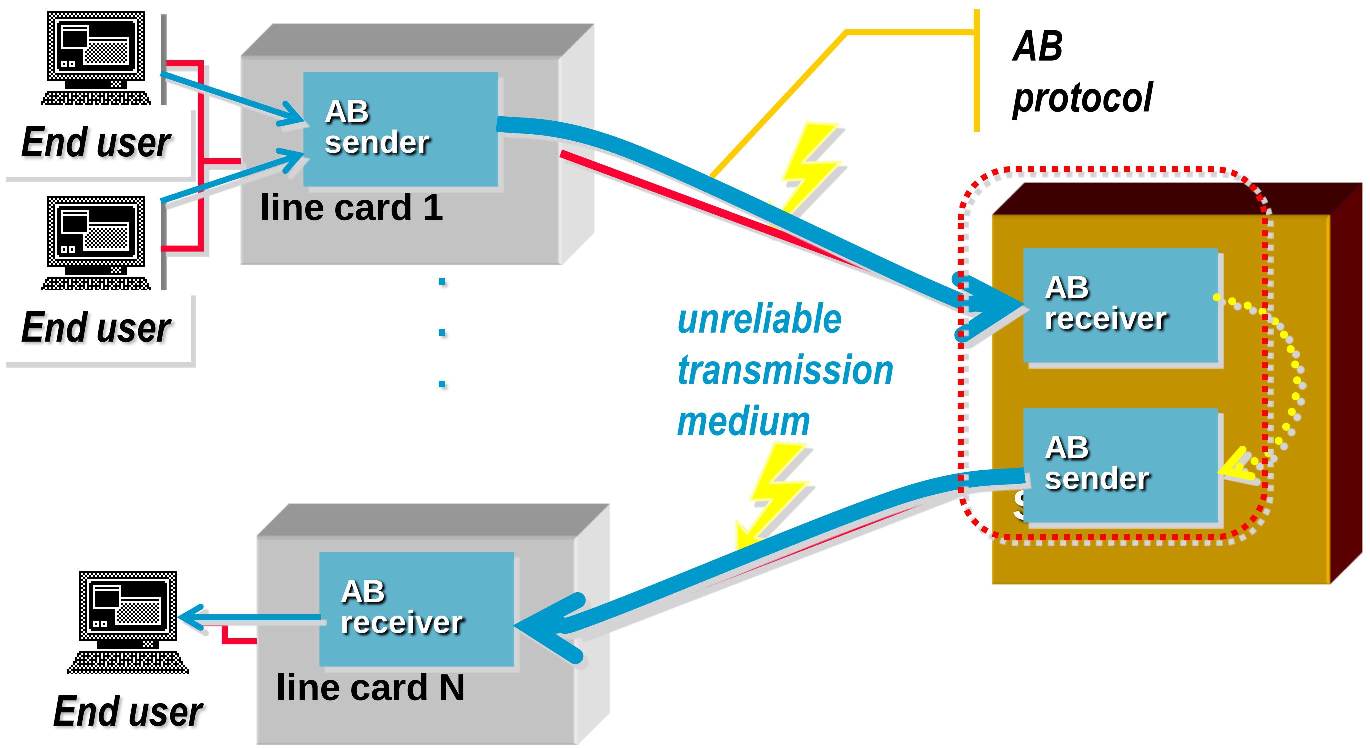
QUESTIONS, COMMENTS, ARGUMENTS...



Example:
The Recursive Control Pattern – A Standard Architecture for MDD of Real-Time Systems

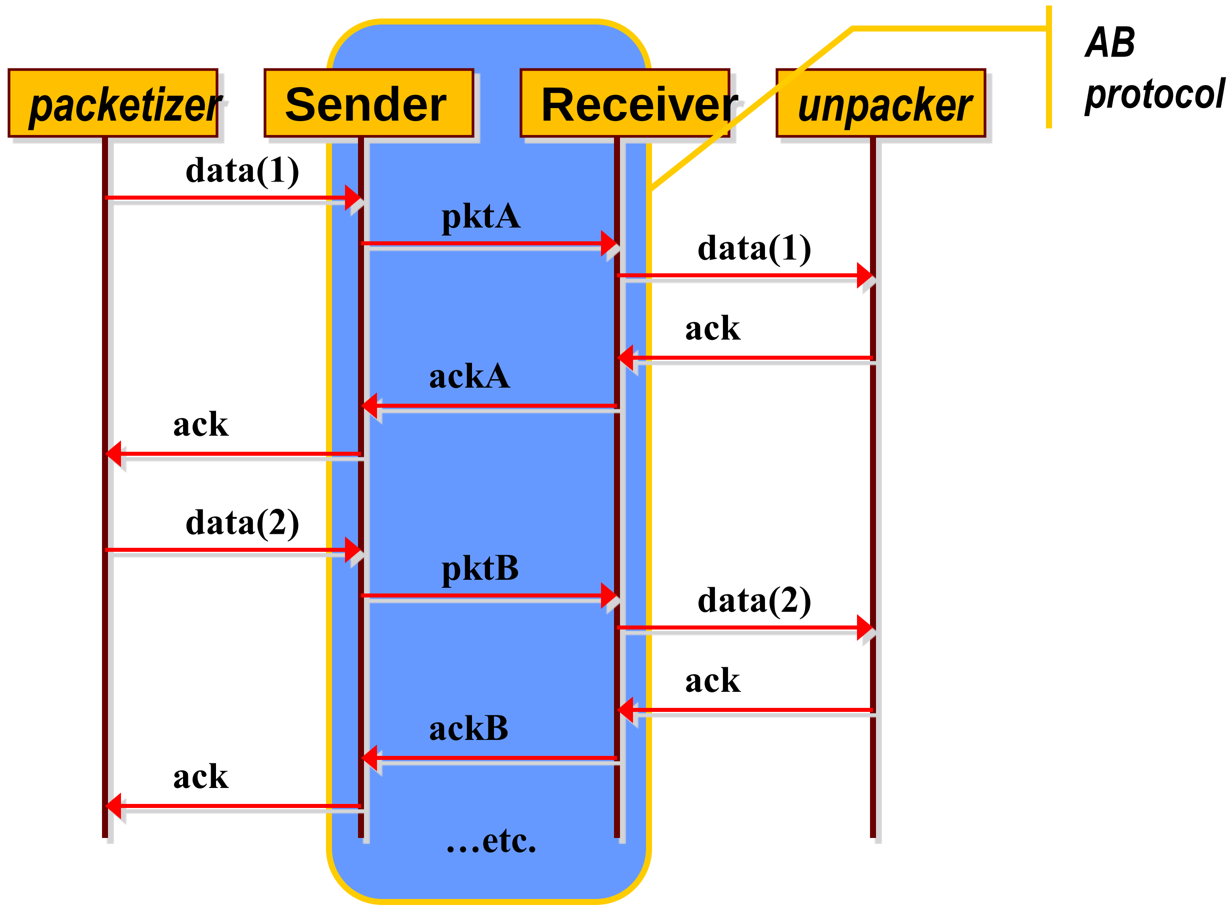
Example System

- A multi-line packet switch that uses the alternating-bit protocol as its link protocol



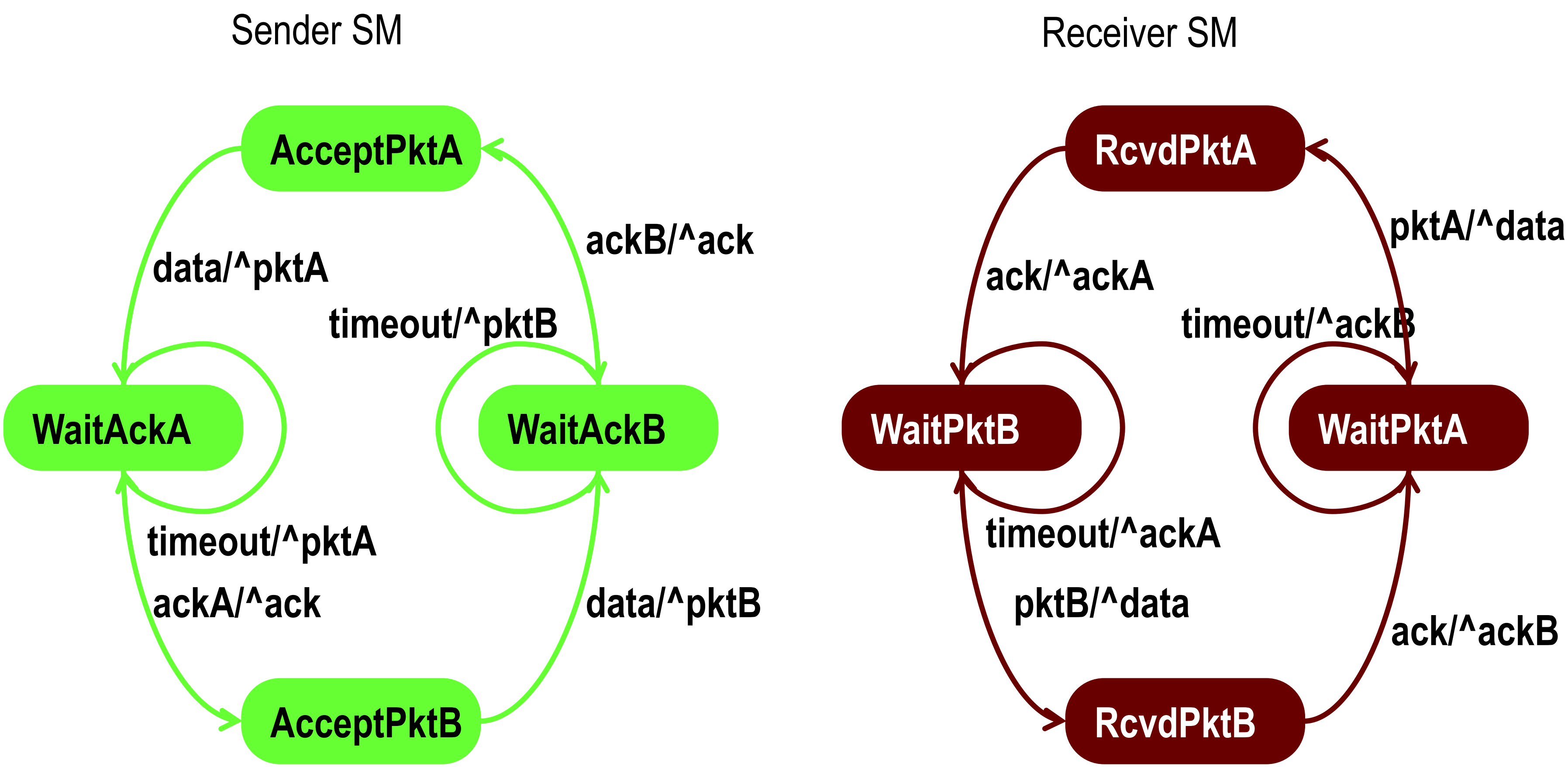
Alternating Bit Protocol (1)

- A simple one-way point-to-point packet protocol



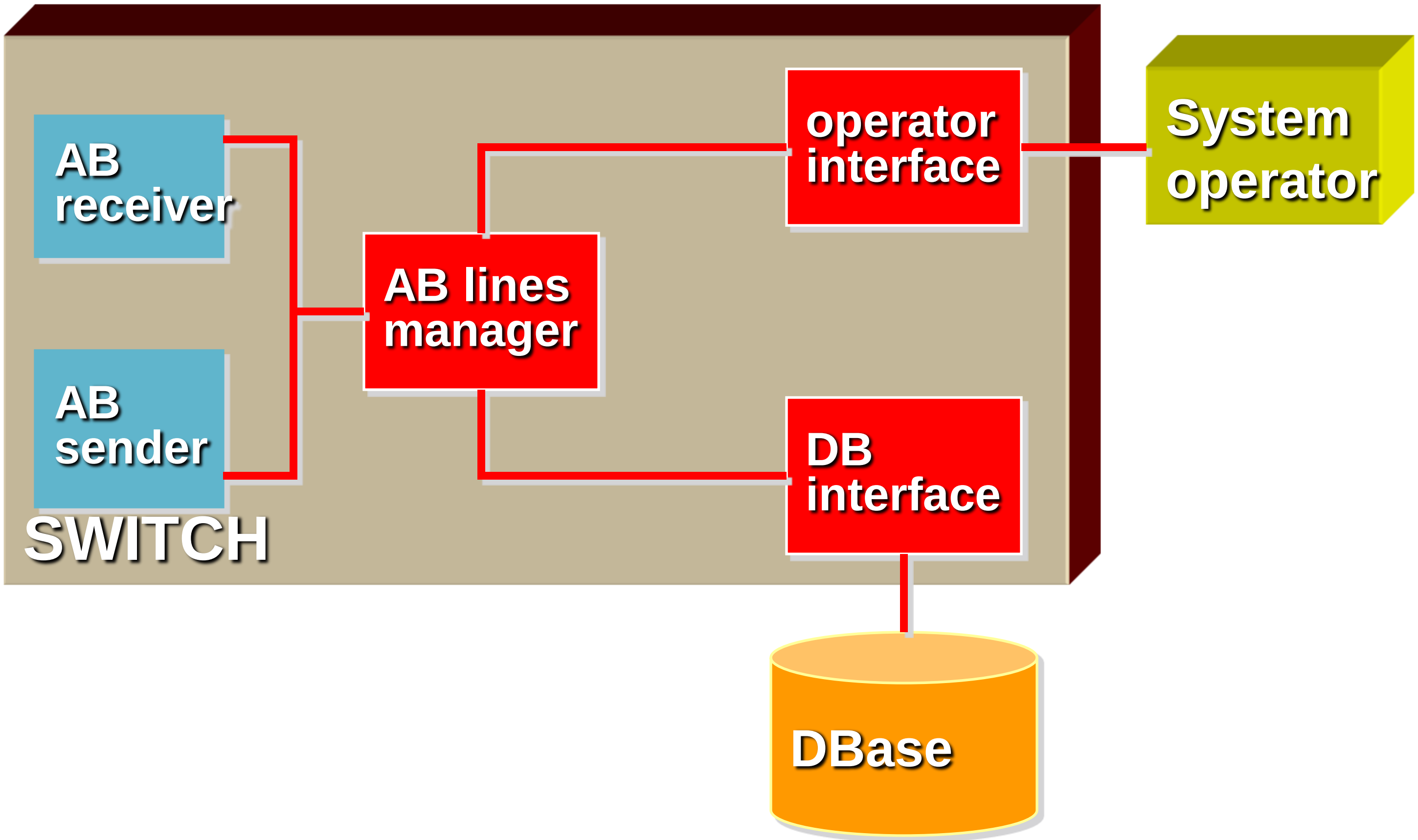
Alternating Bit Protocol (2)

- State machine specification



Additional Considerations

- Support infrastructure



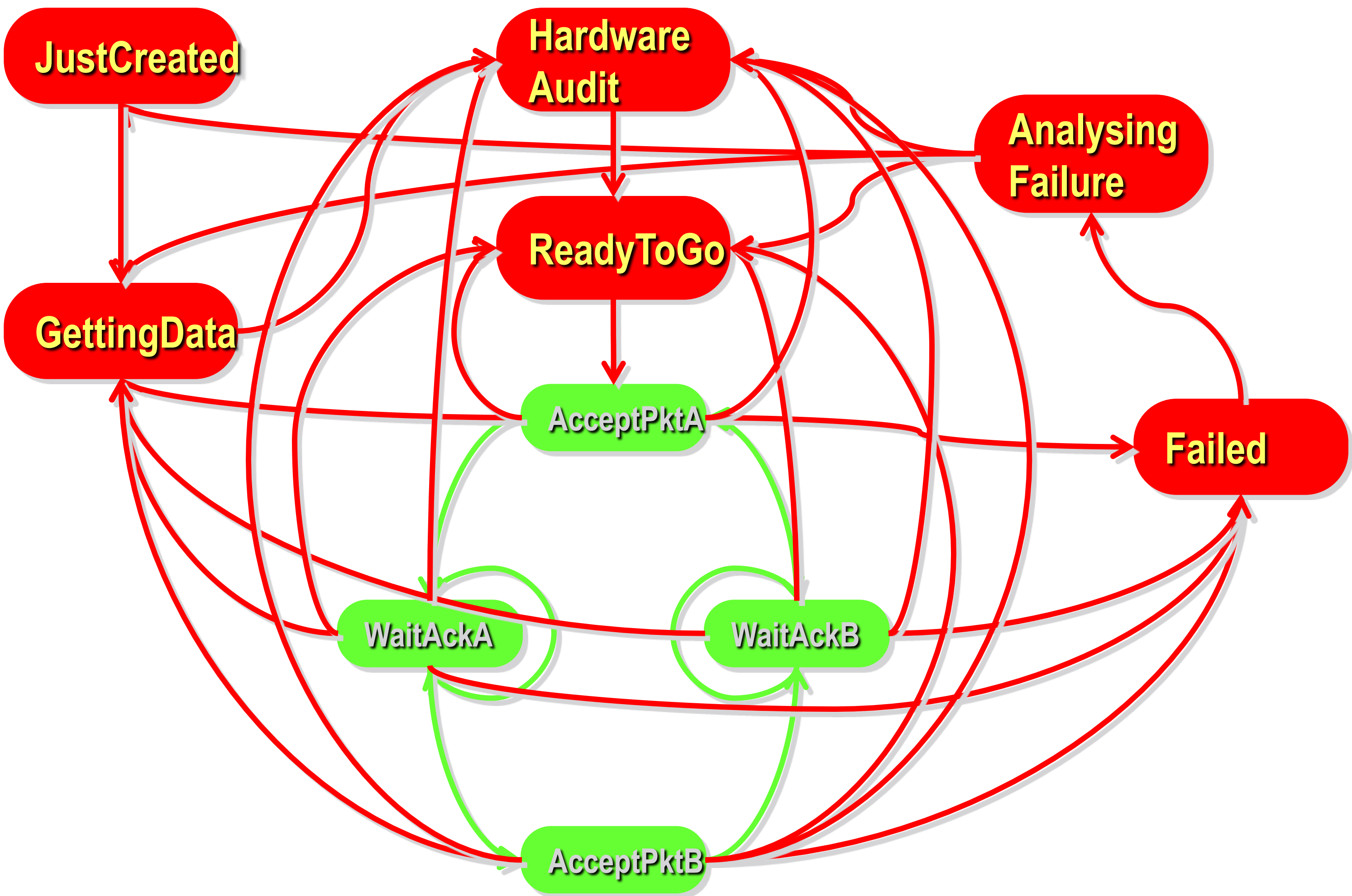
Control

The set of (additional) mechanisms and actions required to bring a system into the desired operational state and to maintain it in that state in the face of various planned and unplanned disruptions

- For software systems this includes:
  - system/component start-up and shut-down
  - failure detection/reporting/recovery
  - system administration, maintenance, and provisioning
  - (on-line) software upgrade

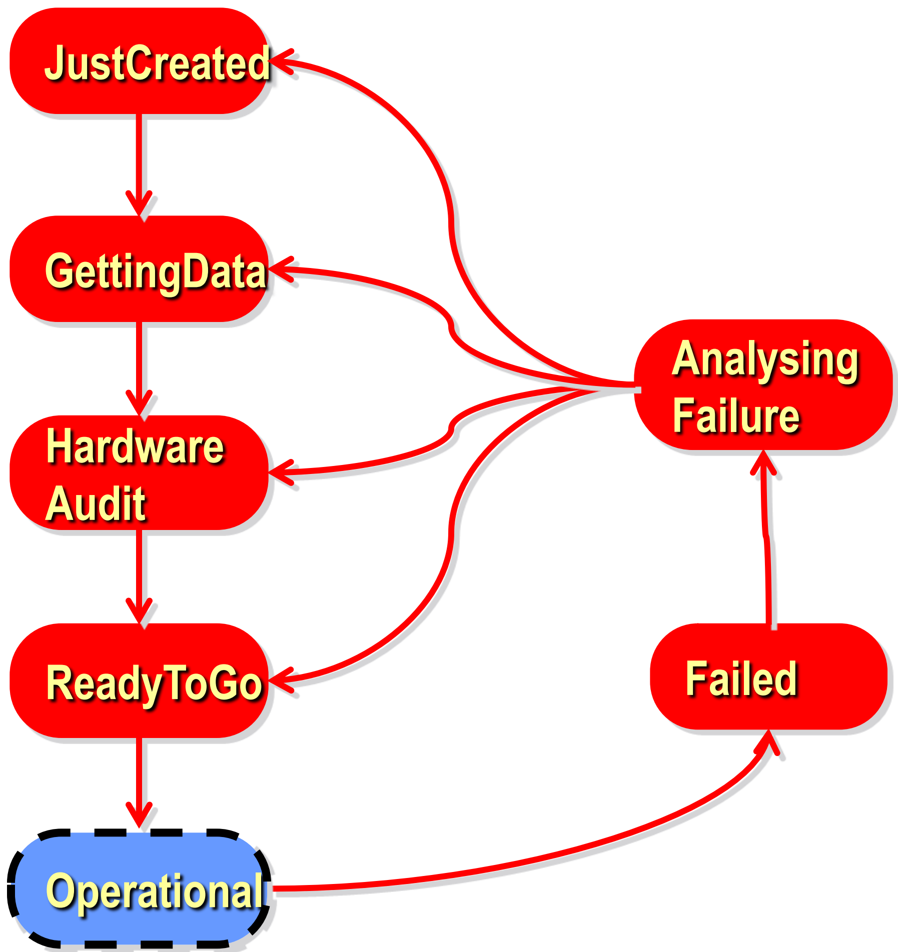
Retrofitting Control Behavior

Malina SOFTWARE CORP.



The Control Automaton

- In isolation, the same control behavior appears much simpler



Control versus Function

- **Control behavior is often treated in an ad hoc manner, since it is not part of the primary system functionality**
  - typically retrofitted into the framework optimized for the functional behavior
  - leads to controllability and stability problems
- ***However, in highly-dependable systems as much as 80% of the system code is dedicated to control behavior!***

Some Important Observations

- ***Control predicates function***
  - before a system can perform its primary function, it first has to reach its operational state
- ***Control behavior is often independent of functional behavior***
  - the process by which a system reaches its operational state is often the same regardless of the specific functionality of the component

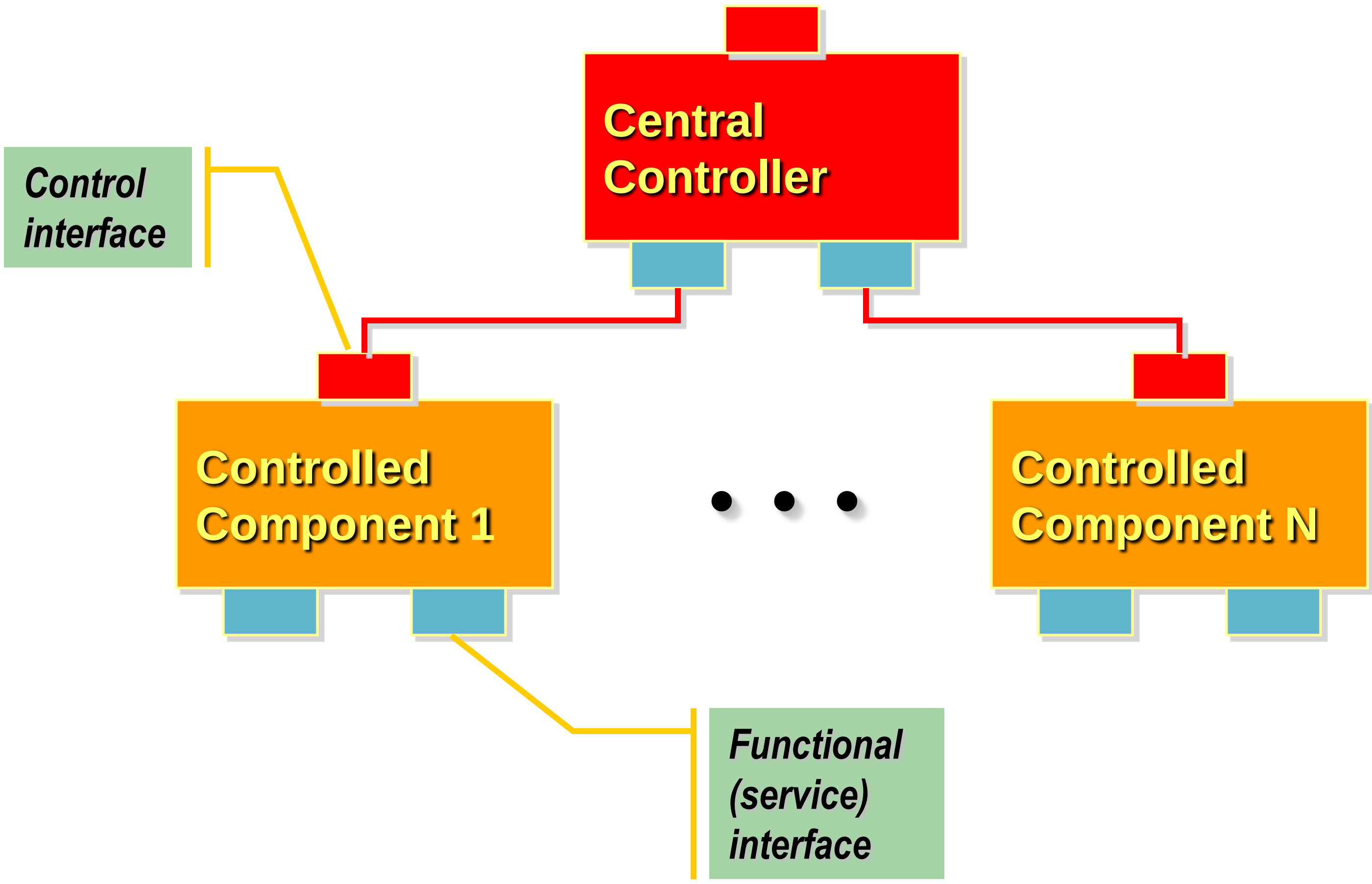
Basic Design Principles

- ***Separate control from function***
  - separate control components from functional components
  - separate control from functional interfaces
  - imbed functional behavior within control behavior
- ***Centralize control (decision making)***
  - if possible, focus control in one component
  - place control policies in the control components and control mechanisms inside the controlled components

The Basic Structural Pattern

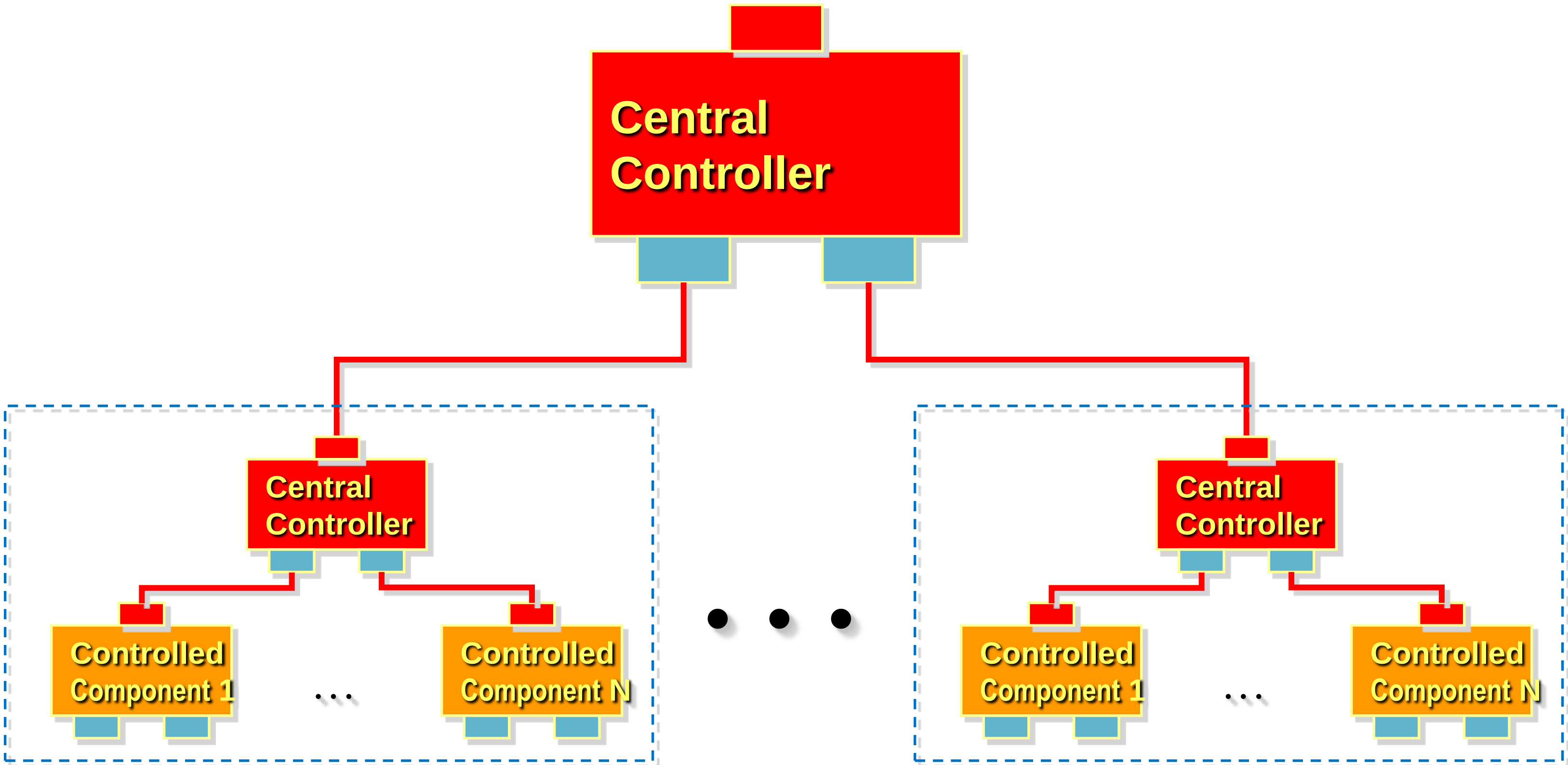
Malina SOFTWARE CORP.

- Set of components that need to be controlled in a coordinated fashion



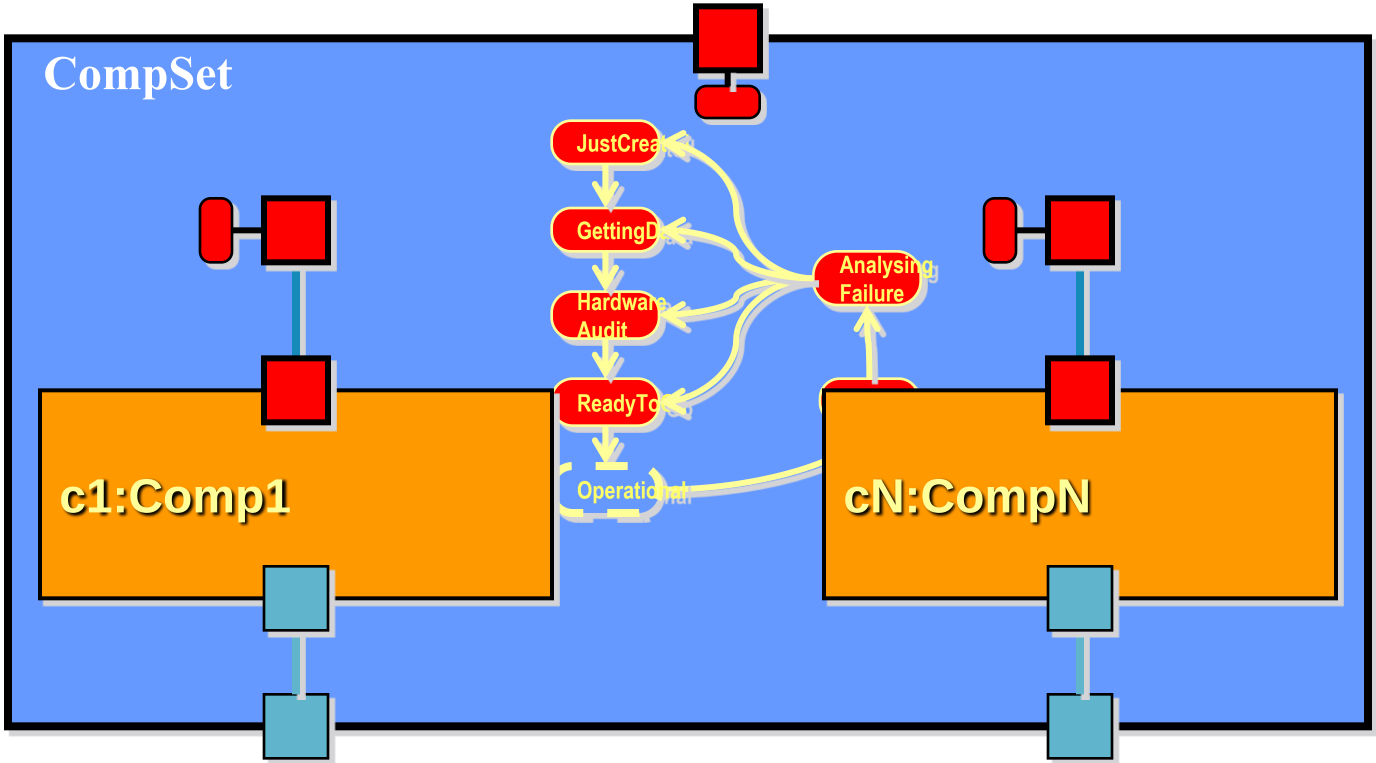
Recursive Application

- Hierarchical control
  - scales up to arbitrary number of levels



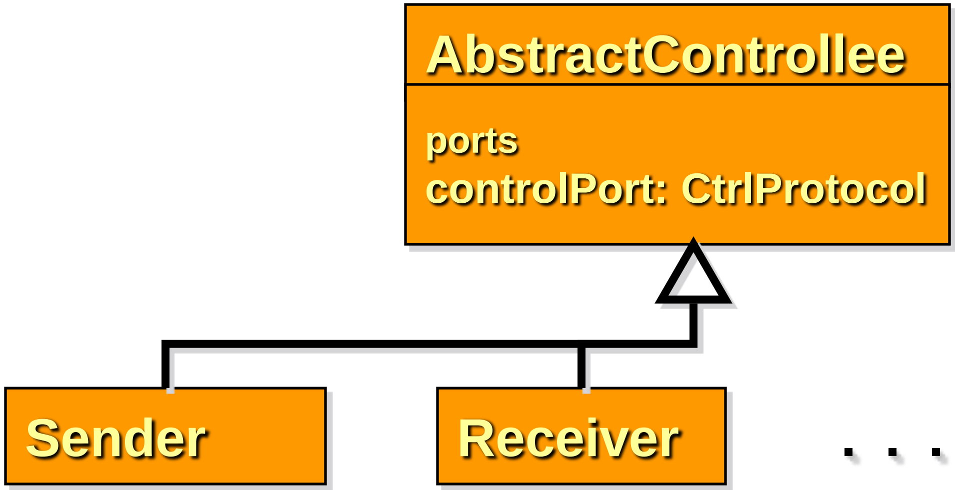
Realization with Active Objects and Ports

- Composite plays role of centralized controller



Exploiting Inheritance

- Abstract control classes can capture common control behavior and structure
- Different subclasses capture function-specific behavior



Exploiting Hierarchical States