mbeddr
C the Difference. C the Future.

An extensible version of the C programming language for Embedded Programming

C the Difference - C the Future

Bundesministerium
für Bildung
und Forschung

gefördert durch das BMBF
Förderkennzeichen 01IS11014

What if…

you could change languages like you can change programs?

```
module WriteATestCase from cdesignpaper.unittest imports nothing {

  var int8_t failedTests;

  int32_t main(int32_t argc, int8_t*[ ] argv) {
    testMultiply();
    return failedTests;
  } main (function)

  void testMultiply() {
    if ( times2(21) != 42 ) { failedTests++; } if
  } testMultiply (function)

  int8_t times2(int8_t a) {
    return 2 * a;
  } times2 (function)
}
```

```
module UnitTestDemo from cdesignpaper.unittest imports nothing {

  int32_t main(int32_t argc, int8_t*[ ] argv) {
    return test testMultiply;
  } main (function)

  exported test case testMultiply {
    assert(0) times2(21) == 42;
  } testMultiply(test case)

  int8_t times2(int8_t a) {
    return 2 * a;
  } times2 (function)
}
```

```
module UnitTestDemo from cdesignpaper.unittest imports nothing {

  int32_t main(int32_t argc, int8_t*[ ] argv) {
    return test testMultiply;
  } main (function)

  exported test case testMultiply {
    assert(0) times2(21) == 42;
  } testMultiply(test case)

  int8_t times2(int8_t a) {
    return 2 * a;
  } times2 (function)
}
```

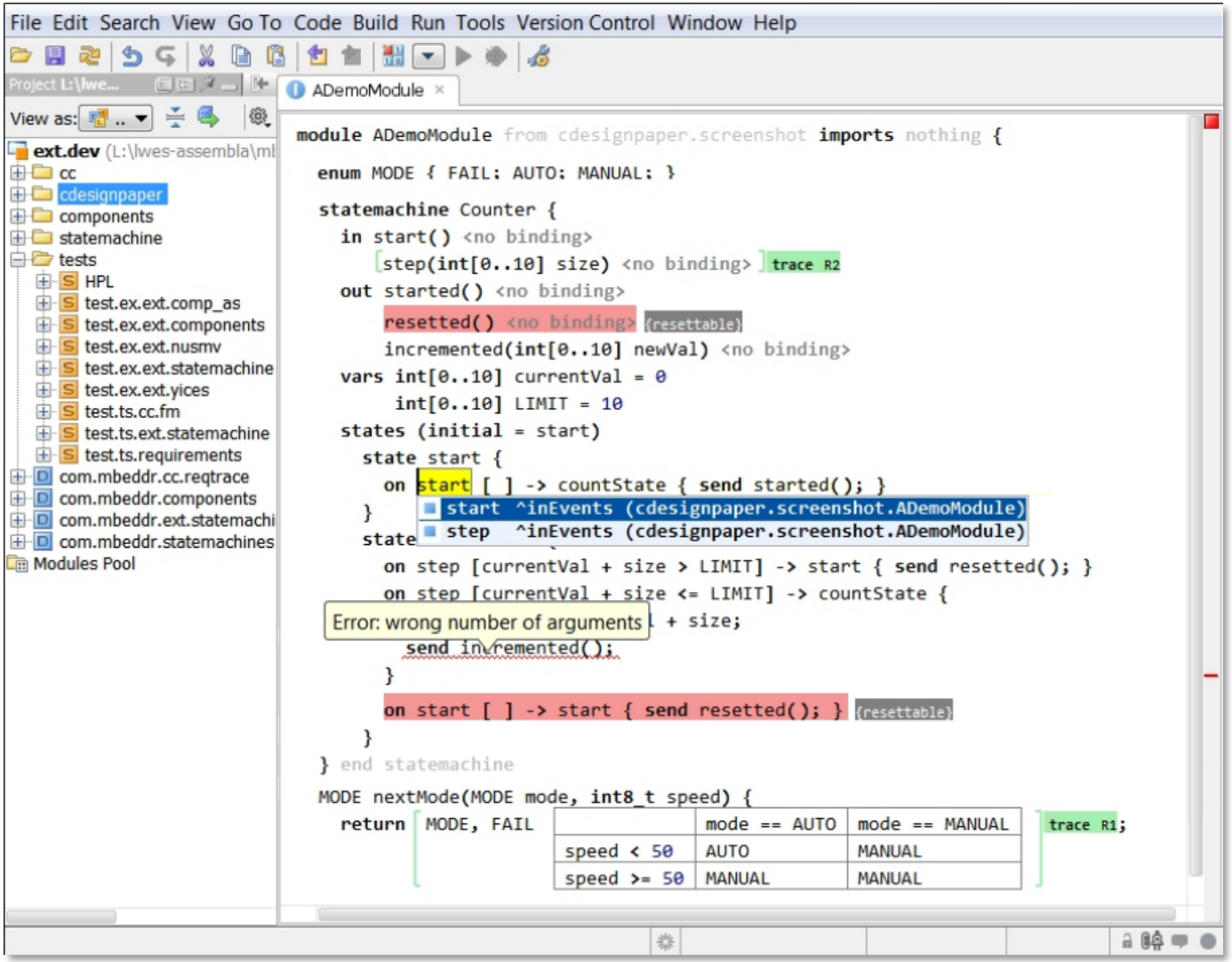
```
module UnitTestDemo from cdesignpaper.unittest imports nothing {

  int32_t main(int32_t argc, int8_t*[ ] argv) {
    return test testMultiply;
  } main (function)

  exported test case testMultiply {
    assert(0) times2(21) == 42;
  } testMultiply(test case)

  int8_t times2(int8_t a) {
    return 2 * a;
  } times2 (function)
}
```

```
module UnitTestDemo from cdesignpaper.unittest imports nothing {

  int32_t main(int32_t argc, int8_t*[ ] argv) {
    return test testMultiply;
  } main (function)

  exported test case testMultiply {
    assert(0) times2(21) == 42;
  } testMultiply(test case)

  int8_t times2(int8_t a) {
    return 2 * a;
  } times2 (function)
}
```

mbeddr C Approach

An extensible C with support for formal methods, requirements and PLE.

IDE for Everything



A debugger
for all of that

SDK for building your own Language Extensions!

IDE for Everything

JetBrains

MPS

Open Source

Language Workbench

Challenges

in embedded software development

Abstraction without Runtime Cost

C considered unsafe

Program Annotations

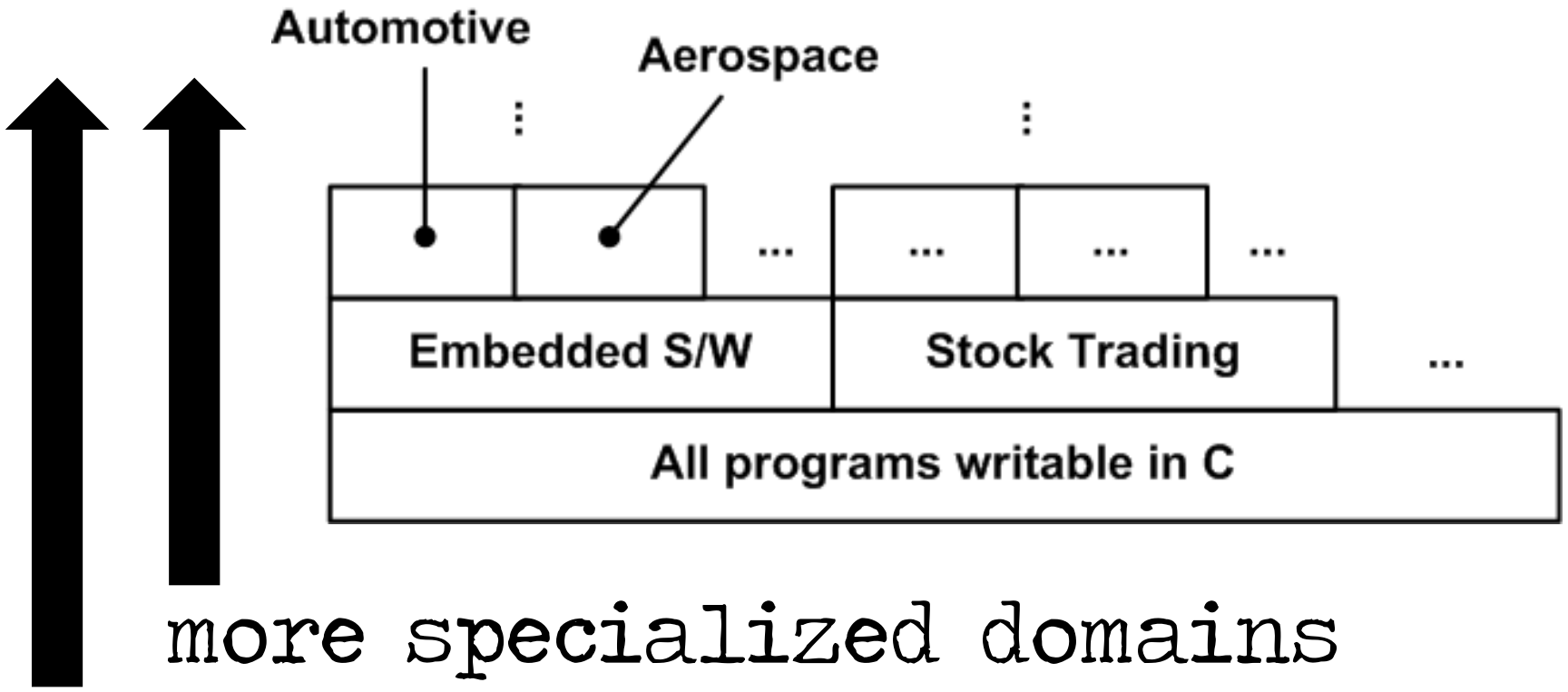
Static Checks and Verification

Product Lines and Requirement Traces

Separate, hard to integrate Tools

mbeddr C Solution Philosophy

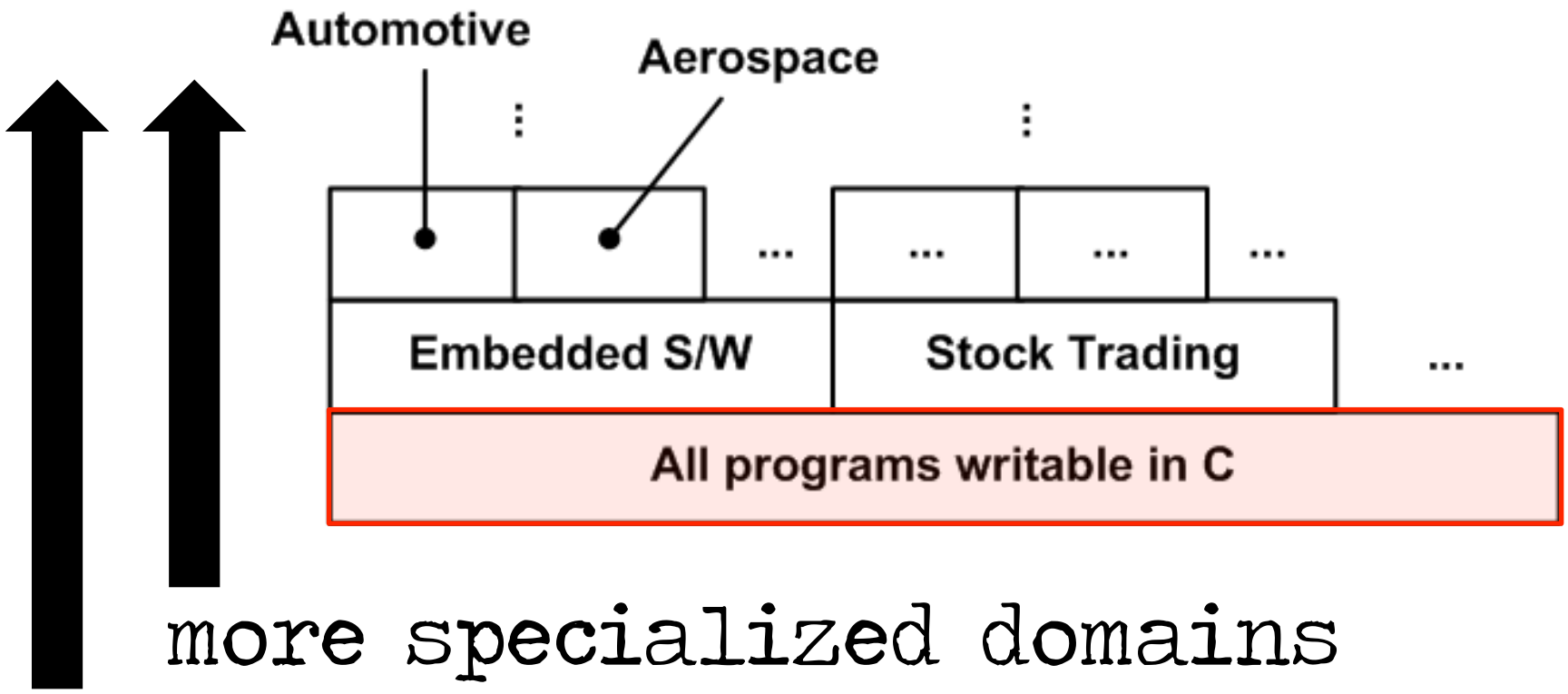
Extension



more specialized domains

more specialized languages

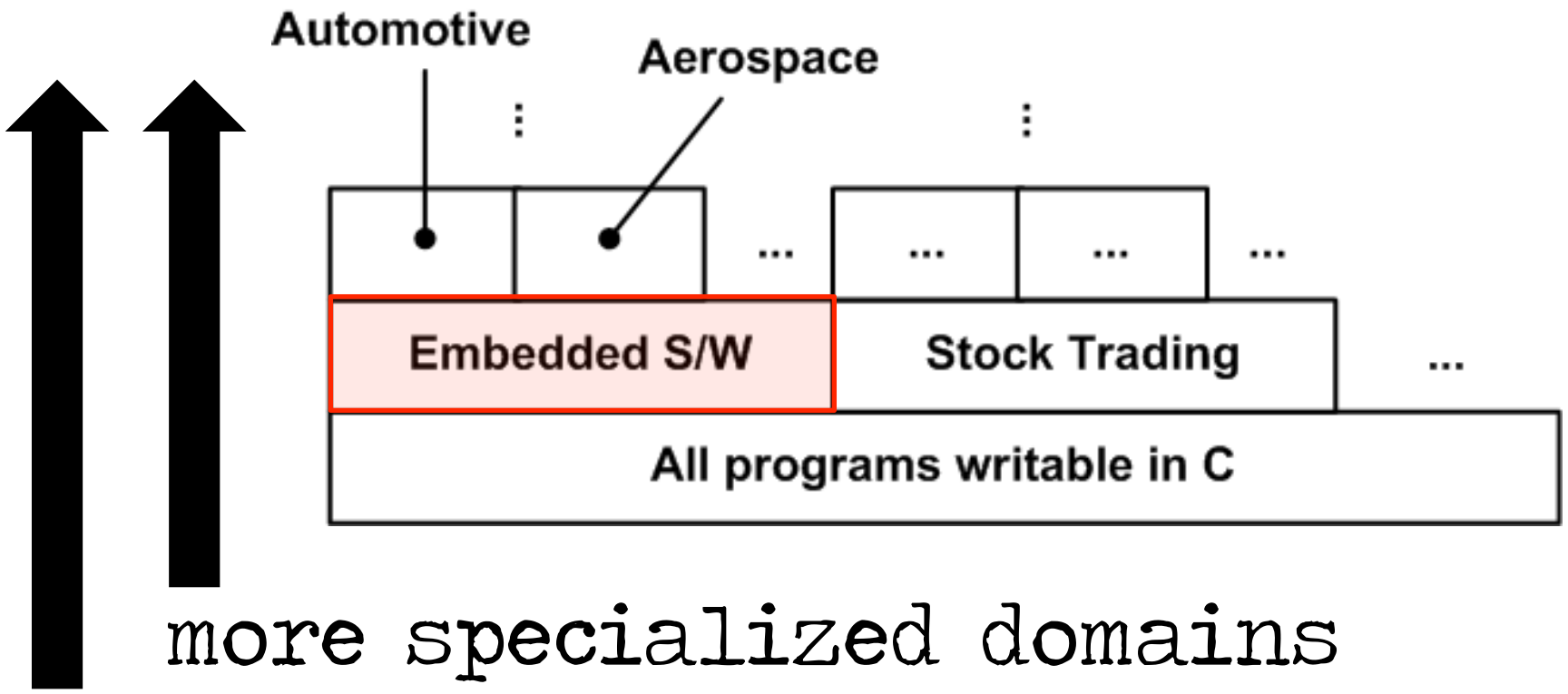
Extension



more specialized domains

more specialized languages

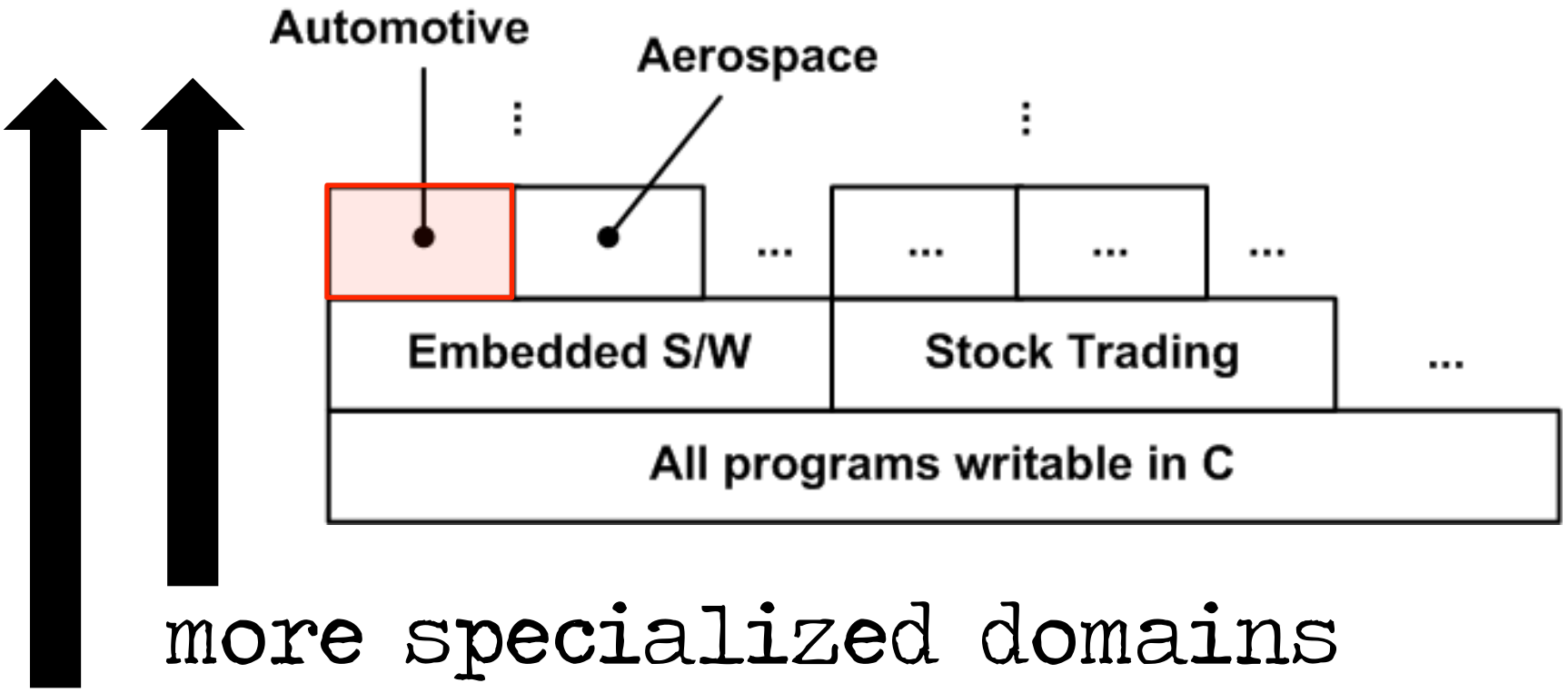
Extension



more specialized domains

more specialized languages

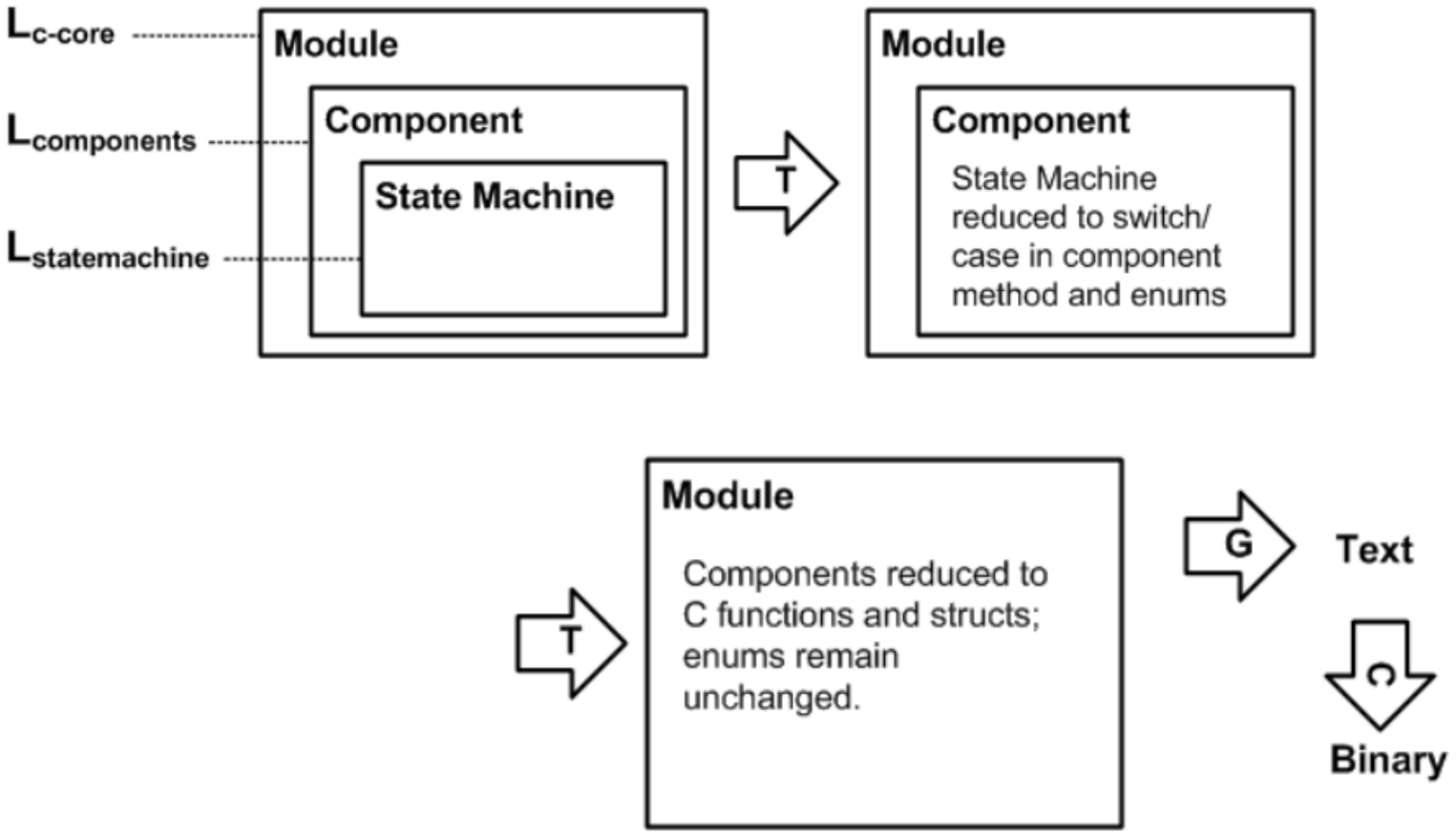
Extension



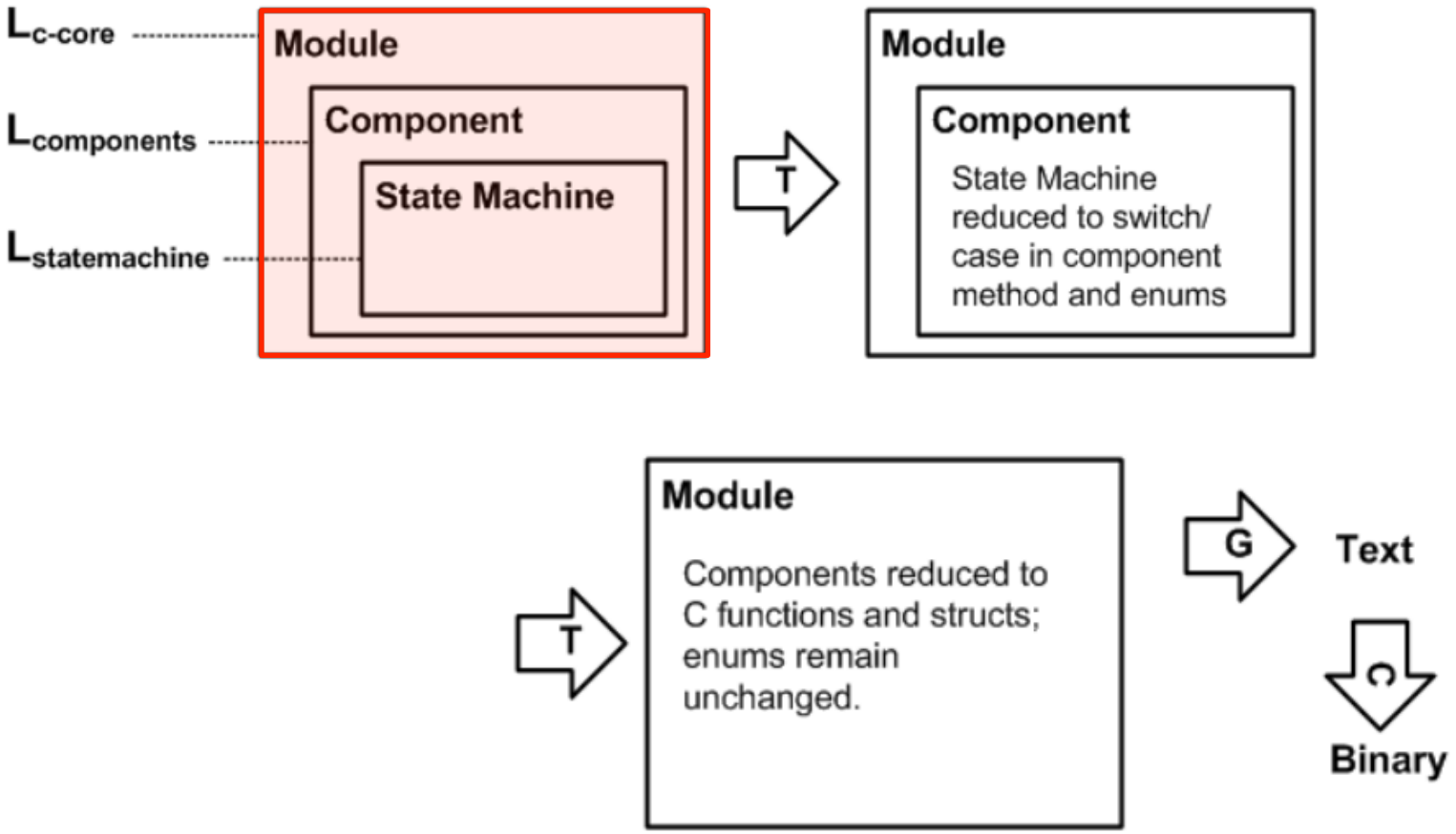
more specialized domains

more specialized languages

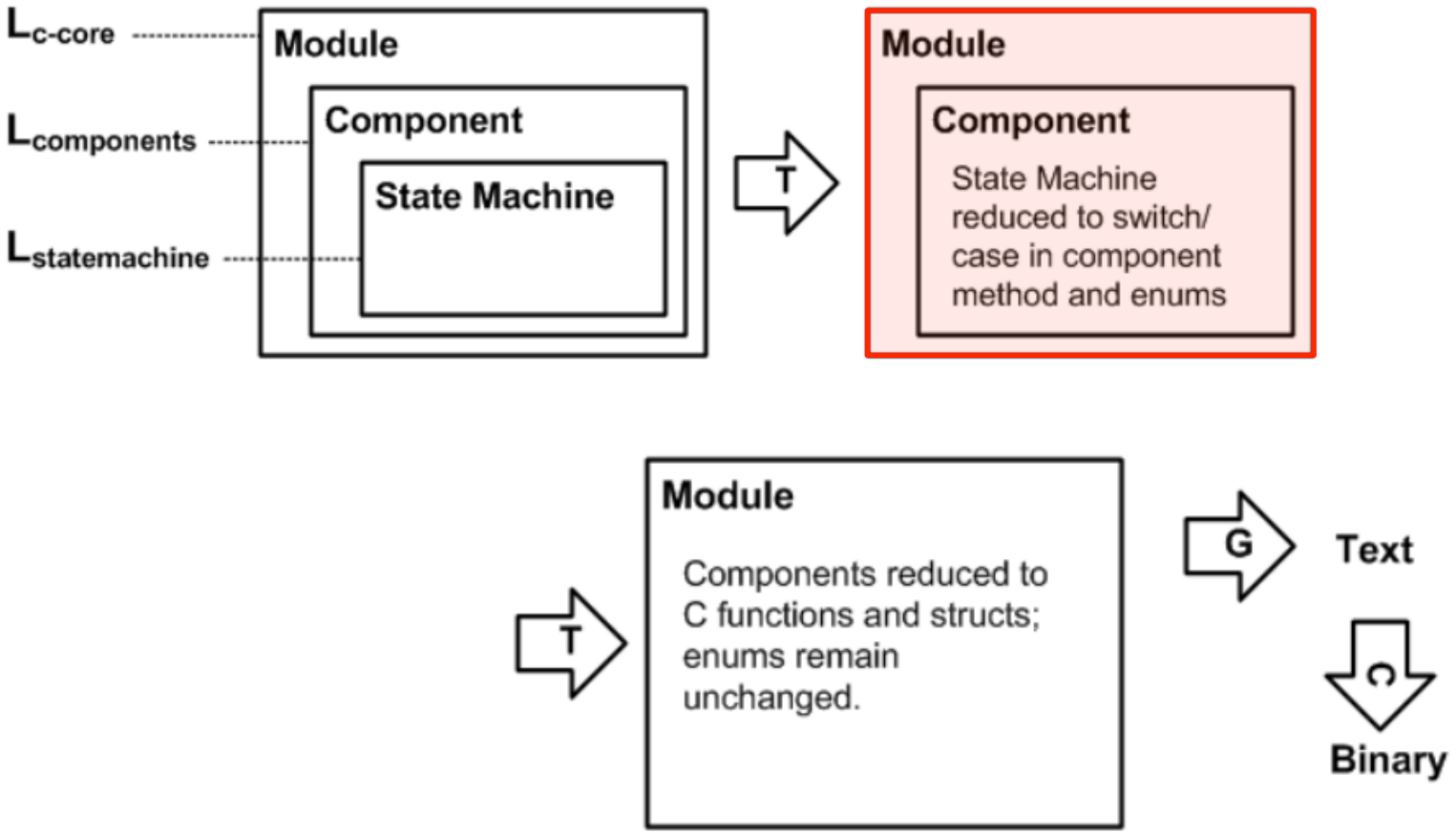
Incremental Trafo



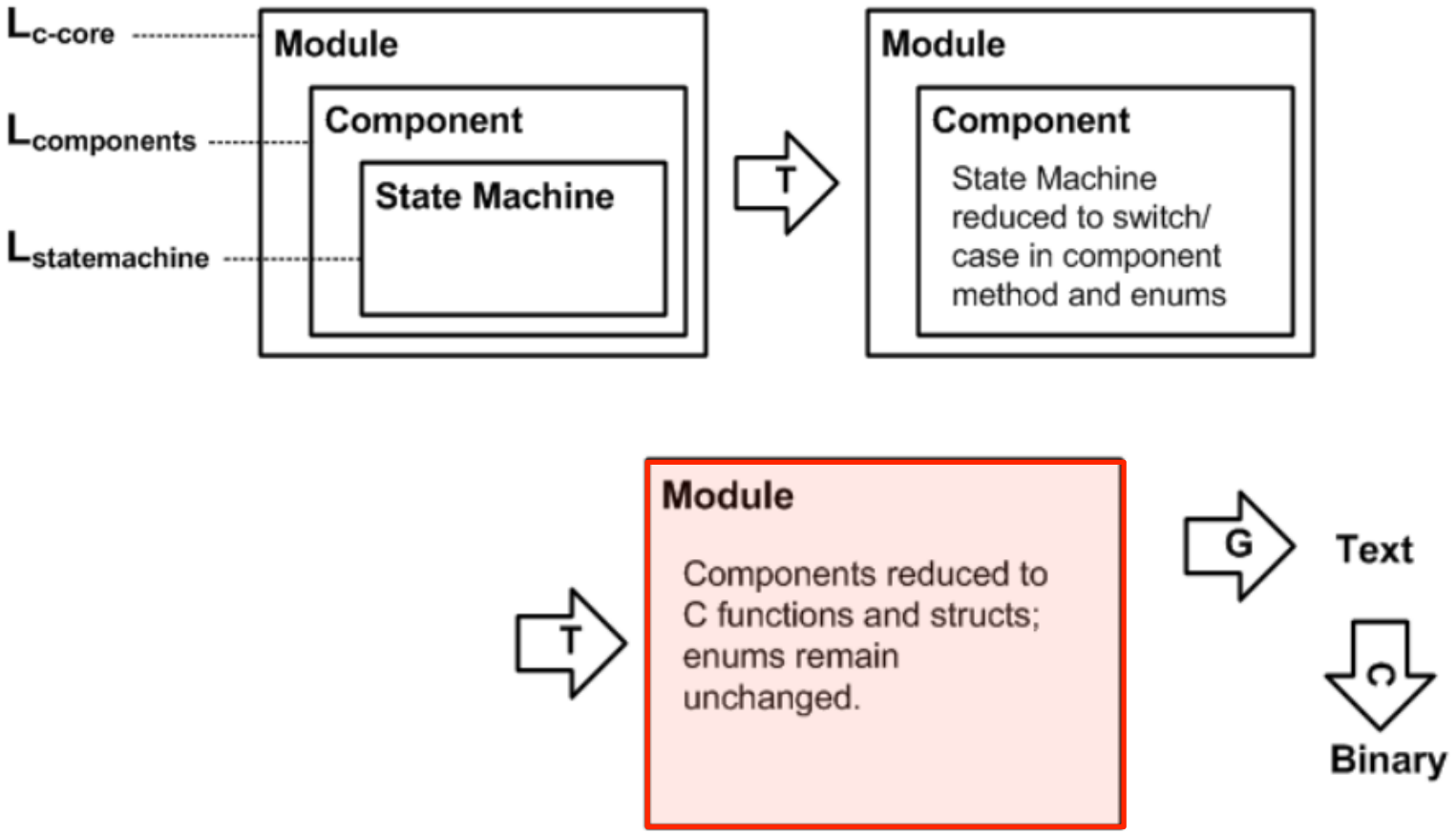
Incremental Trafo



Incremental Trafo

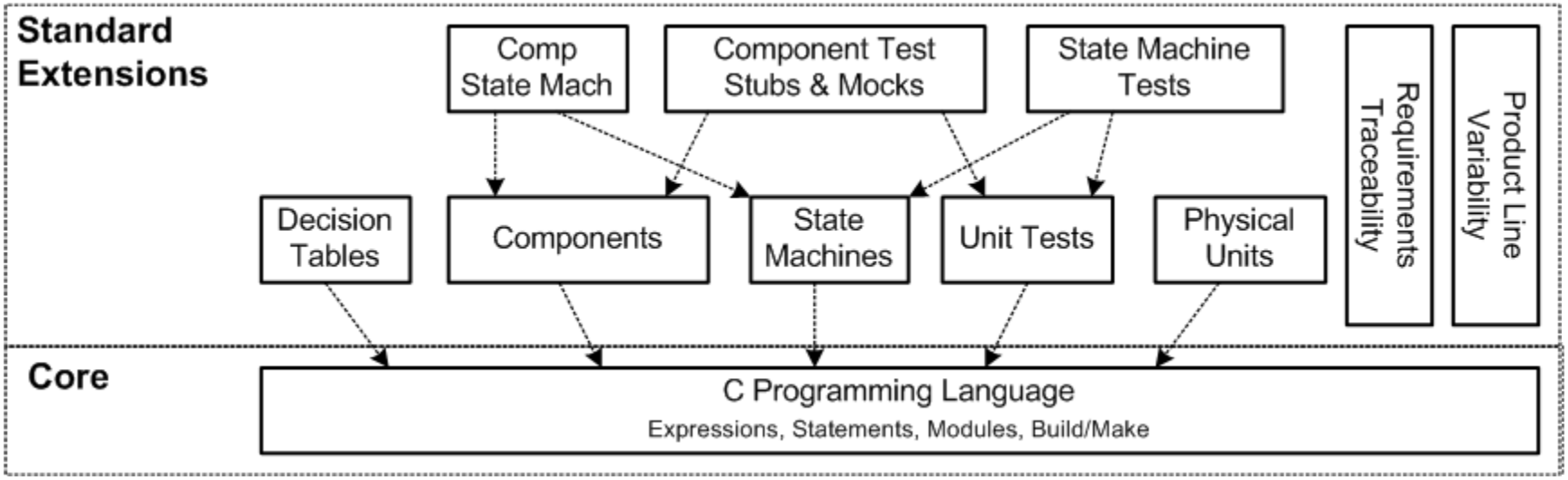


Incremental Trafo

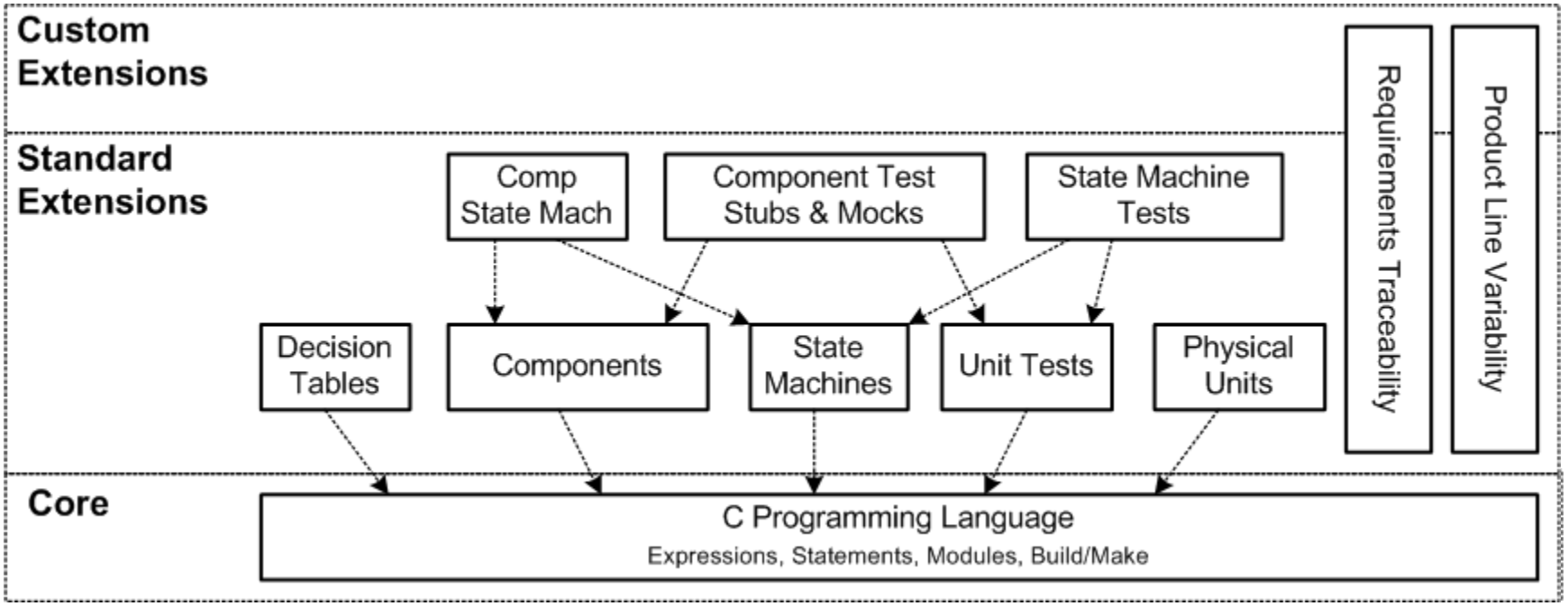


Language Extension

Language Extension



Language Extension



Subset of Available Extensions

All of C (cleaned-up)

```
module Calculator from cdesignpaper.helloWorld imports nothing {

  exported int8_t add(int8_t x, int8_t y) {
    return x + y;
  } add (function)

  exported int8_t multiply(int8_t x, int8_t y) {
    return x * y;
  } multiply (function)
}
```

```
module HelloWorld from cdesignpaper.helloWorld imports Calculator {

  int32_t main(int32_t argc, int8_t*[ ] argv) {
    return add(2, 2) + multiply(10, 2);
  } main (function)
}
```

Retargettable Build Integration

```
Build Configuration for model MutiBot_Test
------------------------------------------------

Target Platform:
  desktop
    compiler: gcc
    compiler options: -std=c99
    debug options: -g

Configuration Items
  reporting: printf
  components: no middleware

Binaries
  executable MultiBotTest isTest: true {
    used libraries
      << ... >>
    included modules
      Driver
      TestDriveTrain
      EcRobotAPI
      Messages
      TestOrienter
      DriveTrain
      Orienter
  }
```

```
Target Platform:
  lego
    oil file: ATMEL_AT91SAM7S256
    path to ecrobot.mak: /opt/lego/nxtOSEK/ecrobot/
```

Native Support for Unit Testing and Logging

```
module UnitTestDemo from cdesignpaper.unittest imports nothing {

  int32_t main(int32_t argc, int8_t*[ ] argv) {
    return test testMultiply;
  } main (function)

  exported test case testMultiply {
    assert(0) times2(21) == 42;
    if ( 1 > 2 ) {
      fail(1);
    } if

  } testMultiply(test case)

  int8_t times2(int8_t a) {
    return 2 * a;
  } times2 (function)
}
```

```
module ARealHelloWorld from cdesignpaper.helloWorld imports nothing {

  message list HelloWorldMessages {
    INFO hello(string name) active: Hello World
    ERROR wrongNumberOfArguments(int8_t expected, int8_t actual) active: Wrong number of Arguments
  }

  int32_t main(int32_t argc, int8_t*[ ] argv) {
    report(0) HelloWorldMessages.wrongNumberOfArguments(1, argc) {
      if ( argc != 1 ) {
        report;
        return 1;
      } if
    };
    report(0) HelloWorldMessages.hello(argv[0]) on/if;
    return 0;
  } main (function)
}
```

```
message list HelloWorldMessages {
  INFO hello(string name) active: Hello World
  ERROR wrongNumberOfArguments(int8_t expected, int8_t actual) inactive: Wrong number of Arguments
}
```

Physical Units

Unit Declarations

```
derived unit N = kg m s^-2 for force

derived unit Pa = N m^-2 for pressure

derived unit N2 = kg m s^-2 for anotherForce

derived unit v = m s^-1 for velocity

derived unit a = m s^-2 for accelearation

convertible unit F for temperature

convertible unit C for temperature
```

Conversion Rules

```
conversion F → C = val * 9 / 5 + 32
conversion C → F = (val - 32) * 5 / 9
```

```
void testBasicUnits() {
  int8_t/N/ n = 3 N;
  int8_t/N/ n2 = 3 N2;
  int8_t/N/ n3 = 3 kg m s^-2 ;
  int8_t/N/ n4 = 3 N * 4s / 3s;
  int8_t/N m/ n5 = n4 * 3 m;

  int8_t/cd/ aLuminousIntensity = 0cd;
  int8_t/m/ length;
  int8_t/s/ time;
  int8_t/m s^-1 / speed = length / time;

  int8_t/v/ thisShouldNotWork = length + time;

  length l1 = length + length;
  length l2 = length * 33;
  length l3 = length / 3;
  length l4 = length + 3;

  if ( 10kg > 20 ) {  } if
  if ( 10kg > 20kg ) {  } if
  if ( 10kg > 20 N ) {  } if

} testBasicUnits (function)
```

```
void fahrenheit() {
  int8_t/C/ temp1 = 10 C;
  int8_t/C/ temp2 = 20 F;
  int8_t/C/ temp3 = [20 F → C];
  int8_t/C/ temp4 = 20 F + 10 C;
} fahrenheit (function)
```

Components
Interfaces
Contracts
Instances
Mocks & Stubs

```
exported c/s interface Orienter on contract error MultibotMessages.prePostconditionFailed {
  int16_t heading()
    post(0) result >= 0 && result <= 359
  void orientTowards(int16_t heading, uint8_t speed, DIRECTION dir)
    pre(0) heading >= 0 && heading <= 359
}
```

```
exported c/s interface Orienter on contract error MultibotMessages.prePostconditionFailed {
  int16_t heading()
    post(0) result >= 0 && result <= 359
  void orientTowards(int16_t heading, uint8_t speed, DIRECTION dir)
    pre(0) heading >= 0 && heading <= 359
}
```

```
exported component OrienterImpl extends nothing {
  ports:
    provides Orienter orienter
    requires EcRobot_Compass compass
    requires EcRobot_Motor motorLeft
    requires EcRobot_Motor motorRight
  contents:
    field int16_t[5] headingBuffer

    void orienter_orientTowards(int16_t heading, uint8_t speed, DIRECTION dir) <-
        op orienter.orientTowards {
      int16_t currentDir = compass.heading();
      if ( dir == COUNTERCLOCKWISE ) {
        motorLeft.set_speed(-1 * ((int8_t) speed));
        motorRight.set_speed(((int8_t) speed));
        while ( currentDir != heading ) { currentDir = compass.heading(); } while
      } else {
        motorLeft.set_speed(((int8_t) speed));
        motorRight.set_speed(-1 * ((int8_t) speed));
        while ( currentDir != heading ) { currentDir = compass.heading(); } while
      } if
      motorLeft.stop();
      motorRight.stop();
    }

    int16_t orienter_heading() <- op orienter.heading {
      return compass.heading();
    }
}
```

```
exported test case testDriveTrain {
  initialize instances;
  assert(0) dt.currentSpeed() == 0;
  dt.driveContinouslyForward(50);
  dt.stop();
  validate mock motorLeft;
  validate mock motorRight;
} testDriveTrain(test case)
```

```
instance configuration instances extends nothing {
  instances:
  instance MotorLeftMock motorLeft
  instance MotorRightMock motorRight
  instance DriveTrainImpl driveTrain
  instance EcUtil util
  connectors:
  connect driveTrain.motorLeft to motorLeft.motor
  connect driveTrain.motorRight to motorRight.motor
  connect driveTrain.util to util.util
  adapter:
  << ... >>
}
```

```
mock component MotorLeftMock {
  report messages: true
  ports:
    provides EcRobot_Motor motor
  expectations:
    total no. of calls is 2
    sequence {
      0: motor.set_speed {
          0: parameter speed: speed == 50
        }
      1: motor.stop
    }
}
```

```
mock component MotorRightMock {
  report messages: true
  ports:
    provides EcRobot_Motor motor
  expectations:
    total no. of calls is 2
    sequence {
      0: motor.set_speed {
          0: parameter speed: speed == 50
        }
      1: motor.stop
    }
}
```

State Machines

+

Model Checking

```
verifiable
statemachine Counter {
  in start() <no binding>
     step(int[0..10] size) <no binding>
  out someEvent(int[0..100] x, boolean b) => handle_someEvent
      resetted() => resetted
  vars int[0..100] currentVal = 0
       int[0..100] LIMIT = 10
  states (initial = initialState)
    state initialState {
      on start [ ] -> countState { send someEvent(100, true && false || true); }
    }
    state countState {
      on step [currentVal + size > LIMIT] -> initialState { send resetted(); }
      on step [currentVal + size <= LIMIT] -> countState { currentVal = currentVal + size; }
      on start [ ] -> initialState {  }
    }
} end statemachine
```

```
verifiable
statemachine Counter {
  in start() <no binding>
     step(int[0..10] size) <no binding>
  out someEvent(int[0..100] x, boolean b) => handle_someE
      resetted() => resetted
  vars int[0..100] currentVal = 0
       int[0..100] LIMIT = 10
  states (initial = initialState)
    state initialState {
      on start [ ] -> countState { send someEvent(100, tr
    }
    state countState {
      on step [currentVal + size > LIMIT] -> initialState
      on step [currentVal + size <= LIMIT] -> countState
      on start [ ] -> initialState {  }
    }
} end statemachine
```

NuSMV Tool

Property	Status	Trace Size
State 'initialState' can be reached	SUCCESS	
State 'countState' can be reached	SUCCESS	
Variable 'currentVal' is always between its defi...	SUCCESS	
Variable 'LIMIT' is always between its defined ...	SUCCESS	
State 'initialState' has deterministic transitions	SUCCESS	
State 'countState' has deterministic transitions	SUCCESS	
Transition 0 of state 'initialState' is not dead	SUCCESS	
Transition 0 of state 'countState' is not dead	SUCCESS	
Transition 1 of state 'countState' is not dead	SUCCESS	
Transition 2 of state 'countState' is not dead	SUCCESS	
Condition 'currentVal == 8' can be true	FAIL	4

Node	Value
State initialState	
LIMIT	10
currentVal	0
State initialState	
in_event: start	star)
LIMIT	10
currentVal	0
State countState	
in_event: step	step(8)
out_event:someEvent	someEvent(100, true)
LIMIT	10
currentVal	0
State countState	
LIMIT	10
currentVal	8

Code Verification

```
requires n >= 0 && \valid_range( t , 0 , n - 1 )
behavior success
assumes
  \forall integer k1; integer k2; : 0 <= k1 && k1 <= k2 && k2 <= n - 1 ==> t[k1] <= t[k2] ;
assumes
  \exists integer k; : 0 <= k && k <= n - 1 && t[k] == v ;
ensures \result >= -1 && \result <= n
int16_t binarySearch(int16_t* t, int16_t n, int16_t v) {
  int16_t l = 0;
  int16_t u = n - 1;

  loop invariant 0 <= l && u <= n - 1
  loop variant u - l
  while ( l <= u ) {
    int16_t m = (l + u - 1) / 2;
    if ( t[m] < v ) {
      l = m + 1;
    } else if ( t[m] > v ) { u = m - 1; } else {
      return m;
    }
  } while

  return -1;
} binarySearch (function)
```

Proof obligations	Alt-Ergo 0.92.2	Simplify 1.5.7	Yices 1.0.29 (SS)	Statistics
Function Algorithms_binarySearch Default behavior	✓			4/4
1. loop invariant initially holds	●	–	–	
2. loop invariant initially holds	●	–	–	
3. loop invariant preserved	●	–	–	
4. loop invariant preserved	●	–	–	
Function Algorithms_binarySearch Normal behavior `success`	✗			5/6
1. postcondition	●	–	–	
2. postcondition	●	–	–	
3. postcondition	●	–	–	
4. postcondition	●	–	–	
5. postcondition	●	–	–	
6. postcondition	?	–	–	

Requirements Tracability

```
requirements HighLevelRequirements
  show traces true

functional Main: Program has to run from the command line ...
  functional Arg2: Argument Count must be 2 ...
    functional FailOtherwise: Otherwise it should return -1 ...
functional Add: The program should return the sum of the two arguments ...
  functional AddFct: Adding should be a separate function for reuse ...
```

```
requirements HighLevelRequirements
  show traces true

functional Main: Program has to run from the command line ...
  functional Arg2: Argument Count must be 2 ...
    functional FailOtherwise: Otherwise it should return -1 ...
functional Add: The program should return the sum of the two arguments ...
  functional AddFct: Adding should be a separate function for reuse ...
```

```
requirements modules: HighLevelRequirements
module ExampleCode from test.ts.requirements.code imports StrUtil {

  int8_t add(int8_t a, int8_t b) {  trace AddFct
    return a + b;
  } add (function)

  int8_t main(string[ ] args, int8_t argc) {                                  trace Main
    if ( argc == 2 ) {                                                  trace Arg2
      return add(str2int(args[0]), str2int(args[1]))  trace Add;
    } else {
      return -1;  trace FailOtherwise
    } if
  } main (function)
}
```

```
requirements HighLevelRequirements
  show traces false

functional Main: Program has to run from the command line ...
  functional Arg2: Argument Count must be 2 ...
    functional FailOtherwise: Otherwise it should return -1 ...
functional Add: The program should return the sum of the two arguments ...
  functional AddFct: Adding should be a separate function for reuse ...
```

```
requirements modules: HighLevelRequirements
module ExampleCode from test.ts.requirements.code imports StrUtil {

  int8_t add(int8_t a, int8_t b) {
    return a + b;
  } add (function)

  int8_t main(string[ ] args, int8_t argc) {
    if ( argc == 2 ) {
      return add(str2int(args[0]), str2int(args[1]));
    } else {
      return -1;
    } if
  } main (function)
}
```

Product Line Variability

```
feature model DeploymentConfiguration
  root ? {
    logging
    test
    valueTest [int8_t value]
  }

configuration model Debug configures DeploymentConfiguration
  root {
    logging
    test
    valueTest [value = 42]
  }

configuration model Production configures DeploymentConfiguration
  root {
    << ... >>
  }
```

```
Variability from FM: DeploymentConfiguration
Rendering Mode: product line
```

```
module ApplicationModule from test.ex.cc.fm imports {!test} SensorModule {

  {logging}
  message list messages {
    INFO beginningMain() active: entering main function
    INFO exitingMain() active: exitingMainFunction
  }

  exported test case testVar {
    {logging}
    report(0) messages.beginningMain() on/if;
    int8_t x = getSensorValue(1) replace if {test} with 42;
    {logging}
    report(1) messages.exitingMain() on/if;
    assert(2) x == 10 replace if {test} with 42;
    {valueTest}
    int8_t vv = value;
    {valueTest}
    assert(3) vv == 42;
    int8_t ww = 22 replace if {valueTest} with 12 + value;
    {!valueTest}
    assert(4) ww == 22;
    {valueTest}
    assert(5) ww == 54;
  } testVar(test case)

  int32_t main(int32_t argc, string[ ] args) {
    return test testVar;
  } main (function)
}
```

```
feature model DeploymentConfiguration
  root ? {
    logging
    test
    valueTest [int8_t value]
  }

configuration model Debug configures DeploymentConfiguration
  root {
    logging
    test
    valueTest [value = 42]
  }

configuration model Production configures DeploymentConfiguration
  root {
    << ... >>
  }
```

```
Variability from FM: DeploymentConfiguration
Rendering Mode: variant rendering config: Debug
module ApplicationModule from test.ex.cc.fm imports {

  message list messages {
    INFO beginningMain() active: entering main function
    INFO exitingMain() active: exitingMainFunction
  }

  exported test case testVar {
    report(0) messages.beginningMain() on/if;
    int8_t x = 42;
    report(1) messages.exitingMain() on/if;
    assert(2) x == 42;
    int8_t vv = value (variant Debug);
    assert(3) vv == 42;
    int8_t ww = 12 + value (variant Debug);
    assert(5) ww == 54;
  } testVar(test case)

  int32_t main(int32_t argc, string[ ] args) {
    return test testVar;
  } main (function)
}
```

```
feature model DeploymentConfiguration
  root ? {
    logging
    test
    valueTest [int8_t value]
  }

configuration model Debug configures DeploymentConfiguration
  root {
    logging
    test
    valueTest [value = 42]
  }

configuration model Production configures DeploymentConfiguration
  root {
    << ... >>
  }
```

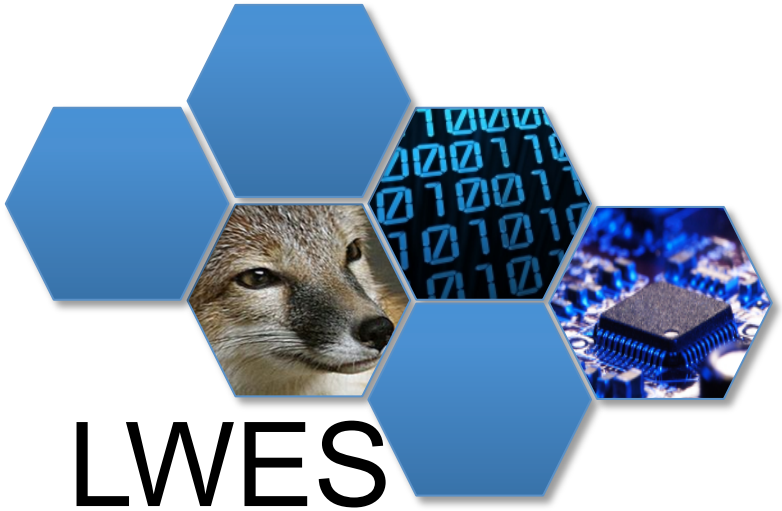
```
Variability from FM: DeploymentConfiguration
Rendering Mode: variant rendering config: Production
module ApplicationModule from test.ex.cc.fm imports SensorModule {


  exported test case testVar {
    int8_t x = getSensorValue(1);
    assert(2) x == 10;
    int8_t ww = 22;
    assert(4) ww == 22;
  } testVar(test case)

  int32_t main(int32_t argc, string[ ] args) {
    return test testVar;
  } main (function)
}
```

```
feature model DeploymentConfiguration
  root ? {
    logging
    test
    valueTest [int8_t value]
  }

configuration model Debug configures DeploymentConfiguration
  root {
    logging
    test
    valueTest [value = 42]
  }

configuration model Production configures DeploymentConfiguration
  root {
    << ... >>
  }
```

Status and Availability

http://mbeddr.com

Developed within



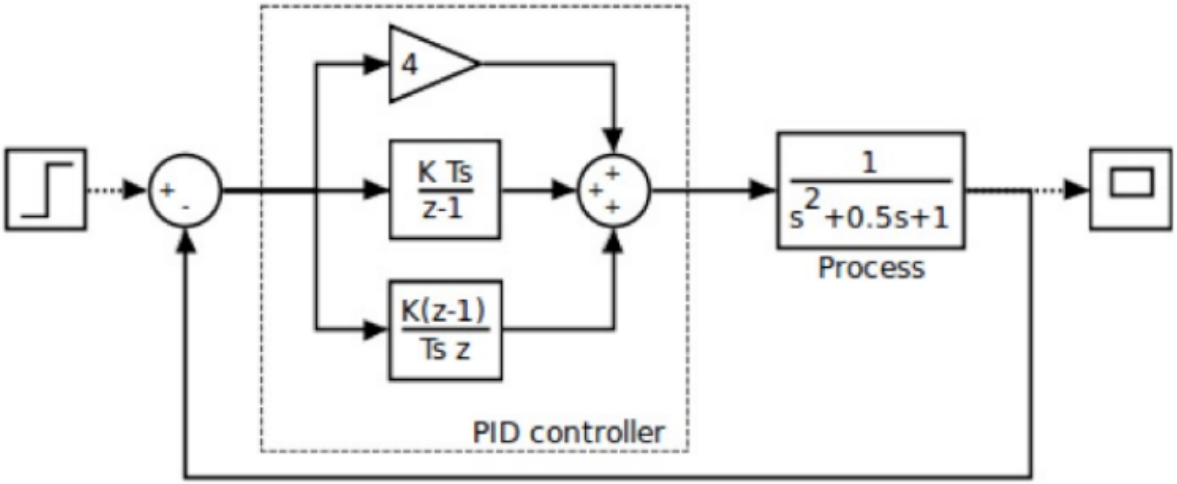
LWES

Language Workbenches
for Embedded Systems

Bundesministerium
für Bildung
und Forschung

gefördert durch das BMBF
Förderkennzeichen 01IS11014

Open Source (EPL)



support for graphical early 2013

integration in early 2013

mbeddr
C the Difference. C the Future.

An extensible version of the C programming language for Embedded Programming

http://mbeddr.com

C the Difference - C the Future

Bundesministerium
für Bildung
und Forschung

gefördert durch das BMBF
Förderkennzeichen 01IS11014

SICK Sensor Intelligence.
fortiss innovation in software and systems
itemis