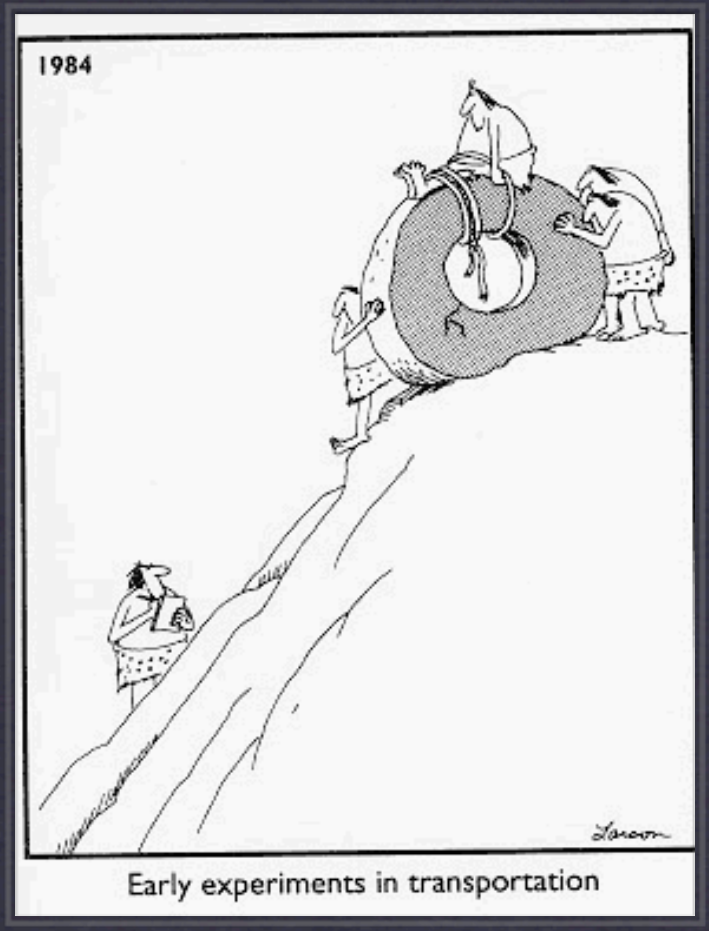
Detailed Design and Verification with JML

Curt Clifton
Rose-Hulman Institute of Technology

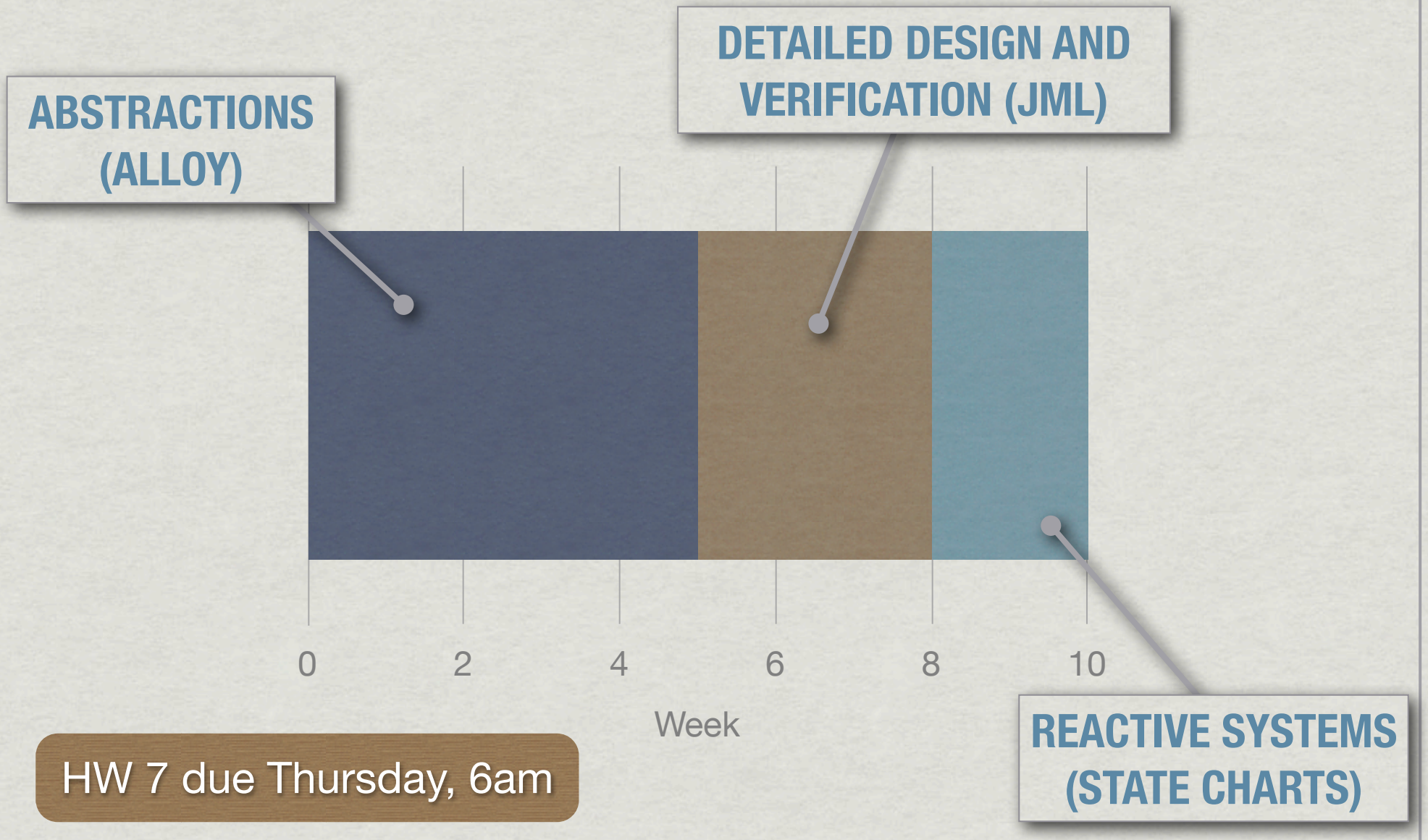
And now for something completely different

And now for something completely different



Early experiments in transportation

Course Topics



HW 7 due Thursday, 6am

Formal Verification

- Formal reasoning about actual code
- Application:
  - Design-by-contract
  - Weakest pre-condition calculation
  - Proving program correctness

The Plan

- First, learn basic reasoning techniques
  - Working by hand
  - Using same notation that we'll automate later
- Then, experiment with prototype tools for automated reasoning

JML: Notation and Tools

What's JML?

- JML stands for "*Java Modeling Language*"
- A Behavioral Interface Specification Language
  - A "BISL" specifies the signatures of functions, methods, and classes—***the interface***
  - Conditions that must hold—***the behavior***
- JML extends Java with special *annotations* for behavioral specifications

JML Annotations

- Annotations in JML are just Java comments
- JML tools recognize comments using '@' symbol
  - `//@` starts a single line annotation
    - `//@ requires x > 0;`
  - `/*@ … */` defines a multi-line comment
- Common mistake—adding a space: ~~`// @`~~

NOT VALID!

BEGIN Q3

Example

PRECONDITION: WHAT CALLER MUST GUARANTEE

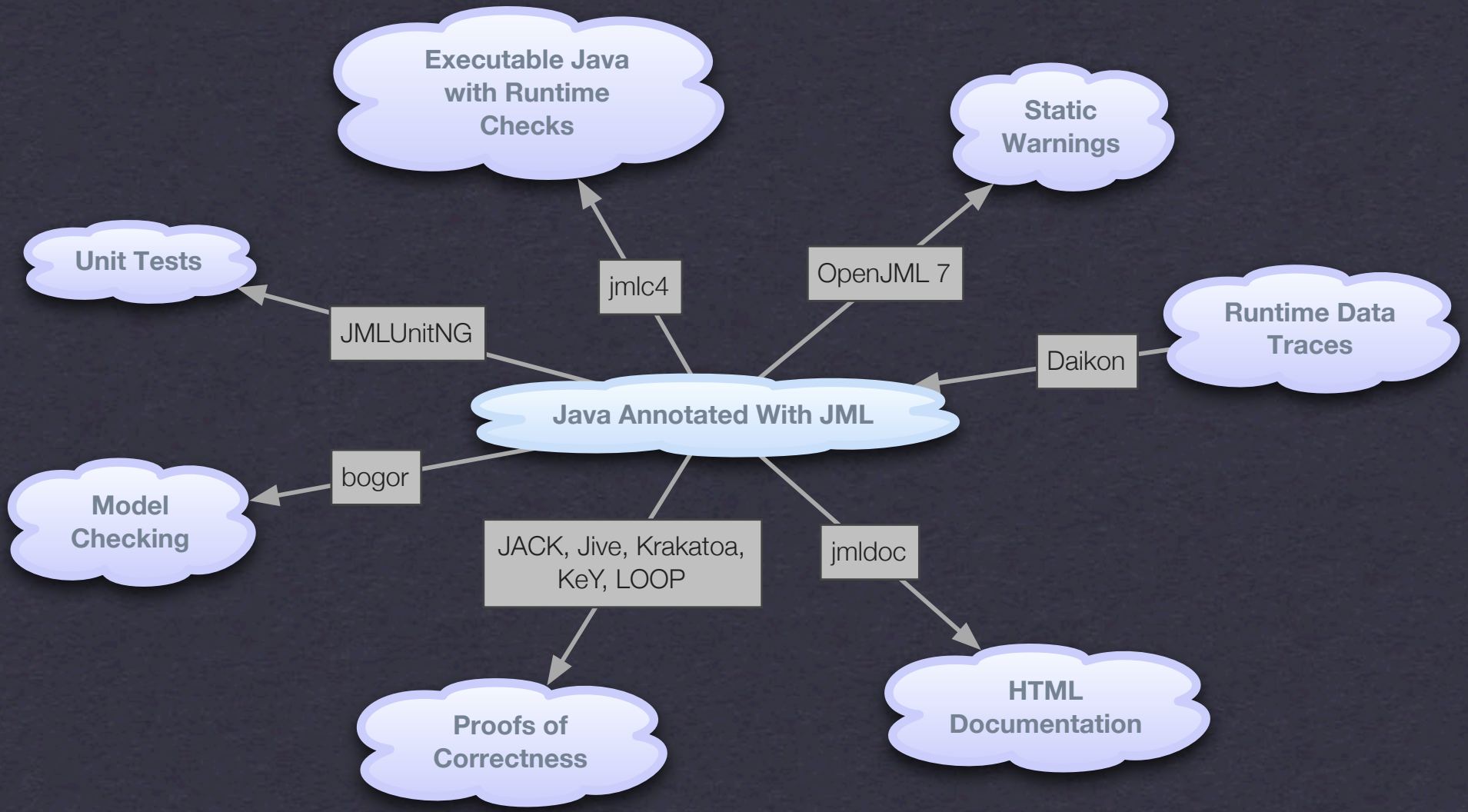
```
/*@ requires x > 0;
  @ ensures \result * \result <= x &&
  @                 x < (\result + 1) * (\result + 1);
  @*/
public static int iroot(int x) {
  ...
}
```

POSTCONDITION: WHAT METHOD MUST ENSURE

FINISH Q3

One use for JML: Design by Contract

- A software development methodology
- Engineer specifies a contract for each method:
  - A precondition and
  - A postcondition
- Programmer implements the methods



WHAT'S THE USE?

THE FAMILY OF JML TOOLS

Techniques for Formal Verification, or …

How Do You Eat an Elephant?

ONE BITE AT A TIME

First Bites

- Assignment
- Sequencing

Detailed Notation

- Low-level verification: use JML's **assert**
- Pre-condition: before a statement or block
- Post-condition: after a statement or block

```
…
    //@ assert n == 0;
i = 0;
    //@ assert n == i;
…
```

WHY NOT JUST USE JAVA'S ASSERT STATEMENT?

Proving Program Properties

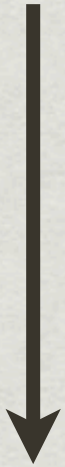
PARTIAL CORRECTNESS

- Specify the program with pre- and post-conditions
- Use “inference rules” to annotate the program
  - proving that from the pre-condition we can reach the post-condition
- Show that loops and recursive functions terminate

TOTAL CORRECTNESS

Assignment Rule

WHATEVER IS TRUE ABOUT *e* BEFORE IS TRUE ABOUT *v* AFTER



```
    //@ assert P(e);
v = e;
    //@ assert P(v);
```

WHAT MUST BE TRUE ABOUT *e* BEFORE IF WE WANT SOME PROPERTY TO BE TRUE ABOUT *v* AFTER?

The Weakest Precondition is…

THE LEAST RESTRICTIVE PRE-CONDITION SUCH THAT THE POST-CONDITION MUST HOLD.

Examples…

```
    //@ assert P(e);
v = e;
    //@ assert P(v);
```

Examples

```
2:     //@ assert 11 == y;
1:     //@ assert 14 - 3 == y;
       x = 3;
           //@ assert 14 - x == y;
```

```
2:     //@ assert 100 <= n * 3 && n * 3 < p;
1: n' = n * 3; // tick trick
       //@ assert 100 <= n' && n' < p;
```

Composition Rule

IF:

//@ assert P_1;
S_1;
//@ assert Q_1;

AND

//@ assert P_2;
S_2;
//@ assert Q_2;

AND

$Q_1 \Longrightarrow P_2$

THEN:

//@ assert P_1;
S_1;
//@ assert Q_1;
//@ assert P_2;
S_2;
//@ assert Q_2;

IMPORTANT: IMPLICATIONS GO DOWN

Composition Example

```
3:     //@ assert 3 * (x + y) > 10;
2: x' = x + y; // tick trick
1:     //@ assert 3 * x' > 10;
   y = 3 * x;
       //@ assert y > 10;
```