Component-Based Software Development

Component-Based Software Development

How to Reason Formally about Your Code

Components

Specification

Reasoning

The Future

Components

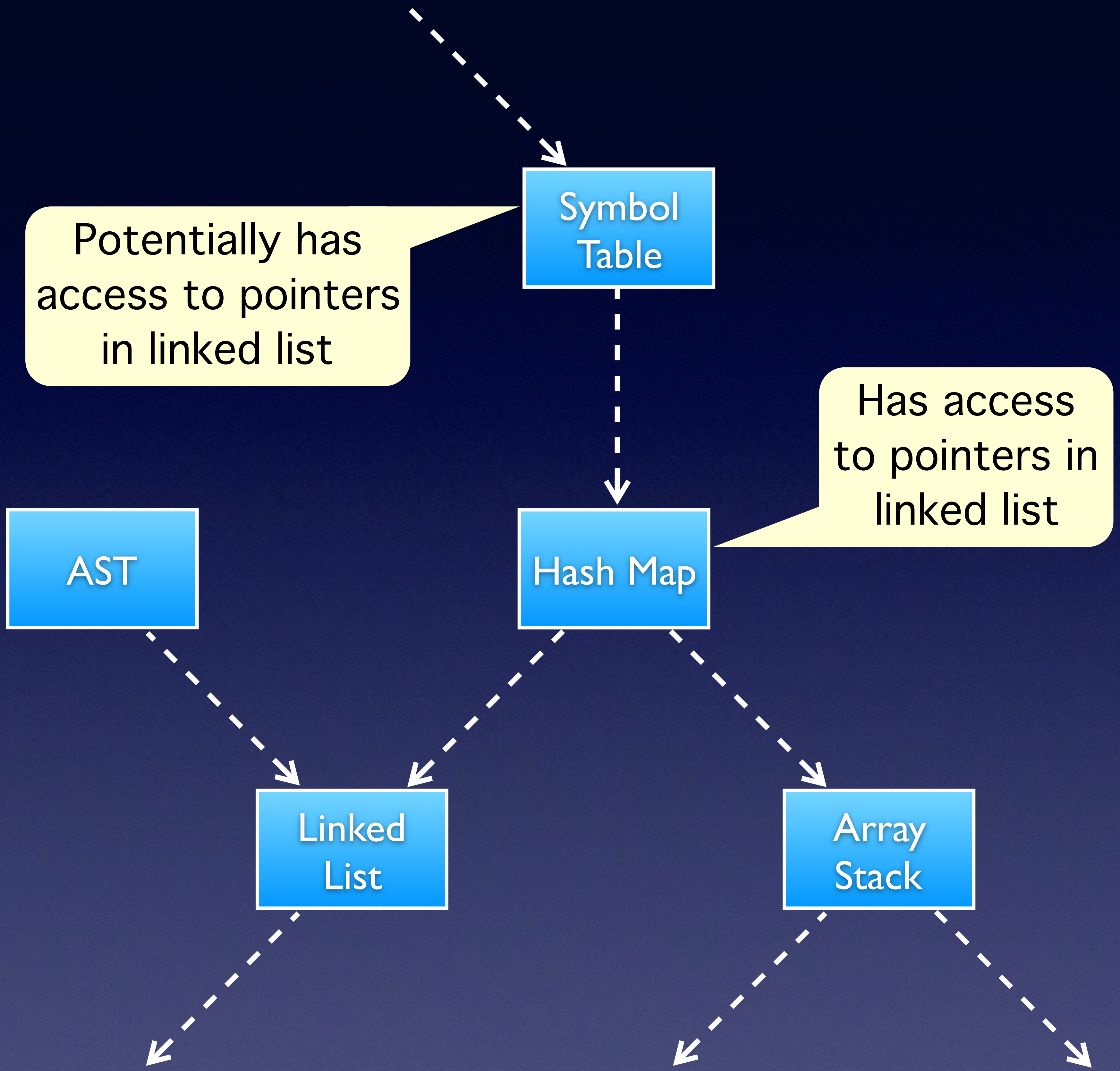

KILLER
TECH
2012
LEADER

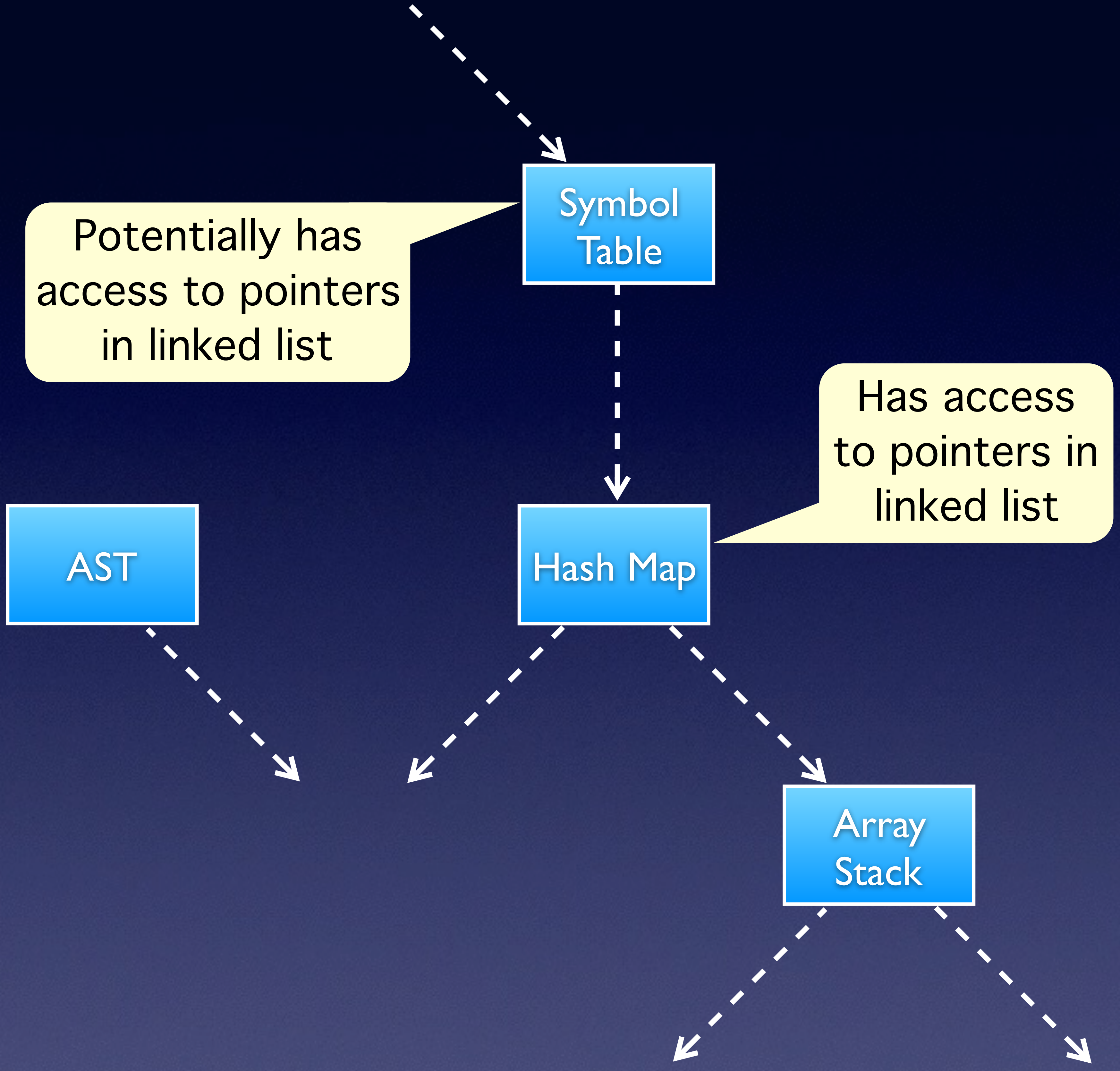
Characteristics

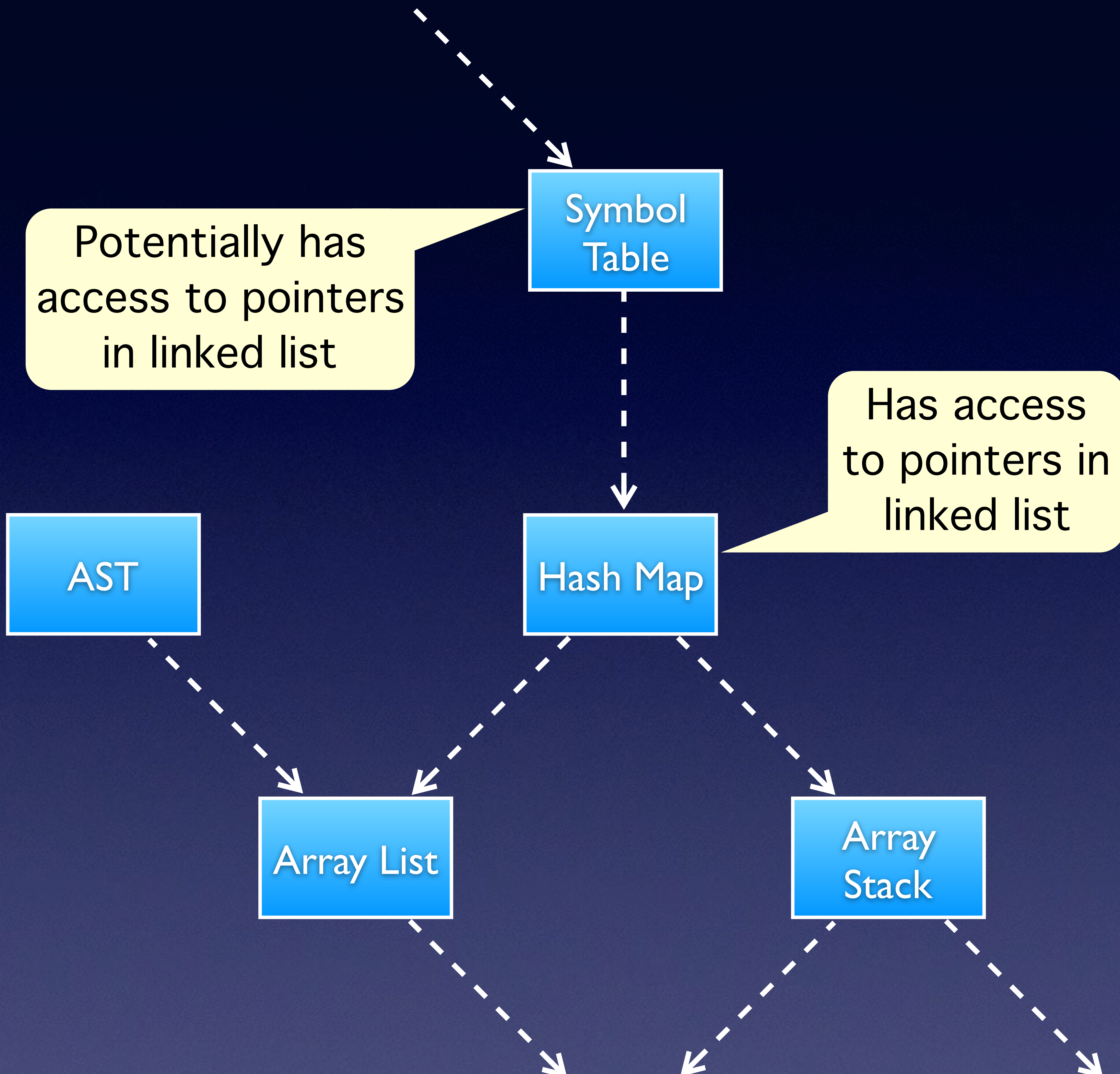
- Components communicate with each other via interfaces
- They are cohesive
- They are substitutable
- They are reusable

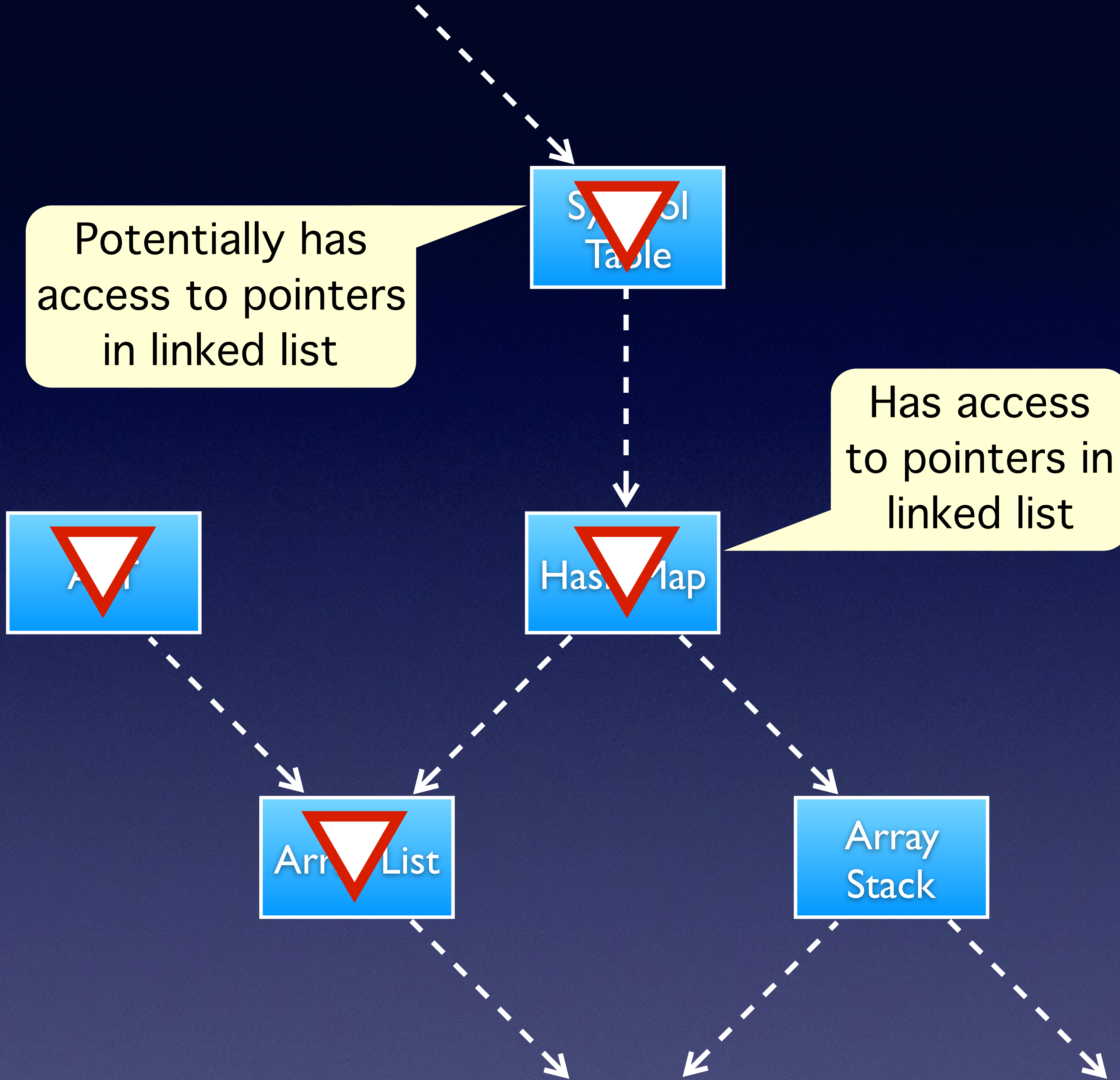
First Principle of Component Design

A component should be usable solely on the basis of its specification

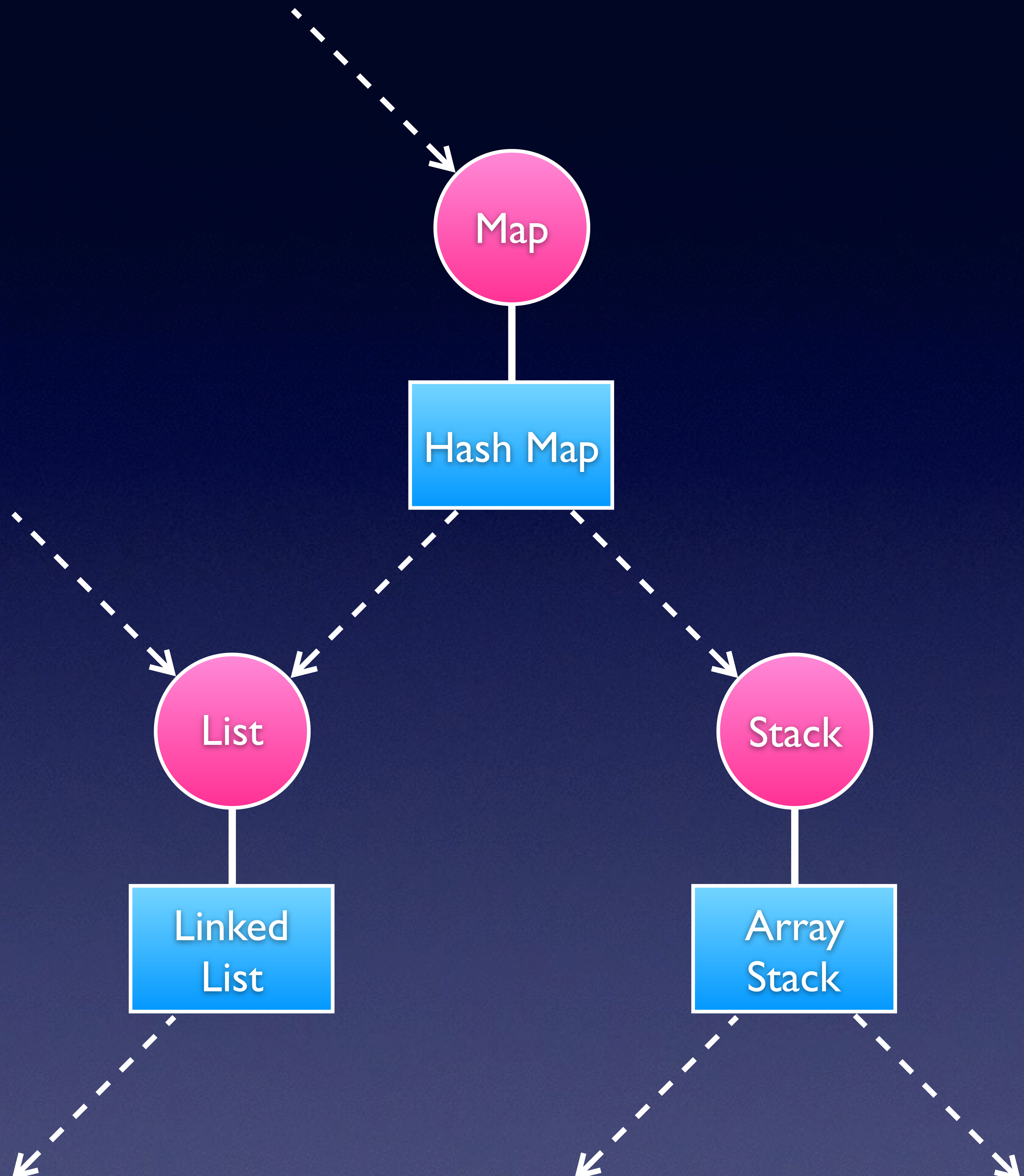

Symbol Table
Potentially has access to pointers in linked list
Has access to pointers in linked list
AST
Hash Map
Linked List
Array Stack

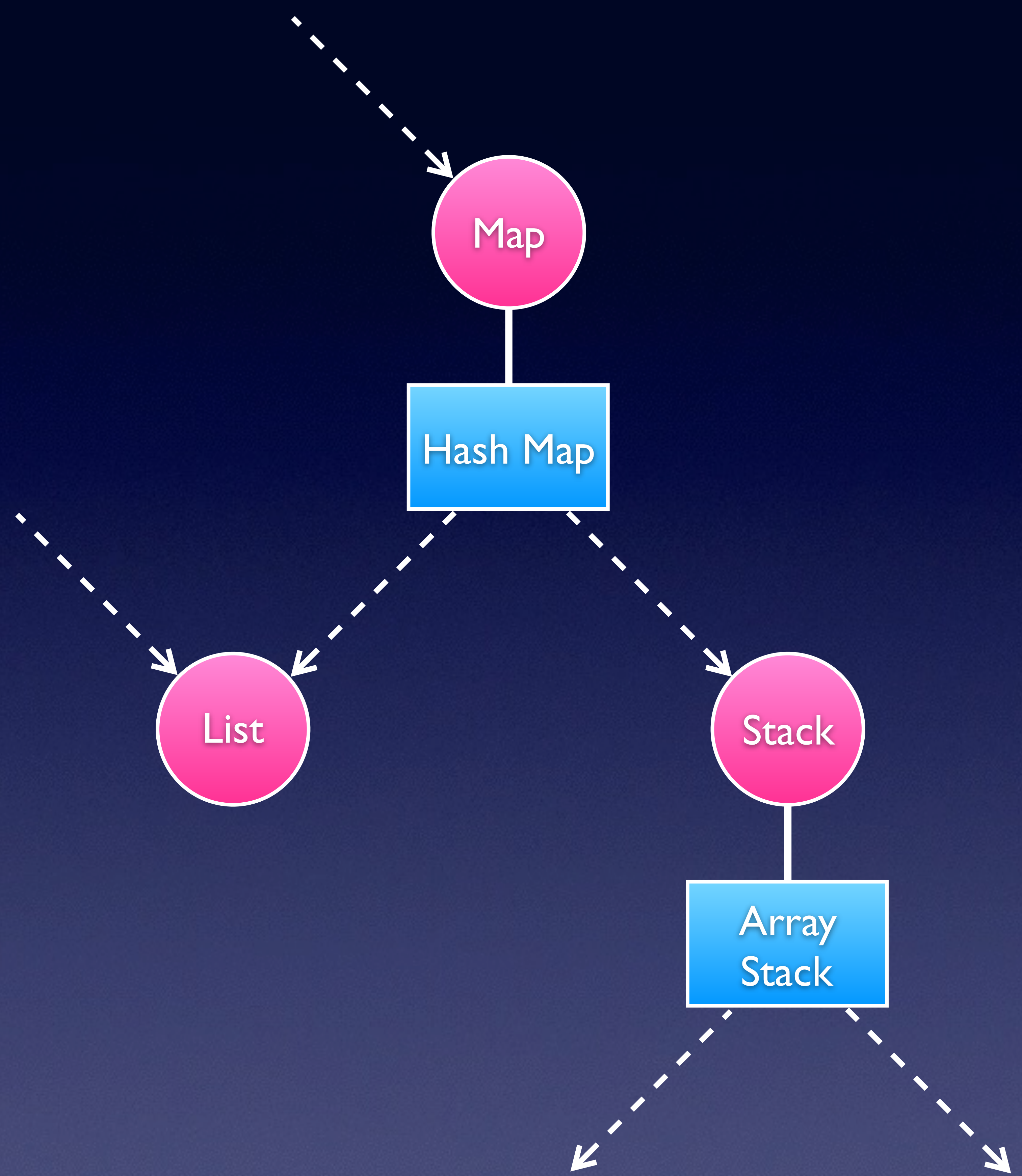

Symbol Table
Potentially has access to pointers in linked list
AST
Hash Map
Has access to pointers in linked list
Array Stack

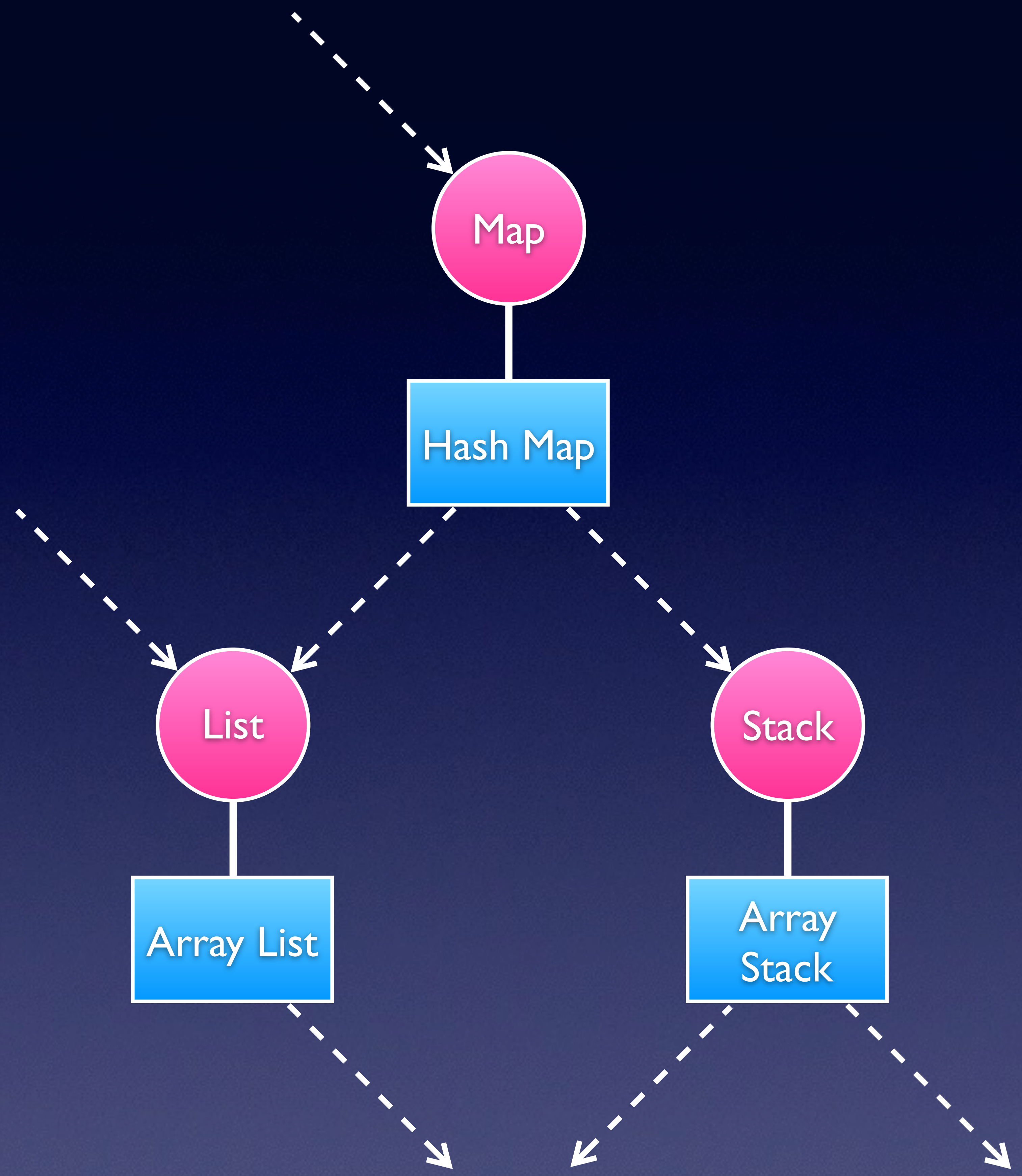

Symbol Table
Potentially has access to pointers in linked list
Has access to pointers in linked list
AST
Hash Map
Array List
Array Stack

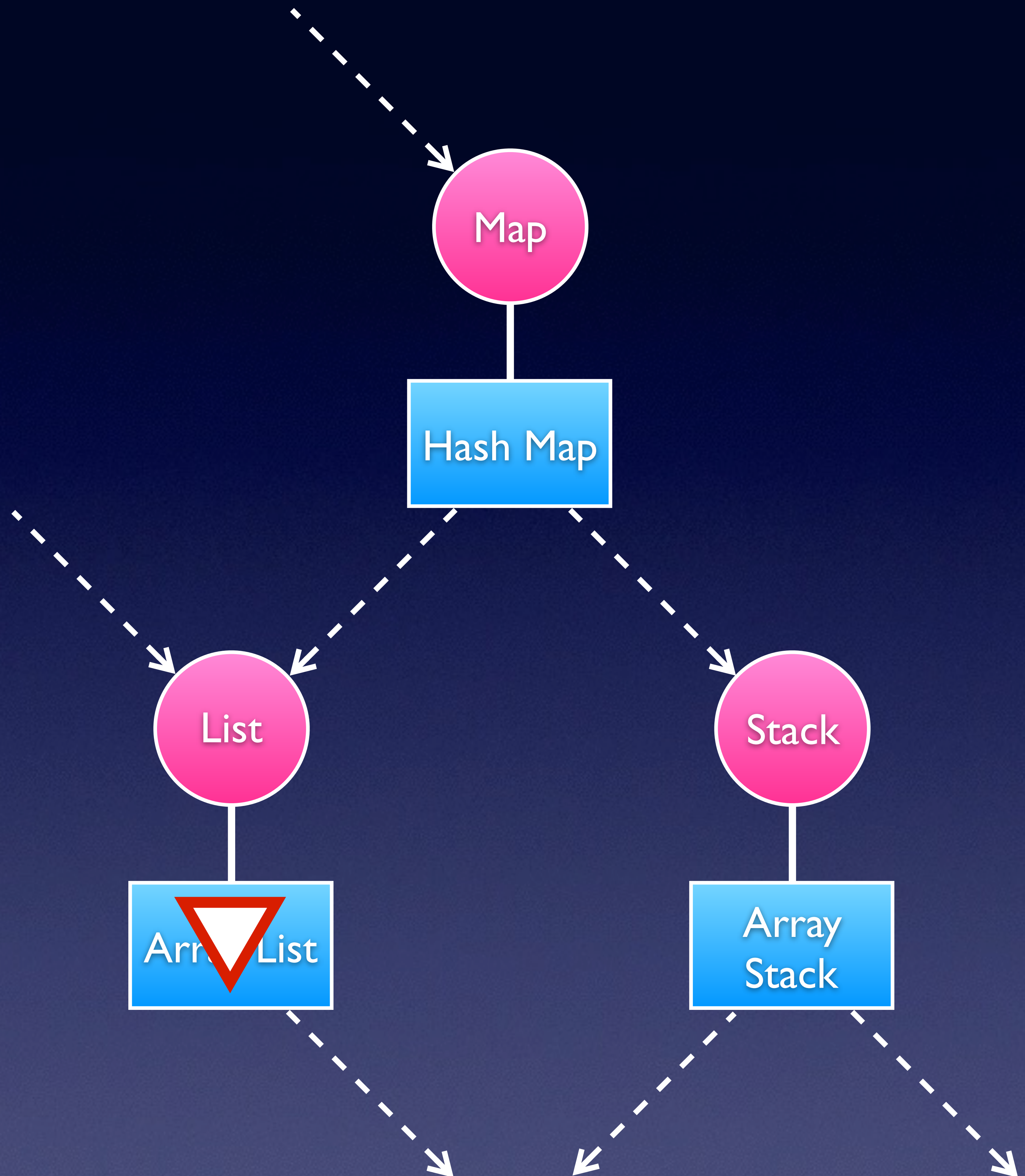


Potentially has access to pointers in linked list
Has access to pointers in linked list
Array Stack


Map
Hash Map
List
Stack
Linked List
Array Stack


Map
Hash Map
List
Stack
Array Stack


Map
Hash Map
List
Stack
Array List
Array
Stack


Map
Hash Map
List
Stack
Arr List
Array Stack

✓ Components

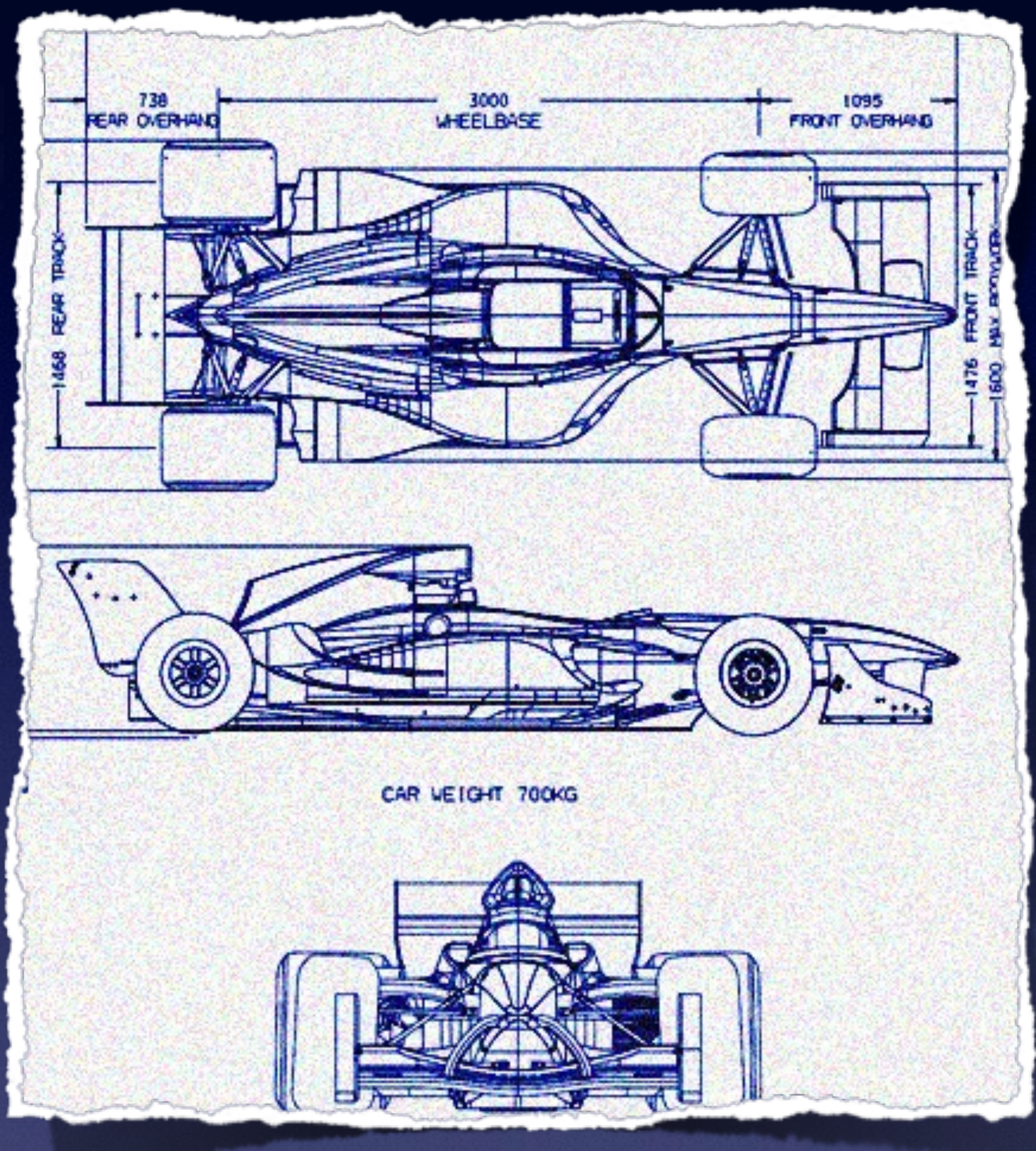
Specification

Reasoning

The Future

Specification

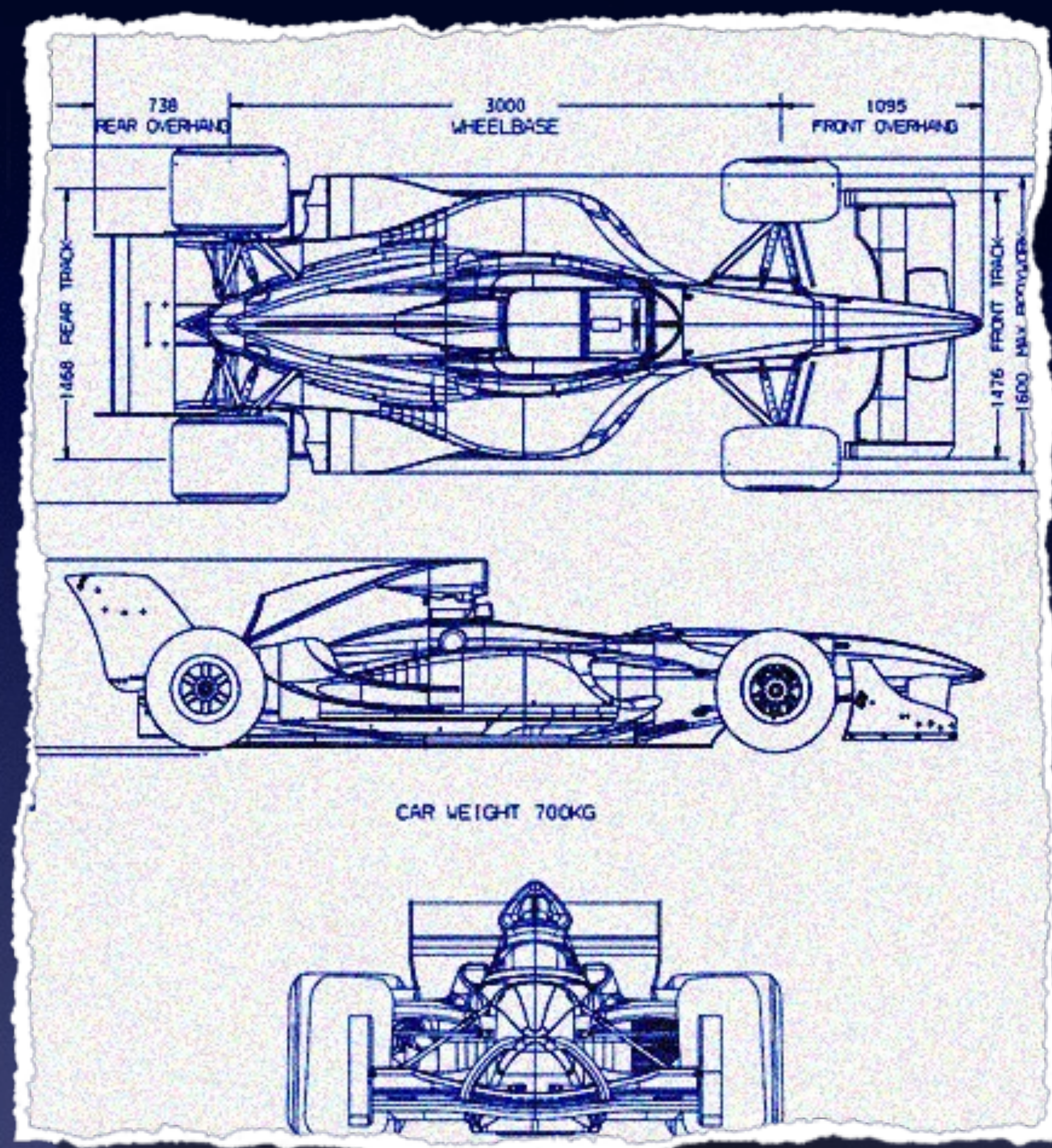
specification



implementation

specification

What it does



implementation

How it does it

Informal Specification

Describe what a component does using natural language, pictures, or real-world metaphors

```java
public interface Stack<E> {

    public Stack();

    public void push(E element);

    public E pop();

    public Integer depth();
}
```

```java
// The Stack class describes a LIFO stack of objects
public interface Stack<E> {

    // Creates an empty stack
    public Stack();

    // Pushes the specified element
    // onto the top of the stack
    public void push(E element);

    // Removes top element from the stack
    // and returns it
    public E pop();

    // Returns the number of items in this stack
    public Integer depth();

}
```

Formal Specification

Formal Specification

Describe what a component does using a mathematical specification language

Formal Specification

Describe what a component does using a mathematical specification language

Z, Resolve, JML, Spec#

```java
public interface Stack<E> {

    public Stack();

    public void push(E element);


    public E pop();



    public Integer depth();

}
```

```
public interface Stack<E> {
    model MathSequence<E>;

    public Stack();
        ensures this = [];

    public void push(E element);
        ensures this = [#element] + #this and
                element = ??;

    public E pop();
        requires this ≠ [];
        ensures this = ALL_BUT_FIRST(#this) and
                result = FIRST(#this);

    public Integer depth();
       ensures result = |#this| and this = #this;
}
```

```java
// Transfers the top element in s to the top of t
public static void transferTop(Stack<E> s;
                               Stack<E> t) {

}
```

Input	Expected output
s = [5, 6, 8] t = [4, 3]	

```
// Transfers the top element in s to the top of t
public static void transferTop(Stack<E> s;
                               Stack<E> t) {

}
```

Input	Expected output
s = [5, 6, 8] t = [4, 3]	s = [6, 8] t = [5, 4, 3]

```
// Transfers the top element in s to the top of t
public static void transferTop(Stack<E> s;
                               Stack<E> t) {
    E temp := s.pop();
    t.push(temp);
}
```

Input	Expected output
s = [5, 6, 8] t = [4, 3]	s = [6, 8] t = [5, 4, 3]

State	Facts
1	`s = [5, 6, 8]` `t = [4, 3]`
`E temp := s.pop();`	
2	
`t.push(temp);`	
3	

we expect:
s = [6, 8]
t = [5, 4, 3]

State	Facts
1	s = [5, 6, 8] t = [4, 3]
E temp := s.pop();	
2	
t.push(temp);	
3	

```
public E pop();
  requires this ≠ [];
  ensures
    this = ALL_BUT_FIRST(#this) and
    result = FIRST(#this);
```

we expect:
s = [6, 8]
t = [5, 4, 3]

State	Facts
1	s = [5, 6, 8] t = [4, 3]
E temp := s.pop();	
2	s = [6, 8] t = [4, 3] temp = 5
t.push(temp);	
3	

```
public E pop();
  requires this ≠ [];
  ensures
    this = ALL_BUT_FIRST(#this) and
    result = FIRST(#this);
```

we expect:
s = [6, 8]
t = [5, 4, 3]

State	Facts
1	s = [5, 6, 8] t = [4, 3]
E temp := s.pop();	
2	s = [6, 8] t = [4, 3] temp = 5
t.push(temp);	
3	

we expect:
s = [6, 8]
t = [5, 4, 3]

State	Facts
1	s = [5, 6, 8] t = [4, 3]
E temp := s.pop();	
2	s = [6, 8] t = [4, 3] temp = 5
t.push(temp);	
3	

```
public void push(E element);
  ensures
    this = [#element] + #this and
    element = ??;
```

State	Facts
1	s = [5, 6, 8] t = [4, 3]
E temp := s.pop();	
2	s = [6, 8] t = [4, 3] temp = 5
t.push(temp);	
3	s = [6, 8] t = [5, 4, 3] temp = ??

```
public void push(E element);
  ensures
    this = [#element] + #this and
    element = ??;
```

State	Facts
1	s = [5, 6, 8] t = [4, 3]
E temp := s.pop();	
2	s = [6, 8] t = [4, 3] temp = 5
t.push(temp);	
3	s = [6, 8] t = [5, 4, 3] temp = ??

we expect:
s = [6, 8]
t = [5, 4, 3]

Pause and Think

Question:

By tracing through the code as we did, what did we prove about the code relative to it’s specification?

Pause and Think

Question:

By tracing through the code as we did,
what did we prove about the code
relative to it's specification?

Answer:

We have shown that the code is correct
with respect to the specification,
for one particular input.

```
public interface Stack<E> {
   model MathSequence<E>;

    public Stack();
        ensures this = [];

    public void push(E element);
        ensures this = [#element] + #this and
                element = ??;

    public E pop();
        requires this ≠ [];
        ensures this = ALL_BUT_FIRST(#this) and
                result = FIRST(#this);

    public Integer depth();
       ensures result = |#this| and this = #this;
}
```

Informal Specification	Formal Specification
• Easier to write than formal specifications • Understanding and writing them does not require a lot of math • Pictures and diagrams help clients quickly grasp the big picture • They are often vague and ambiguous • They cannot be understood by computer programs	• More concise than informal specifications • Precise and unambiguous • Can be understood by other computer programs, which is important if you want tools to help you analyze, test, or verify your code • Understanding them requires basic math knowledge • Writing good specifications requires a solid mathematical background

- ✓ Components
- ✓ Specification
- Reasoning
- The Future

Reasoning


THE COFFEEBLOGGER SEZ...
TRUST...
BUT VERIFY!

```
public static void transferTop(Stack<E> s;
                               Stack<E> t)
    requires s ≠ [];
    ensures s = ALL_BUT_FIRST(#s) and
            t = [FIRST(#s)] + #t;
{
    E temp := s.pop();
    t.push(temp);
}
```

```
public static void transferTop(Stack<E> s;
                               Stack<E> t)
    requires s ≠ [];
    ensures s = ALL_BUT_FIRST(#s) and
            t = [FIRST(#s)] + #t;
{
    E temp := s.pop();
    t.push(temp);
}
```

Input (state 1)	Expected output (state 3)

```
public static void transferTop(Stack<E> s;
                               Stack<E> t)
    requires s ≠ [];
    ensures s = ALL_BUT_FIRST(#s) and
            t = [FIRST(#s)] + #t;
{
    E temp := s.pop();
    t.push(temp);
}
```

Input (state 1)	Expected output (state 3)
$s_1 \neq []$	

```
public static void transferTop(Stack<E> s;
                               Stack<E> t)
    requires s ≠ [];
    ensures s = ALL_BUT_FIRST(#s) and
            t = [FIRST(#s)] + #t;
{
    E temp := s.pop();
    t.push(temp);
}
```

Input (state 1)	Expected output (state 3)
$s_1 \neq []$	$s_3 = \text{ALL\_BUT\_FIRST}(s_1)$ $t_3 = \text{FIRST}(s_1) + t_1$

State	Facts	Obligations
1	$s_1 \neq []$	
E temp := s.pop();		
2		
t.push(temp);		
3		$s_3 =$ $\text{ALL\_BUT\_FIRST}(s_1)$ $t_3 = [\text{FIRST}(s_1)] + t_1$

State	Facts	Obligations
1	$s_1 \neq []$	
E temp := s.pop();		
2		
t.push(temp);		
3		

```
public E pop();
  requires this ≠ [];
  ensures
    this = ALL_BUT_FIRST(#this) and
    result = FIRST(#this);
```

State	Facts	Obligations
1	$s_1 \neq []$	$s_1 \neq []$
E temp := s.pop();		
2		
t.push(temp);		
3		

```
public E pop();
  requires this ≠ [];
  ensures
    this = ALL_BUT_FIRST(#this) and
    result = FIRST(#this);
```

State	Facts	Obligations
1	$s_1 \neq []$	$s_1 \neq []$
E temp := s.pop();		
2	s_2 = ALL_BUT_FIRST(s_1) $t_2 = t_1$ $temp_2$ = FIRST(s_1)	
t.push(temp);		
3		

```
public E pop();
  requires this ≠ [];
  ensures
    this = ALL_BUT_FIRST(#this) and
    result = FIRST(#this);
```

<table>
<tr><th>State</th><th>Facts</th><th>Obligations</th></tr>
<tr><td>1</td><td>$s_1 \neq []$</td><td>$s_1 \neq []$</td></tr>
<tr><td colspan="3"><code>E temp := s.pop();</code></td></tr>
<tr><td>2</td><td>s_2 = ALL_BUT_FIRST(s_1)
$t_2 = t_1$
$temp_2$ = FIRST(s_1)</td><td></td></tr>
<tr><td colspan="3"><code>t.push(temp);</code></td></tr>
<tr><td>3</td><td></td><td>s_3 =
ALL_BUT_FIRST(s_1)
t_3 = [FIRST(s_1)] + t_1</td></tr>
</table>

```
public void push(E element);
  ensures
    this = [#element] + #this and
    element = ??;
```

State	Facts	Obligations
		$\neq$ []
2	s_2 = ALL_BUT_FIRST(s_1) t_2 = t_1 $temp_2$ = FIRST(s_1)	
t.push(temp);		
3		s_3 = ALL_BUT_FIRST(s_1) t_3 = [FIRST(s_1)] + t_1

```
public void push(E element);
  ensures
    this = [#element] + #this and
    element = ??;
```

State	Facts	Obligations
		≠ []
2	s_2 = ALL_BUT_FIRST(s_1) t_2 = t_1 $temp_2$ = FIRST(s_1)	/* no obligations */
t.push(temp);		
3		s_3 = ALL_BUT_FIRST(s_1) t_3 = [FIRST(s_1)] + t_1

State	Facts	Obligations
		$\neq$ []
2	s_2 = ALL_BUT_FIRST(s_1) t_2 = t_1 $temp_2$ = FIRST(s_1)	/* no obligations */
t.push(temp);		
3	s_3 = s_2 t_3 = [$temp_2$] + t_2 $temp_3$ = ??	s_3 = ALL_BUT_FIRST(s_1) t_3 = [FIRST(s_1)] + t_1

```
public void push(E element);
  ensures
    this = [#element] + #this and
    element = ??;
```

State	Facts	Obligations
1	$s_1 \neq []$	$s_1 \neq []$
E temp := s.pop();		
2	s_2 = ALL_BUT_FIRST(s_1) $t_2 = t_1$ $temp_2$ = FIRST(s_1)	/* no obligations */
t.push(temp);		
3	$s_3 = s_2$ $t_3 = [temp_2] + t_2$ $temp_3$ = ??	s_3 = ALL_BUT_FIRST(s_1) t_3 = [FIRST(s_1)] + t_1

Pause and Think

Question:

By completing the symbolic reasoning table, what did we prove about the code relative to it's specification?

Pause and Think

Question:
By completing the symbolic reasoning table,
what did we prove about the code
relative to it's specification?

Answer:
Nothing! (yet)
To prove that the code is correct
with respect to the specification,
we need to prove <u>all</u> the obligations.

$s_1 \neq []$	$s_2 = ALL\_BUT\_FIRST(s_1)$	$s_3 = s_2$
	$t_2 = t_1$	$t_3 = [temp_2] + t_2$
	$temp_2 = FIRST(s_1)$	$temp_3 = ??$

$s_1 \neq []$	$s_2 = \text{ALL\_BUT\_FIRST}(s_1)$	$s_3 = s_2$
	$t_2 = t_1$	$t_3 = [temp_2] + t_2$
	$temp_2 = \text{FIRST}(s_1)$	$temp_3 = ??$

$t_3 = [\text{FIRST}(s_1)] + t_1$

$s_1 \neq []$	s_2 = ALL_BUT_FIRST(s_1)	$s_3 = s_2$
	$t_2 = t_1$	t_3 = [$temp_2$] + t_2
	$temp_2$ = FIRST(s_1)	$temp_3$ = ??

t_3 = [FIRST(s_1)] + t_1

➡ [$temp_2$] + t_2 = [FIRST(s_1)] + t_1 Substitution from Fact 3

$s_1 \neq []$ | $s_2 = \text{ALL\_BUT\_FIRST}(s_1)$ | $s_3 = s_2$
$t_2 = t_1$ | $t_3 = [temp_2] + t_2$
$temp_2 = \text{FIRST}(s_1)$ | $temp_3 = ??$

$t_3 = [\text{FIRST}(s_1)] + t_1$

⇒ $[temp_2] + t_2 = [\text{FIRST}(s_1)] + t_1$ Substitution from Fact 3

⇒ $[temp_2] + t_1 = [\text{FIRST}(s_1)] + t_1$ Substitution from Fact 2

$s_1 \neq []$	$s_2 = \text{ALL\_BUT\_FIRST}(s_1)$	$s_3 = s_2$
	$t_2 = t_1$	$t_3 = [temp_2] + t_2$
	$temp_2 = \text{FIRST}(s_1)$	$temp_3 = ??$

$t_3 = [\text{FIRST}(s_1)] + t_1$

⇒ $[temp_2] + t_2 = [\text{FIRST}(s_1)] + t_1$ Substitution from Fact 3

⇒ $[temp_2] + t_1 = [\text{FIRST}(s_1)] + t_1$ Substitution from Fact 2

⇒ $[temp_2] = [\text{FIRST}(s_1)]$ Algebra on sequences

$s_1 \neq []$	$s_2 = \text{ALL\_BUT\_FIRST}(s_1)$	$s_3 = s_2$
	$t_2 = t_1$	$t_3 = [temp_2] + t_2$
	$temp_2 = \text{FIRST}(s_1)$	$temp_3 = ??$

$t_3 = [\text{FIRST}(s_1)] + t_1$

⇒ $[temp_2] + t_2 = [\text{FIRST}(s_1)] + t_1$ Substitution from Fact 3

⇒ $[temp_2] + t_1 = [\text{FIRST}(s_1)] + t_1$ Substitution from Fact 2

⇒ $[temp_2] = [\text{FIRST}(s_1)]$ Algebra on sequences

⇒ $temp_2 = \text{FIRST}(s_1)$ Algebra on sequences

$s_1 \neq []$	$s_2 = \text{ALL\_BUT\_FIRST}(s_1)$	$s_3 = s_2$
	$t_2 = t_1$	$t_3 = [temp_2] + t_2$
	$temp_2 = \text{FIRST}(s_1)$	$temp_3 = ??$

$t_3 = [\text{FIRST}(s_1)] + t_1$

➡ $[temp_2] + t_2 = [\text{FIRST}(s_1)] + t_1$ Substitution from Fact 3

➡ $[temp_2] + t_1 = [\text{FIRST}(s_1)] + t_1$ Substitution from Fact 2

➡ $[temp_2] = [\text{FIRST}(s_1)]$ Algebra on sequences

➡ $temp_2 = \text{FIRST}(s_1)$ Algebra on sequences

True from Fact 2 Q.E.D

- ✓ Components
- ✓ Specification
- ✓ Reasoning
- The Future

The Future


C11 (L) TOTAL
C!
25
A B C D
1 ITEM NO. UNIT COST
2 ---- --- ---- ----
3 MUCK RAKE 43 12.95 556.85
4 BUZZ CUT 15 6.75 101.25
5 TOE TONER 250 49.95 12487.50
6 EYE SNUFF 2 4.95 9.90
7 ----------
8 SUBTOTAL 13155.50
9 9.75% TAX 1282.66
10 ----------
11 TOTAL 14438.16
12
13
14
15
16
17
18
19
20


C11 (L) TOTAL
C!
25
A B C D
ITEM NO. UNIT COST
MUCK RAKE 43 12.95 556.85
BUZZ CUT 15 6.75 101.25
TOE TONER 250 49.95 12487.50
EYE SNUFF 2 4.95 9.90
SUBTOTAL 13155.50
9.75% TAX 1282.66
TOTAL 14438.16

VisiCalc

Grand Challenge

for Computing Research

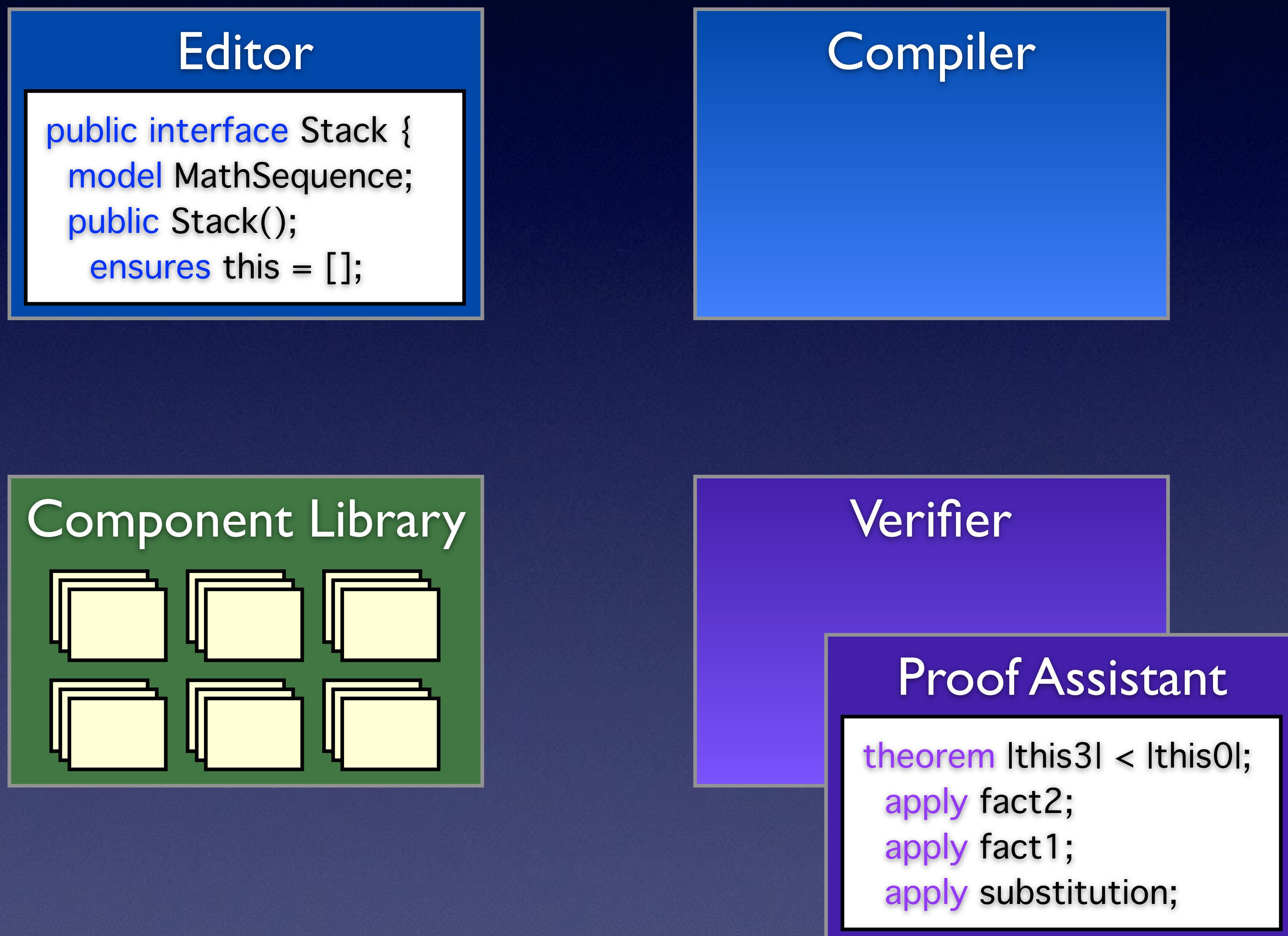
The construction and application of a verifying compiler that guarantees correctness of a program before running it.

– Tony Hoare, 2003

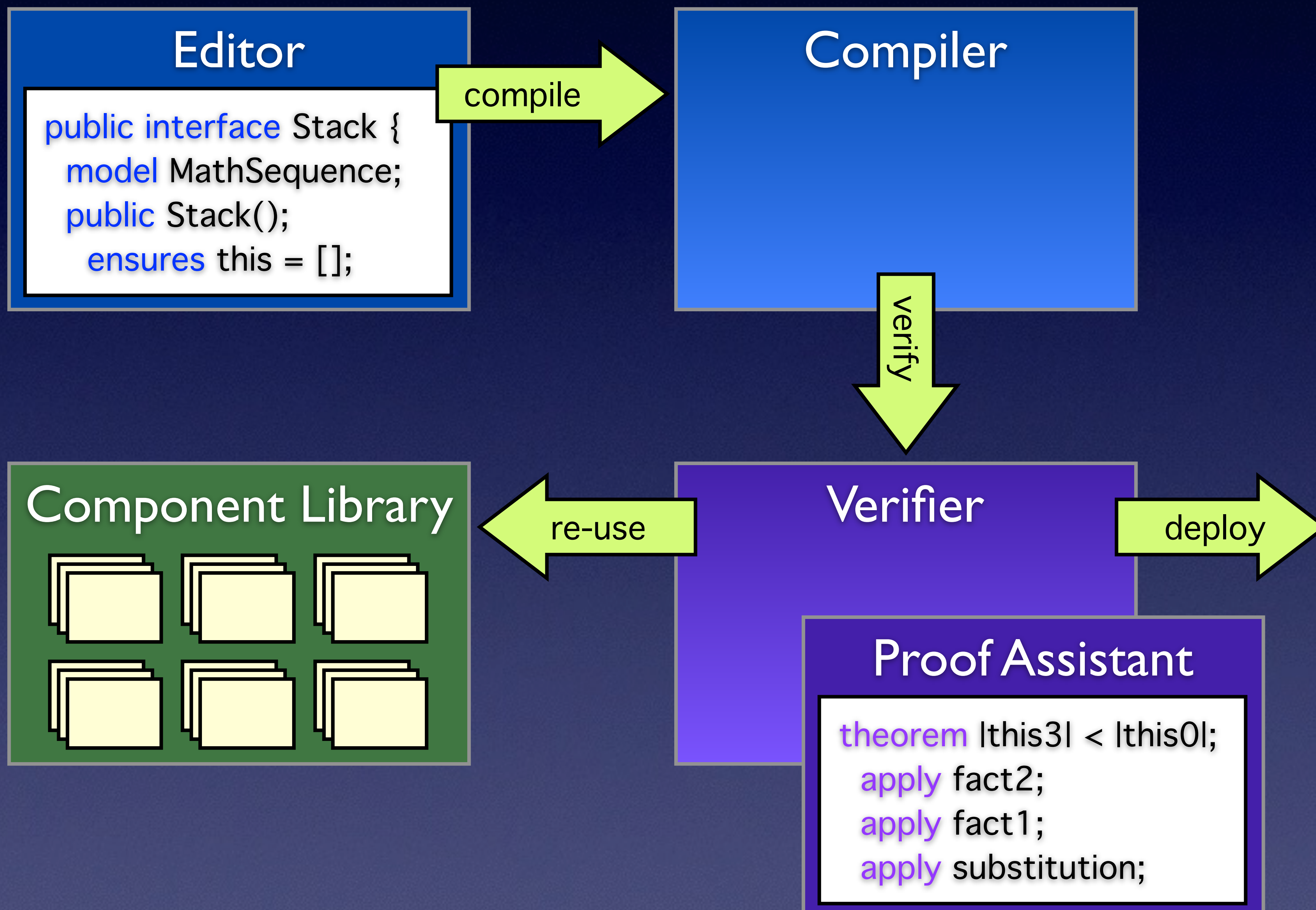
Typical Grand Challenges

- Prove Fermat’s last theorem (done)
- Put a man on the moon (done)
- Cure cancer in 10 years (failed in 1970’s)
- Prove that P is not equal to NP (open)
- Turing test (done)
- Championship chess program (done)

Verifying Compiler

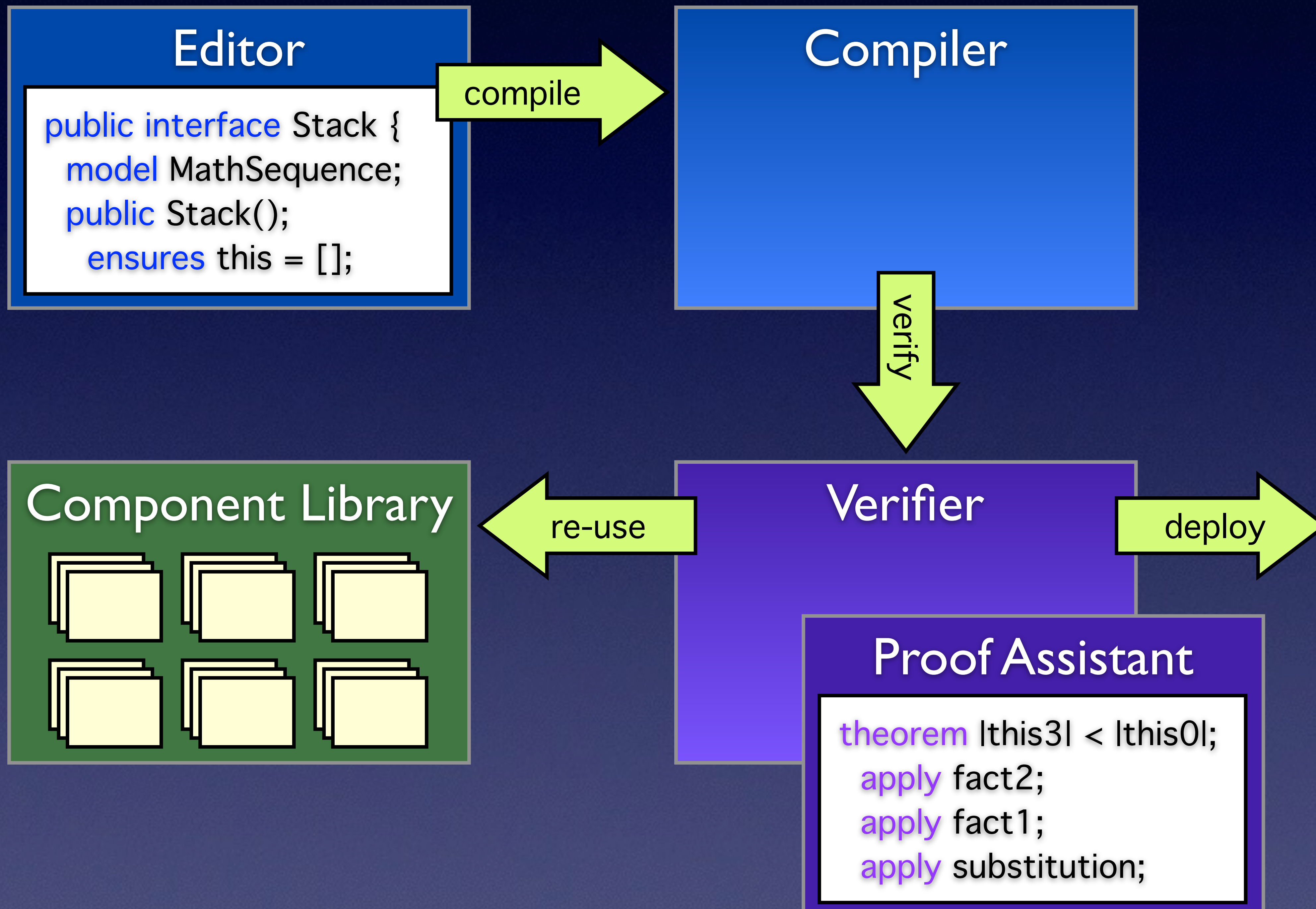


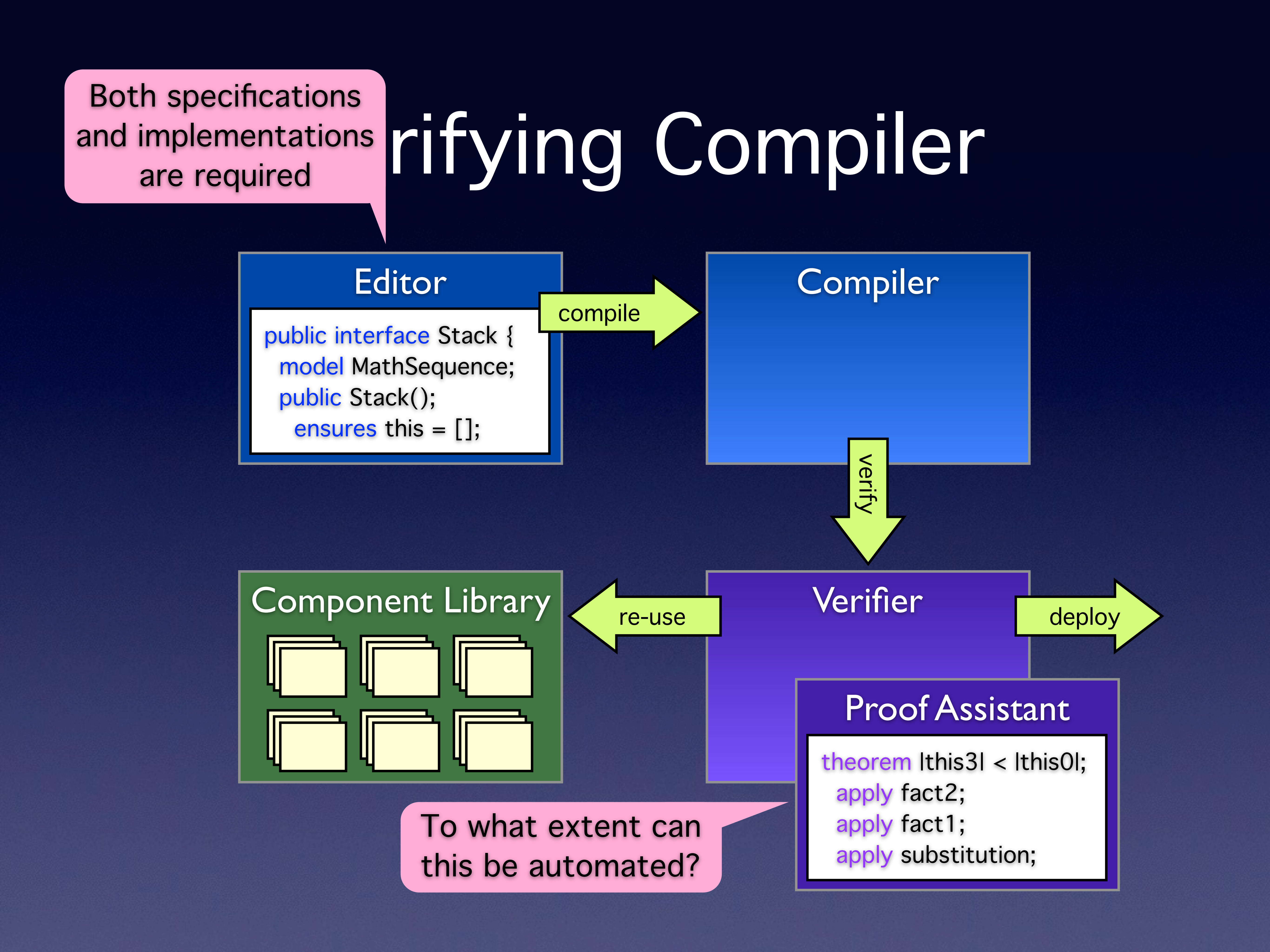
Verifying Compiler



Both specifications and implementations are required

rifying Compiler




Both specifications and implementations are required
rifying Compiler
Editor
public interface Stack {
model MathSequence;
public Stack();
ensures this = [];
compile
Compiler
verify
Component Library
re-use
Verifier
deploy
Proof Assistant
theorem |this3| < |this0|;
apply fact2;
apply fact1;
apply substitution;
To what extent can this be automated?

References

- Dafny Tutorial at rise4fun.com/Dafny/tutorial
- "The Seven Myths of Formal Methods" by Anthony Hall
- "Reasoning about Software Component Behavior" by Sitaraman et al.
- "The Verifying Compiler: A Grand Challenge for Computing Research" by Tony Hoare