Outline

0. Goals

   What we are trying to do, for whom, and how.

1. Process & Architecture

   Organizing Software as Components, Packages, & Package Groups.

2. Design & Implementation

   Using Class Categories, Value Semantics, & Vocabulary Types.

3. Verification & Testing

   Component-Level Test Drivers, Peer Review, & Defensive Checks.

4. Bloomberg Development Environment (BDE)

   Rendered as Fine-Grained *Hierarchically* *Reusable* Components.

Outline

0. Goals

   What we are trying to do, for whom, and how.

1. Process & Architecture

   Organizing Software as Components, Packages, & Package Groups.

2. Design & Implementation

   Using Class Categories, Value Semantics, & Vocabulary Types.

3. Verification & Testing

   Component-Level Test Drivers, Peer Review, & Defensive Checks.

4. Bloomberg Development Environment (BDE)

   Rendered as Fine-Grained *Hierarchically* *Reusable* Components.

3. Verification & Testing

Essential Strategies and Techniques

Ensuring our own reliability while improving that of our clients:

a) Component-Level Testing
b) Peer Review
c) Static Analysis Tools
d) Defensive (Precondition) Checks

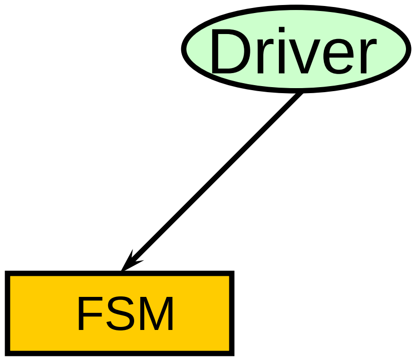
Essential Strategies and Techniques

Ensuring our own reliability while improving that of our clients:

a) Component-Level Testing
b) Peer Review
c) Static Analysis Tools
d) Defensive (Precondition) Checks

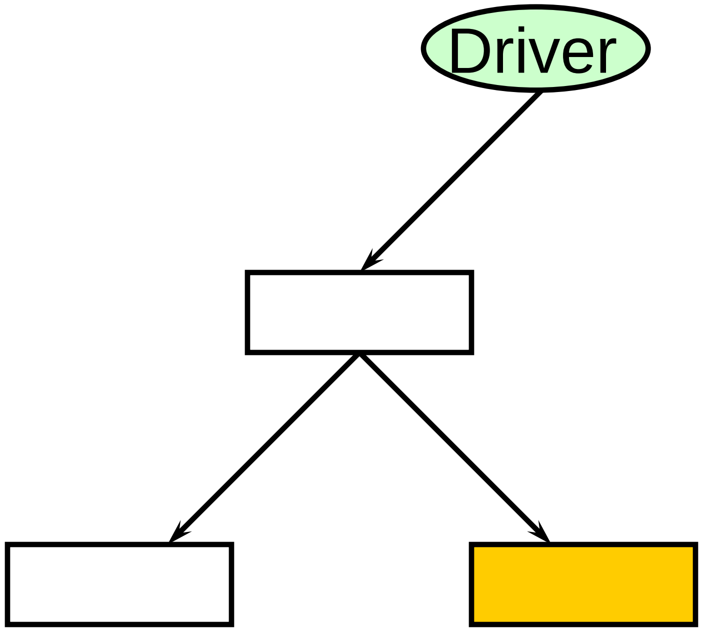
Testing Proximately?

A small state machine is easy to test.



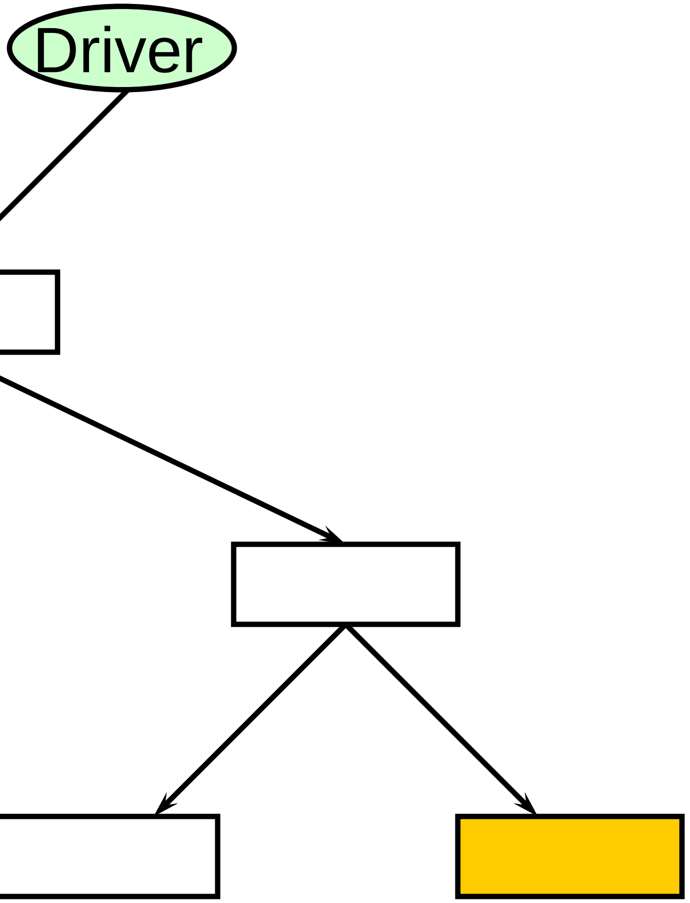
Testing Proximately?

But even if all states are theoretically reachable …



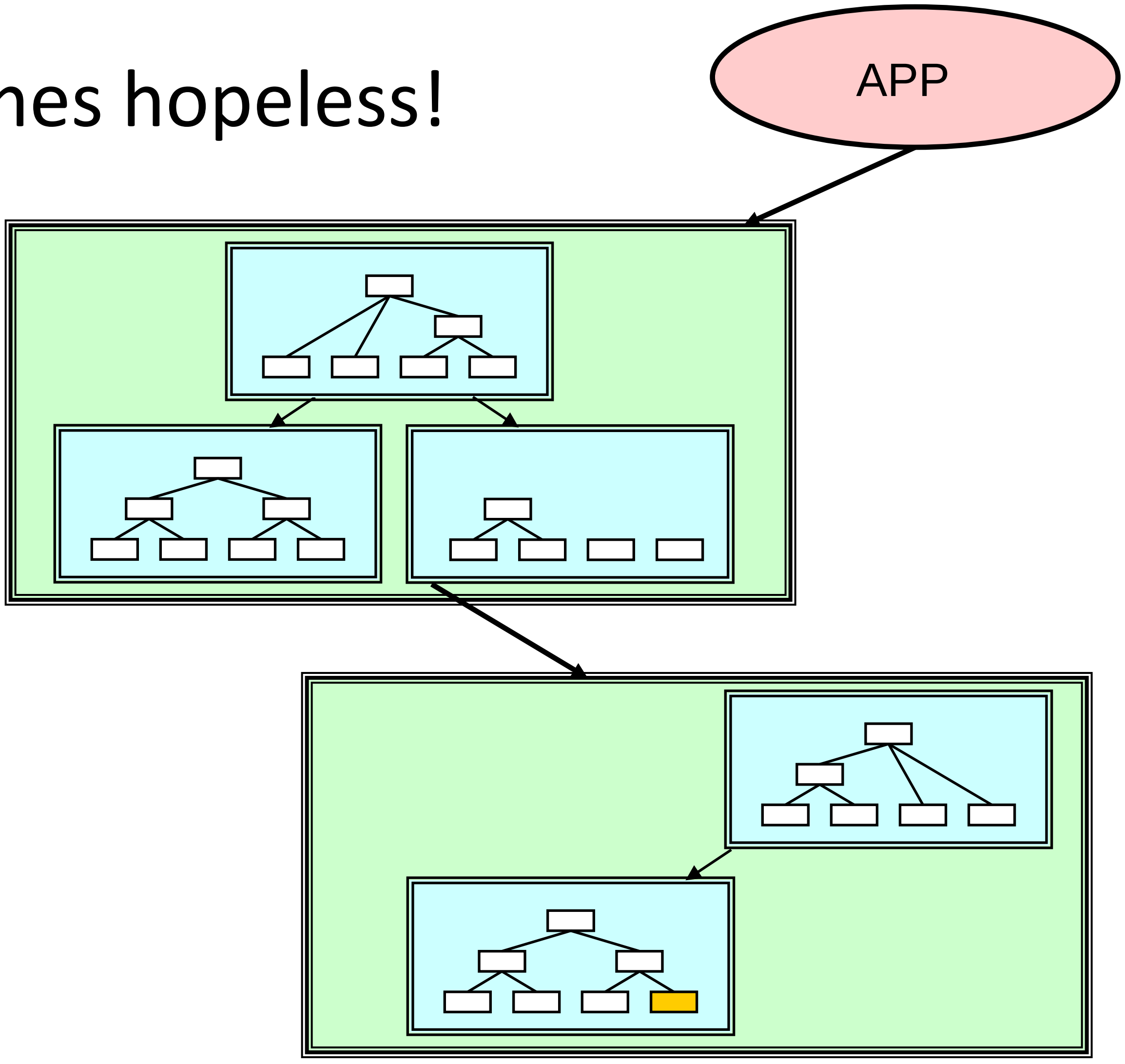
Testing Proximately?

As the system grows …



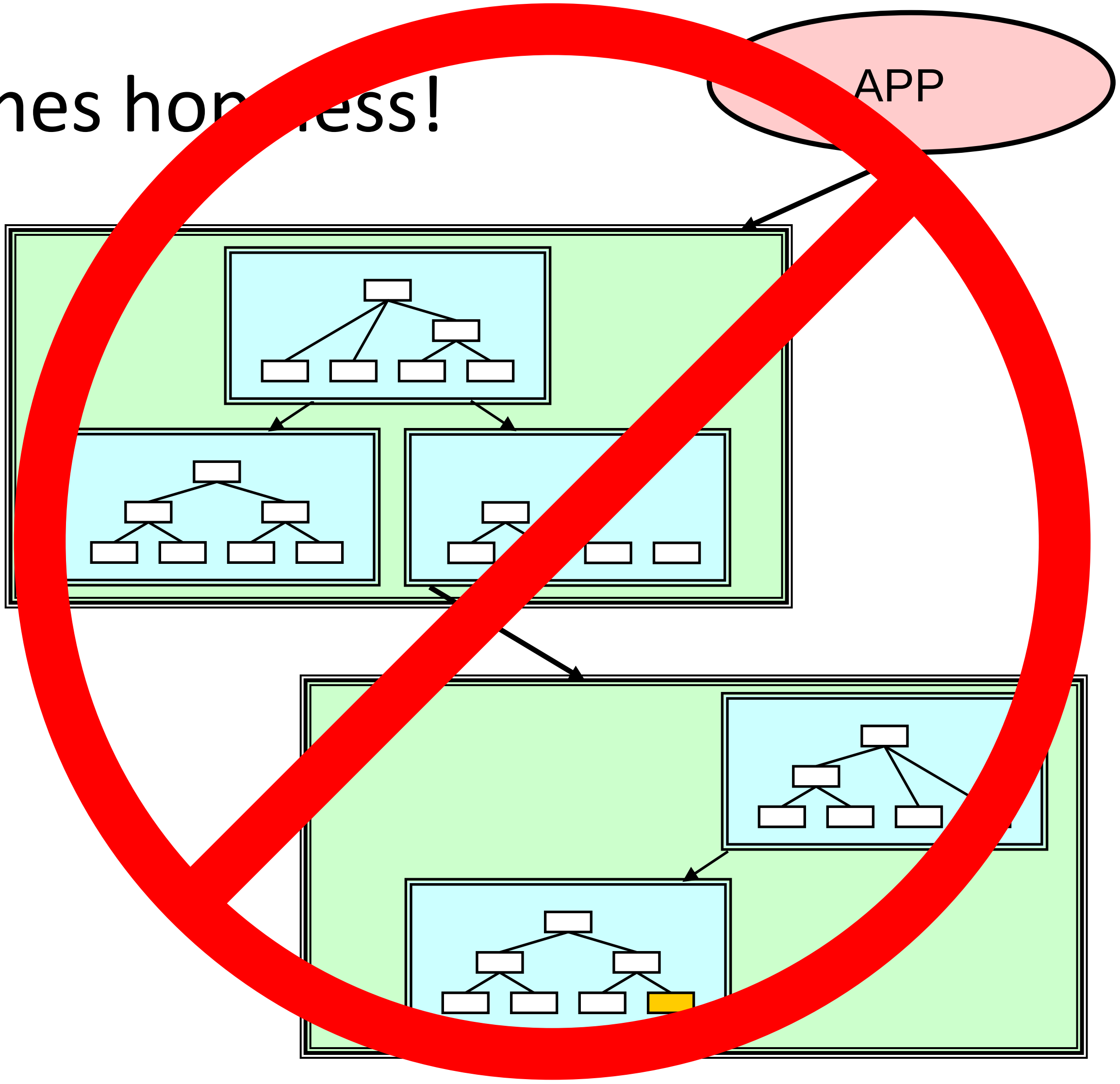
Testing Proximately?

It becomes hopeless!



Testing Proximately?

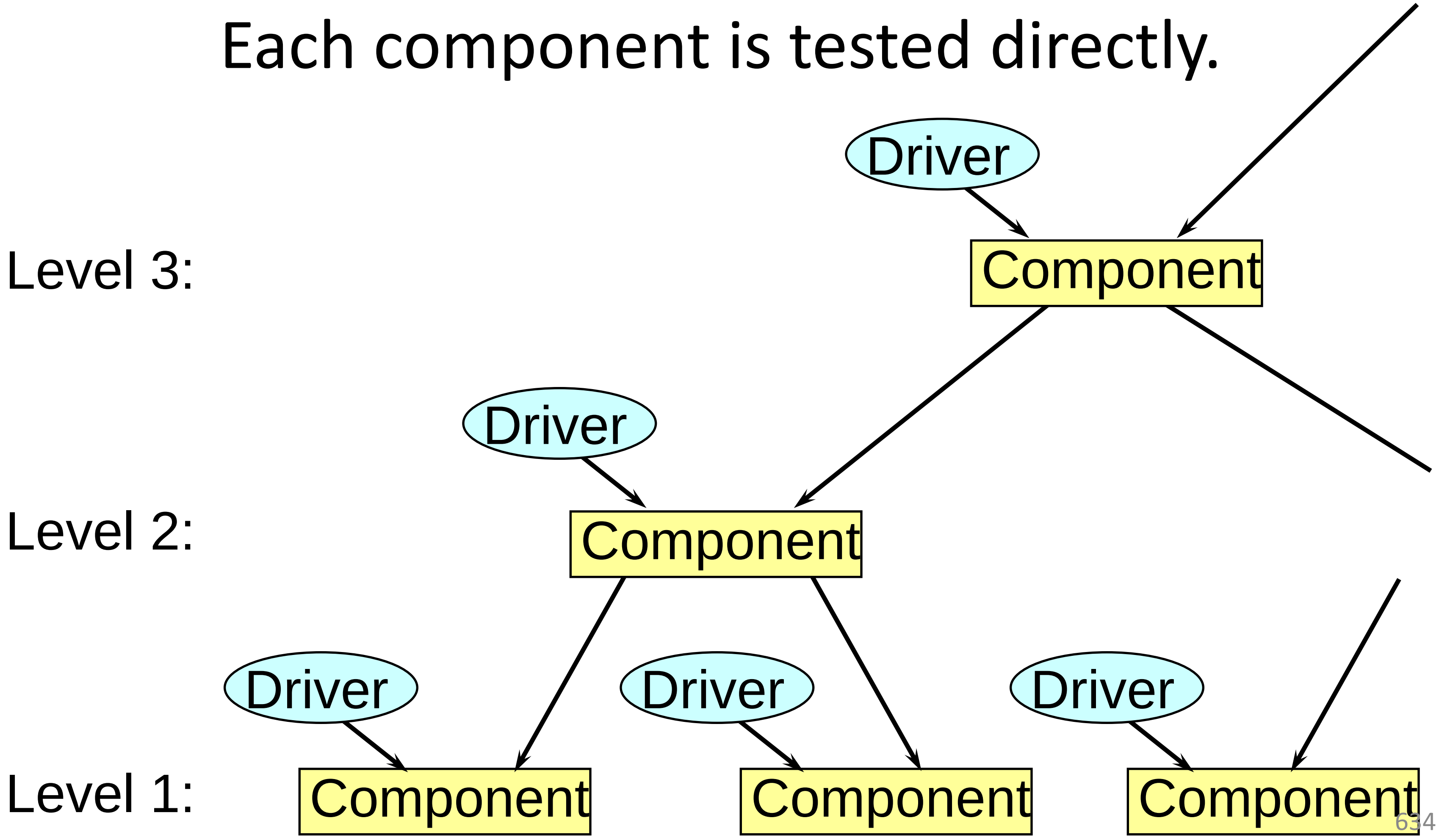
It becomes hopeless!



Component-Level Testing

Hierarchical Testing Strategy:

Each component is tested directly.

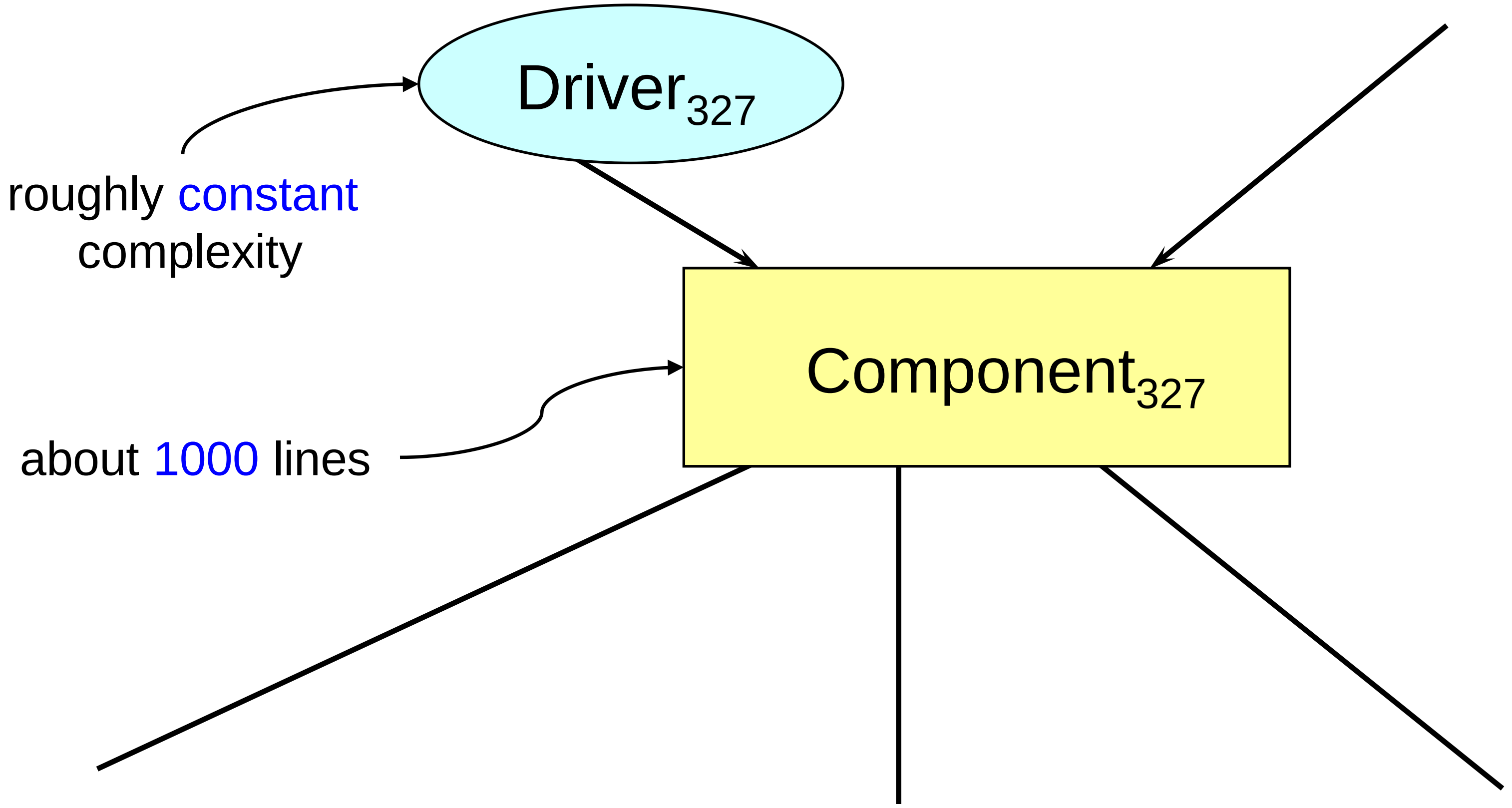


Component-Level Testing

Incremental Functionality Testing:

Test only the **value added** by a component.

No need to retest subordinate functionality.



3. Verification & Testing

Component-Level Testing

Component-level testing methodology overview:

Component-Level Testing

Component-level testing methodology overview:

1. Provide a fundamentally different representation of behavior.

Component-Level Testing

Component-level testing methodology overview:

1. Provide a fundamentally different representation of behavior.

2. Use one of our systematic *Test Data Selection Methods*.

Component-Level Testing

Component-level testing methodology overview:

1. Provide a fundamentally different representation of behavior.
2. Use one of our systematic *Test Data Selection Methods*.
3. Apply our standard *Test Case Implementation Techniques*.

Component-Level Testing

Component-level testing methodology overview:

1. Provide a fundamentally different representation of behavior.

2. Use one of our systematic *Test Data Selection Methods*.

3. Apply our standard *Test Case Implementation Techniques*.

4. Order test cases so as to exploit already tested functionality.

Component-Level Testing

Component-level testing methodology overview:

1. Provide a fundamentally different representation of behavior.
2. Use one of our systematic *Test Data Selection Methods*.
3. Apply our standard *Test Case Implementation Techniques*.
4. Order test cases so as to exploit already tested functionality.

Lots more to this story!

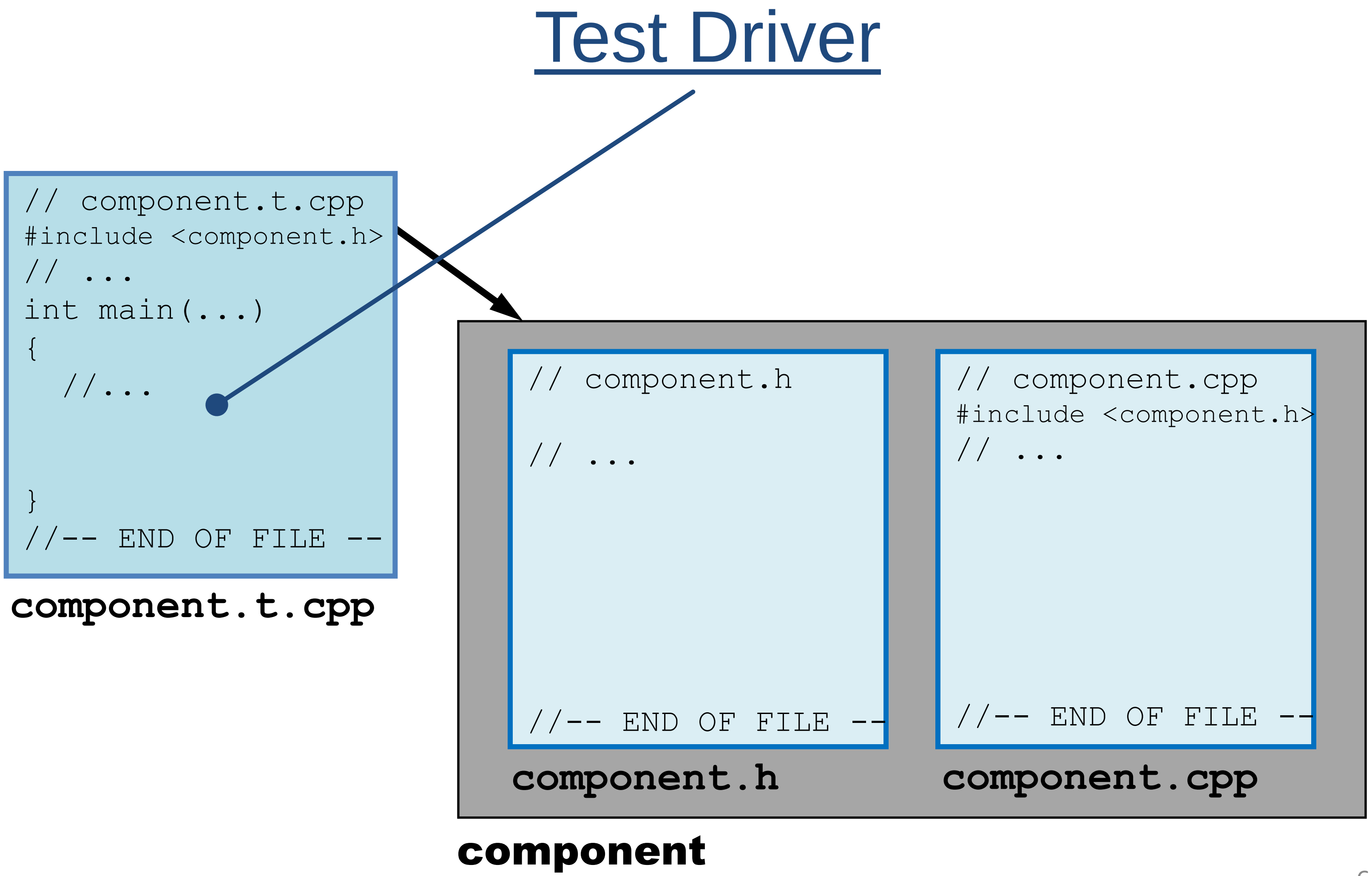
3. Verification & Testing

The Component-Level Test Driver

The Component-Level Test Driver

What is a Test Driver?

The Component-Level Test Driver



```
// component.t.cpp
#include <component.h>
// ...
int main(...)
{
  //...

}
//-- END OF FILE --
```

```
// component.h

// ...

//-- END OF FILE --
```

```
// component.cpp
#include <component.h>
// ...

//-- END OF FILE --
```

3. Verification & Testing

The Component-Level Test Driver

What is a Test Driver?

- It’s a **tool** for developers
  – used during the initial development process.

The Component-Level Test Driver

What is a Test Driver?

- It’s a **tool** for developers
  - used during the initial development process.
- It’s a **“cartridge”** for an automated regression-testing system
  - used throughout the lifetime of the component.

3. Verification & Testing

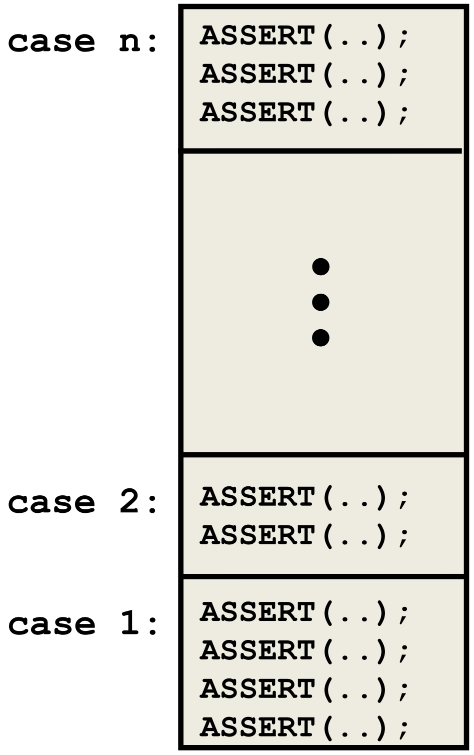
The Component-Level Test Driver

What does a BDE Test Driver comprise?

The Component-Level Test Driver

What does a BDE Test Driver comprise?

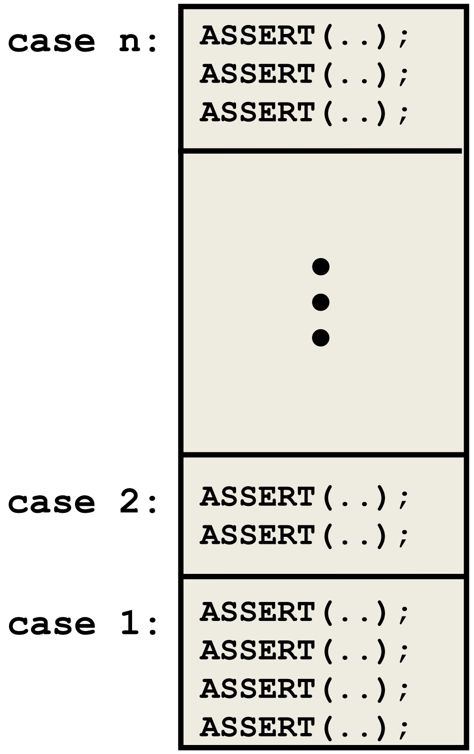
- Set of consecutively numbered ***test cases***.



The Component-Level Test Driver

What does a BDE Test Driver comprise?

- Set of consecutively numbered ***test cases***.
- Each *test case* performs some number of individual ***ASSERTIONS***.



3. Verification & Testing

The Component-Level Test Driver

What is the *User Experience*?

The Component-Level Test Driver

What is the *User Experience*?

- A test driver should succeed quietly in production.

The Component-Level Test Driver

What is the *User Experience*?

- A test driver should succeed quietly in production.
- When an error occurs, the test driver should report the offending expression along with the line number:

```
filename(line #): 2 == sqrt(4) (failed)
```

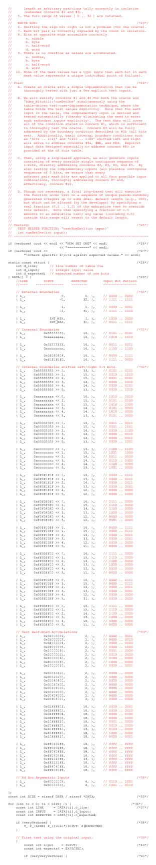
The Component-Level Test Driver

Verbose Mode:

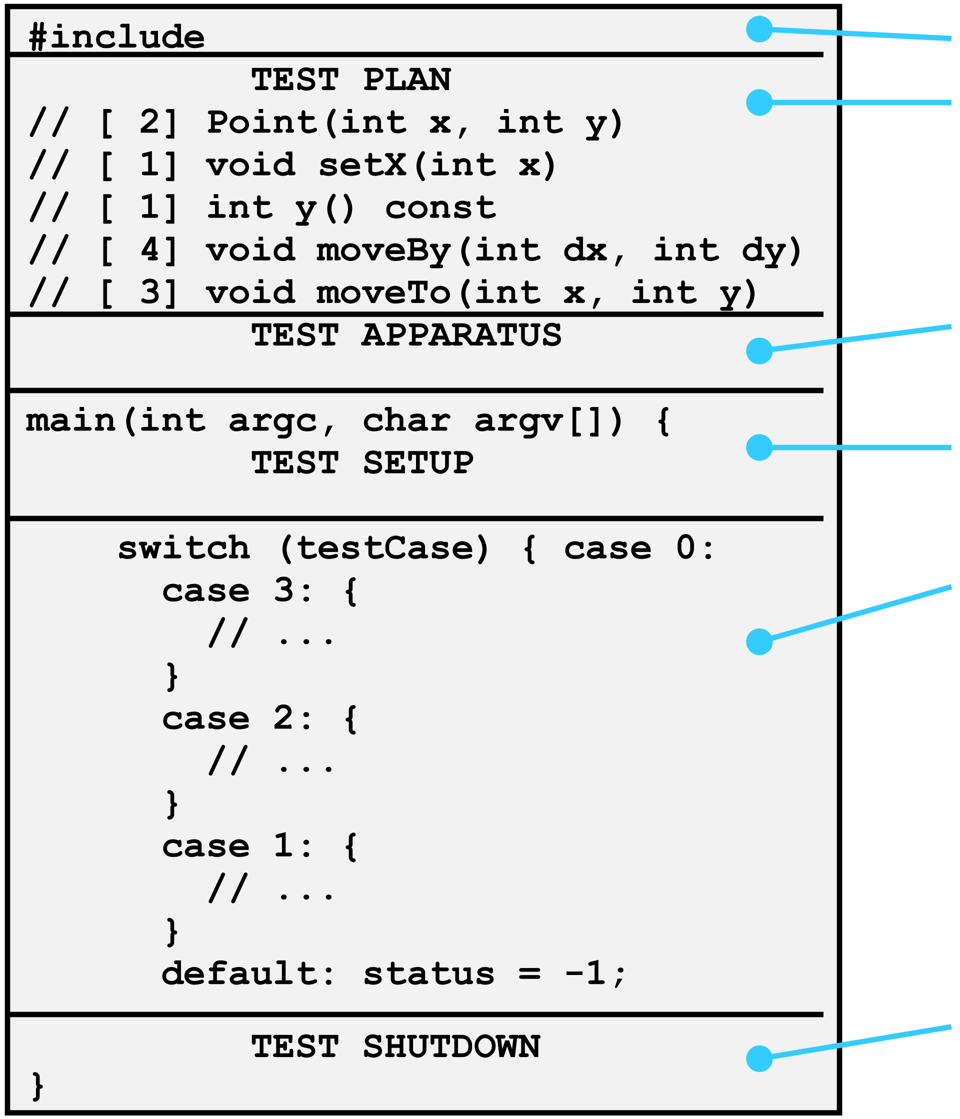
```
Testing length 0
      without aliasing
      with aliasing
Testing length 1
      without aliasing
      with aliasing
Testing length 2
      without aliasing
      with aliasing

...
```

3. Verification & Testing

BDE Test-Driver Layout



BDE Test-Driver Layout

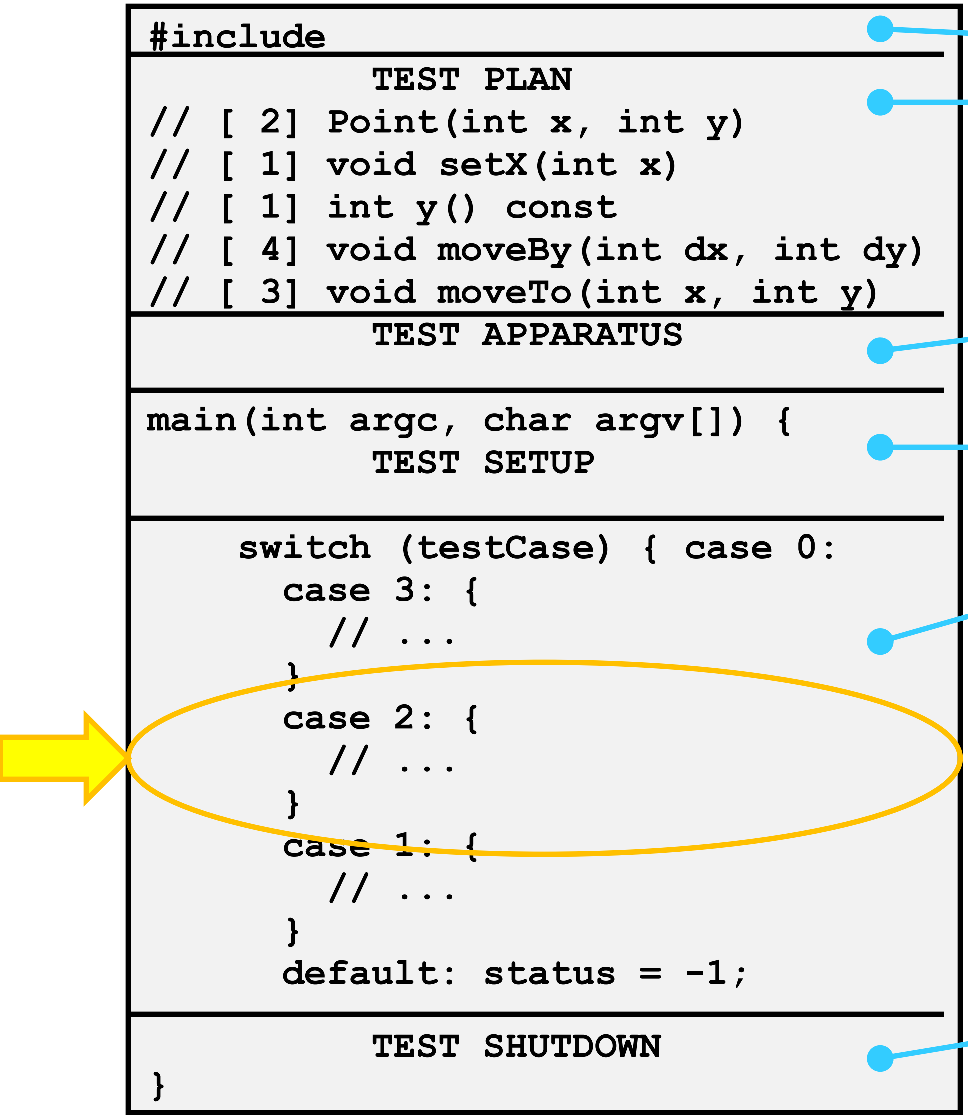


```
#include
            TEST PLAN
// [ 2] Point(int x, int y)
// [ 1] void setX(int x)
// [ 1] int y() const
// [ 4] void moveBy(int dx, int dy)
// [ 3] void moveTo(int x, int y)
            TEST APPARATUS

main(int argc, char argv[]) {
            TEST SETUP

     switch (testCase) { case 0:
       case 3: {
         // ...
       }
       case 2: {
         // ...
       }
       case 1: {
         // ...
       }
       default: status = -1;

            TEST SHUTDOWN
}
```

- include directives
- test plan identifying case in which each public function is fully tested
- ASSERT macro definition, supporting functions, etc.
- common setup for all test cases
- switch on test case number (actual test code goes here)
- any common cleanup code (rare)

3. Verification & Testing

Test Case



```
#include
            TEST PLAN
// [ 2] Point(int x, int y)
// [ 1] void setX(int x)
// [ 1] int y() const
// [ 4] void moveBy(int dx, int dy)
// [ 3] void moveTo(int x, int y)
            TEST APPARATUS

main(int argc, char argv[]) {
            TEST SETUP

     switch (testCase) { case 0:
       case 3: {
         // ...
       }
       case 2: {
         // ...
       }
       case 1: {
         // ...
       }
       default: status = -1;

            TEST SHUTDOWN
}
```

- include directives
- test plan identifying case in which each public function is fully tested
- ASSERT macro definition, supporting functions, etc.
- common setup for all test cases
- switch on test case number (actual test code goes here)
- any common cleanup code (rare)

3. Verification & Testing

Test Case

- TITLE
  - Short Label (printed in verbose mode) + optional intro.
- CONCERNS
  - Precise (and concise) description of “what could go wrong” <u>with this particular implementation.</u>
- PLAN
  - How this test case will address each of our concerns.
- TESTING
  - Copy-and-paste cross-reference from the overall test plan.

3. Verification & Testing

Test Case

- TITLE
  - Short Label (printed in verbose mode) + optional intro.
- CONCERNS
  - Precise (and concise) description of "what could go wrong" with this particular implementation.
- PLAN
  - How this test case will address each of our concerns.
- TESTING
  - Copy-and-paste cross-reference from the overall test plan.

BDE Test-Case Layout

```
} break;
case 2: {
  //----------------------------------------------------------------
  // UNIQUE BIRTHDAY
  //   The value returned for an input of 365 is small.
  //
  // Concerns:
  //   1. That it can represent the result as a double.
  //   2. …
  //   …
  //   6. That the special-case input of 0 returns 1.
  //   7. …
  //
  // Plan:
  //   Test for explicit values near 0, 365, and INT_MAX.
  //
  // Testing:
  //   double uniqueBirthday(int value);
  //----------------------------------------------------------------

  if (verbose) cout << endl << "UNIQUE BIRTHDAY" << endl
                            << "===============" << endl;

  // …
```

test code goes here

```
} break;
 case 1: {
```

BDE Test-Case Layout

```
} break;
case 2: {
  //---------------------------------------------------------
  // UNIQUE BIRTHDAY
  //   The value returned for an input of 365 is small.
  //
  // Concerns:
  //   1. That it can represent the result as a double.
  //   2. …
  //   …
  //   6. That the special-case input of 0 returns 1.
  //   7. …
  //
  // Plan:
  //   Test for explicit values near 0, 365, and INT_MAX.
  //
  // Testing:
  //   double uniqueBirthday(int value);
  //---------------------------------------------------------

  if (verbose) cout << endl << "UNIQUE BIRTHDAY" << endl
                            << "===============" << endl;

  ASSERT(1 == uniqueBirthday(0));
  ASSERT(1 == uniqueBirthday(1));
  ASSERT(1 >  uniqueBirthday(2));
  // …
  ASSERT(0 <  uniqueBirthday(365);
  ASSERT(0 == uniqueBirthday(366);
```

BDE Test-Case Layout

```
} break;
case 2: {
  //--------------------------------------------------------------------
  // UNIQUE BIRTHDAY
  //   The value returned for an input of 365 is small.
  //
  // Concerns:
  //   1. That it can represent the result as a double.
  //   2. …
  //   …
  //   6. That the special-case input of 0 returns 1.
  //   7. …
  //
  // Plan:
  //   Test for explicit values near 0, 365, and INT_MAX.
  //
  // Testing:
  //   double uniqueBirthday(int value);
  //--------------------------------------------------------------------

  if (verbose) cout << endl << "UNIQUE BIRTHDAY" << endl
                            << "===============" << endl;

  ASSERT(1 == uniqueBirthday(0));
  ASSERT(1 == uniqueBirthday(1));
  ASSERT(1 >  uniqueBirthday(2));
  // …
  ASSERT(0 <  uniqueBirthday(365);
  ASSERT(0 == uniqueBirthday(366);
```

Essential Strategies and Techniques

Ensuring our own reliability while improving that of our clients:

a) Component-Level Testing

b) Peer Review

c) Static Analysis Tools

d) Defensive (Precondition) Checks

Essential Strategies and Techniques

Ensuring our own reliability while improving that of our clients:

a) Component-Level Testing

b) Peer Review

c) Static Analysis Tools

d) Defensive (Precondition) Checks

3. Verification & Testing

Peer Review

Having another developer review your code helps to ensure that:

- Documentation
- Code
- Tests

are clear, correct, and effective.

3. Verification & Testing
Peer Review

Having another developer review your code helps to ensure that:

- **Documentation**
- **Code**
- **Tests**

are clear, correct, and effective.

3. Verification & Testing

Peer Review

Having another developer review your code helps to ensure that:

- Documentation
- Code
- Tests

are clear, correct, and effective.

Peer Review

Having another developer review your code that:

- Doc
- Coc
- Tes

are clear, correct, a

Is
Complementary to
and
Synergistic with
Component-Level Testing.

Essential Strategies and Techniques

Ensuring our own reliability while improving that of our clients:

a) Component-Level Testing

b) Peer Review

c) Static Analysis Tools

d) Defensive (Precondition) Checks

Essential Strategies and Techniques

Ensuring our own reliability while improving that of our clients:

a) Component-Level Testing

b) Peer Review

c) Static Analysis Tools

d) Defensive (Precondition) Checks

3. Verification & Testing

Static Analysis Tools

- Tools (e.g., clang-based) provide additional consistency checks…

3. Verification & Testing
Static Analysis Tools

❖ Tools (e.g., clang-based) provide additional consistency checks **that can also be used by our clients!**

Static Analysis Tools

- Tools (e.g., clang-based) provide additional consistency checks **that can also be used by our clients!**
- Having a highly **consistent** and **regular** implementation structure…

Static Analysis Tools

- Tools (e.g., clang-based) provide additional consistency checks **that can also be used by our clients!**
- Having a highly **consistent** and **regular** implementation structure **makes** the use of such **tools** all the more **practical** and **effective**.

Essential Strategies and Techniques

Ensuring our own reliability while improving that of our clients:

a) Component-Level Testing
b) Peer Review
c) Static Analysis Tools
d) Defensive (Precondition) Checks

Essential Strategies and Techniques

Ensuring our own reliability while improving that of our clients:

a) Component-Level Testing

b) Peer Review

c) Static Analysis Tools

d) Defensive (Precondition) Checks

3. Verification & Testing

Addressing Client Misuse

As library developers, …

3. Verification & Testing

Addressing Client Misuse

As library developers, how much CPU should we spend detecting misuse?

3. Verification & Testing

Addressing Client Misuse

As library developers, how much CPU should we spend detecting misuse?

a. Less than 5%

Addressing Client Misuse

As library developers, how much CPU should we spend detecting misuse?

a. Less than 5%

b. 5% to 20%

Addressing Client Misuse

As library developers, how much CPU should we spend detecting misuse?

a. Less than 5%

b. 5% to 20%

c. More than 20%, but not more than a constant factor.

Addressing Client Misuse

As library developers, how much CPU should we spend detecting misuse?

a. Less than 5%

b. 5% to 20%

c. More than 20%, but not more than a constant factor.

d. Sky's the limit: factor of O[log(n)], O[n], or more.

3. Verification & Testing

Addressing Client Misuse

As library developers, how much CPU should we spend detecting misuse?

a. Less than 5%
b. 5% to 20%
c. More than 20%, but not more than a constant factor.
d. Sky's the limit: factor of O[log(n)], O[n], or more.

3. Verification & Testing

Addressing Client Misuse

As library developers, …

3. Verification & Testing

Addressing Client Misuse

As library developers, what should happen if we detect misuse?

3. Verification & Testing
Addressing Client Misuse

As library developers, what should happen if we detect misuse?

a. Be fired?

Addressing Client Misuse

As library developers, what should happen if we detect misuse?

a. Be fired?

b. Ignore it, and proceed on? (See a.)

3. Verification & Testing

Addressing Client Misuse

As library developers, what should happen if we detect misuse?

a. Be fired?

b. Ignore it, and proceed on? (See a.)

c. Return immediately, but normally? (See a.)

Addressing Client Misuse

As library developers, what should happen if we detect misuse?

a. Be fired?
b. Ignore it, and proceed on? (See a.)
c. Return immediately, but normally? (See a.)
d. Immediately terminate the program?

Addressing Client Misuse

As library developers, what should happen if we detect misuse?

a. Be fired?
b. Ignore it, and proceed on? (See a.)
c. Return immediately, but normally? (See a.)
d. Immediately terminate the program?
e. Throw an exception?

Addressing Client Misuse

As library developers, what should happen if we detect misuse?

a. Be fired?
b. Ignore it, and proceed on? (See a.)
c. Return immediately, but normally? (See a.)
d. Immediately terminate the program?
e. Throw an exception?
f. Spin, waiting to break into a debugger?

Addressing Client Misuse

As library developers, what should happen if we detect misuse?

a. Be fired?
b. Ignore it, and proceed on? (See a.)
c. Return immediately, but normally? (See a.)
d. Immediately terminate the program?
e. Throw an exception?
f. Spin, waiting to break into a debugger?
g. Something else?

3. Verification & Testing

Addressing Client Misuse

As library developers, what should happen if we detect misuse?

3. Verification & Testing

Addressing Client Misuse

How do we ***as an enterprise*** decide what to do?

3. Verification & Testing

Addressing Client Misuse

How do we **as an enterprise** decide what to do?

It depends…

3. Verification & Testing

Addressing Client Misuse

How do we **as an enterprise** decide what to do?

It depends…

1. How mature is the software?

3. Verification & Testing

Addressing Client Misuse

How do we **as an enterprise** decide what to do?

It depends…

1. How mature is the software?
2. Are we in *alpha*, *beta*, or *production*?

Addressing Client Misuse

How do we **as an enterprise** decide what to do?

It depends…

1. How mature is the software?
2. Are we in *alpha*, *beta*, or *production*?
3. Is this a performance-critical application?

Addressing Client Misuse

How do we **as an enterprise** decide what to do?

It depends…

1. How mature is the software?
2. Are we in *alpha*, *beta*, or *production*?
3. Is this a performance-critical application?
4. Is there something sensible to do?
   a. Save client work before terminating the program.
   b. Log the error, abandon the current transaction, & proceed.
   c. Send a message to the console room and just wait.

3. Verification & Testing

Addressing Client Misuse

Who should decide…

Addressing Client Misuse

Who should decide…

1. **How much time** the library component should spend checking for preconditions?

Addressing Client Misuse

Who should decide…

1. **How much time** the library component should spend checking for preconditions?
2. **What happens** if preconditions are violated?

Addressing Client Misuse

Who should decide…

1. **How much time** the library component should spend checking for preconditions?
2. **What happens** if preconditions are violated?

Should it be…

a. The (reusable) library component developer?

Addressing Client Misuse

Who should decide…

1. **How much time** the library component should spend checking for preconditions?
2. **What happens** if preconditions are violated?

Should it be…

a. The (reusable) library component developer?

b. The developer of the immediate client?

Addressing Client Misuse

Who should decide…

1. **How much time** the library component should spend checking for preconditions?
2. **What happens** if preconditions are violated?

Should it be…

a. The (reusable) library component developer?
b. The developer of the immediate client?
c. The owner of the application, who:

Addressing Client Misuse

Who should decide…

1. **How much time** the library component should spend checking for preconditions?
2. **What happens** if preconditions are violated?

Should it be…

a. The (reusable) library component developer?
b. The developer of the immediate client?
c. The owner of the application, who:
   i. Is responsible for building the application.

Addressing Client Misuse

Who should decide…

1. **How much time** the library component should spend checking for preconditions?
2. **What happens** if preconditions are violated?

Should it be…

a. The (reusable) library component developer?

b. The developer of the immediate client?

c. The owner of the application, who:

   i. Is responsible for building the application.

   ii. Owns `main`.

Addressing Client Misuse

Who should decide…

1. **How much time** the library component should spend checking for preconditions?
2. **What happens** if preconditions are violated?

It should be:

a. The (reusable) library component developer?

b. The developer of the immediate client?

c. The owner of the application, who:
   i. Is responsible for building the application.
   ii. Owns `main`.

Addressing Client Misuse

Who should decide…

1. **How much time** the library component should spend checking for preconditions?
2. **What happens** if preconditions are violated?

It should be:

a. The (reusable per?

b. The develope

See the **`bsls_assert`** component.

c. The owner of the application, who:

   i. Is responsible for building the application.

   ii. Owns `main`.

3. Verification & Testing

Addressing Client Misuse

CPU Usage for Checking

Specified at Compile Time

Behavior if Misuse is Detected

Specified at Runtime

c. The owner of the application, who:

   i. Is responsible for building the application.

   ii. Owns `main`.

Outline

0. Goals

   What we are trying to do, for whom, and how.

1. Process & Architecture

   Organizing Software as Components, Packages, & Package Groups.

2. Design & Implementation

   Using Class Categories, Value Semantics, & Vocabulary Types.

3. Verification & Testing

   Component-Level Test Drivers, Peer Review, & Defensive Checks.

4. Bloomberg Development Environment (BDE)

   Rendered as Fine-Grained *Hierarchically* *Reusable* Components.

Outline

0. Goals

    What we are trying to do, for whom, and how.

1. Process & Architecture

    Organizing Software as Components, Packages, & Package Groups.

2. Design & Implementation

    Using Class Categories, Value Semantics, & Vocabulary Types.

3. Verification & Testing

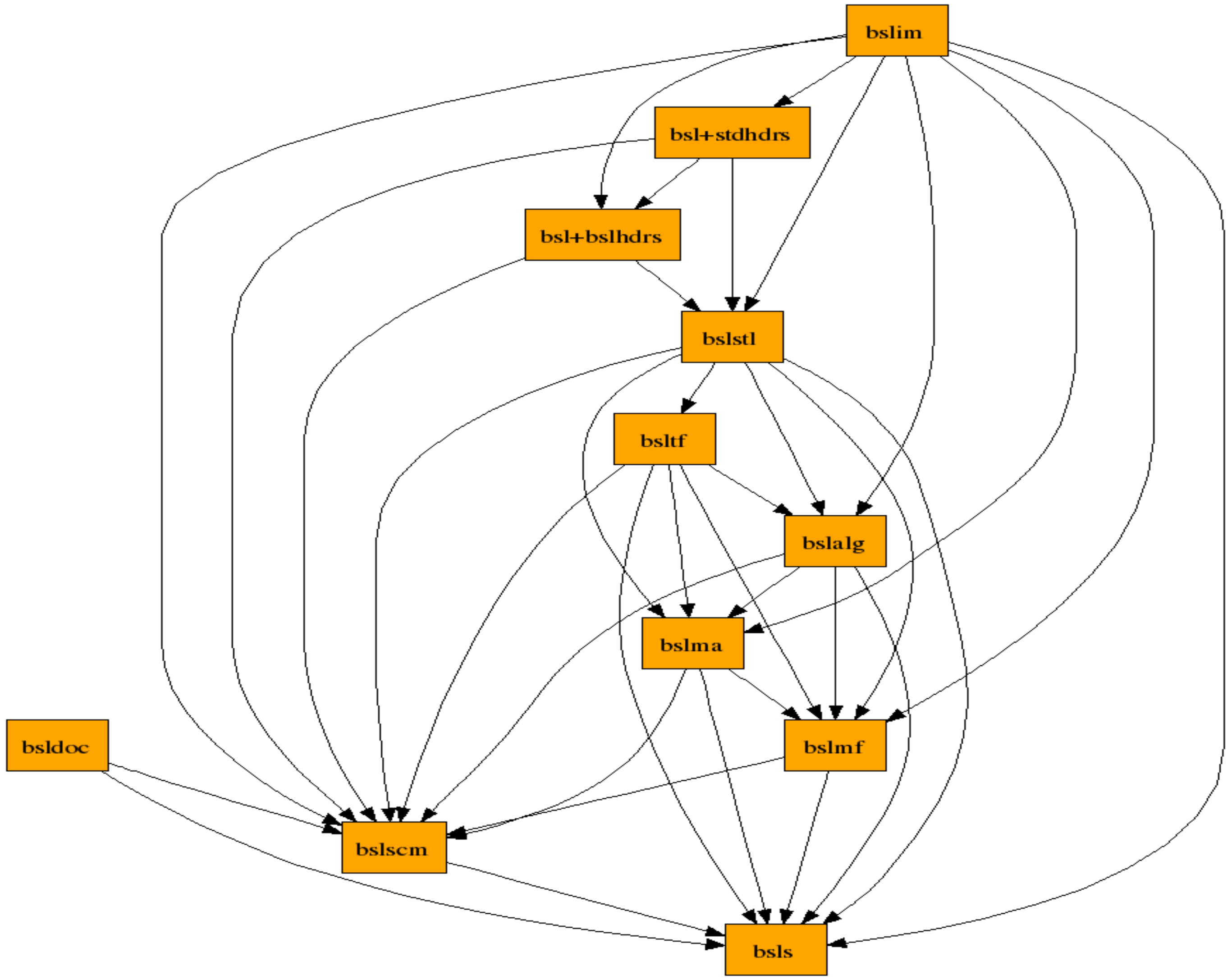
    Component-Level Test Drivers, Peer Review, & Defensive Checks.

4. Bloomberg Development Environment (BDE)

    Rendered as Fine-Grained *Hierarchically Reusable* Components.

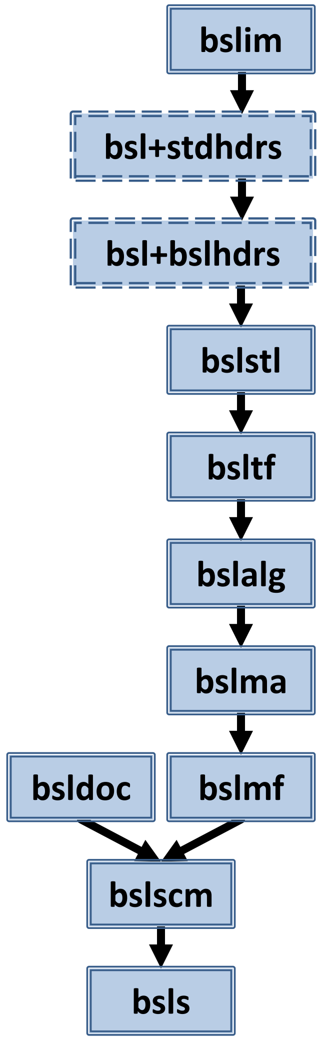
4. Bloomberg Development Environment

The BSL Package Group



4. Bloomberg Development Environment

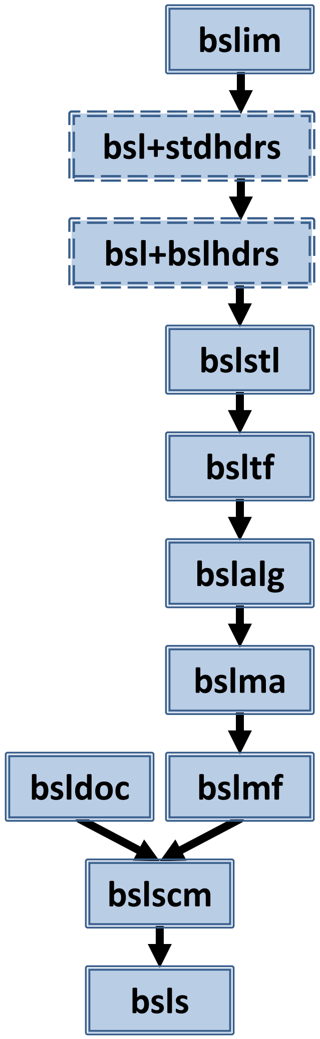
The BSL Package Group



4. Bloomberg Development Environment

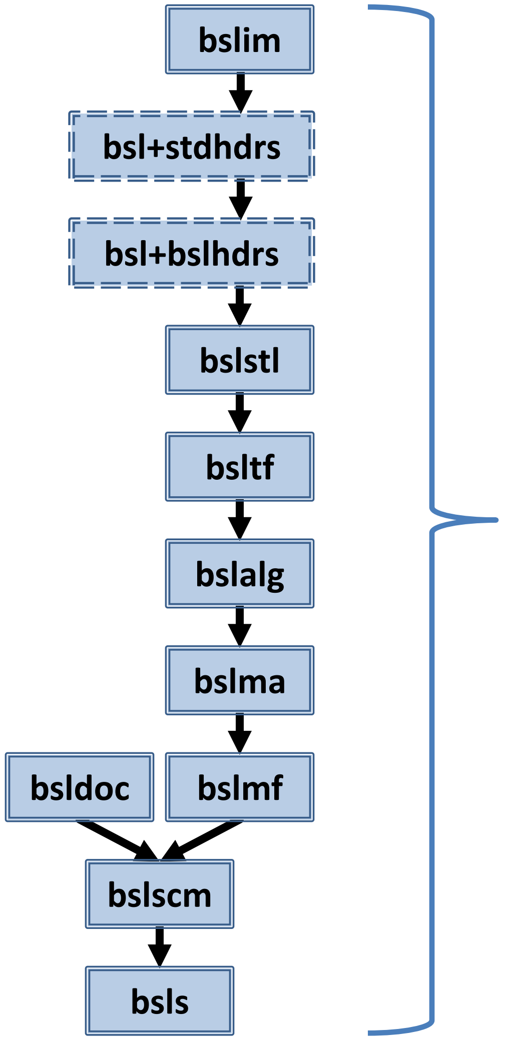
The BSL Package Group

11 Packages



The BSL Package Group

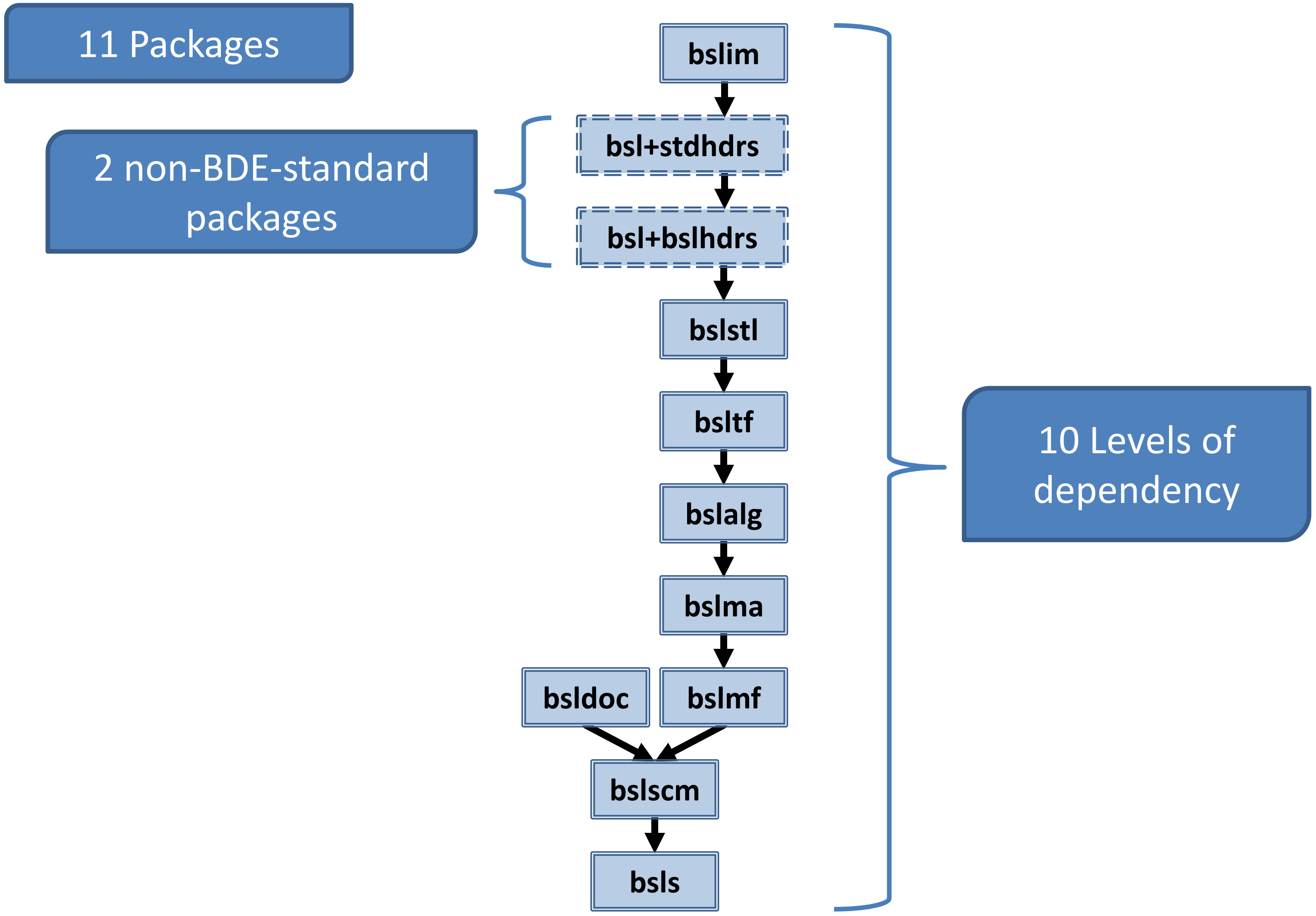
11 Packages



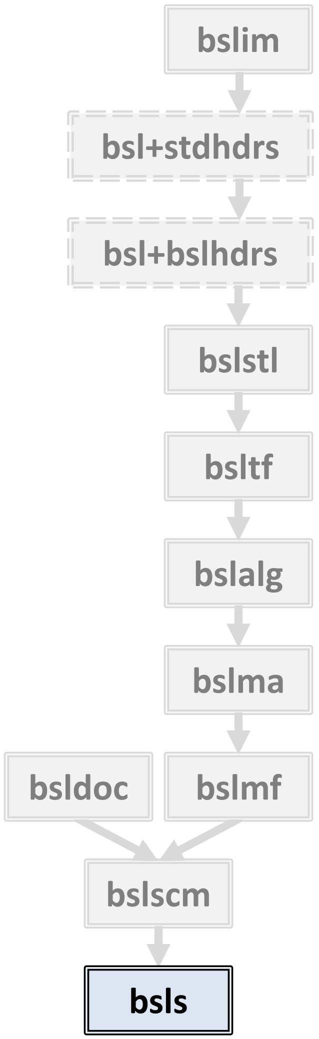
10 Levels of dependency

4. Bloomberg Development Environment

The BSL Package Group



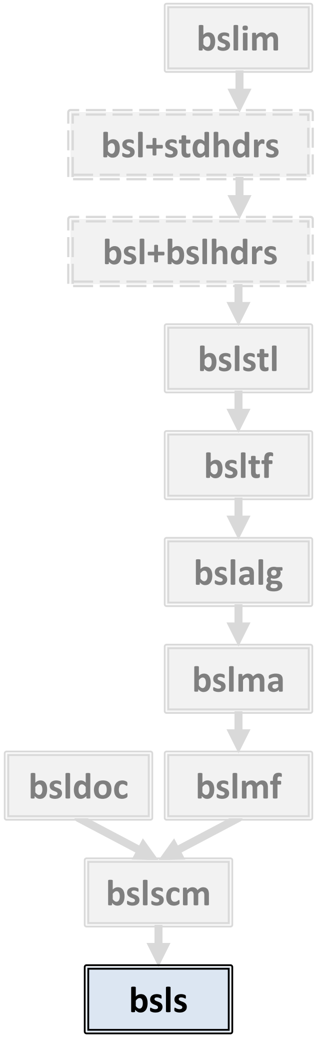
Package `bsls`



- **S**ystem utilities

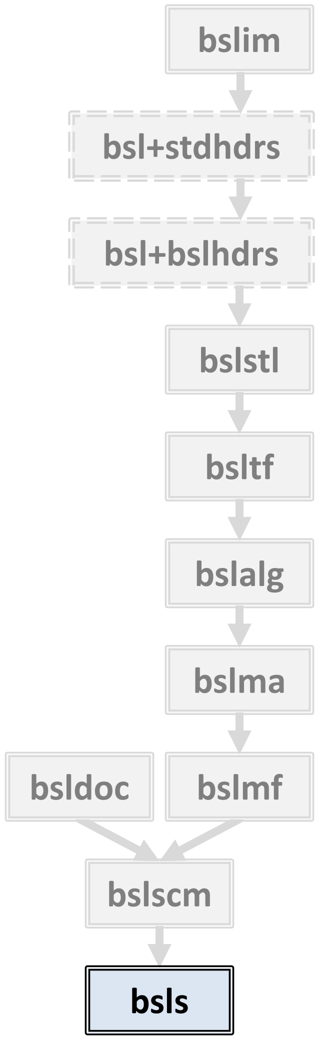
4. Bloomberg Development Environment

Package `bsls`



- **S**ystem utilities
- Provides uniform handling of:
  - alignment, endian-ness, integer sizes, …
  - clocks, atomic ops, and other system facilities

Package **`bsls`**



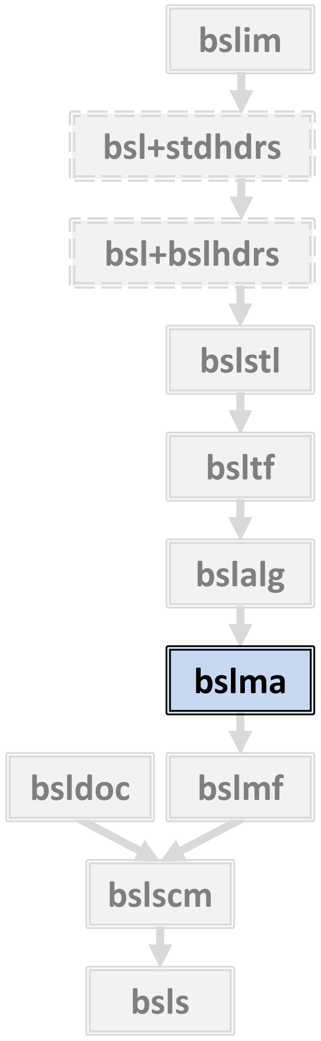
- System utilities
- Provides uniform handling of:
  - alignment, endian-ness, integer sizes, …
  - clocks, atomic ops, and other system facilities
- Support for BDE methodology: e.g.,
  - **`bsls_bsltestutil`**
  - **`BSLS_ASSERT*`** macros

Package **bsls**

bsls_alignedbuffer	Provide raw buffers with user-specified size and alignment.
bsls_alignmentfromtype	Provide a meta-function that maps a `TYPE` to its alignment.
bsls_alignment	Provide a namespace for enumerating memory alignment strategies.
bsls_alignmentimp	Provide implementation meta-functions for alignment computation.
bsls_alignmenttotype	Provide a meta-function mapping an `ALIGNMENT` to a primitive type.
bsls_alignmentutil	Provide constants, types, and operations related to alignment.
bsls_annotation	Provide support for compiler annotations for compile-time safety.
bsls_assert	Provide build-specific, runtime-configurable assertion macros.
bsls_asserttestexception	Provide an exception type to support testing for failed assertions.
bsls_asserttest	Provide a test facility for assertion macros.
bsls_atomic	Provide types with atomic operations.
bsls_atomicoperations	Provide platform-independent atomic operations.
bsls_blockgrowth	Provide a namespace for memory block growth strategies.
bsls_bsltestutil	Provide test utilities for `bsl` that do not use `<iostream>`.
bsls_buildtarget	Provide build-target information in the object file.
bsls_byteorder	Provide byte-order manipulation macros.
bsls_compilerfeatures	Provide macros to identify compiler support for C++11 features.
bsls_exceptionutil	Provide simplified exception constructs for non-exception builds.
bsls_ident	Provide macros for inserting SCM Ids into source files.
bsls_macroincrement	Provide a macro to increment preprocessor numbers.
bsls_nativestd	Define the namespace `native_std` as an alias for `::std`.
bsls_nullptr	Provide a distinct type for null pointer literals.
bsls_objectbuffer	Provide raw buffer with size and alignment of user-specified type.
bsls_performancehint	Provide performance hints for code optimization.
bsls_platform	Provide compile-time support for platform/attribute identification.
bsls_protocoltest	Provide classes and macros for testing abstract protocols.
bsls_stopwatch	Provide access to user, system, and wall times of current process.
bsls_timeutil	Provide a platform-neutral functional interface to system clocks.
bsls_types	Provide a consistent interface for platform-dependent types.
bsls_unspecifiedbool	Provide a class supporting the "unspecified `bool`" idiom.
bsls_util	Provide essential, low-level support for portable generic code.

4. Bloomberg Development Environment

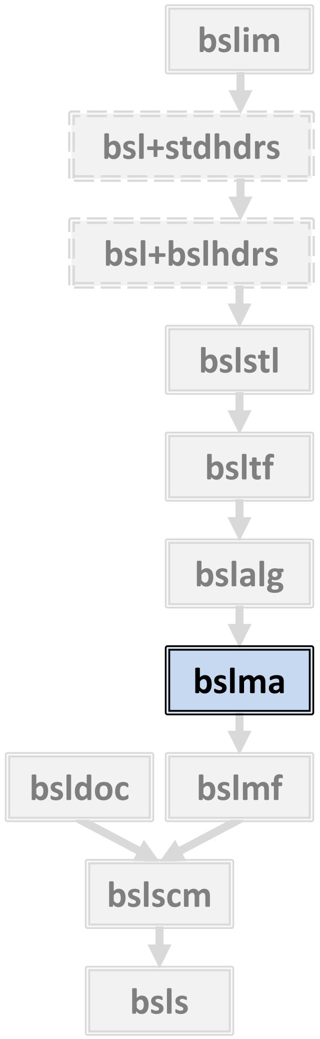
Package `bslma`



- **M**emory **A**llocators

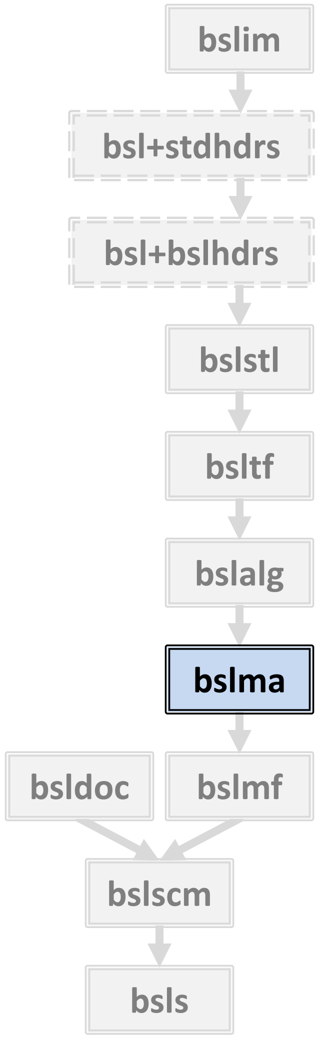
4. Bloomberg Development Environment

Package **`bslma`**



- **M**emory **A**llocators
- Allocator protocol: **`bslma_allocator`**

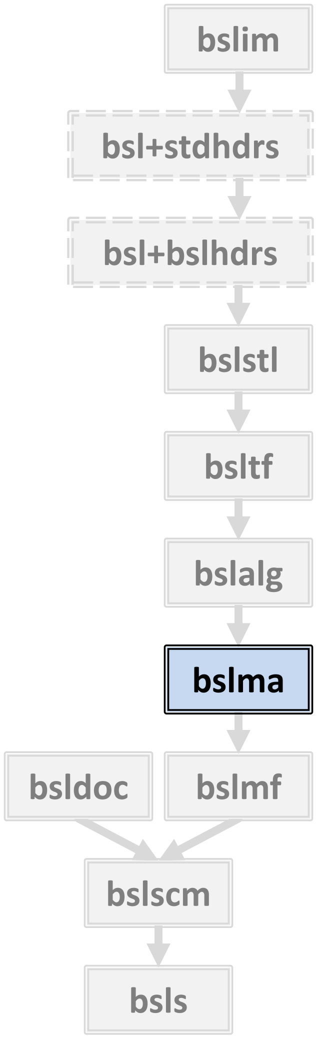
Package **bslma**



- **M**emory **A**llocators
- Allocator protocol: **`bslma_allocator`** } Quintessential Vocabulary Type

4. Bloomberg Development Environment

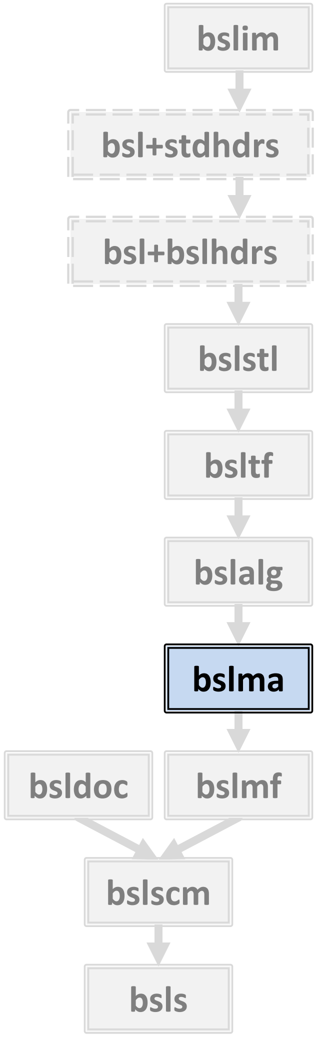
Package **bslma**



- **M**emory **A**llocators
- Allocator protocol: **`bslma_allocator`**
- Mechanisms

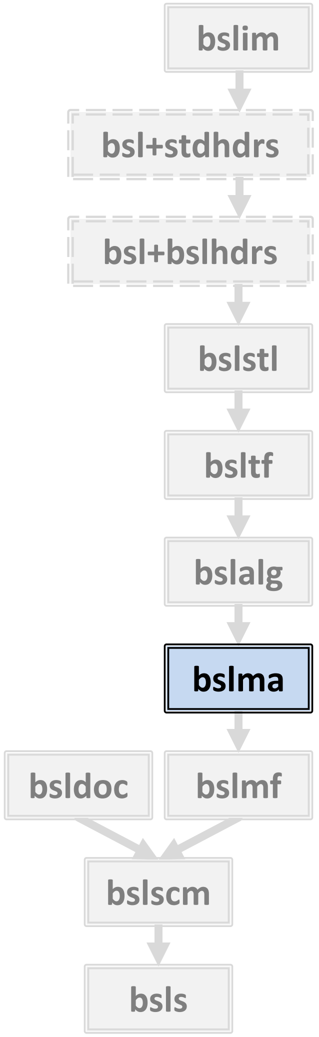
Quintessential Vocabulary Type

Package `bslma`



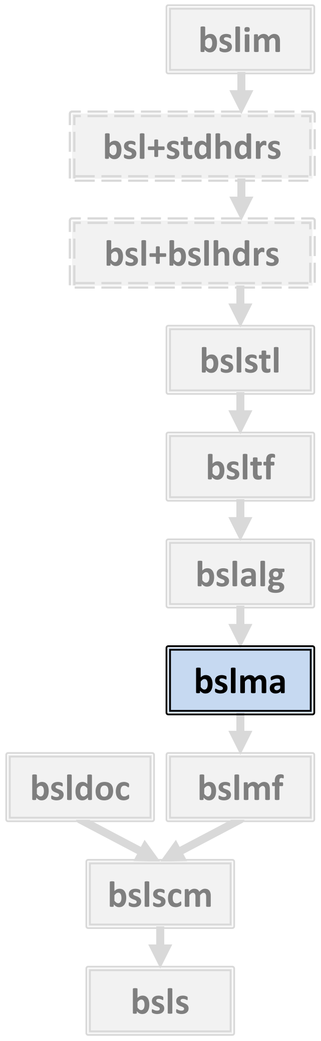
- **M**emory **A**llocators
- Allocator protocol: **`bslma_allocator`** } Quintessential Vocabulary Type
- Mechanisms
  - The default default-allocator, **`bslma_newdeleteallocator`**

Package **`bslma`**



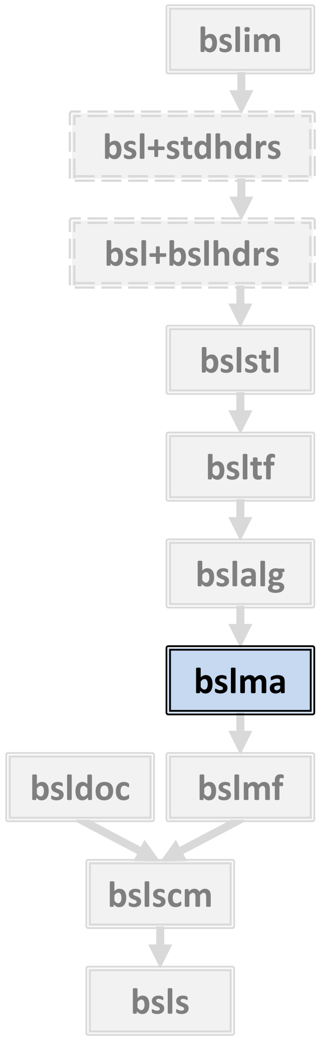
- **M**emory **A**llocators
- Allocator protocol: **`bslma_allocator`** } Quintessential Vocabulary Type
- Mechanisms
  - The default default-allocator, **`bslma_newdeleteallocator`**
  - Managing the default, **`bslma_default`**

Package **`bslma`**



- **M**emory **A**llocators
- Allocator protocol: **`bslma_allocator`** } Quintessential Vocabulary Type
- Mechanisms
  - The default default-allocator, **`bslma_newdeleteallocator`**
  - Managing the default, **`bslma_default`**
  - Development, **`bslma_testallocator`**

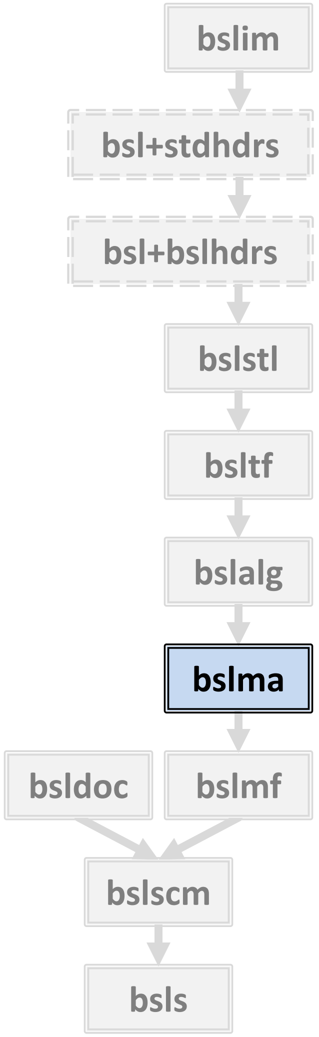
Package `bslma`



- **M**emory **A**llocators
- Allocator protocol: **`bslma_allocator`** } Quintessential Vocabulary Type
- Mechanisms
  – The default default-allocator, **`bslma_newdeleteallocator`**
  – Managing the default, **`bslma_default`**
  – Development, **`bslma_testallocator`**
- Guards and proctors for single objects and ranges

4. Bloomberg Development Environment

Package **`bslma`**



- **M**emory **A**llocators
- Allocator protocol: **`bslma_allocator`**

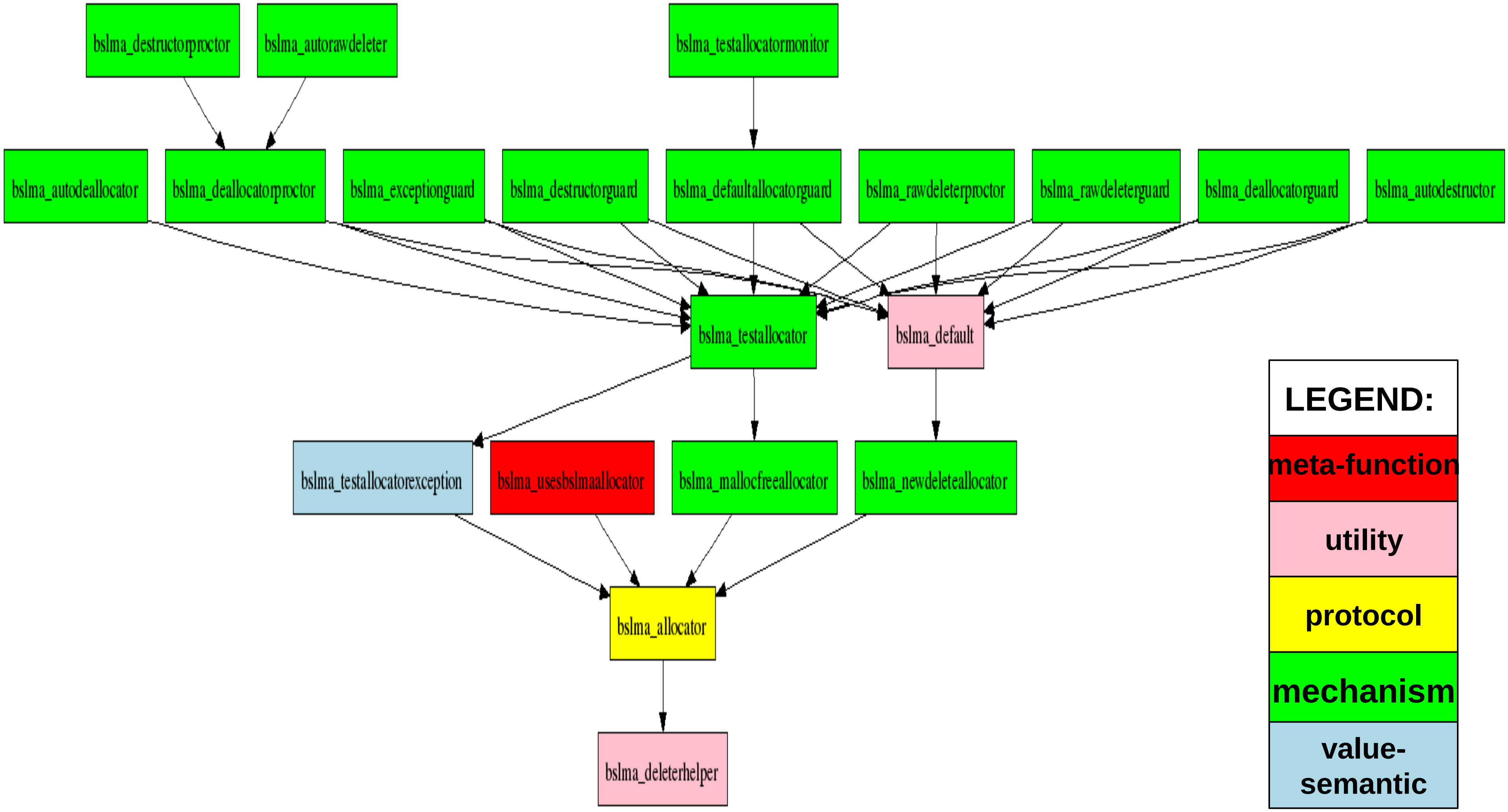
Quintessential Vocabulary Type

- Mechanisms
  - The default default-allocator, **`bslma_newdeleteallocator`**
  - Managing the default, **`bslma_default`**
  - Development, **`bslma_testallocator`**
- Guards and proctors for single objects and ranges

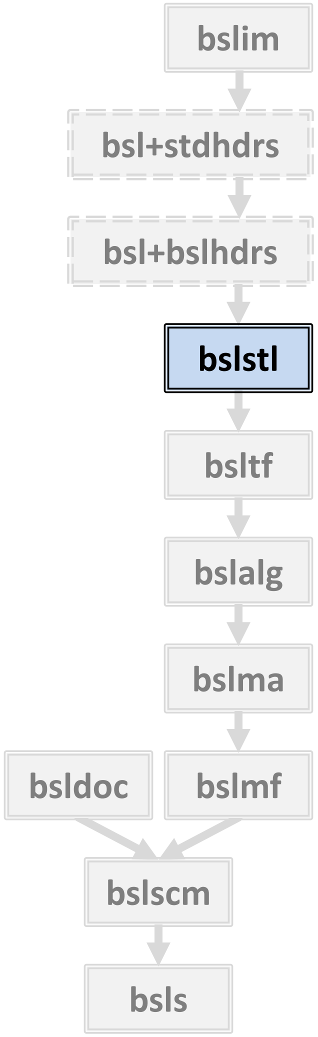
Have **`release`** method

4. Bloomberg Development Environment

Package `bslma`

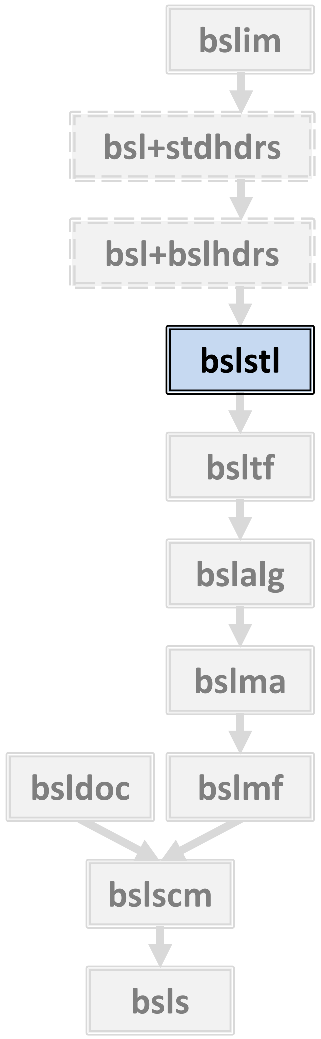


Package `bslstl`



- **STL**

Package **`bslstl`**

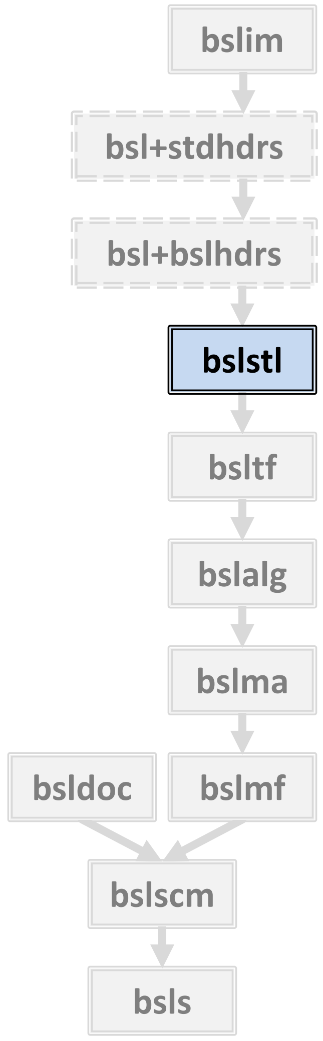


- **STL**
- C++ Standard library from BDE allows
  - Standard allocators, *and*
  - BDE runtime polymorphic allocators

  for allocator-aware types (e.g., **`vector`**, **`list`**, **`unordered_map`**)

4. Bloomberg Development Environment

Package `bslstl`

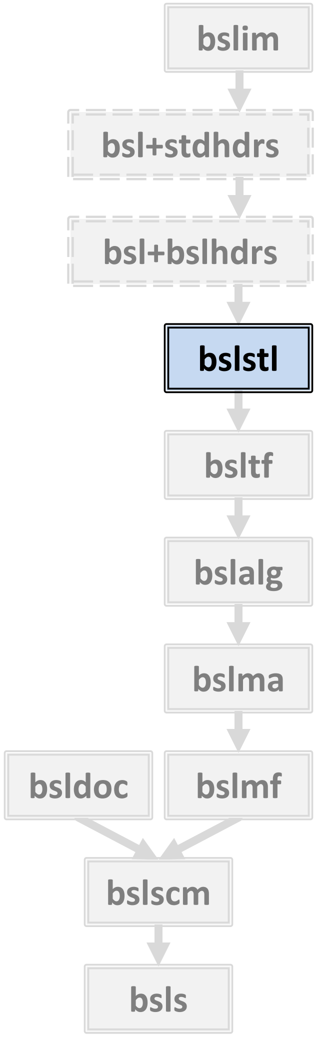


- **STL**
- C++ Standard library from BDE allows
  - Standard allocators, *and*
  - BDE runtime polymorphic allocators

  for allocator-aware types (e.g., **`vector`**, **`list`**, **`unordered_map`**)
- Non-allocator facilities pass through to native library

4. Bloomberg Development Environment

Package **`bslstl`**



- **STL**
- C++ Standard library from BDE allows
  - Standard allocators, *and*
  - BDE runtime polymorphic allocators

  for allocator-aware types (e.g., **`vector`**, **`list`**, **`unordered_map`**)
- Non-allocator facilities pass through to native library
- Used via **`bsl+bslhdrs`** (not directly)

4. Bloomberg Development Environment
Our Open Source Distribution

How do you find what you need?

Our Open Source Distribution

How do you find what you need?

- BDE group, package, and component-level doc converted to **`doxygen`** markup.

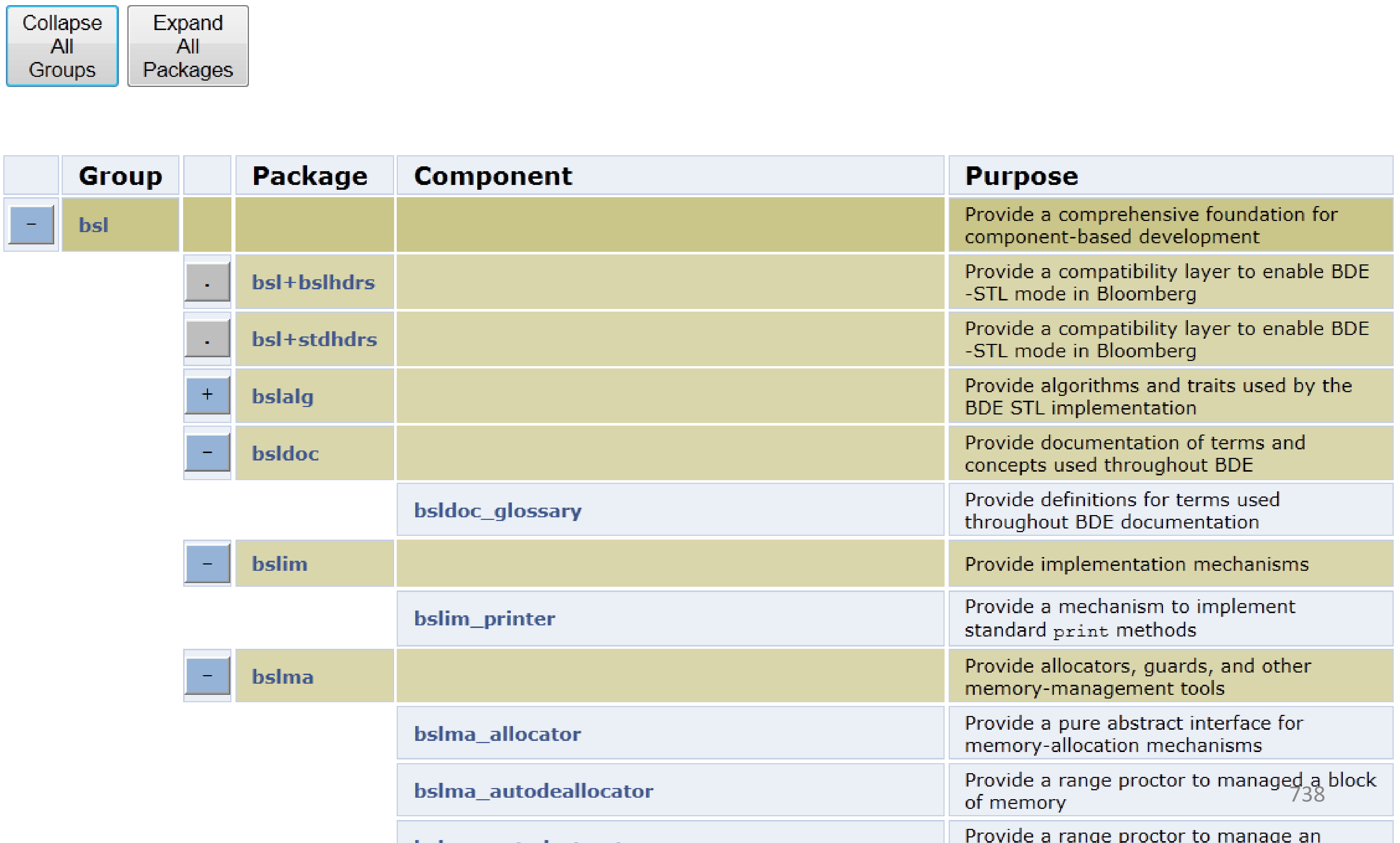
Our Open Source Distribution

How do you find what you need?

- BDE group, package, and component-level doc converted to **`doxygen`** markup.
- Hierarchically organized home page provides overview of all components.

4. Bloomberg Development Environment

Our Open Source Distribution



	Group		Package	Component	Purpose
−	bsl				Provide a comprehensive foundation for component-based development
		.	bsl+bslhdrs		Provide a compatibility layer to enable BDE -STL mode in Bloomberg
		.	bsl+stdhdrs		Provide a compatibility layer to enable BDE -STL mode in Bloomberg
		+	bslalg		Provide algorithms and traits used by the BDE STL implementation
		−	bsldoc		Provide documentation of terms and concepts used throughout BDE
				bsldoc_glossary	Provide definitions for terms used throughout BDE documentation
		−	bslim		Provide implementation mechanisms
				bslim_printer	Provide a mechanism to implement standard `print` methods
		−	bslma		Provide allocators, guards, and other memory-management tools
				bslma_allocator	Provide a pure abstract interface for memory-allocation mechanisms
				bslma_autodeallocator	Provide a range proctor to managed a block of memory
				[illegible]	Provide a range proctor to manage an

4. Bloomberg Development Environment
Our Open Source Distribution

- What License Applies?

4. Bloomberg Development Environment
Our Open Source Distribution

- What License Applies?
  - Software License: MIT

4. Bloomberg Development Environment

Our Open Source Distribution

- What License Applies?
  - Software License: MIT
  - Deliberately liberal

4. Bloomberg Development Environment

Our Open Source Distribution

- What License Applies?
  - Software License: MIT
  - Deliberately liberal
  - We intend that anyone can use our software freely

4. Bloomberg Development Environment

Our Open Source Distribution

- What License Applies?
  - Software License: MIT
  - Deliberately liberal
  - We intend that anyone can use our software freely
    - for any legitimate purpose

Our Open Source Distribution

- What License Applies?
  - Software License: MIT
  - Deliberately liberal
  - We intend that anyone can use our software freely
    - for any legitimate purpose
    - including as part of a product for sale

4. Bloomberg Development Environment

Our Open Source Distribution

- Find our open-source distribution at: **`http://www.openbloomberg.com/bsl`**
- Moderator: **`kpfleming@bloomberg.net`**
- How to contribute? *See our site.*
- All comments and criticisms welcome...

Our Open Source Distribution

- Find our open-source distribution at: **`http://www.openbloomberg.com/bsl`**
- Moderator: **`kpfleming@bloomberg.net`**
- How to contribute? *See our site.*
- All comments and criticisms welcome…

We will come back to this…

4. Bloomberg Development Environment

Moving Upward and Onward

Beyond BSL:

4. Bloomberg Development Environment

Moving Upward and Onward

Beyond BSL:

- The `Allocator` protocol is defined in **`bslma`**

4. Bloomberg Development Environment

Moving Upward and Onward

Beyond BSL:

- The `Allocator` protocol is defined in **`bslma`**
- Most concrete allocators reside above **`bsl`**

Moving Upward and Onward

Beyond BSL:

- The `Allocator` protocol is defined in **`bslma`**
- Most concrete allocators reside above **`bsl`**
- Some will be in **`bdlma`** (when released)

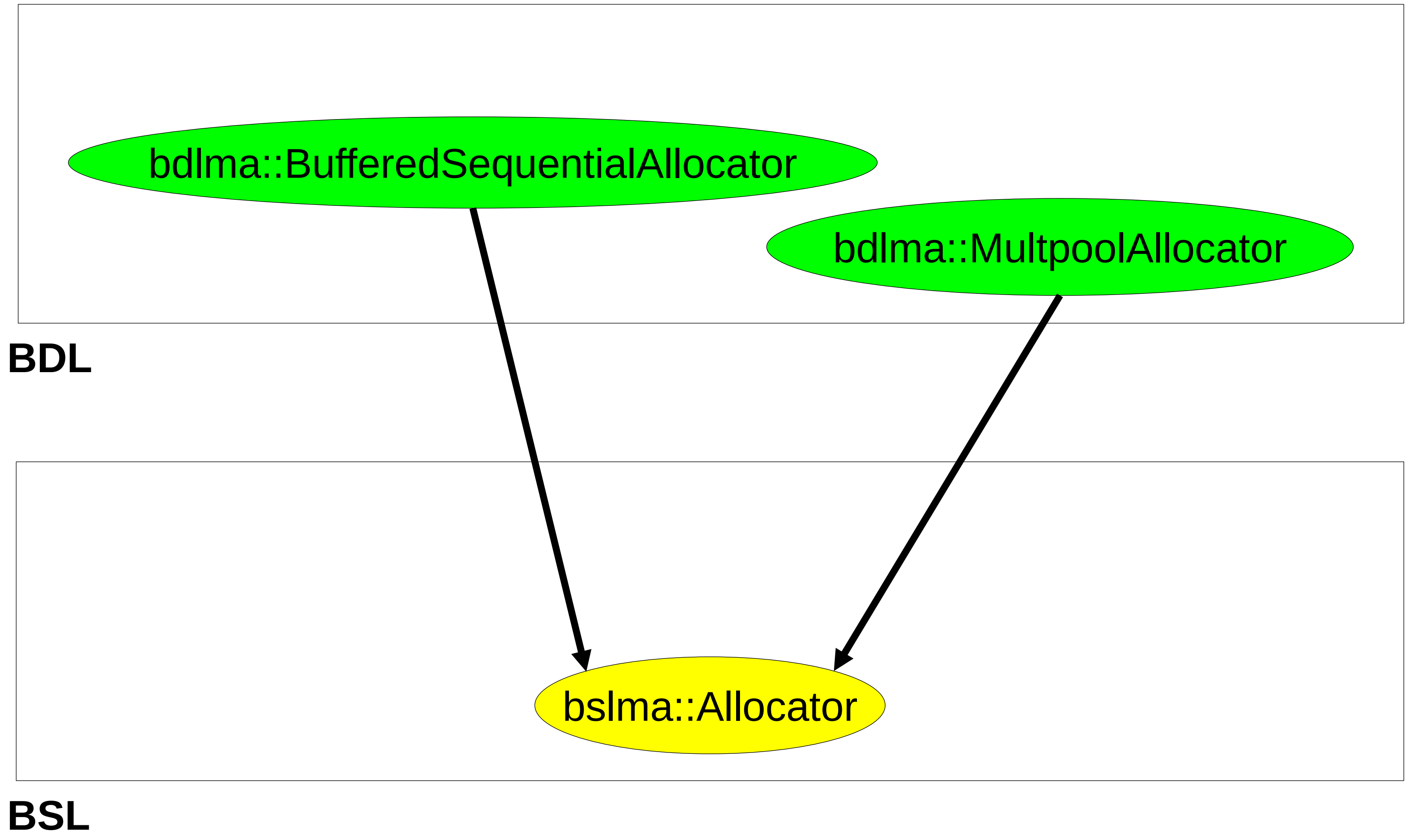
Moving Upward and Onward

Beyond BSL:

- The `Allocator` protocol is defined in **`bslma`**
- Most concrete allocators reside above **`bsl`**
- Some will be in **`bdlma`** (when released)
- Examples:
  - **Buffered Sequential Allocator**
  - **Multipool Allocator**

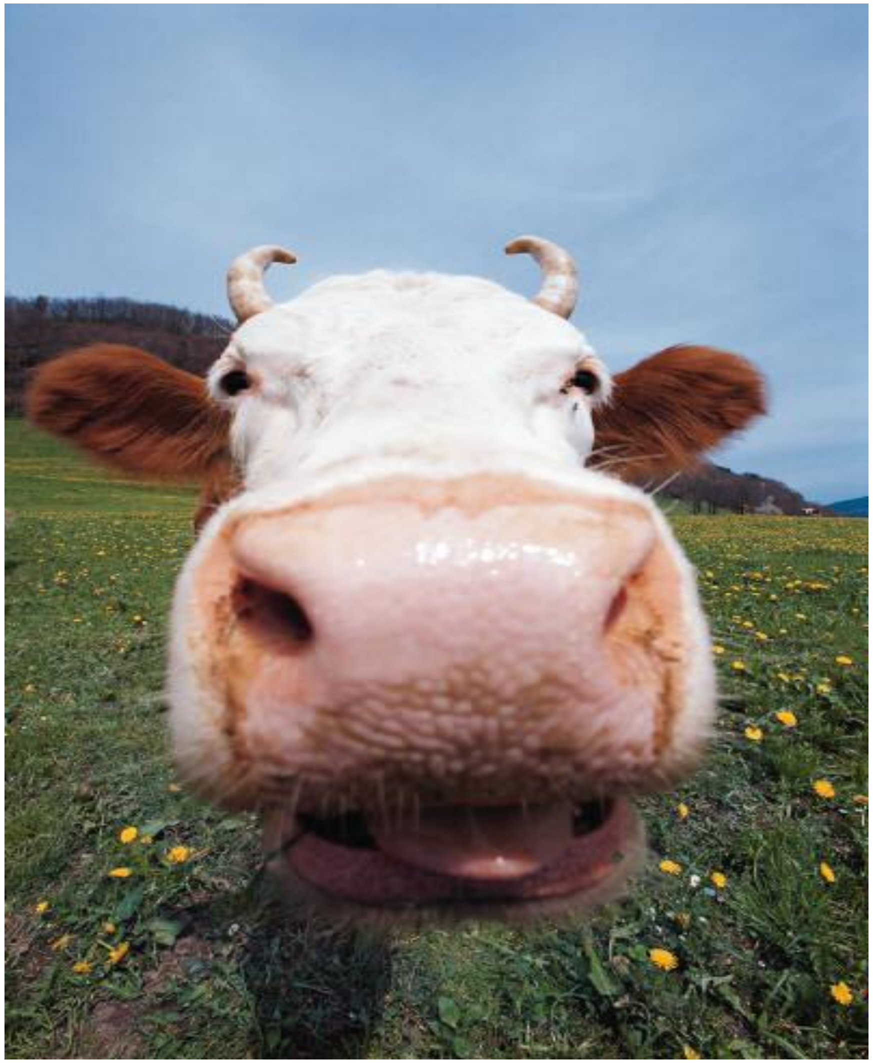
4. Bloomberg Development Environment

Moving Upward and Onward



4. Bloomberg Development Environment

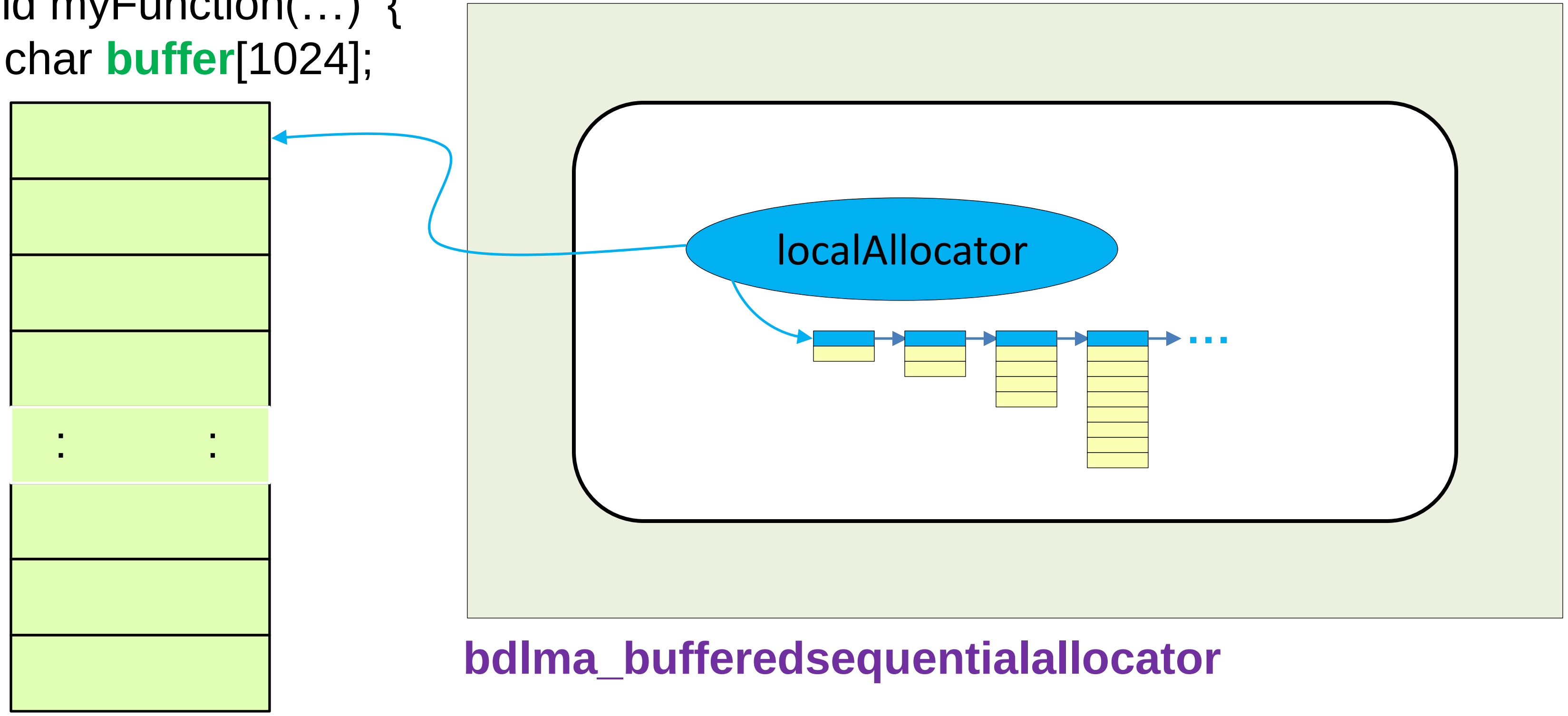
Buffered Sequential Allocator



4. Bloomberg Development Environment

Buffered Sequential Allocator

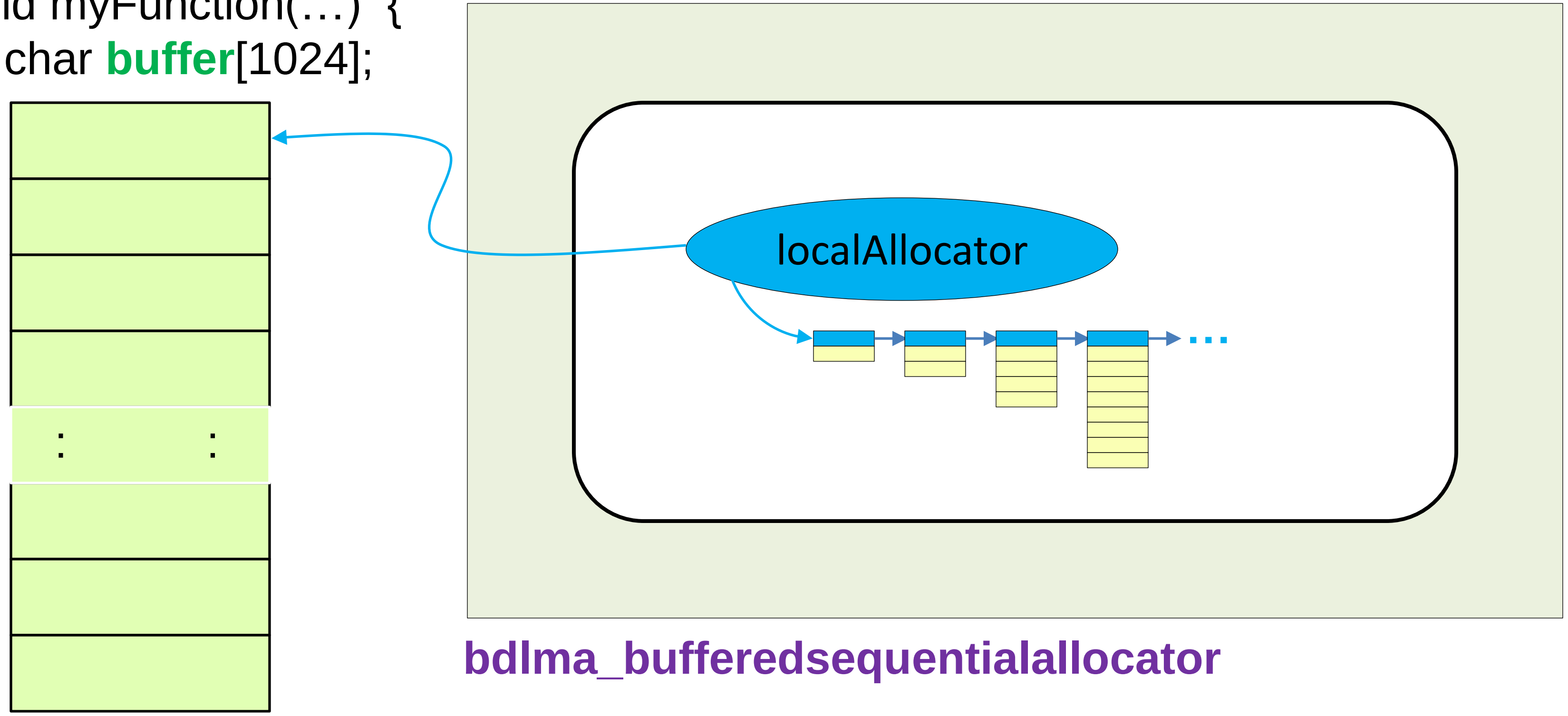
```
void myFunction(…) {
    char buffer[1024];
```



```
    bdlma::BufferedSequentialAllocator local Allocator(buffer, sizeof buffer);
    bsl::vector(&local Allocator);
     // …
}
```

Buffered Sequential Allocator

```
void myFunction(…)  {
   char buffer[1024];
```



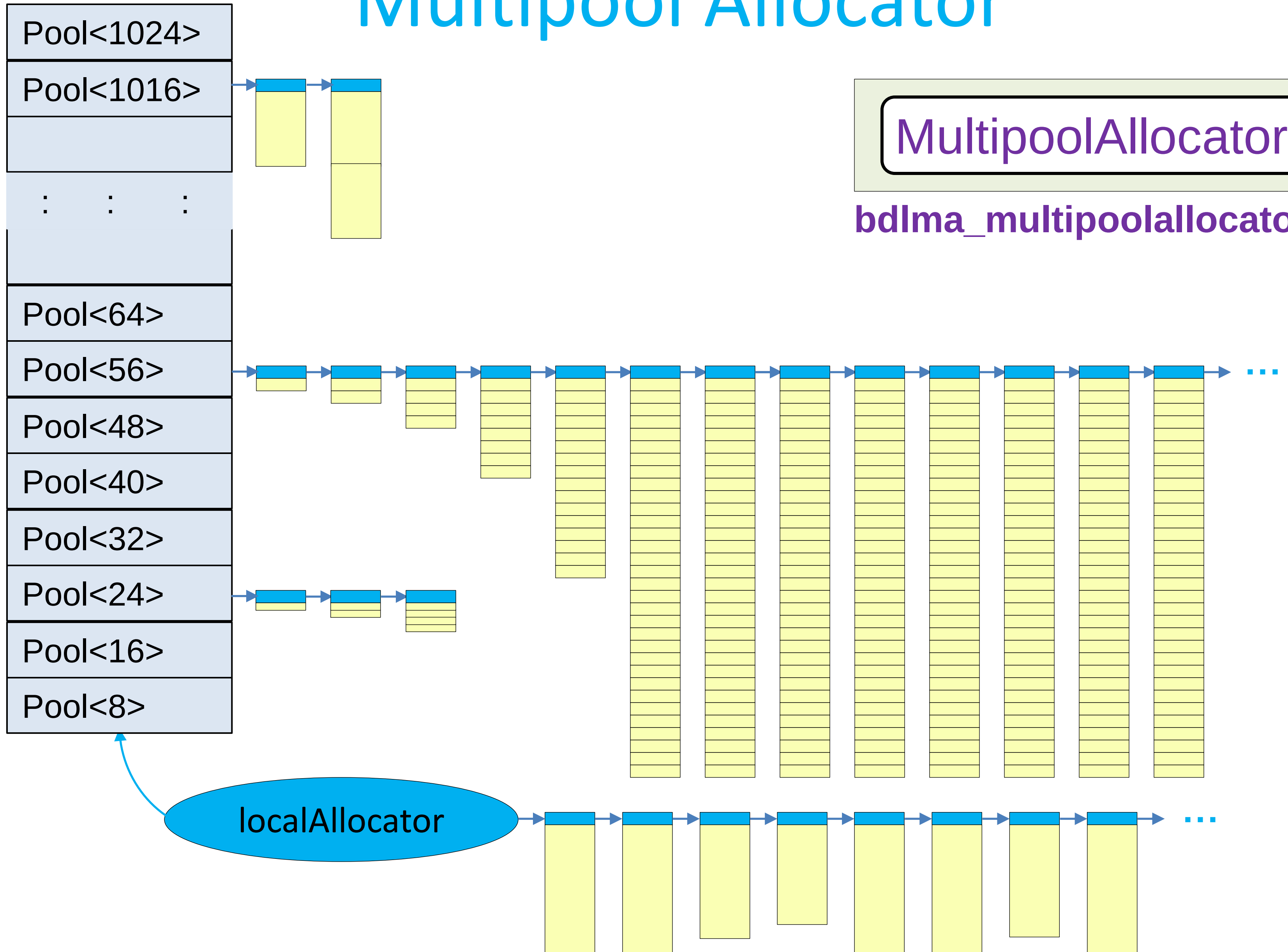
```
   bdlma::BufferedSequentialAllocator local Allocator(buffer, sizeof buffer);
   bsl::vector(&local Allocator);
    // …
}
```

Note that deallocate is a No-Op!

4. Bloomberg Development Environment

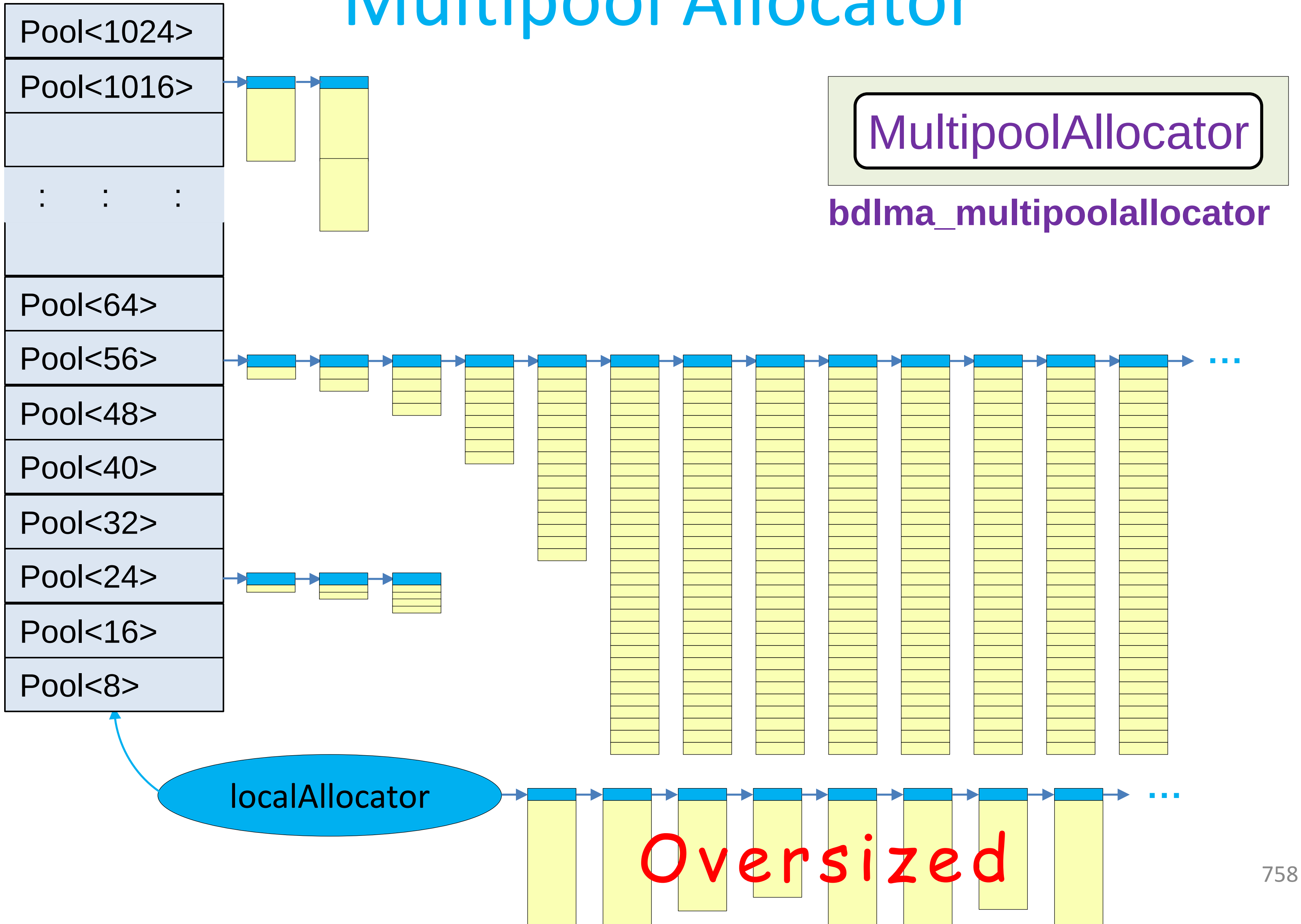
Multipool Allocator

Multipool Allocator



bdlma_multipoolallocator

Multipool Allocator



A Business Request

Suppose you are asked to provide some business functionality:

"Write me a ’Date’ class that tells me whether today is a business day."

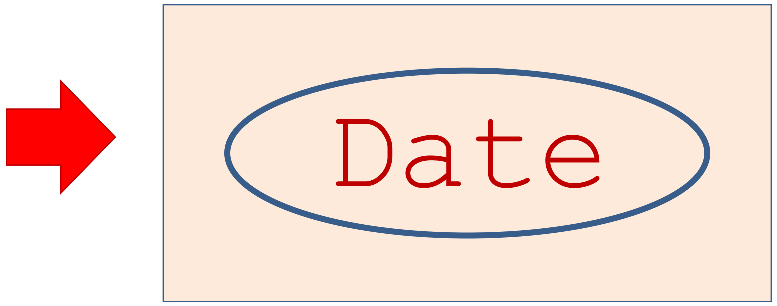
4. Bloomberg Development Environment

What’s the Problem?

"Write me a 'Date' class that tells me whether **today** is a **business day**."

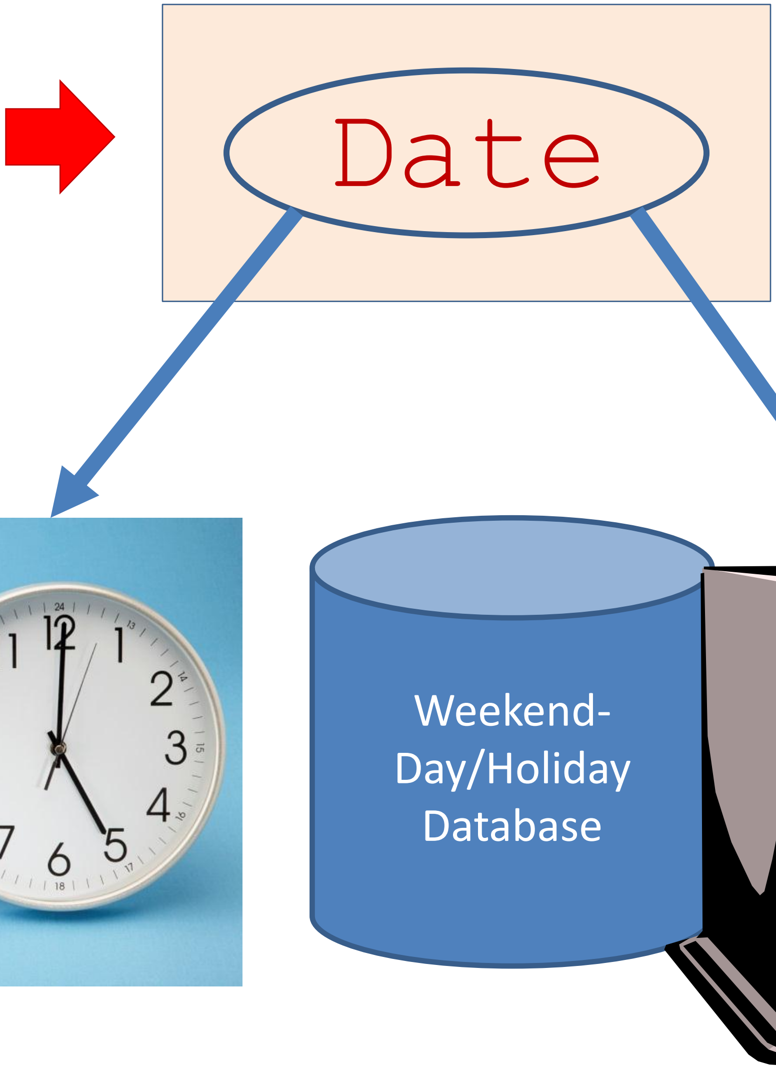
What's the Problem?

"Write me a 'Date' class that tells me whether **today** is a **business day**."



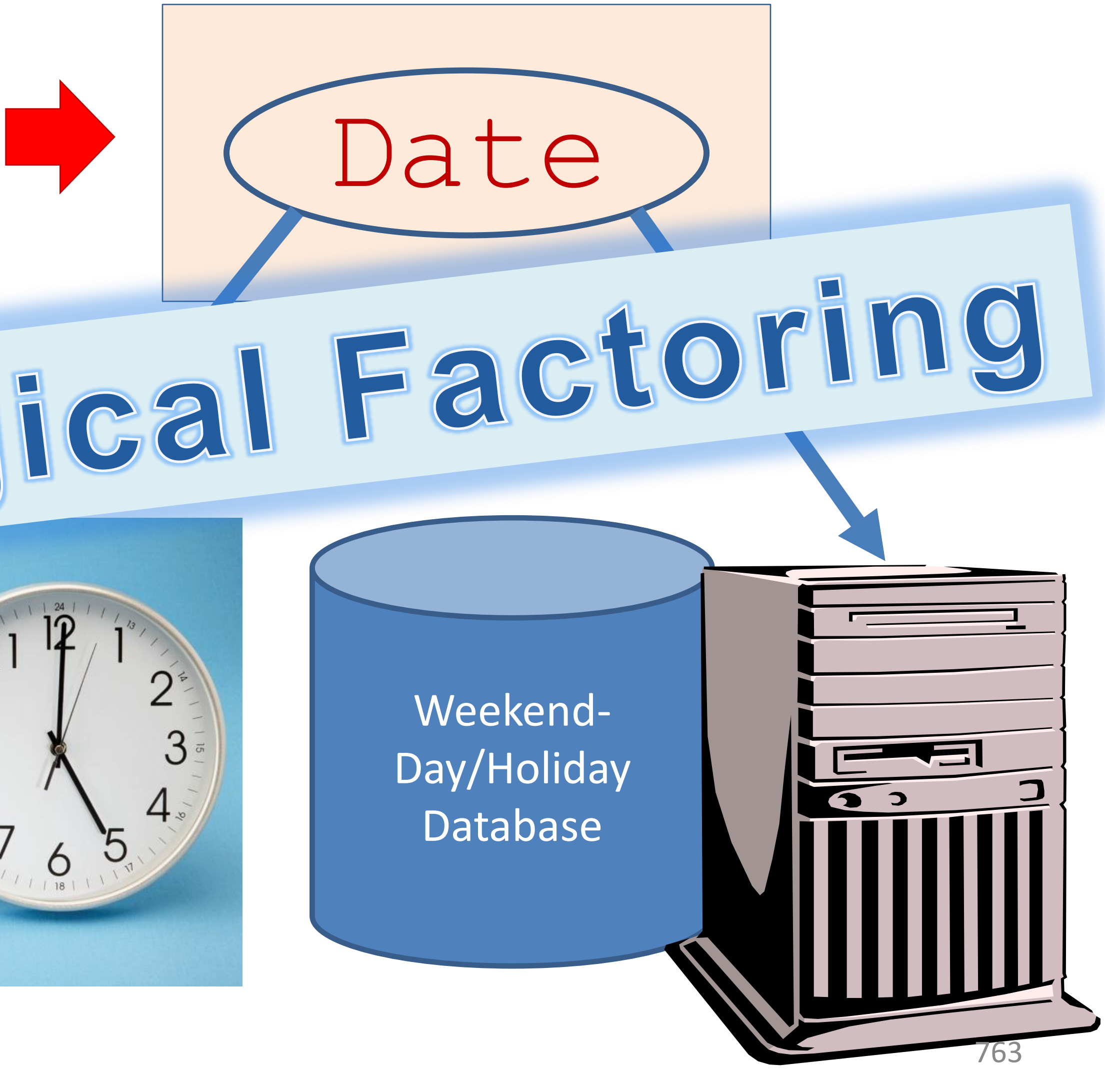
What’s the Problem?

"Write me a 'Date' class that tells me whether **today** is a **business day**."



What's the Problem?

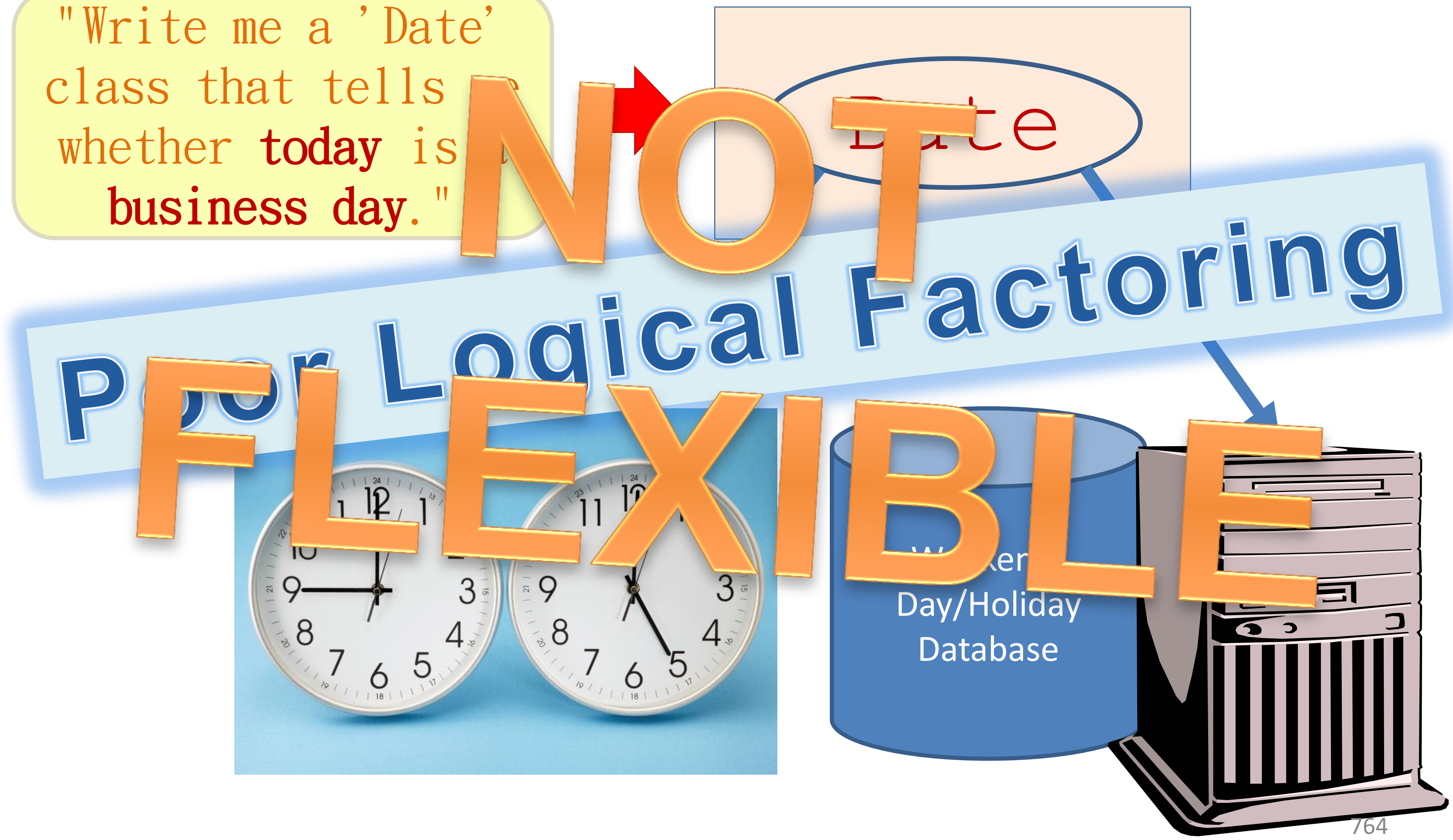
"Write me a 'Date' class that tells me whether **today** is a **business day**."



Poor Logical Factoring

What's the Problem?

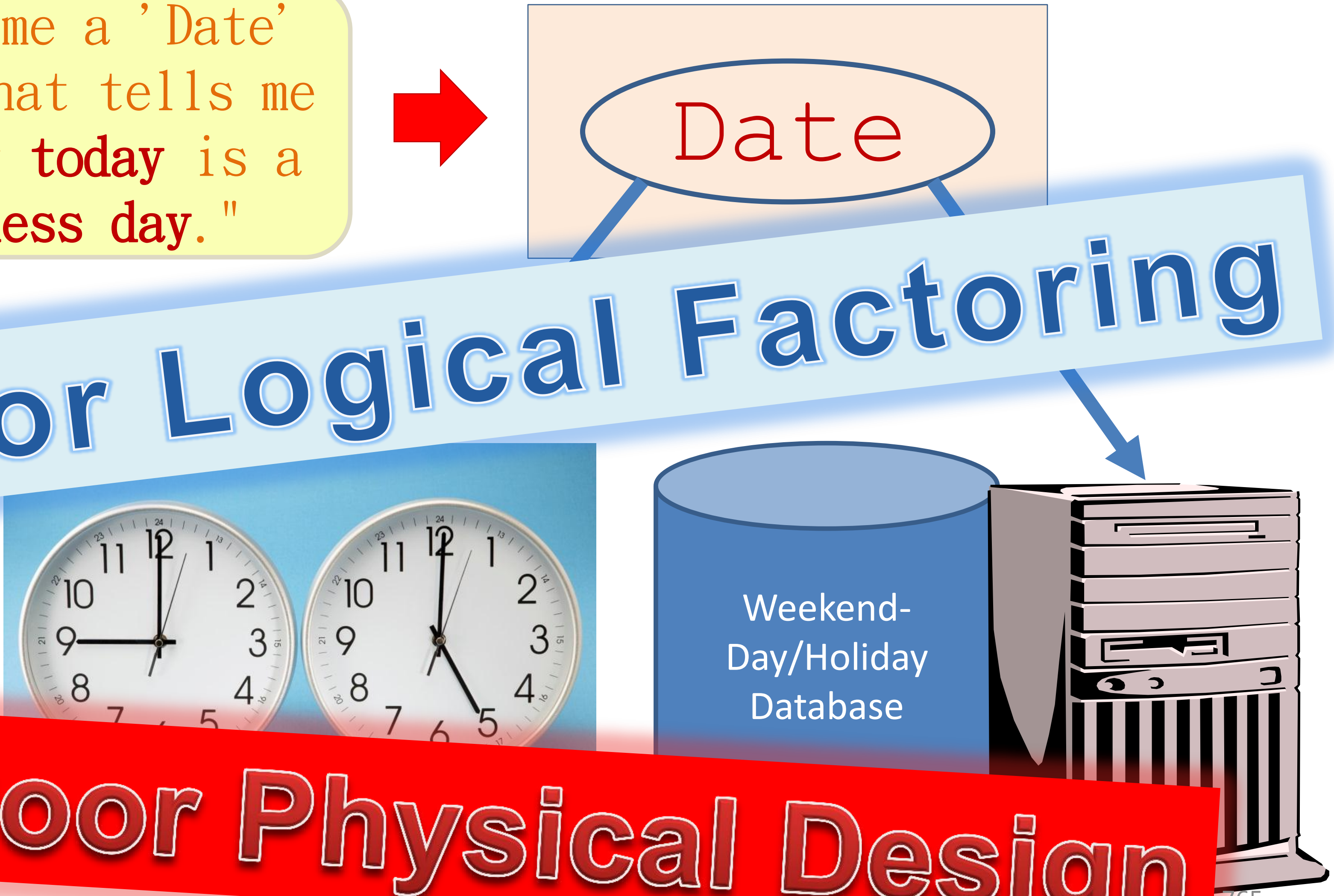
"Write me a 'Date' class that tells me whether **today** is a **business day.**"



4. Bloomberg Development Environment

What's the Problem?

"Write me a 'Date' class that tells me whether **today** is a **business day**."

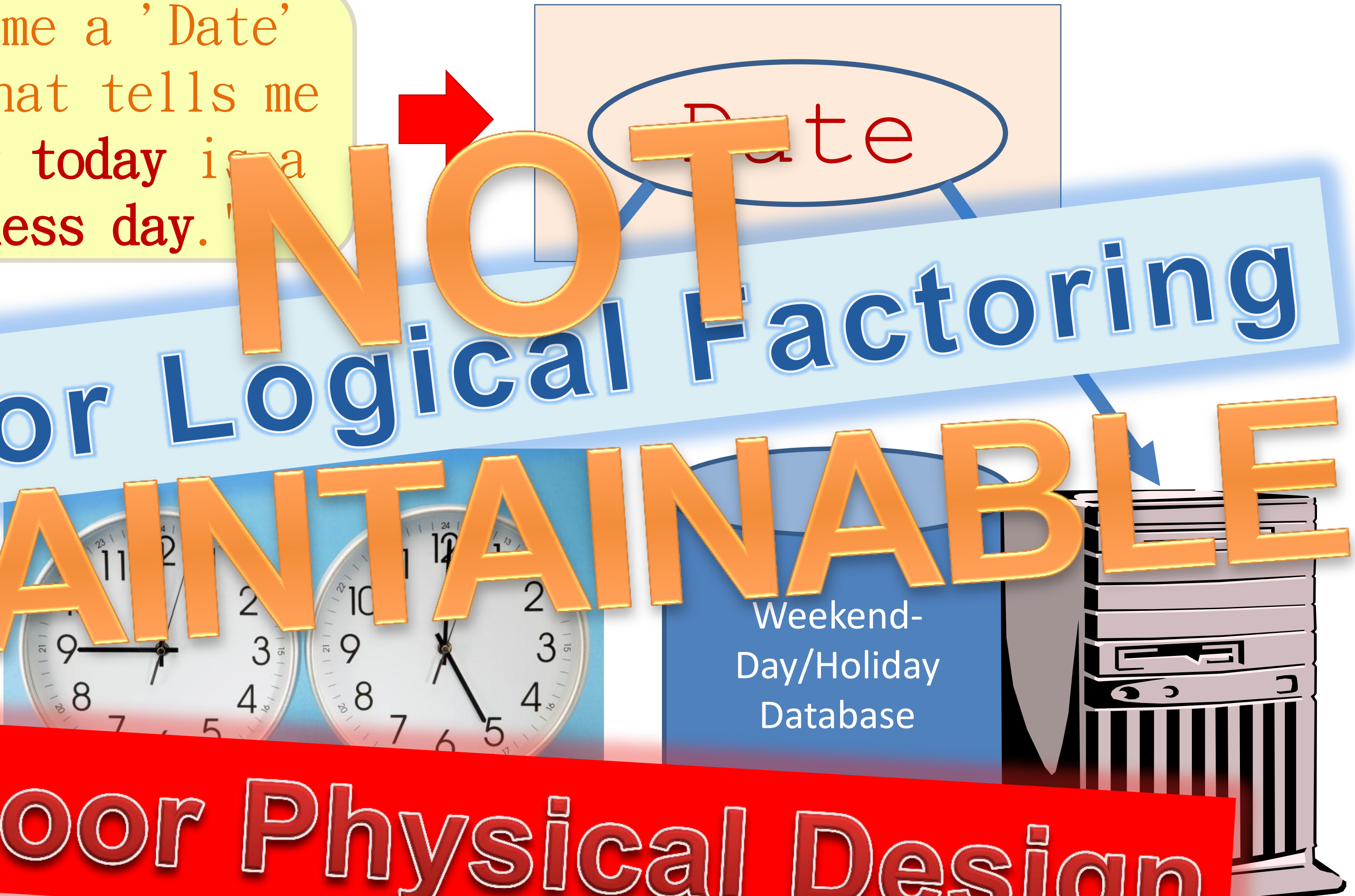


Poor Logical Factoring

Poor Physical Design

What’s the Problem?

"Write me a 'Date' class that tells me whether **today** is a **business day**."



The Original Request

"Write me a 'Date' class that tells me whether today is a business day."

What are the *real* requirements?

The Original Request

"Write me a 'Date' class that tells me whether today is a business day."

What are the *real* requirements?

1. Represent a *date value* as a C++ Type.

The Original Request

"Write me a 'Date' class that tells me whether today is a business day."

What are the *real* requirements?

1. Represent a *date value* as a C++ Type.
2. Determine what date value *today* is.

4. Bloomberg Development Environment

The Original Request

"Write me a 'Date' class that tells me whether today is a business day."

What are the *real* requirements?

1. Represent a *date value* as a C++ Type.
2. Determine what date value *today* is.
3. Determine if a date value is a *business day*.

The Original Request

"Write me a 'Date' class that tells me whether today is a business day."

What are the *real* requirements?

1. Represent a *date value* as a C++ Type.
2. Determine what date value *today* is.
3. Determine if a date value is a *business day*.
4. **Provide well-factored useful components that we'll need over and over again!**

4. Bloomberg Development Environment

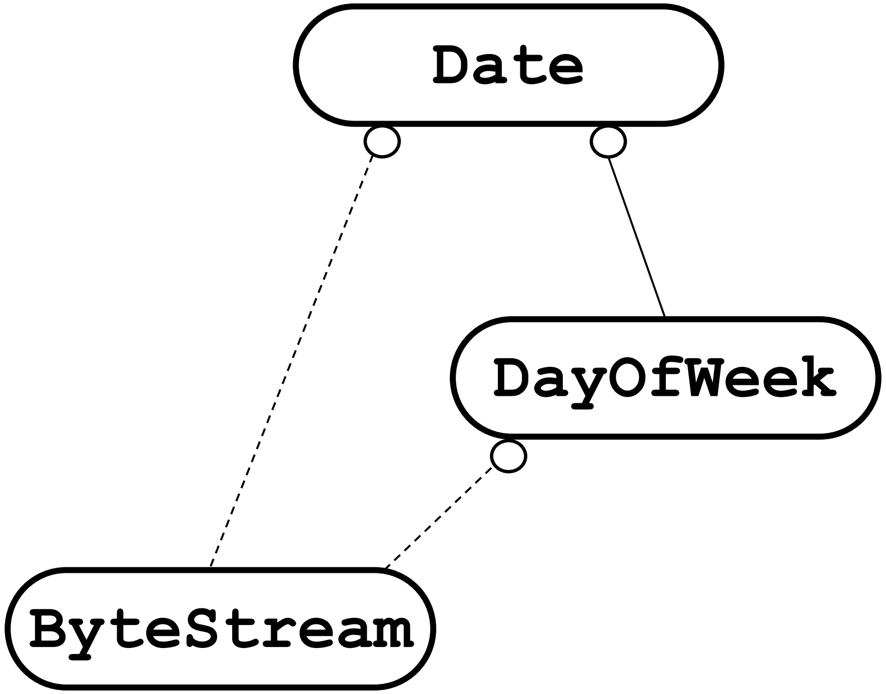
The Original Request

"Write me a 'Date' class that tells me whether today is a business day."

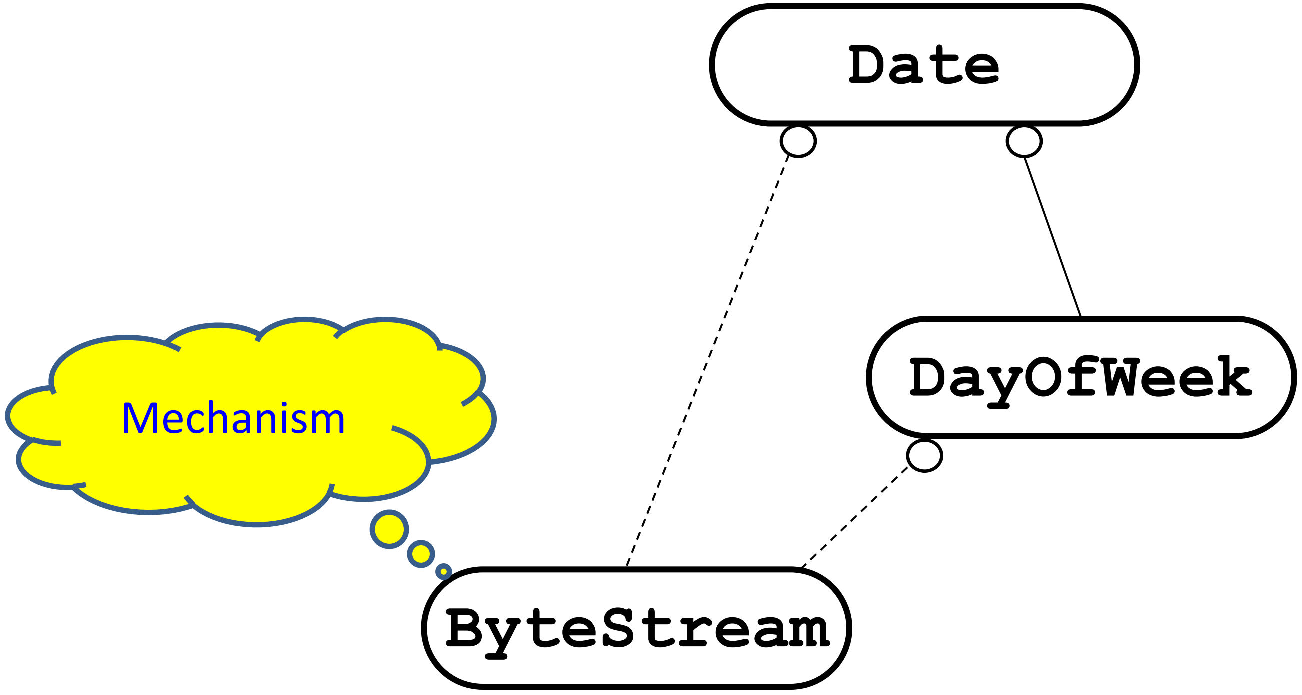
What are the *real* requirements?

1. **Represent a *date value* as a C++ Type.**
2. Determine what date value *today* is.
3. Determine if a date value is a *business day*.
4. **Provide well-factored useful components that we'll need over and over again!**

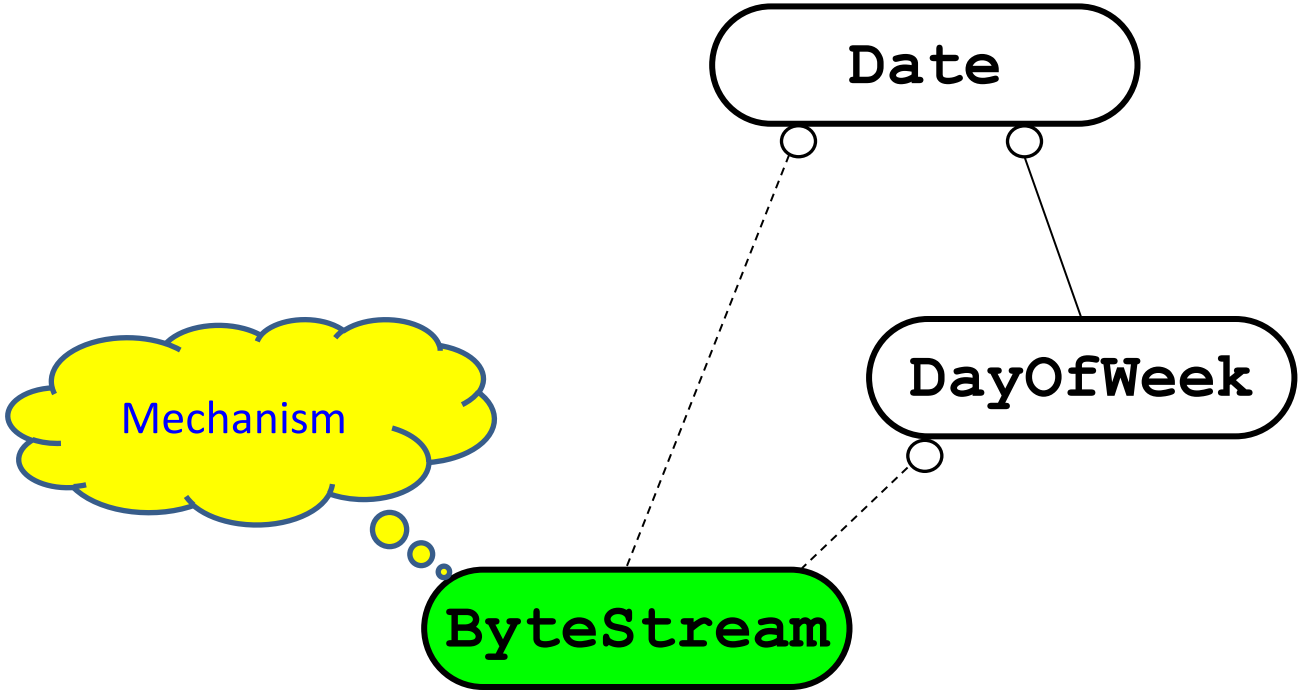
Represent a *Date Value* as a C++ Type



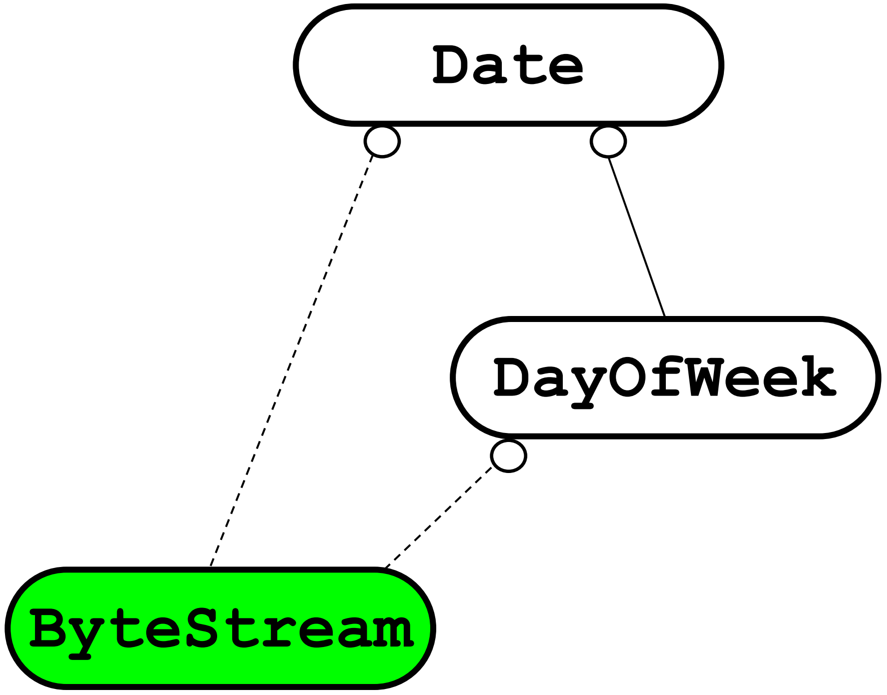
Represent a *Date Value* as a C++ Type



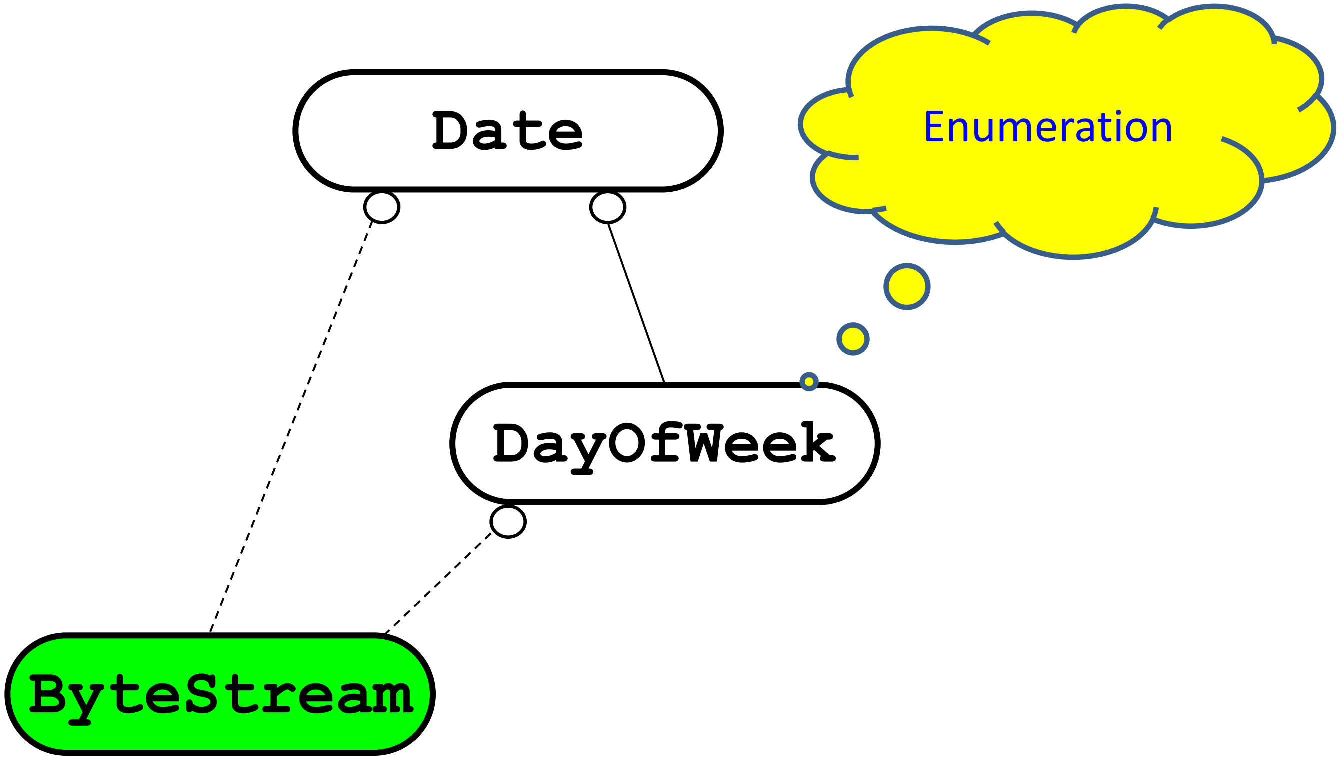
Represent a *Date Value* as a C++ Type



Represent a *Date Value* as a C++ Type

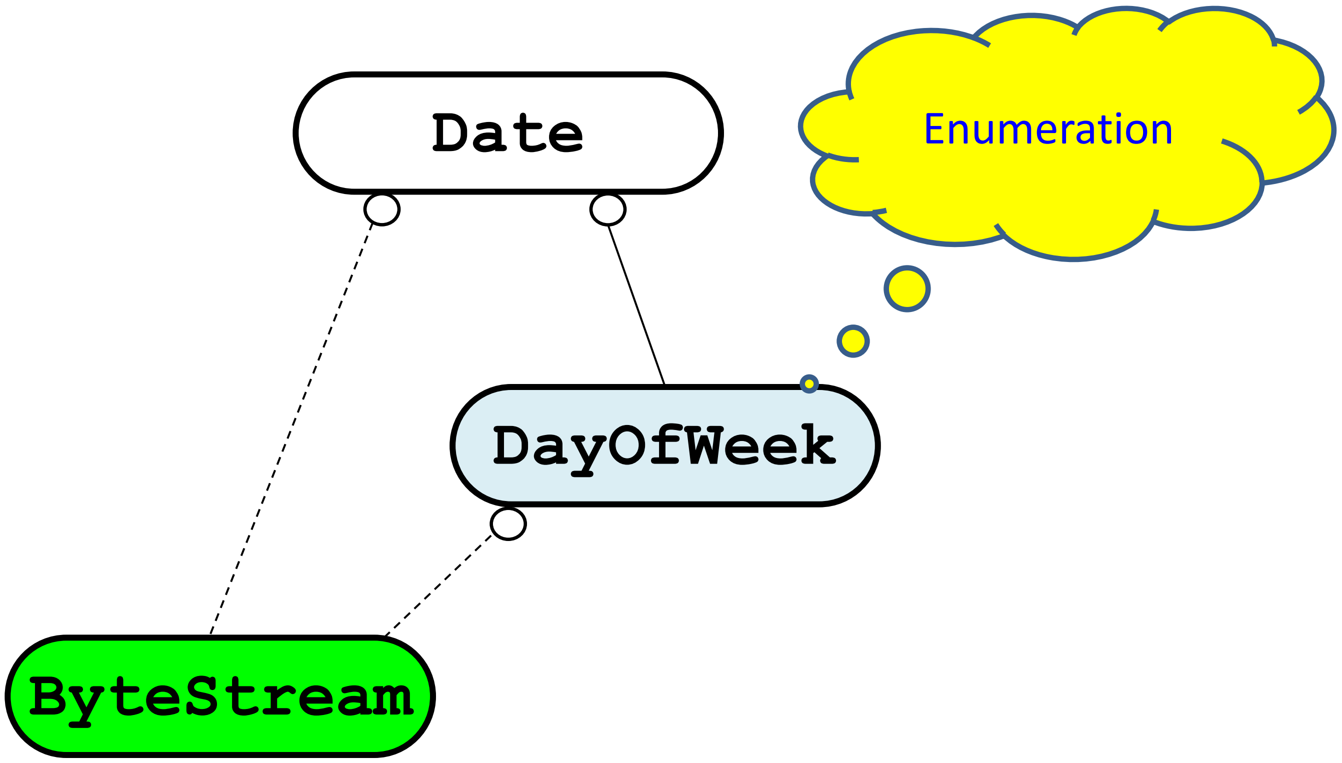


Represent a *Date Value* as a C++ Type

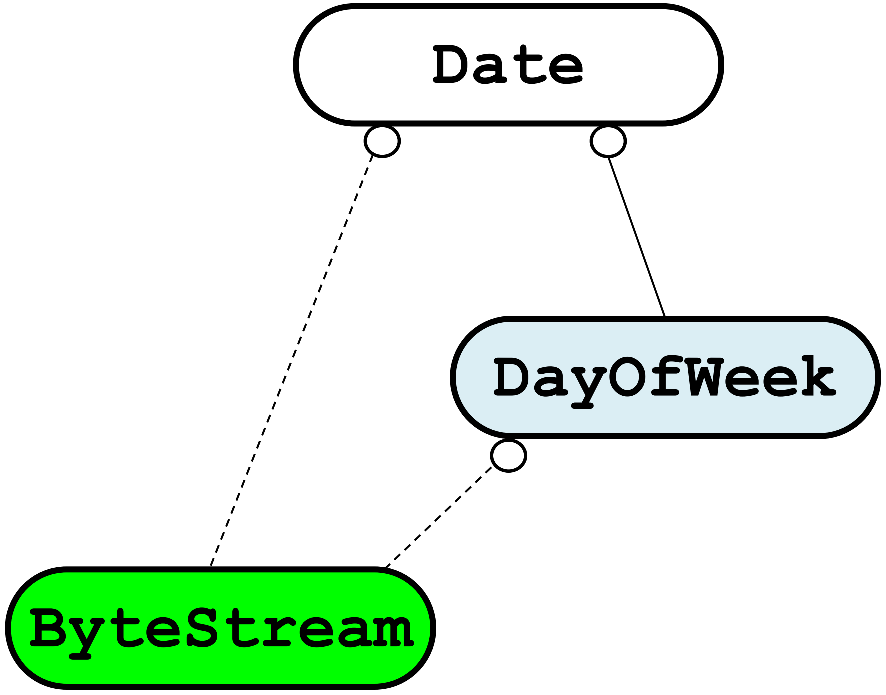


4. Bloomberg Development Environment

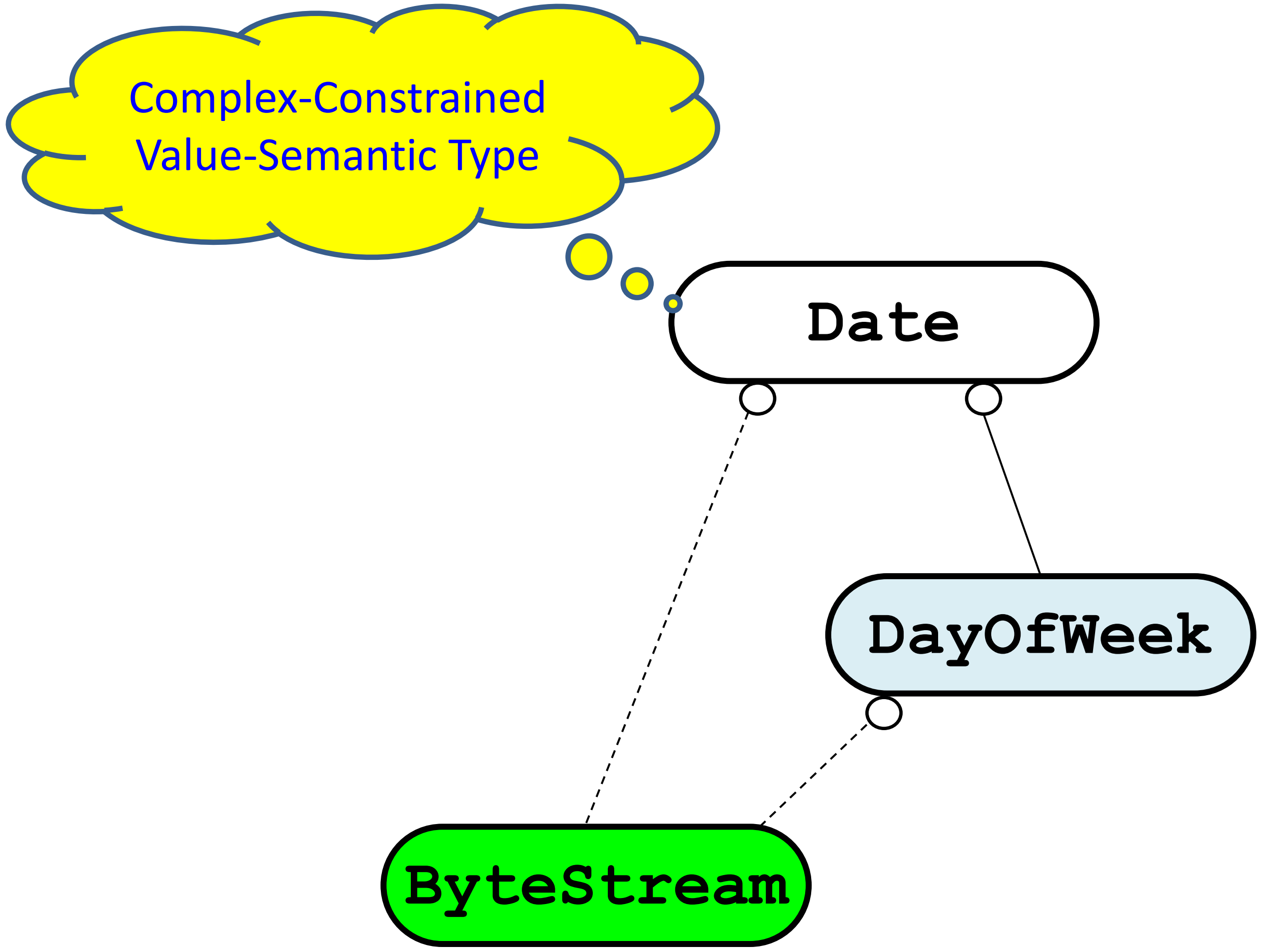
Represent a *Date Value* as a C++ Type



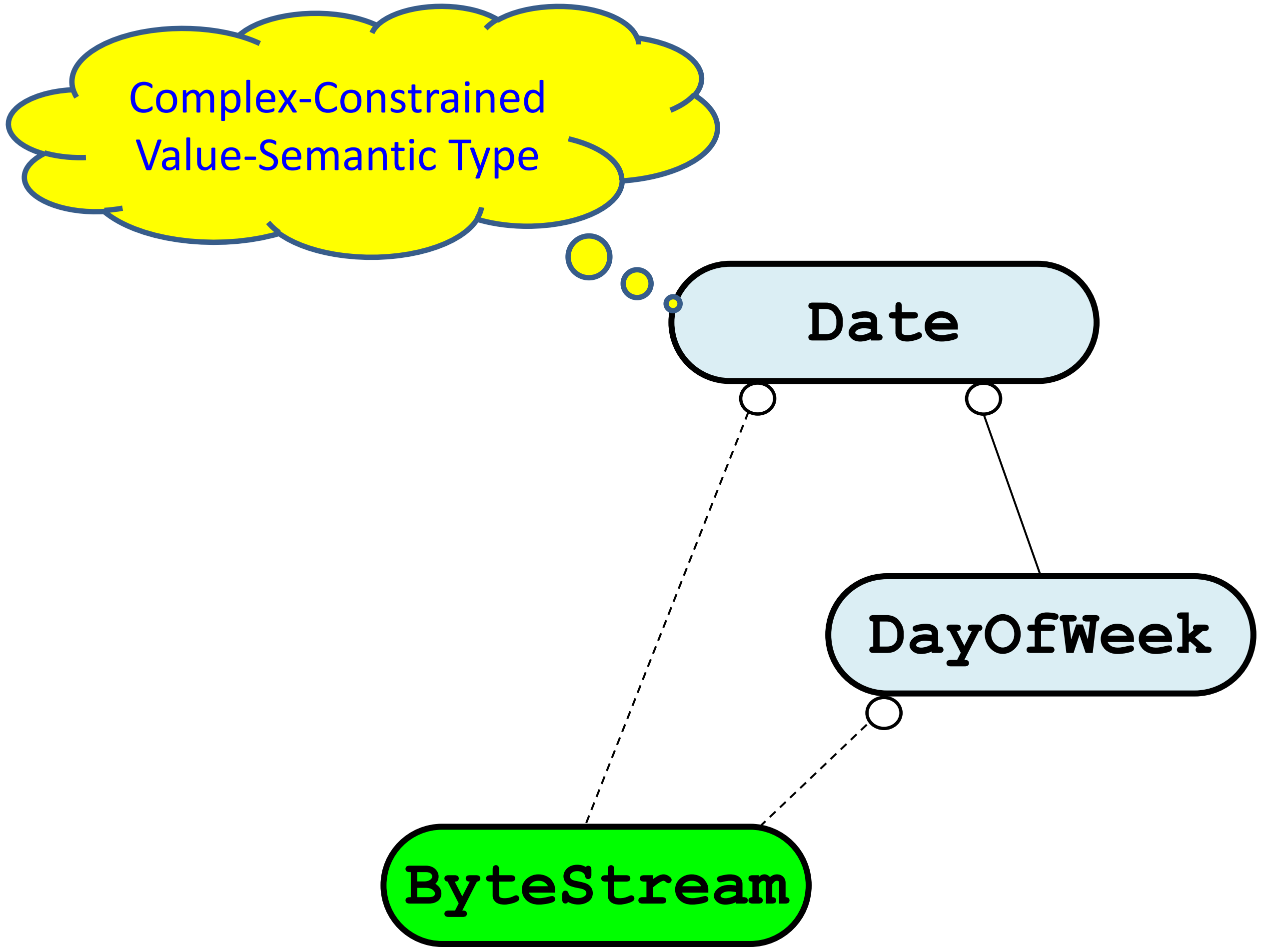
Represent a *Date Value* as a C++ Type



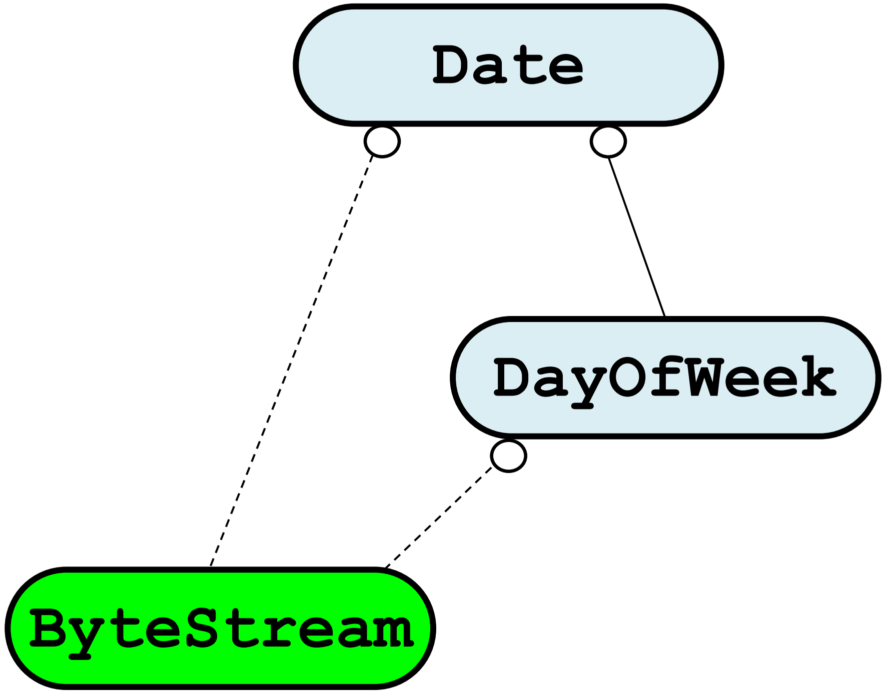
Represent a *Date Value* as a C++ Type



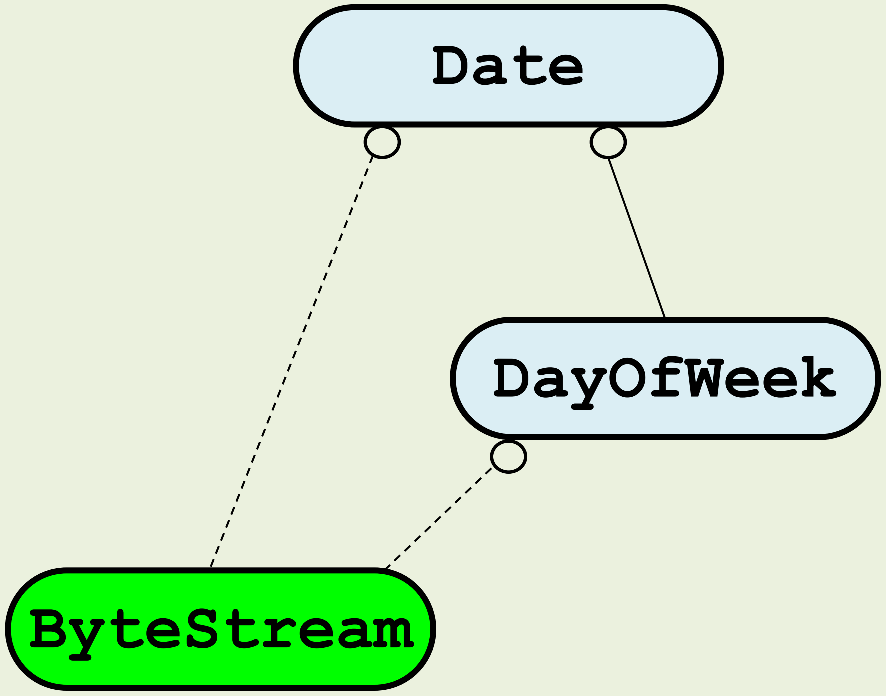
Represent a *Date Value* as a C++ Type



Represent a *Date Value* as a C++ Type



Solution 1: Represent a `Date` Value.



The Original Request

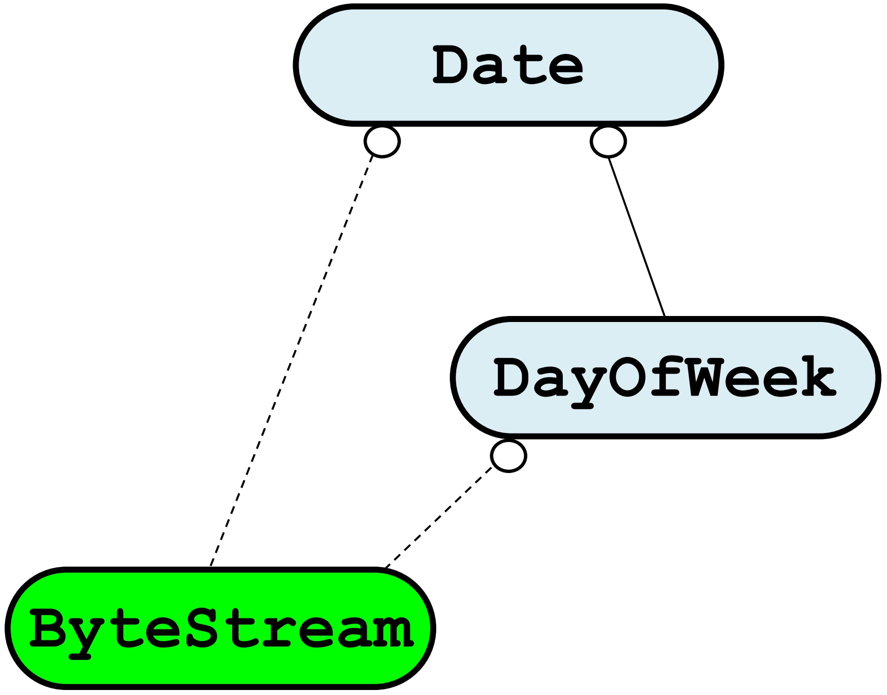
"Write me a 'Date' class that tells me whether today is a business day."

What are the *real* requirements?

1. Represent a *date value* as a C++ Type.
2. **Determine what date value *today* is.**
3. Determine if a date value is a *business day*.
4. **Provide well-factored useful components that we'll need over and over again!**

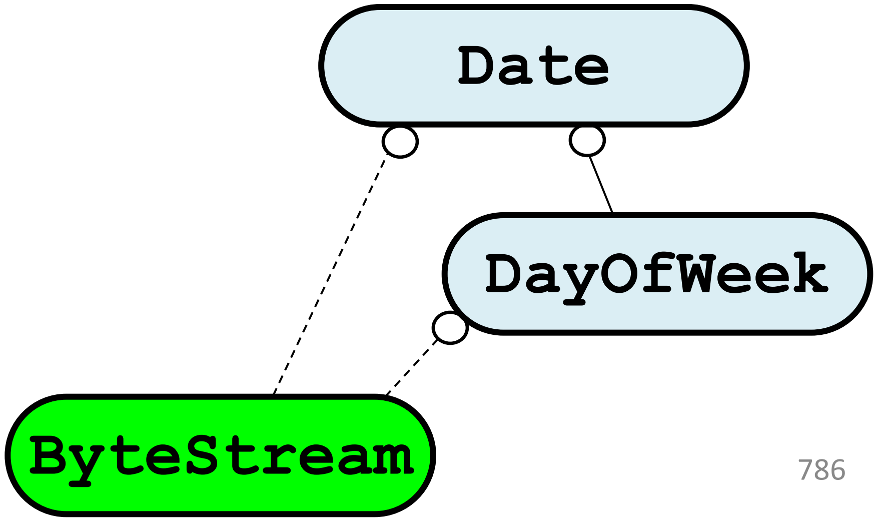
4. Bloomberg Development Environment

Determine what Date Value *today* is

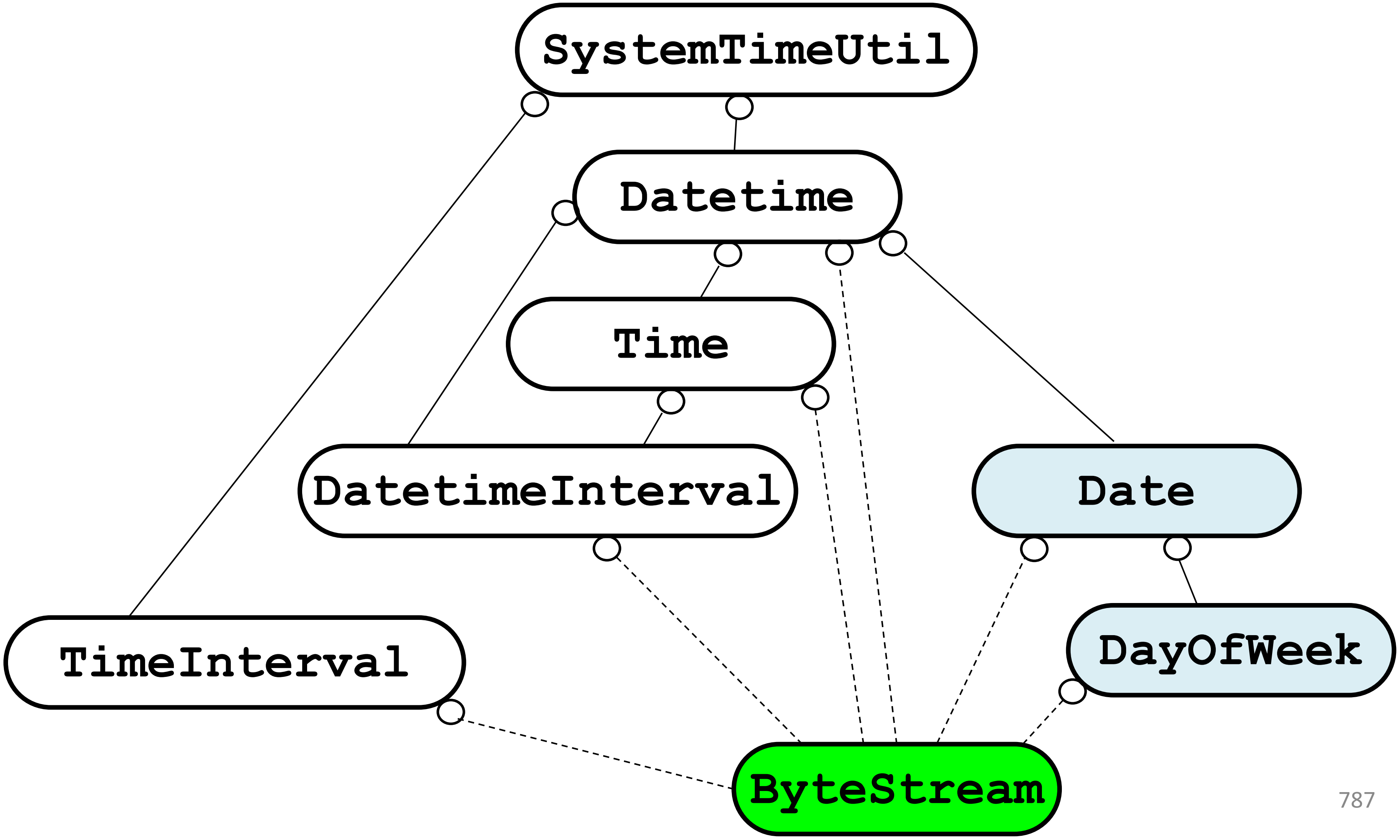


4. Bloomberg Development Environment

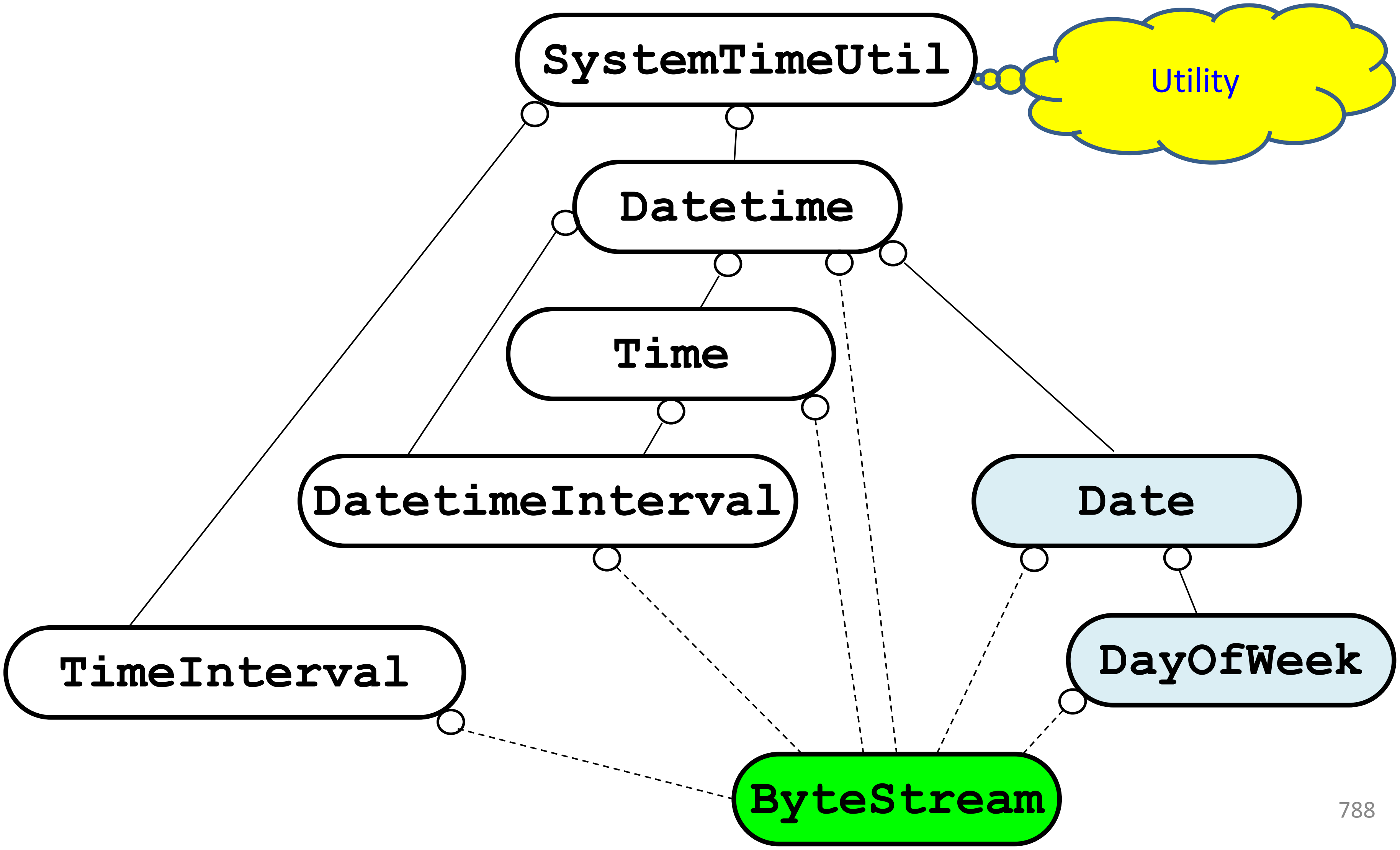
Determine what Date Value *today* is



Determine what Date Value *today* is

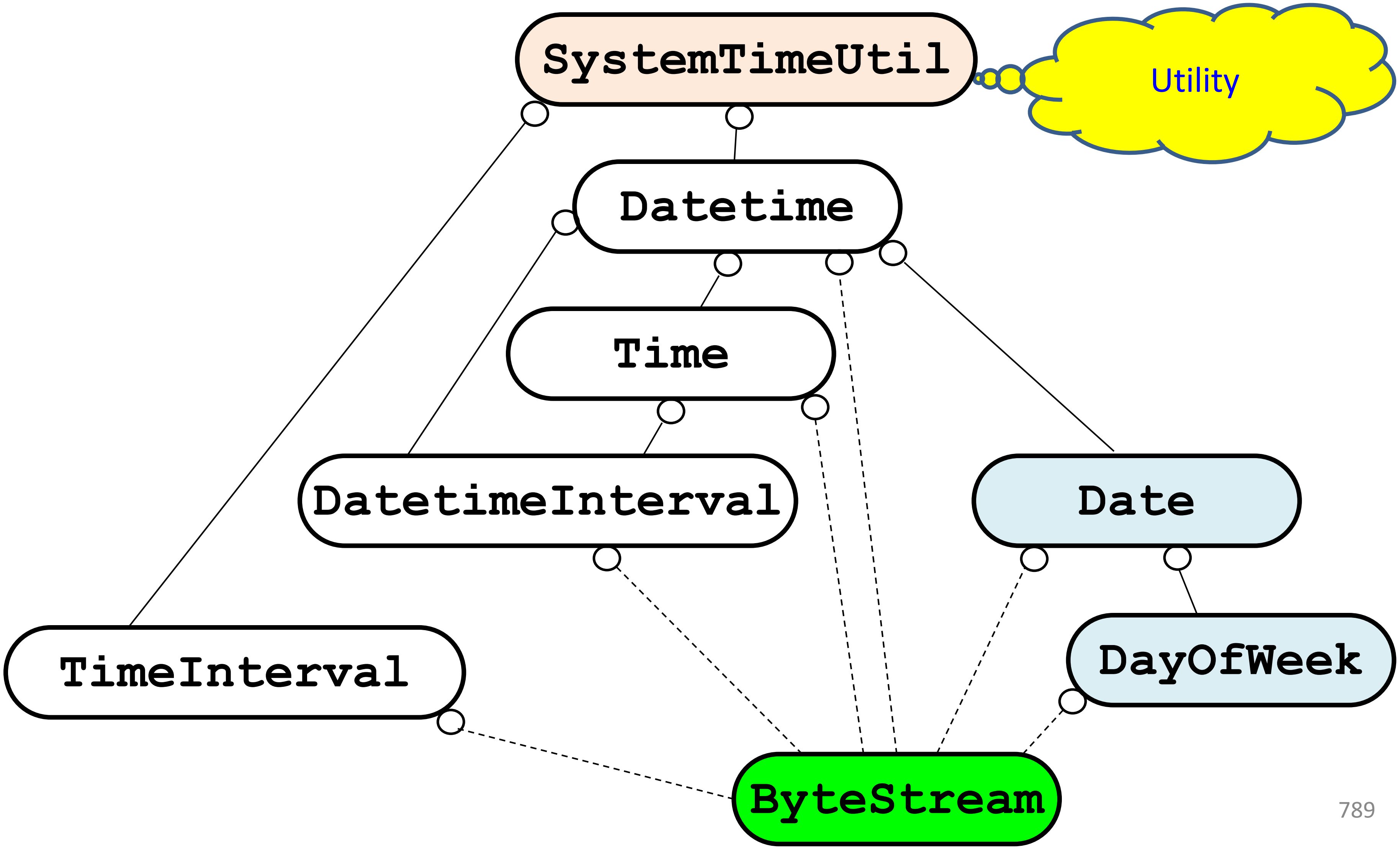


Determine what Date Value *today* is

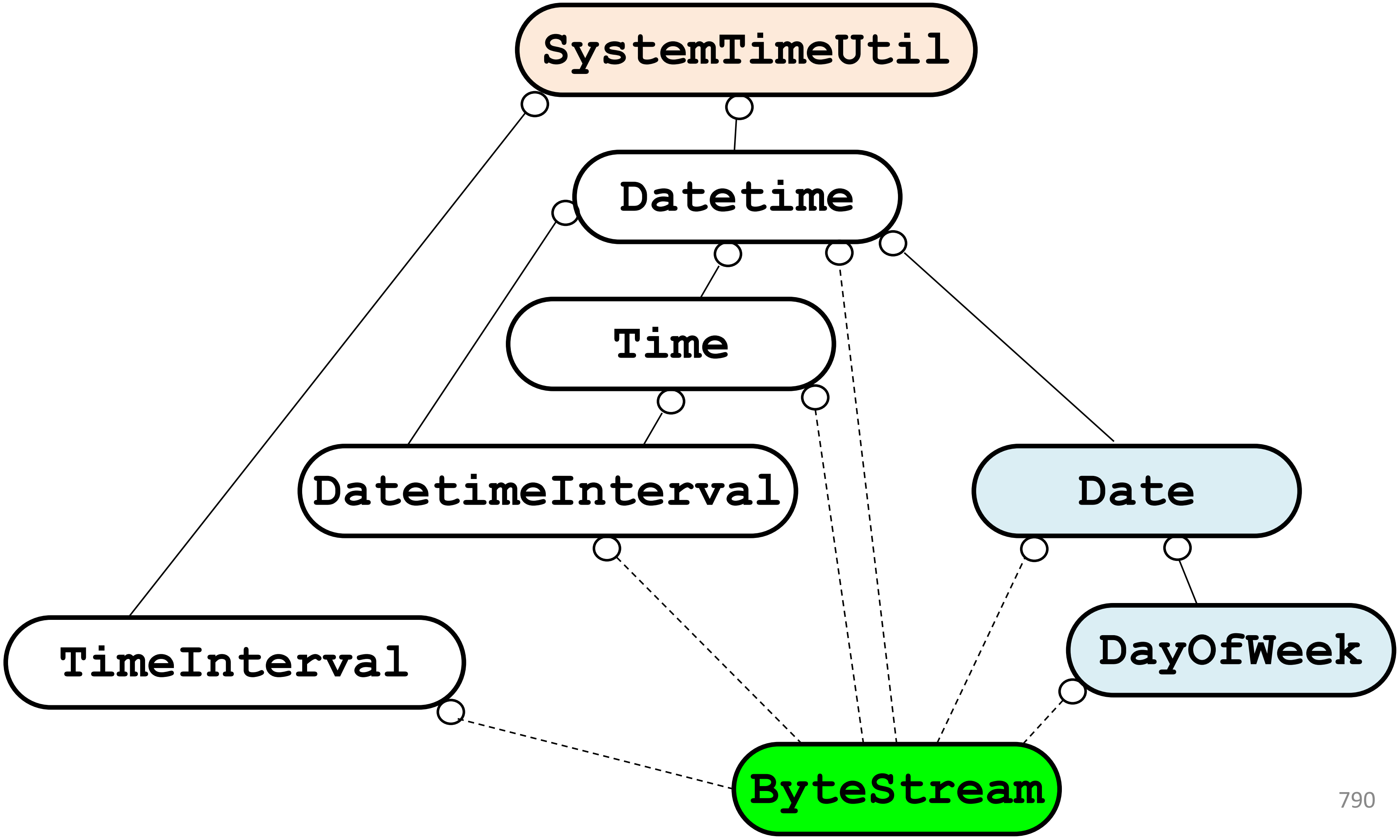


4. Bloomberg Development Environment

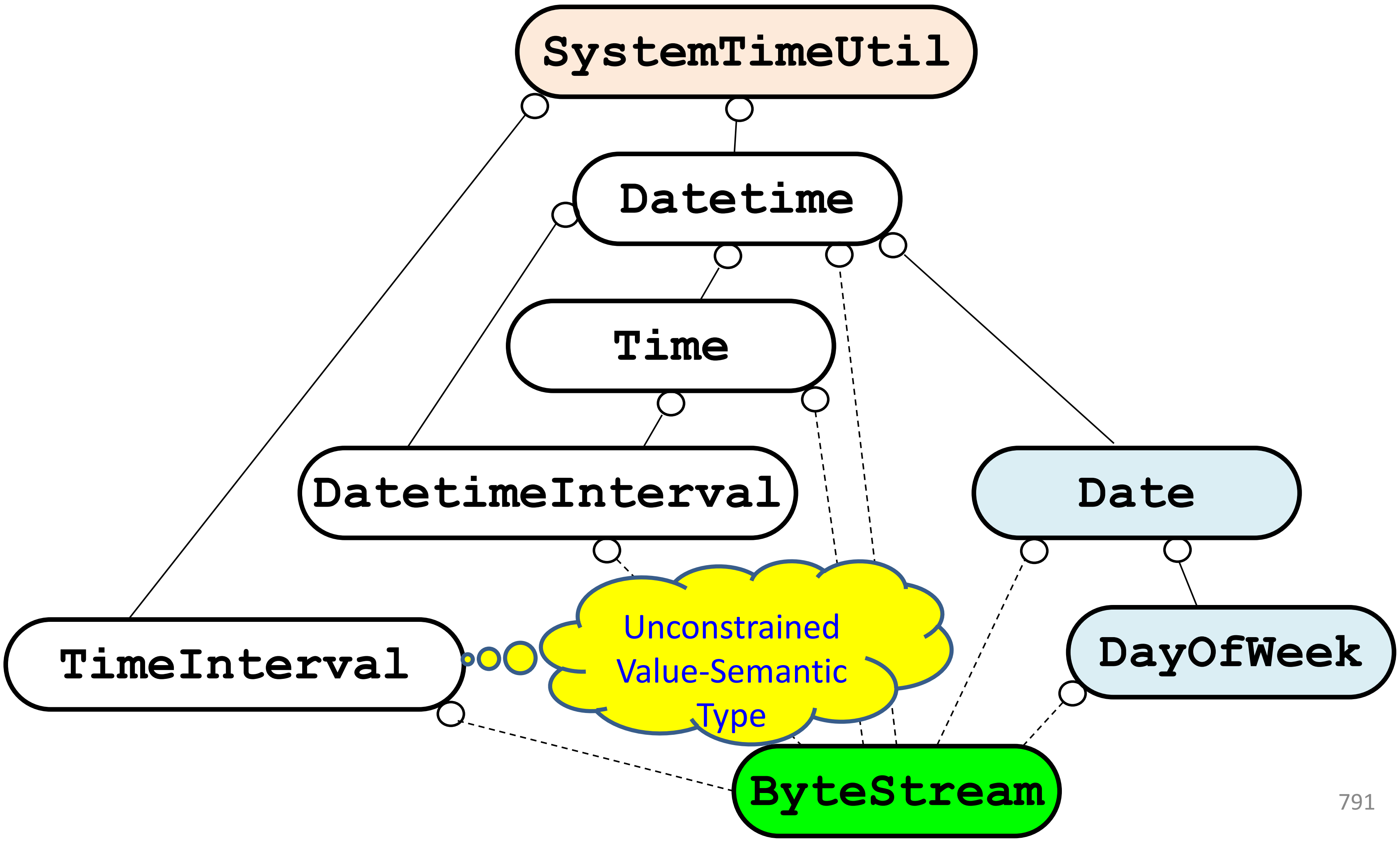
Determine what Date Value *today* is



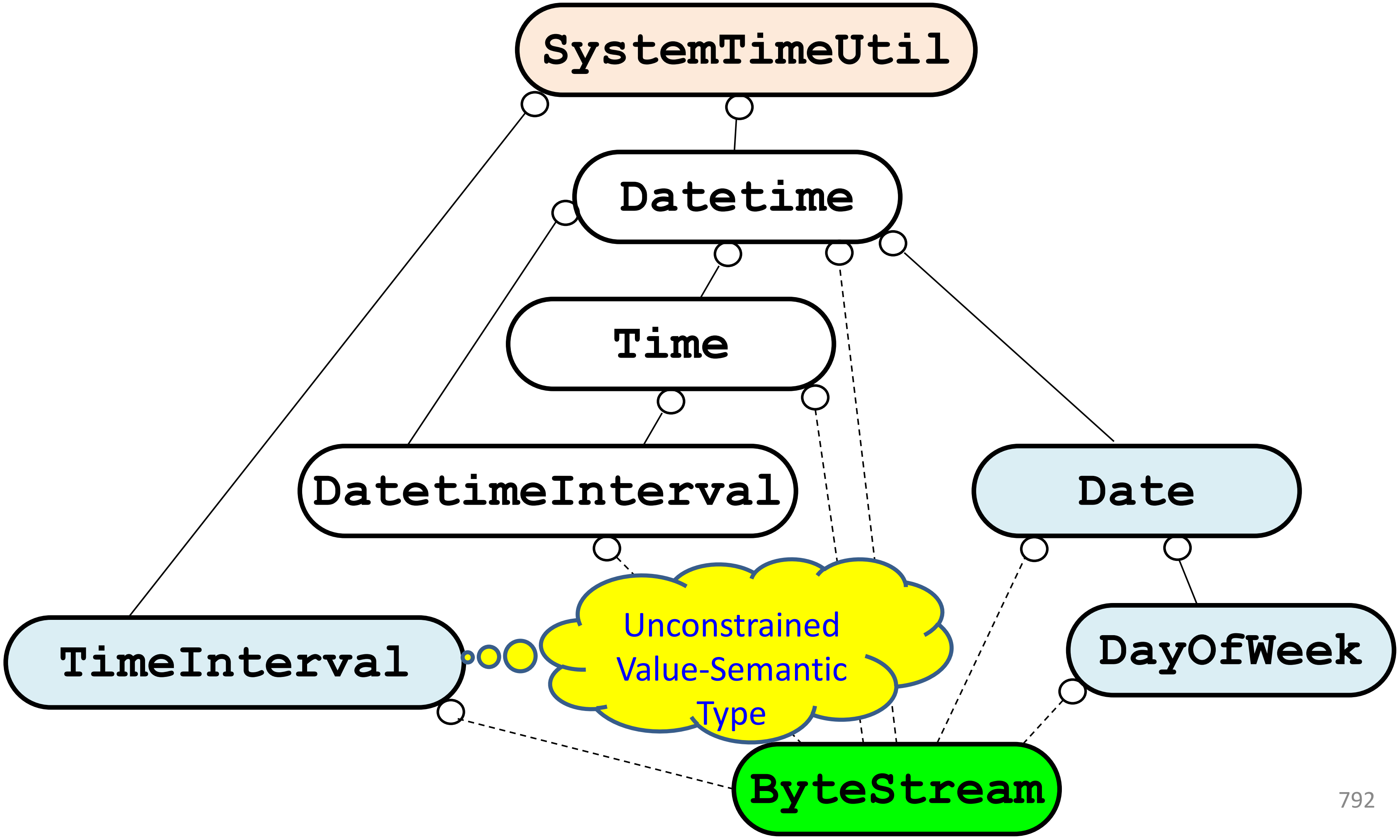
Determine what Date Value *today* is



Determine what Date Value *today* is

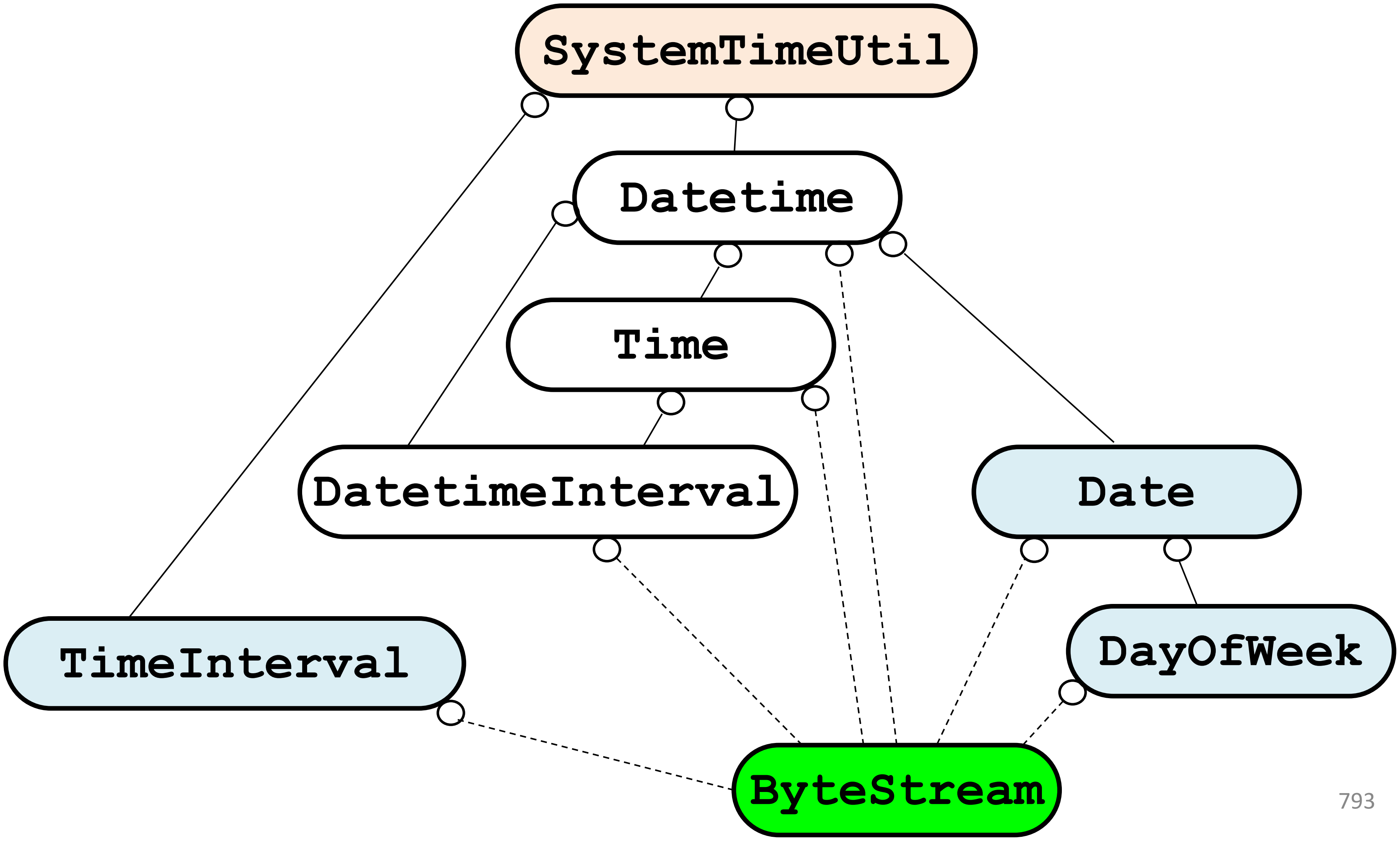


Determine what Date Value *today* is

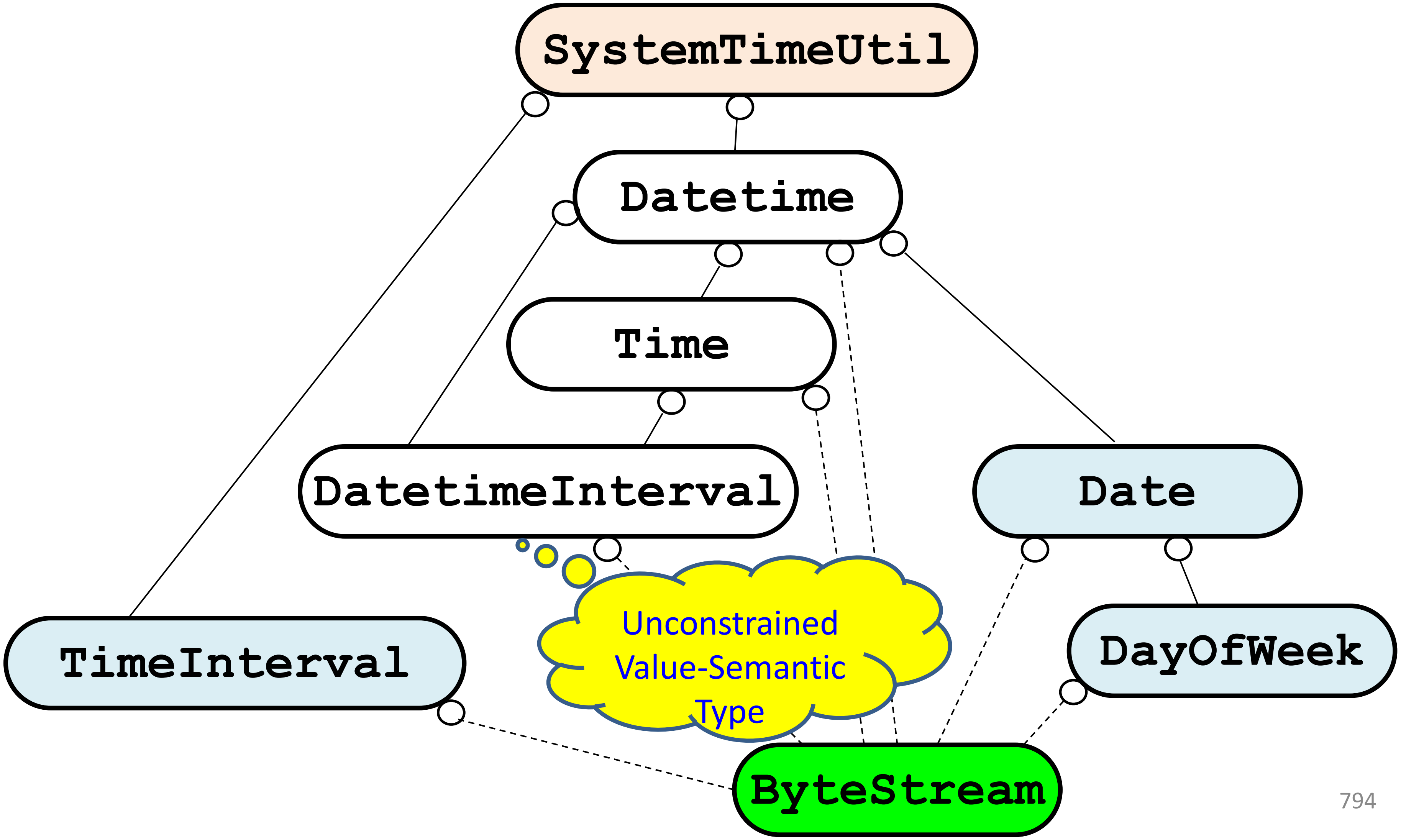


4. Bloomberg Development Environment

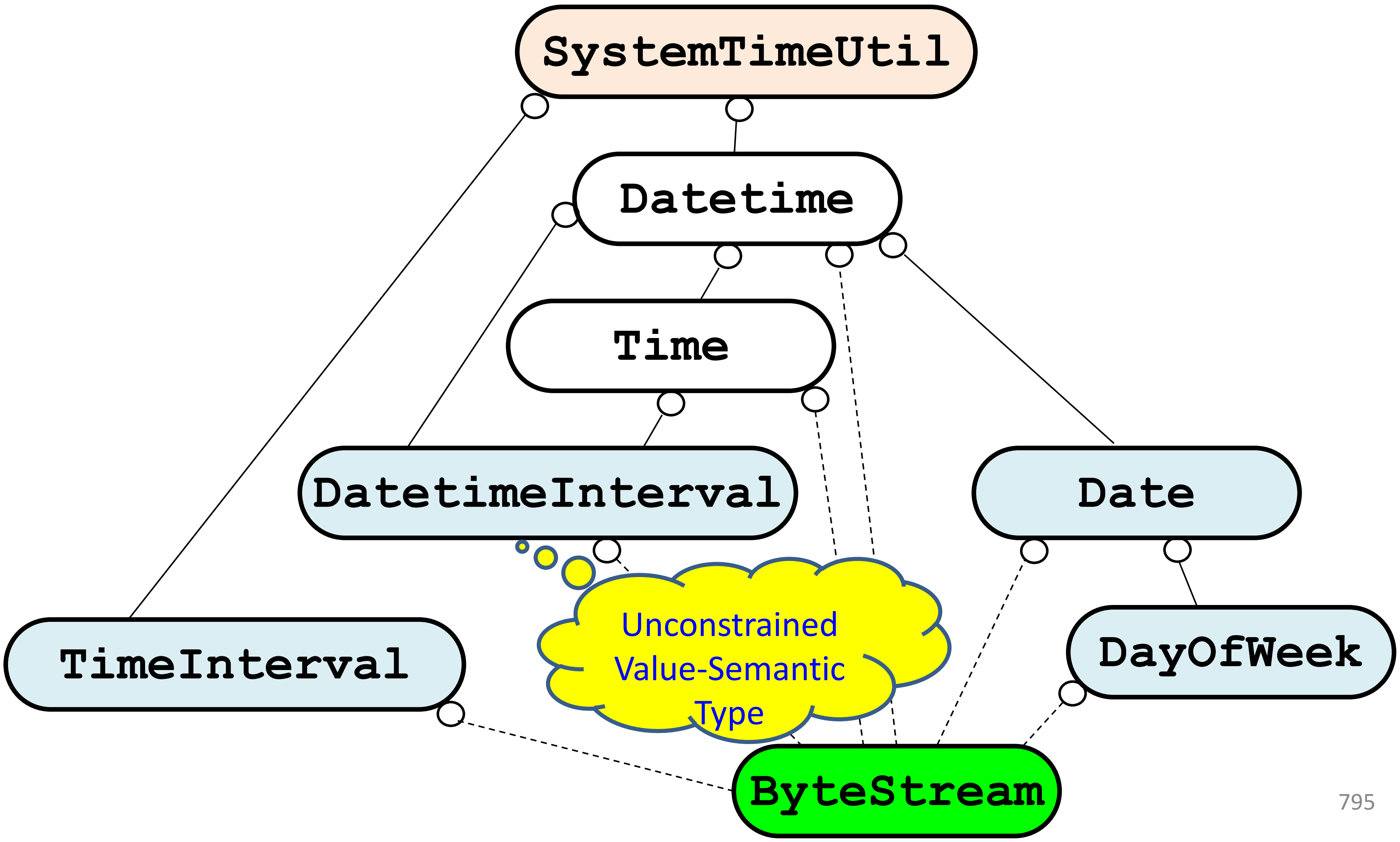
Determine what Date Value *today* is



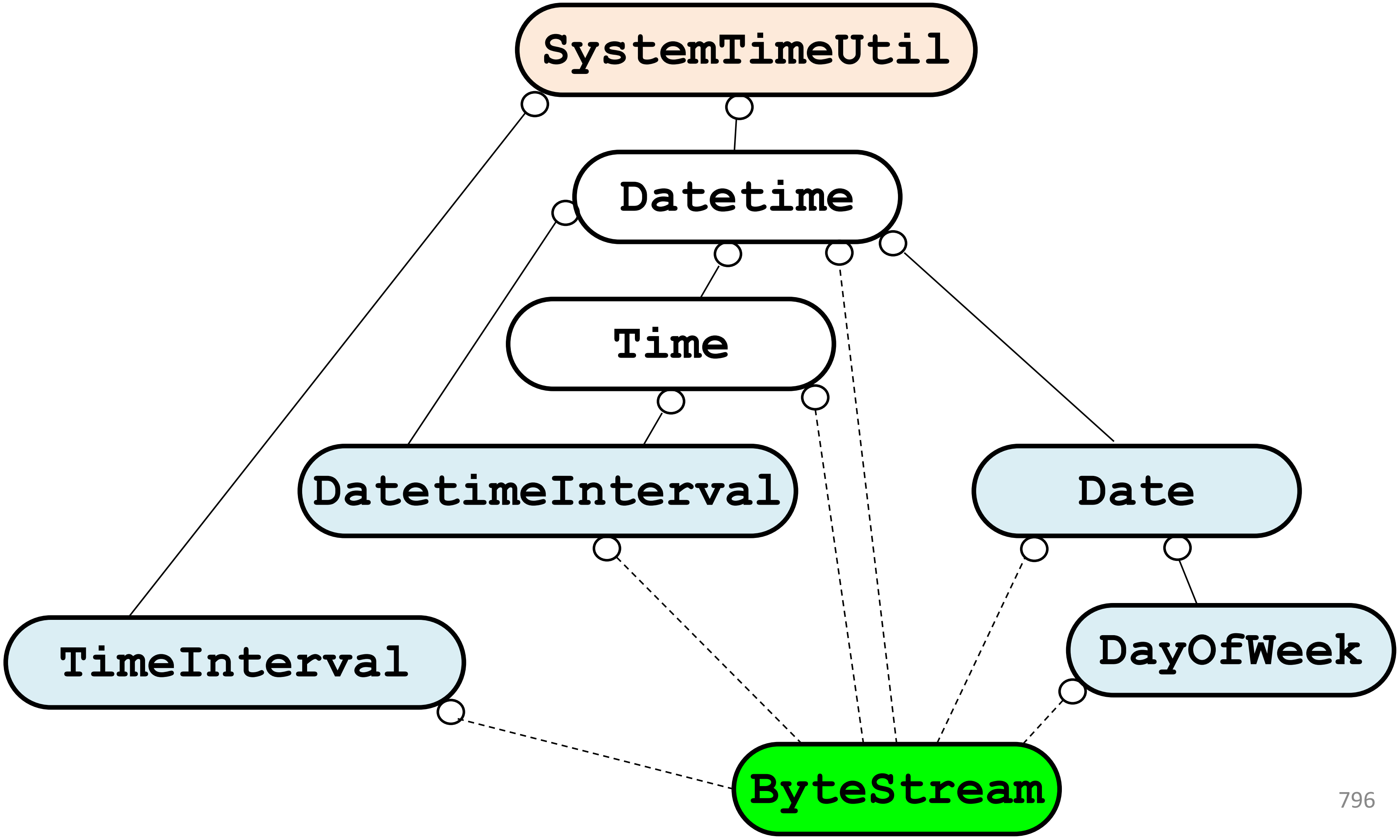
Determine what Date Value *today* is



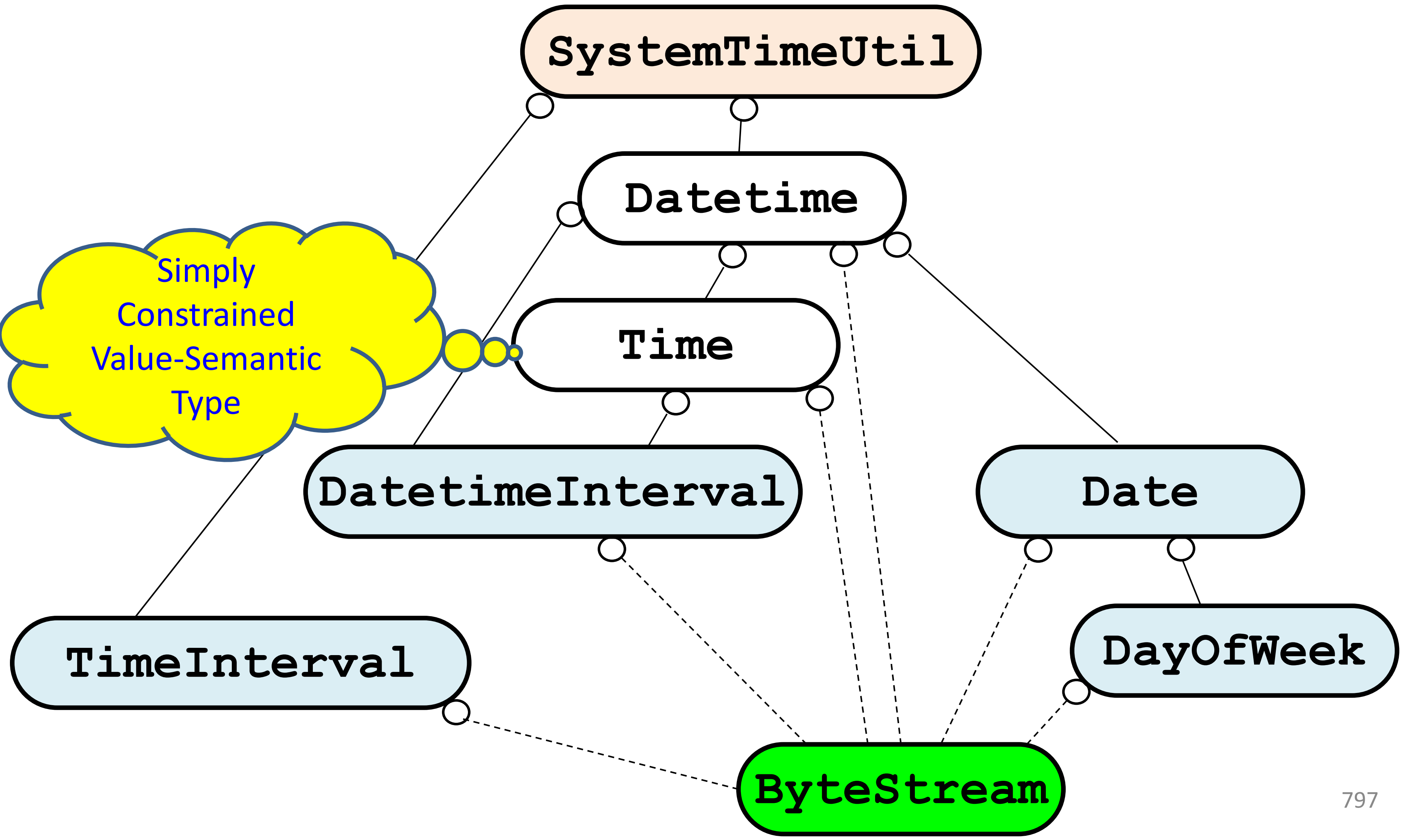
Determine what Date Value *today* is



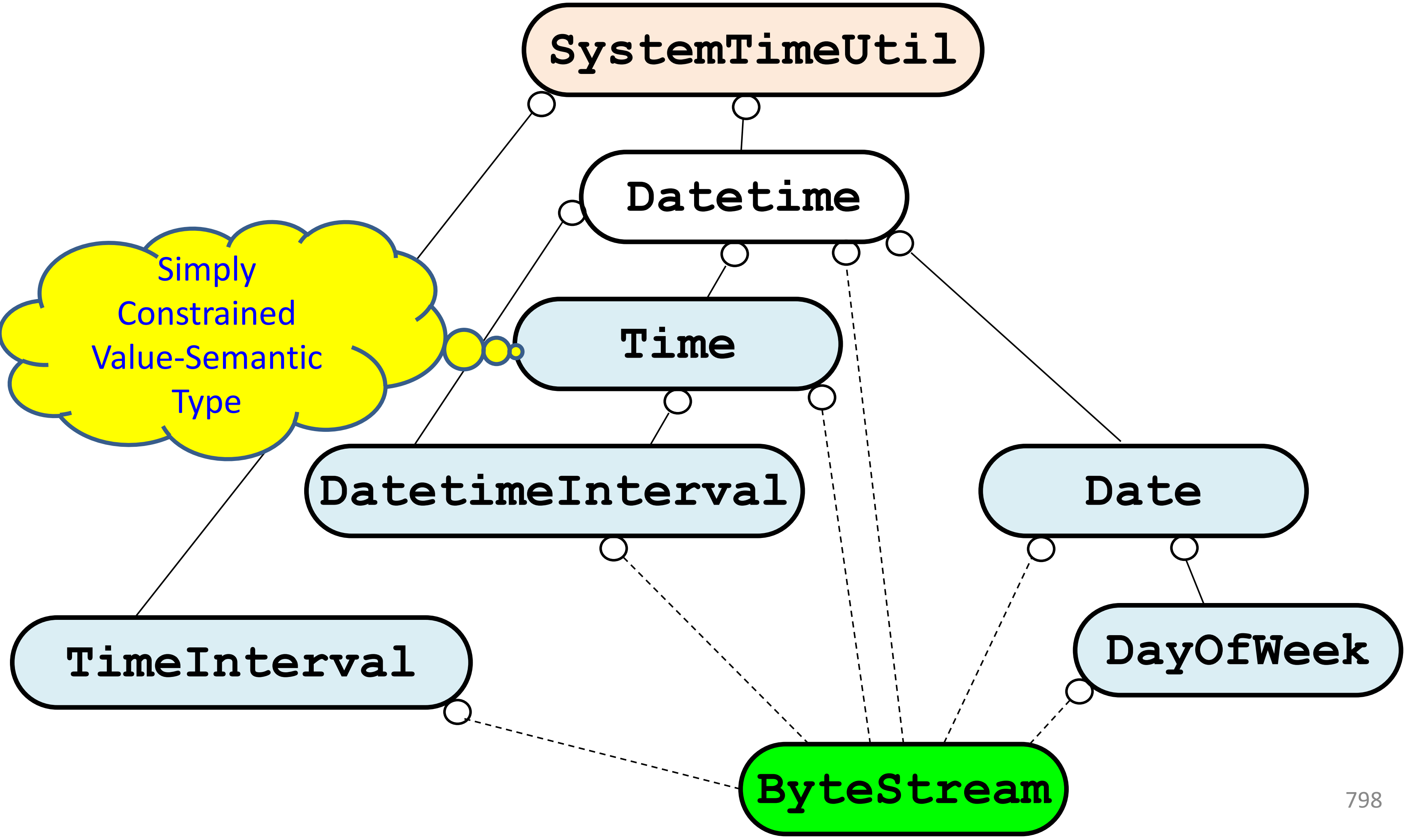
Determine what Date Value *today* is



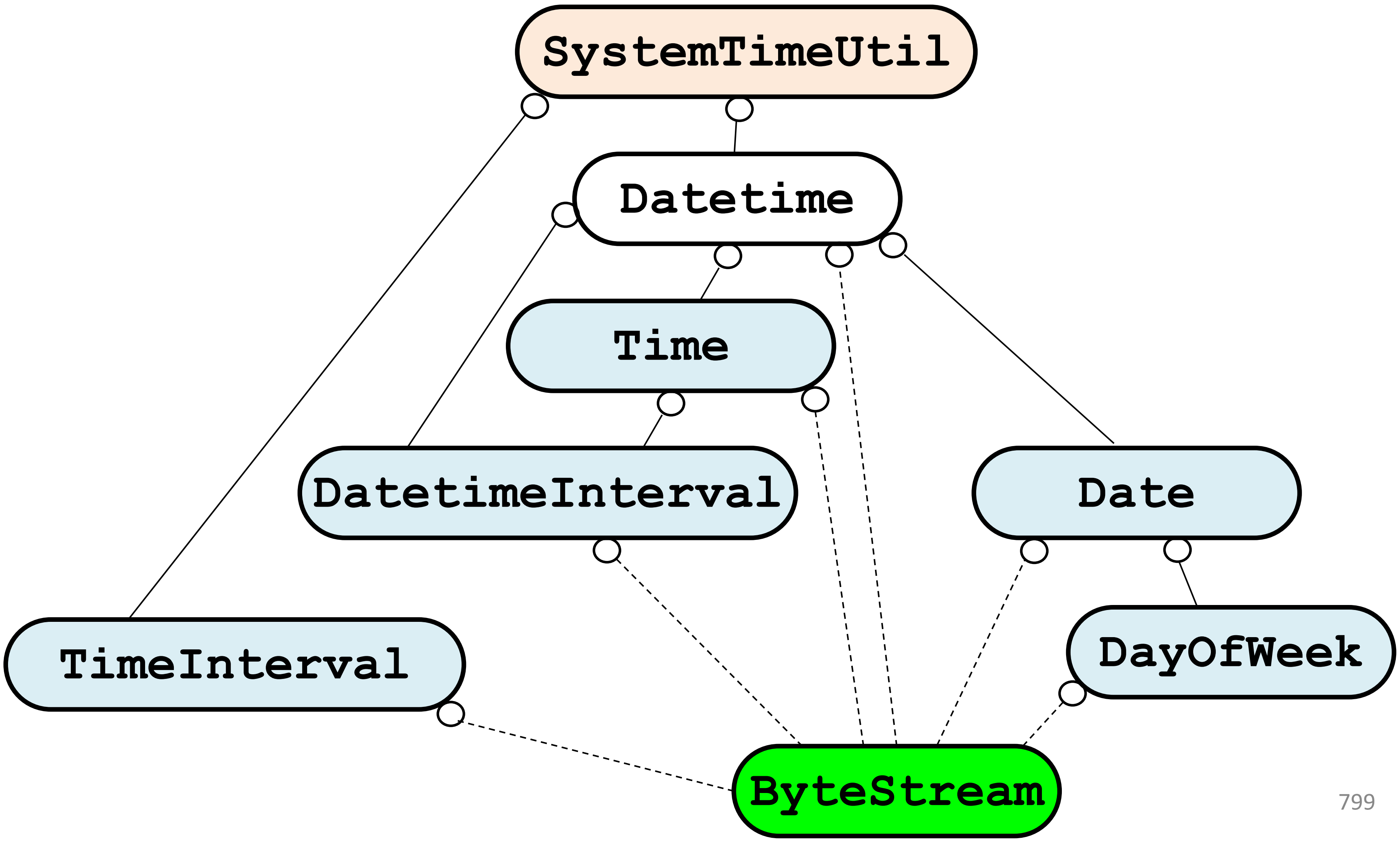
Determine what Date Value *today* is



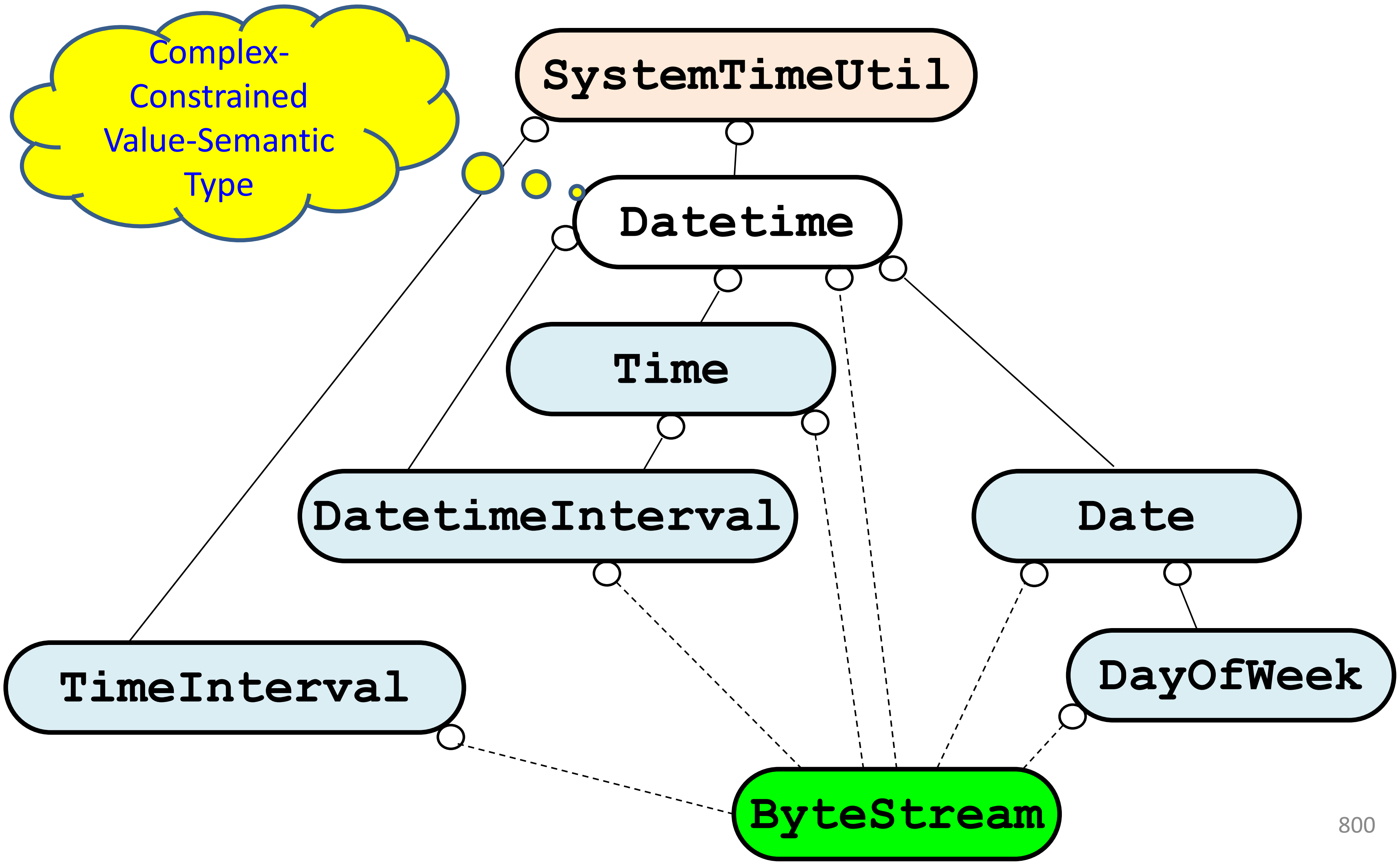
Determine what Date Value *today* is



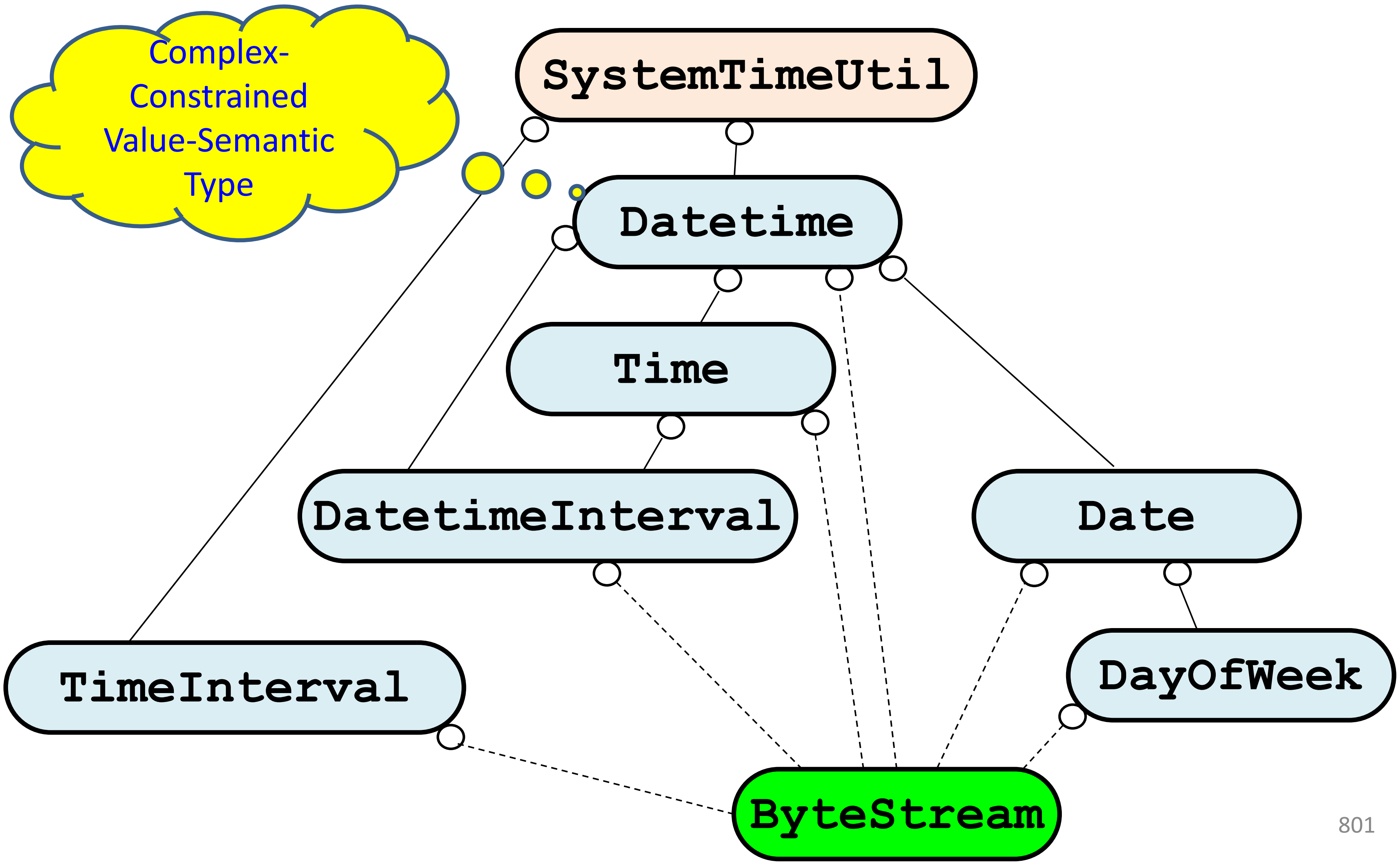
Determine what Date Value *today* is



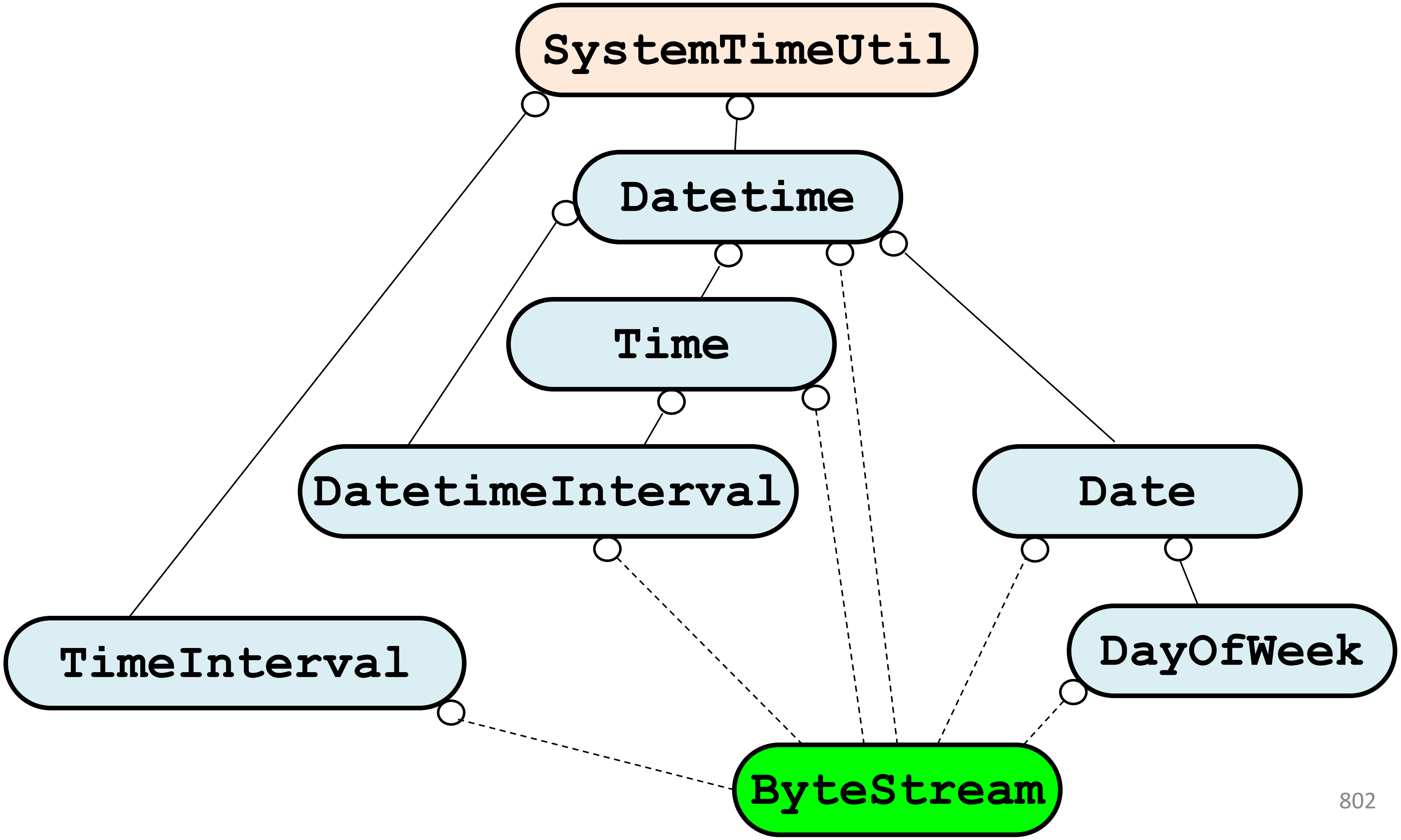
Determine what Date Value *today* is



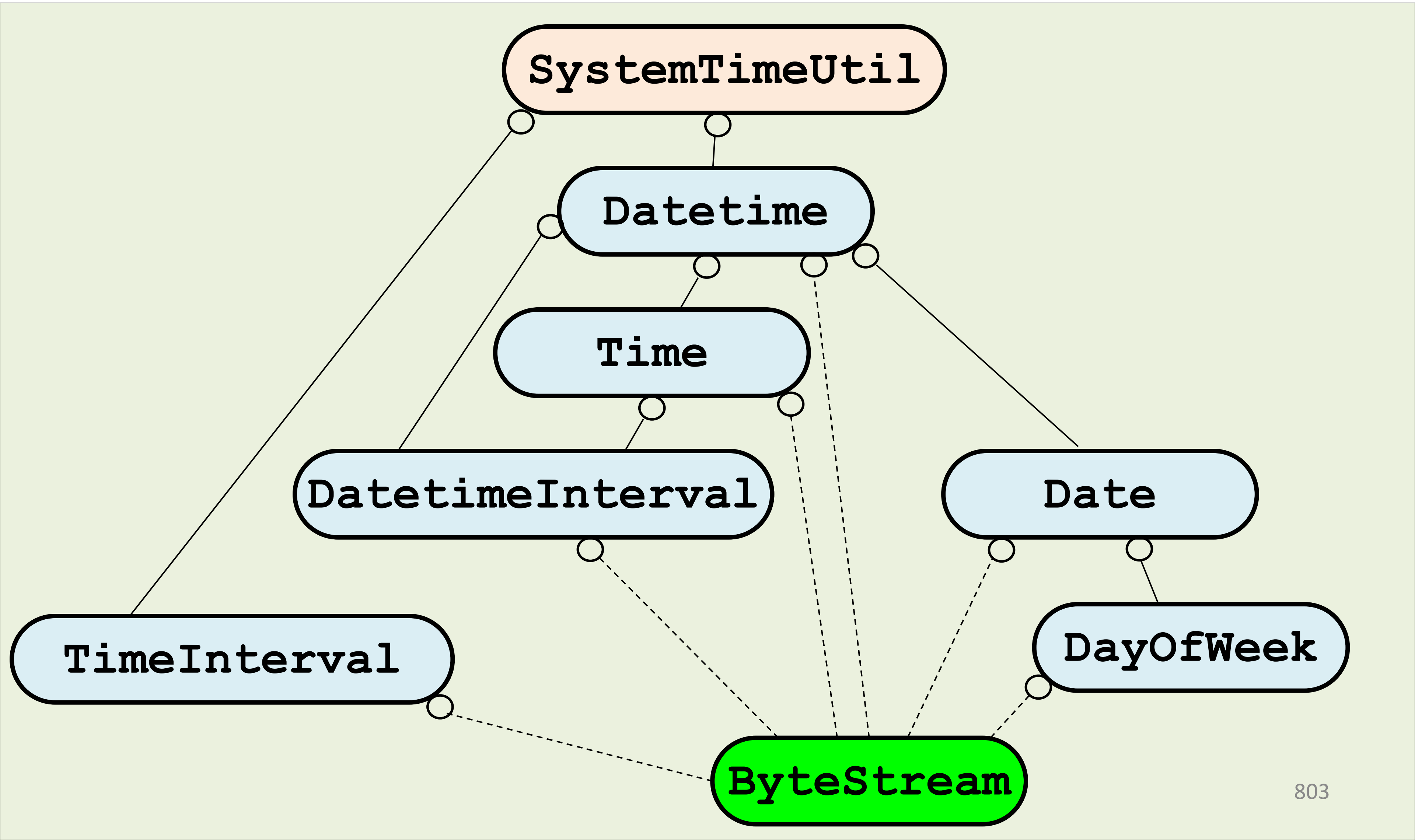
Determine what Date Value *today* is



Determine what Date Value *today* is



Solution 2: What `Date` is Today?



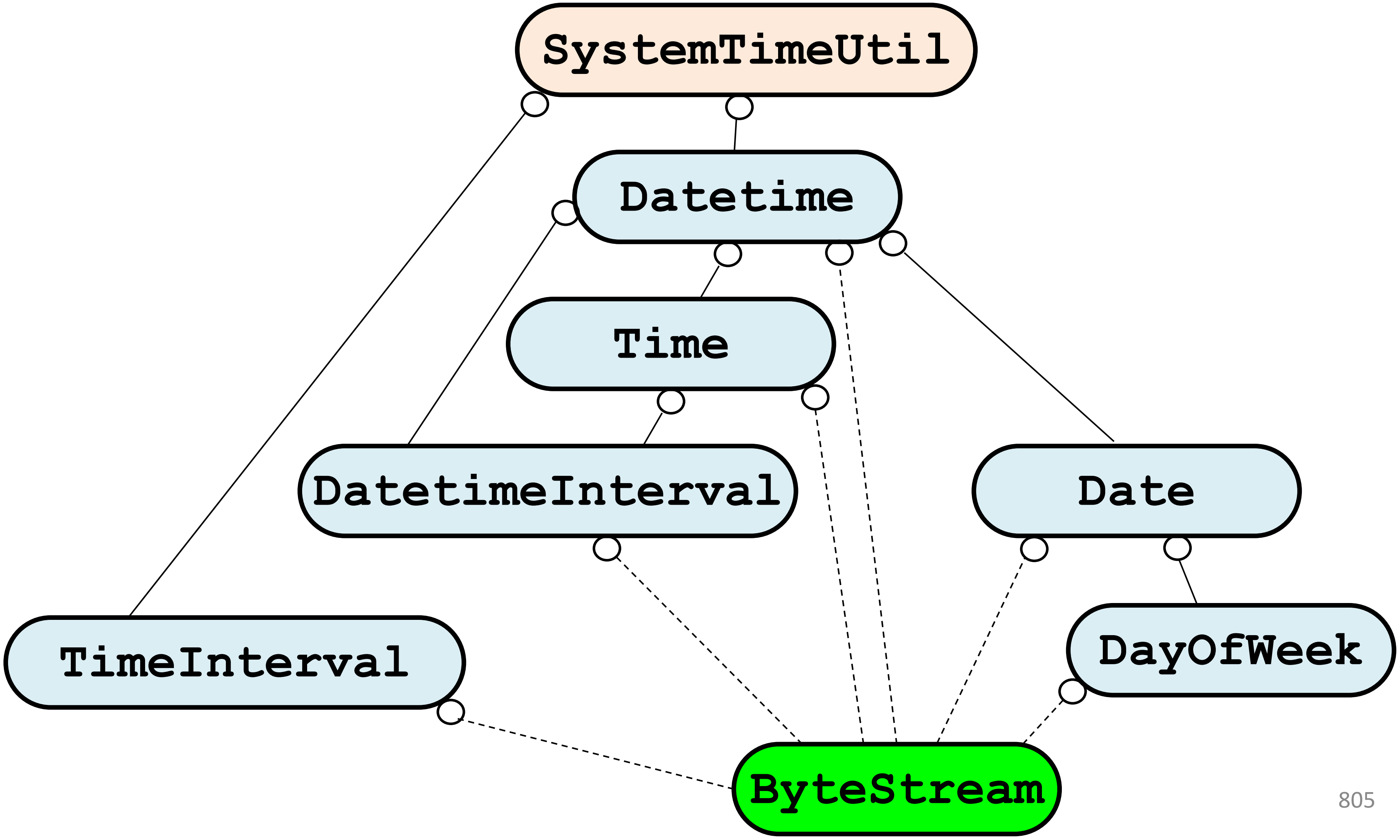
The Original Request

"Write me a 'Date' class that tells me whether today is a business day."

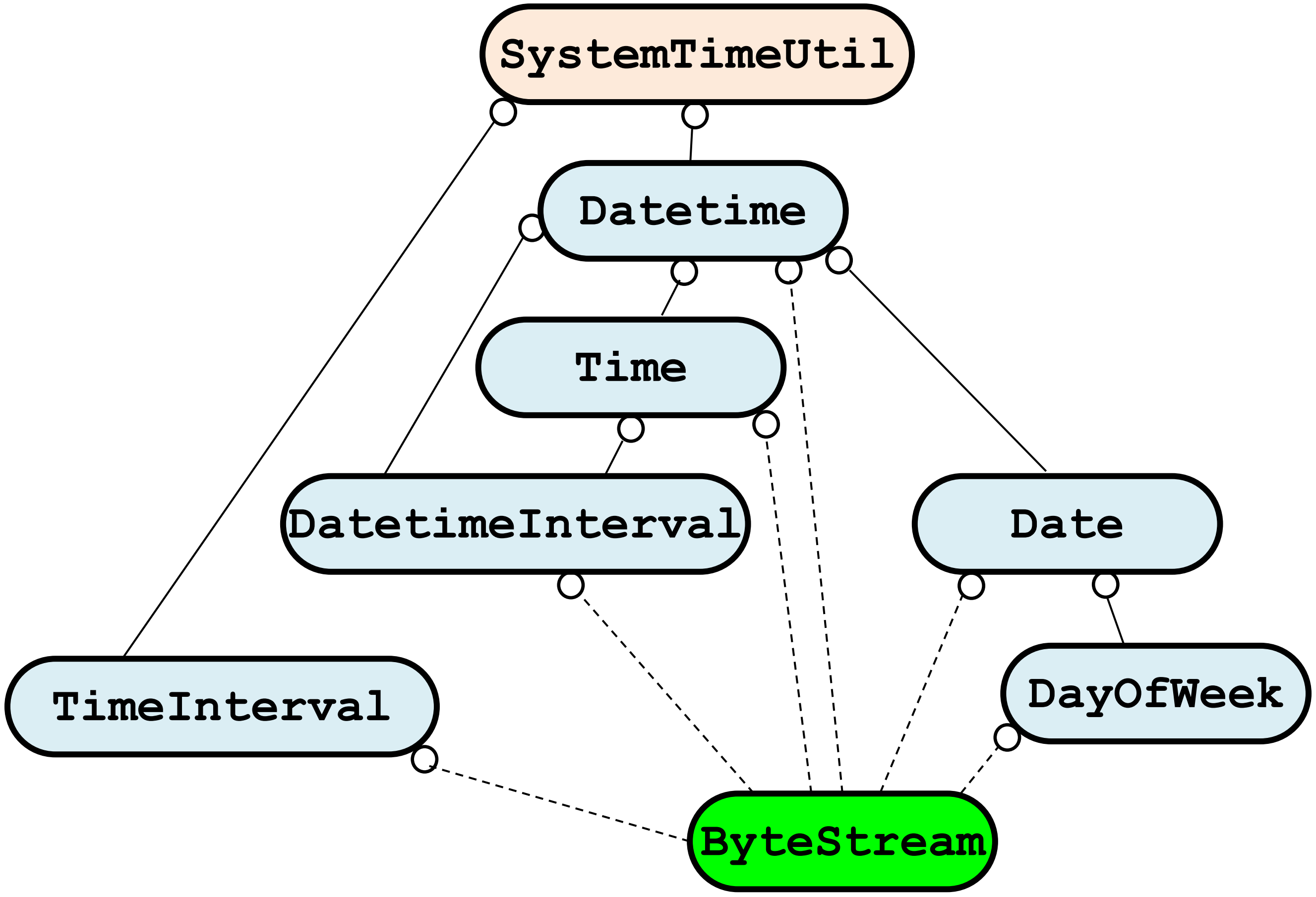
What are the *real* requirements?

1. Represent a *date value* as a C++ Type.
2. Determine what date value *today* is.
3. **Determine if a date value is a *business day*.**
4. **Provide well-factored useful components that we'll need over and over again!**

Determine if a Date Value is a *Business Day*

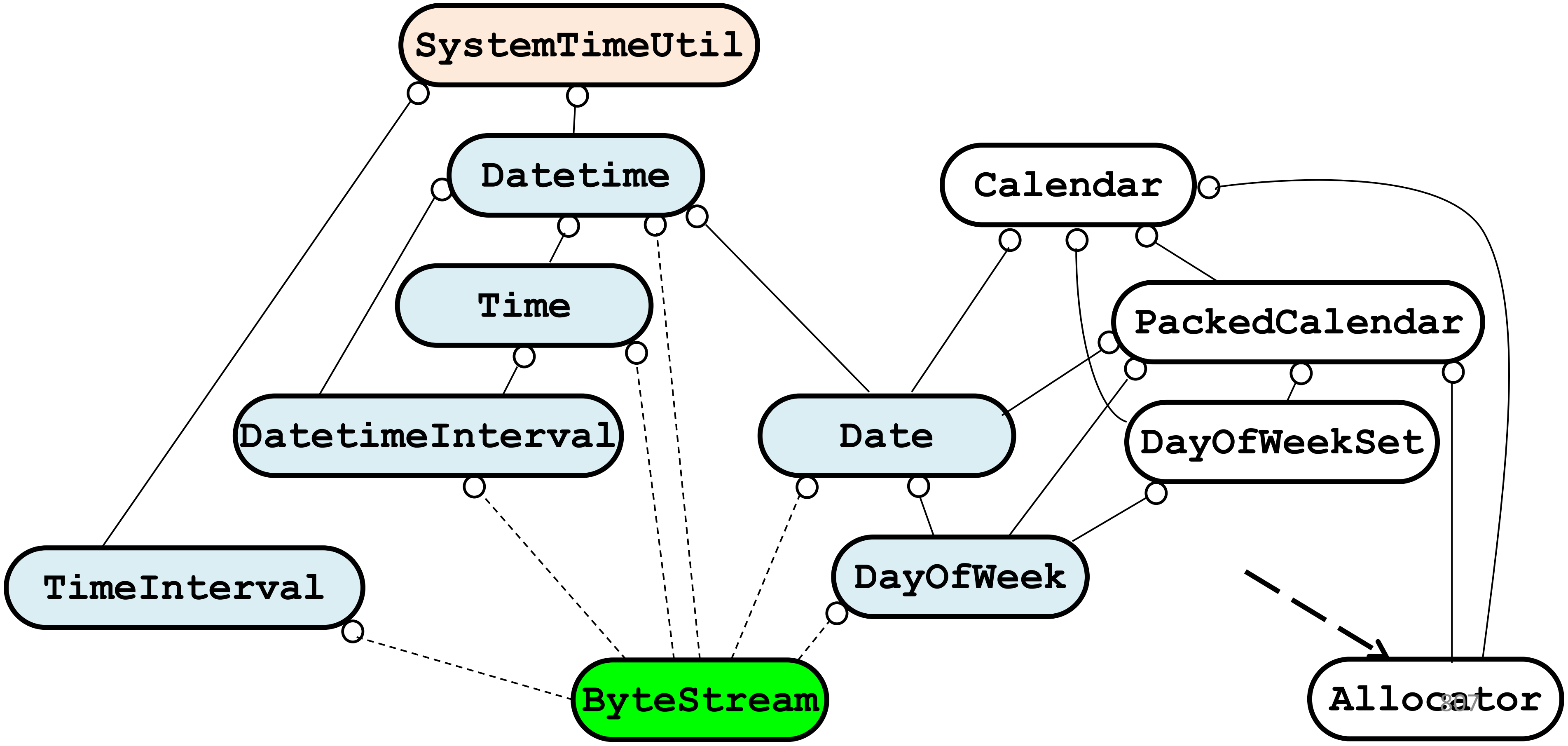


Determine if a Date Value is a *Business Day*

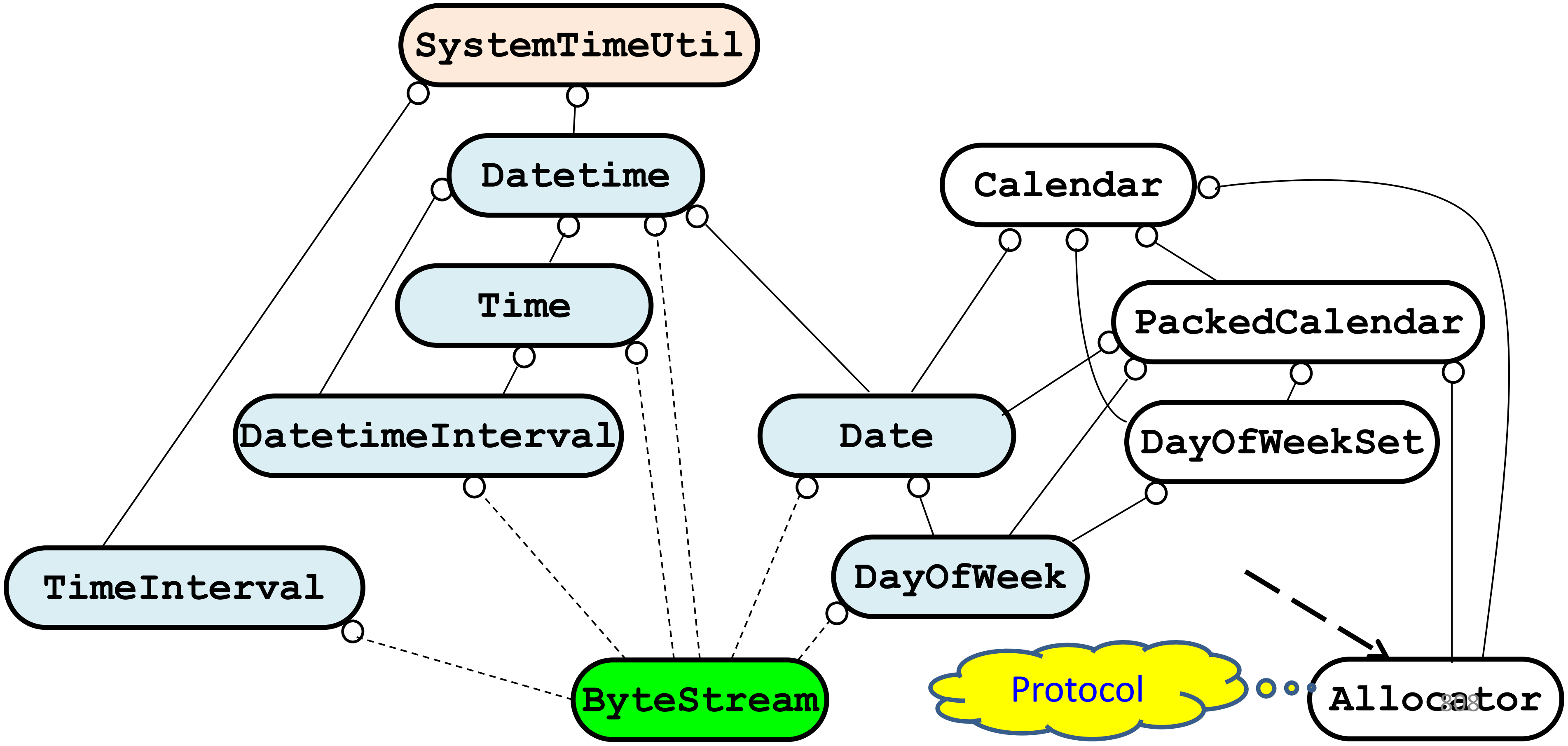


4. Bloomberg Development Environment

Determine if a Date Value is a *Business Day*

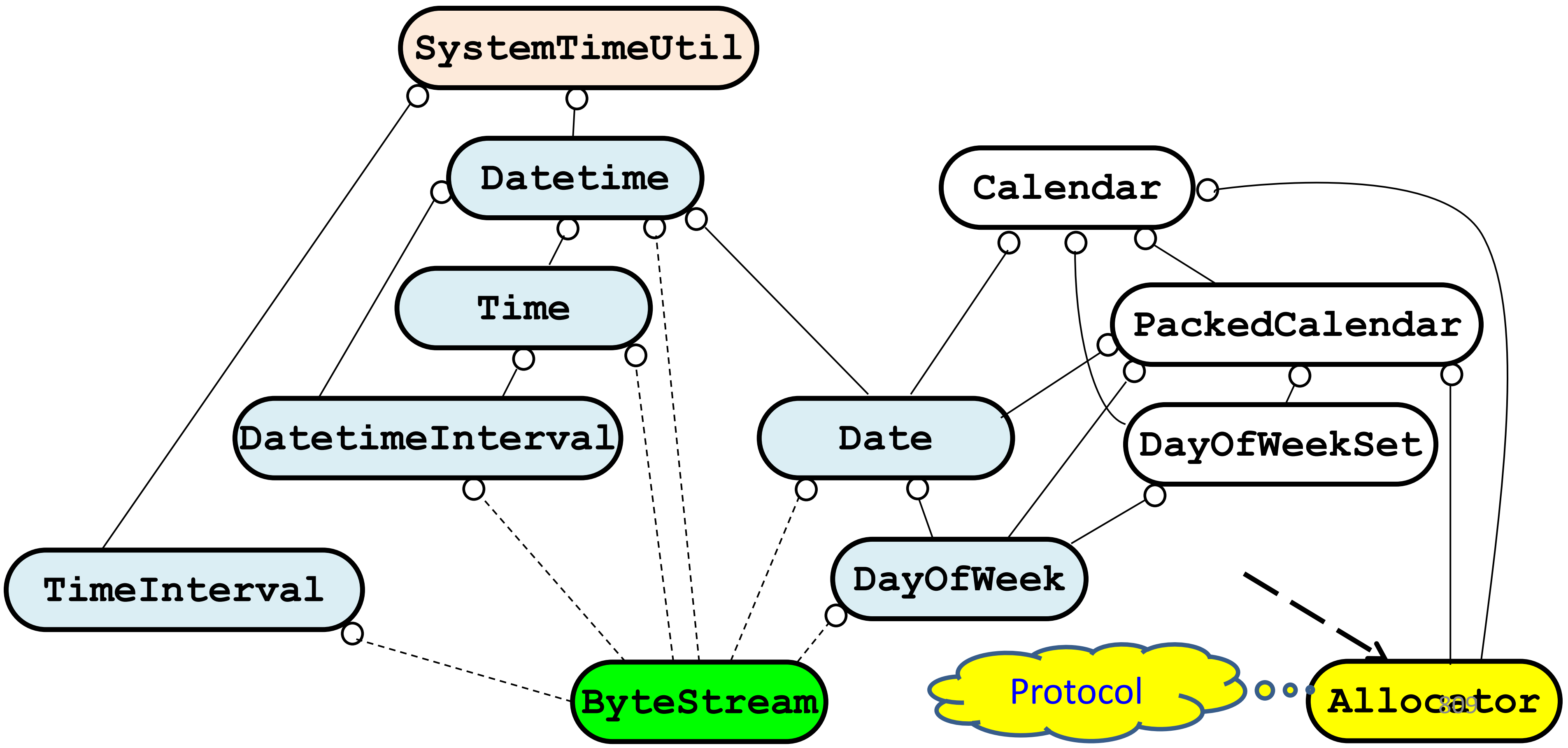


Determine if a Date Value is a *Business Day*



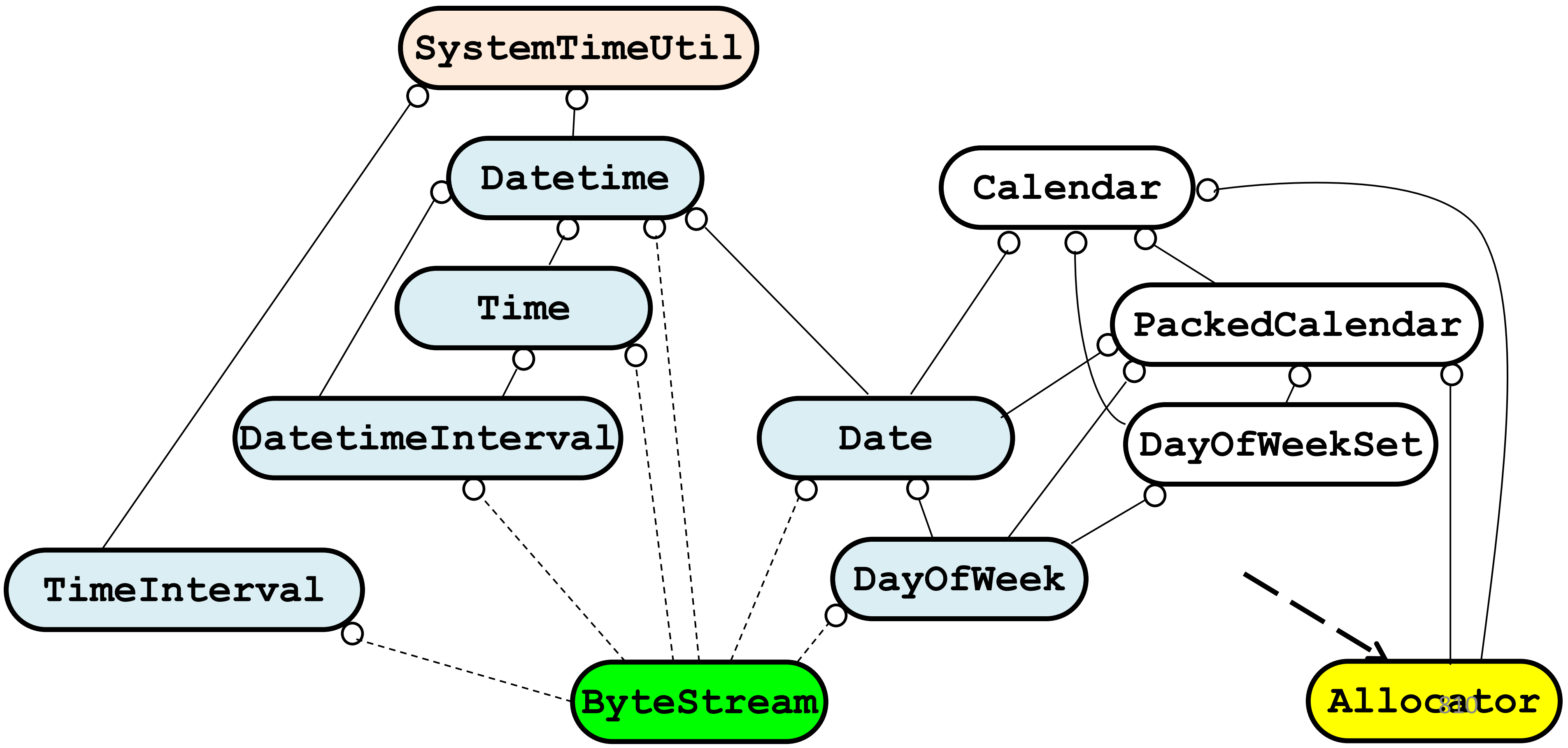
4. Bloomberg Development Environment

Determine if a Date Value is a *Business Day*



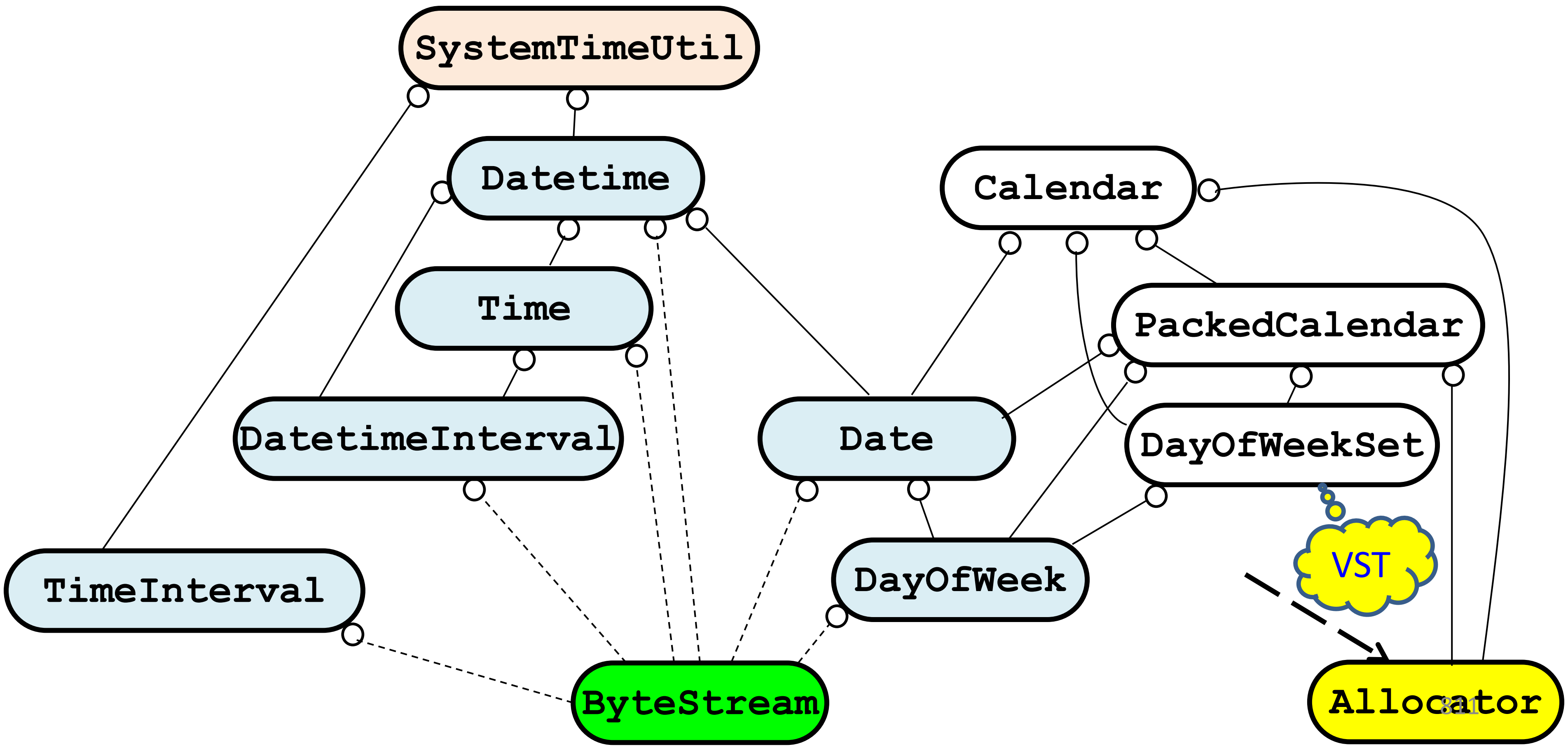
4. Bloomberg Development Environment

Determine if a Date Value is a *Business Day*



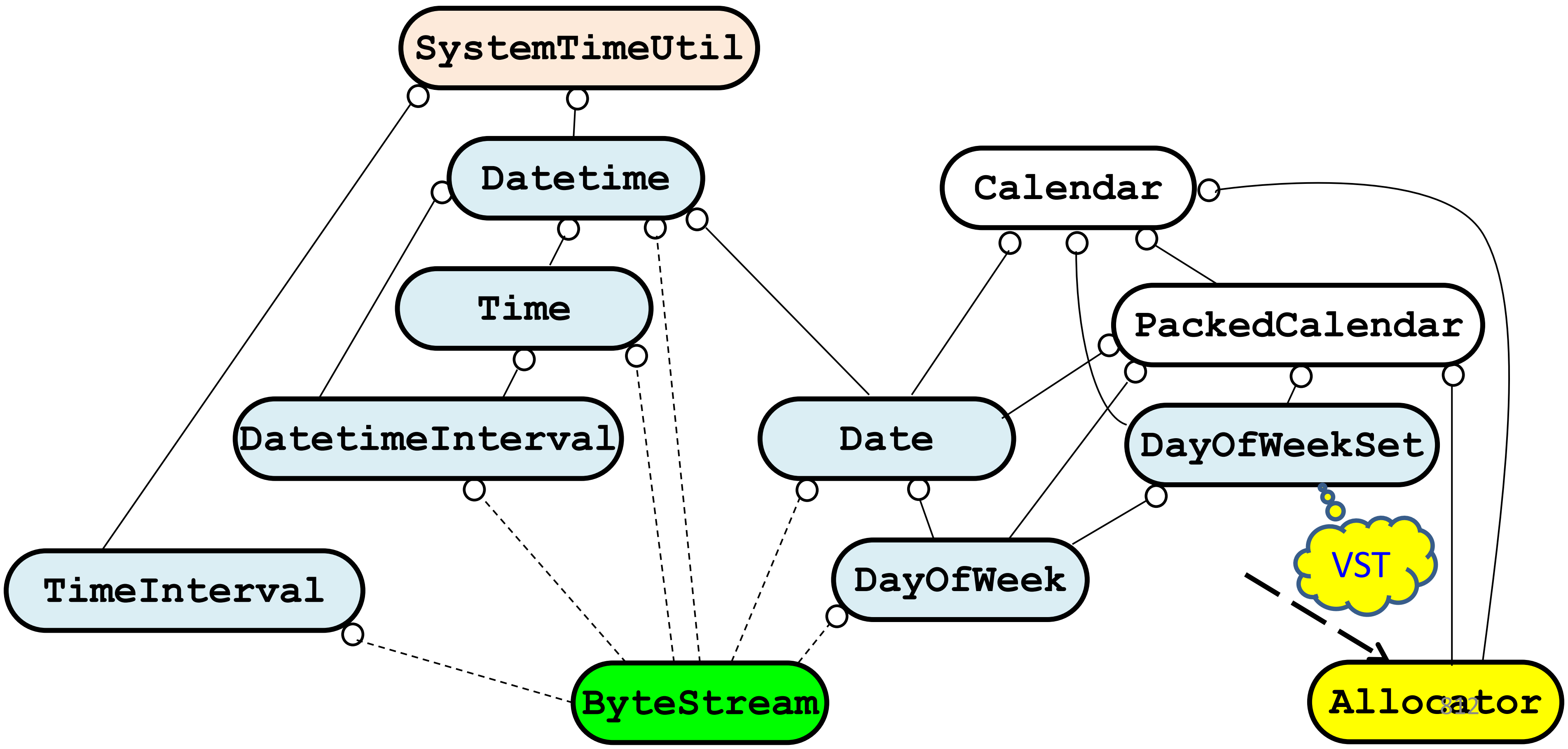
4. Bloomberg Development Environment

Determine if a Date Value is a *Business Day*



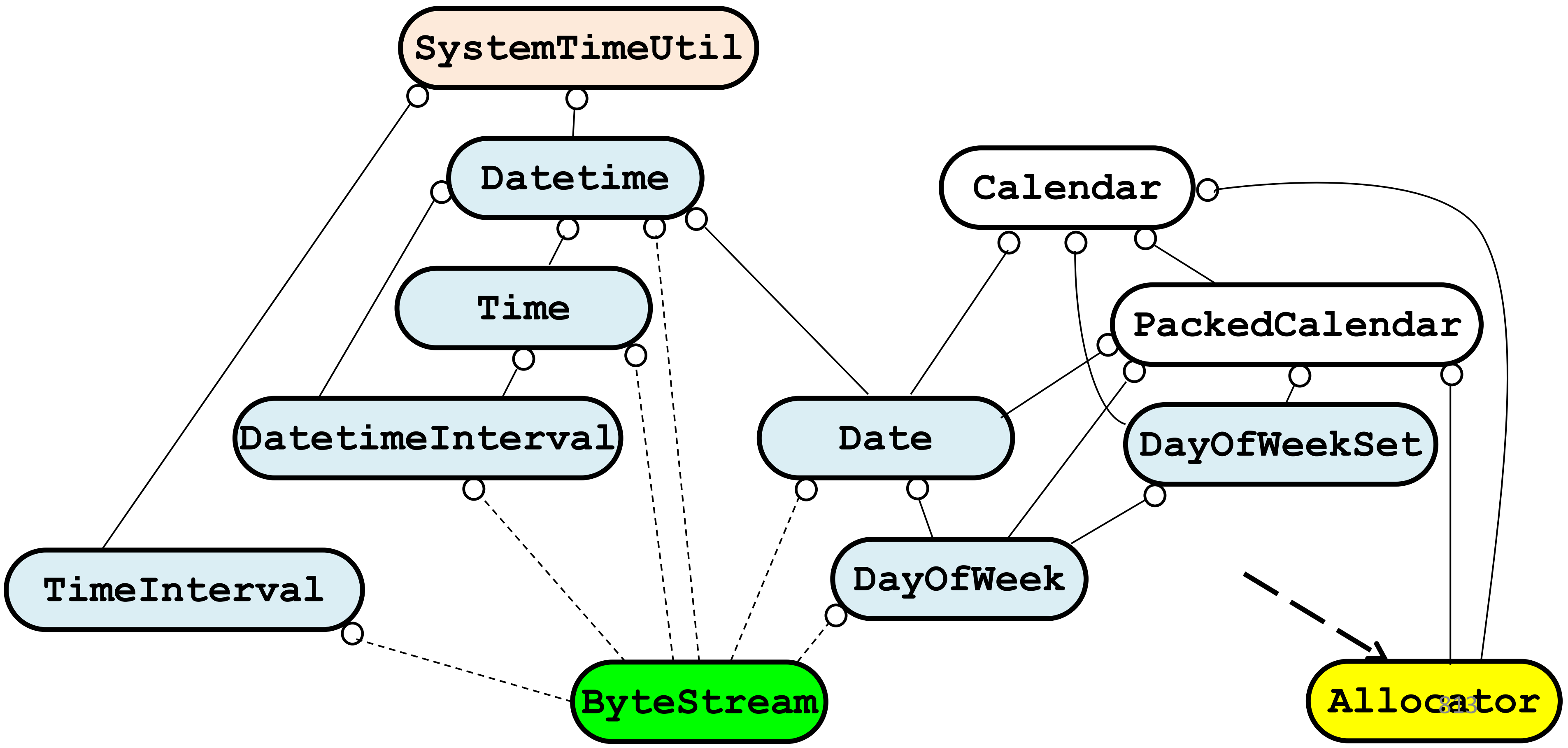
4. Bloomberg Development Environment

Determine if a Date Value is a *Business Day*

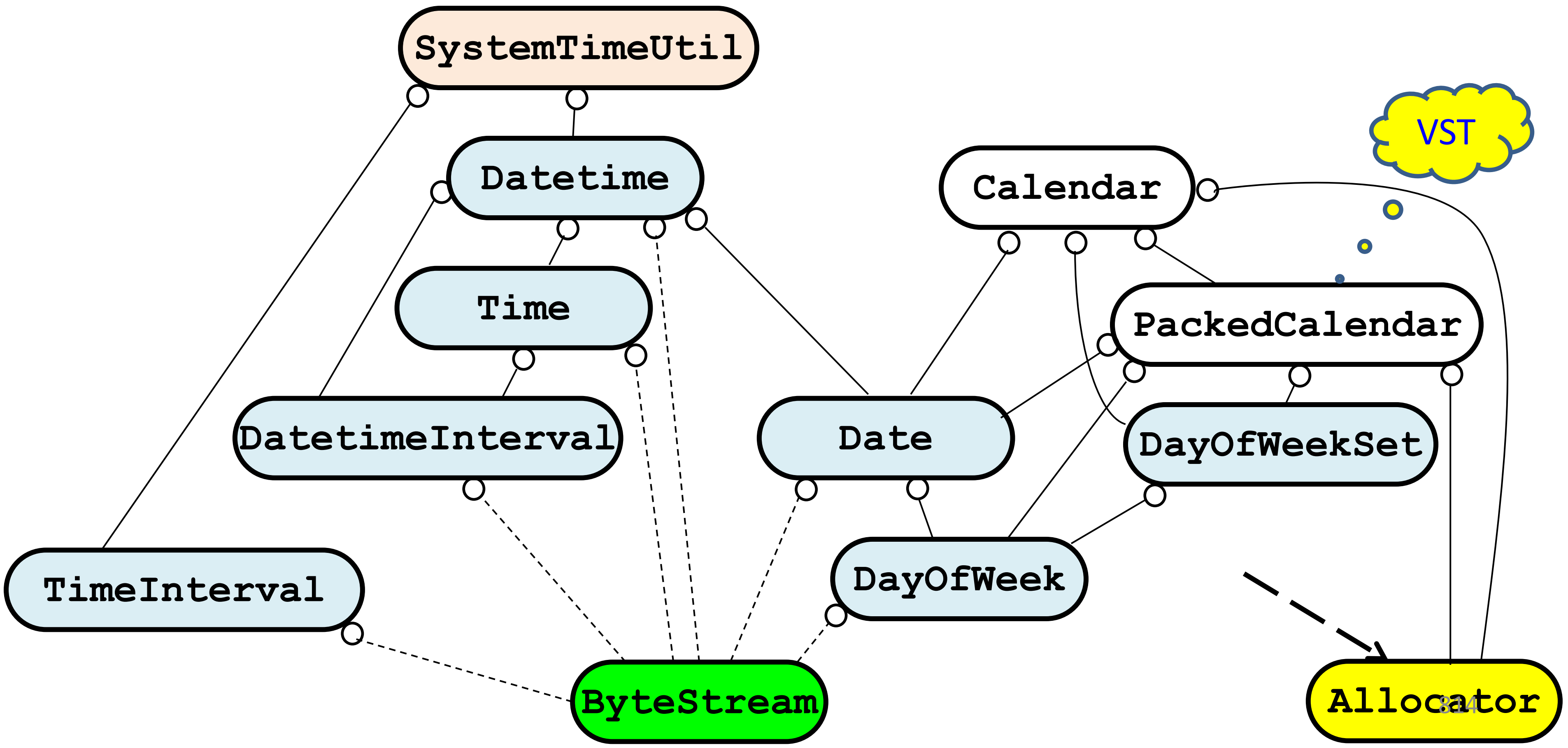


4. Bloomberg Development Environment

Determine if a Date Value is a *Business Day*

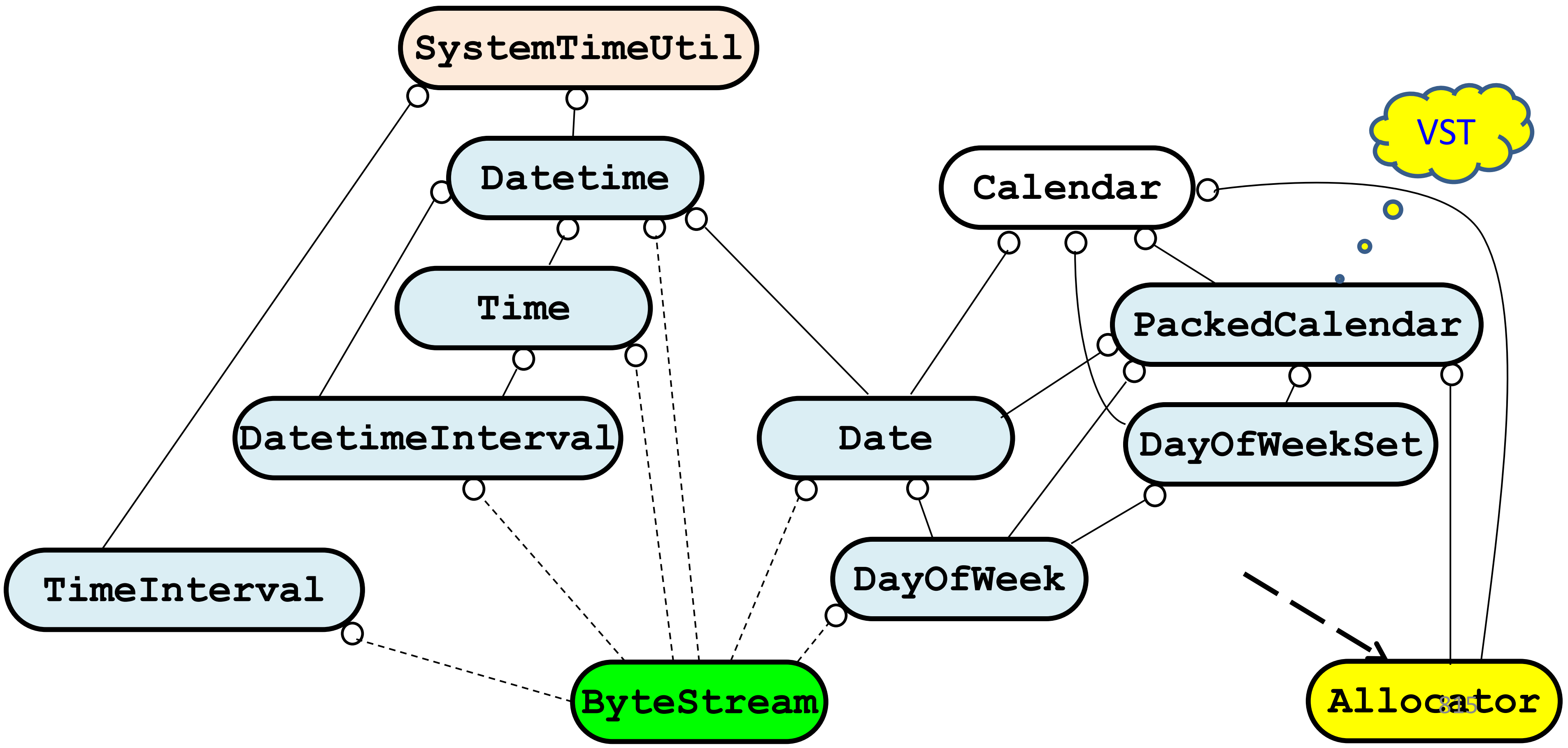


Determine if a Date Value is a *Business Day*



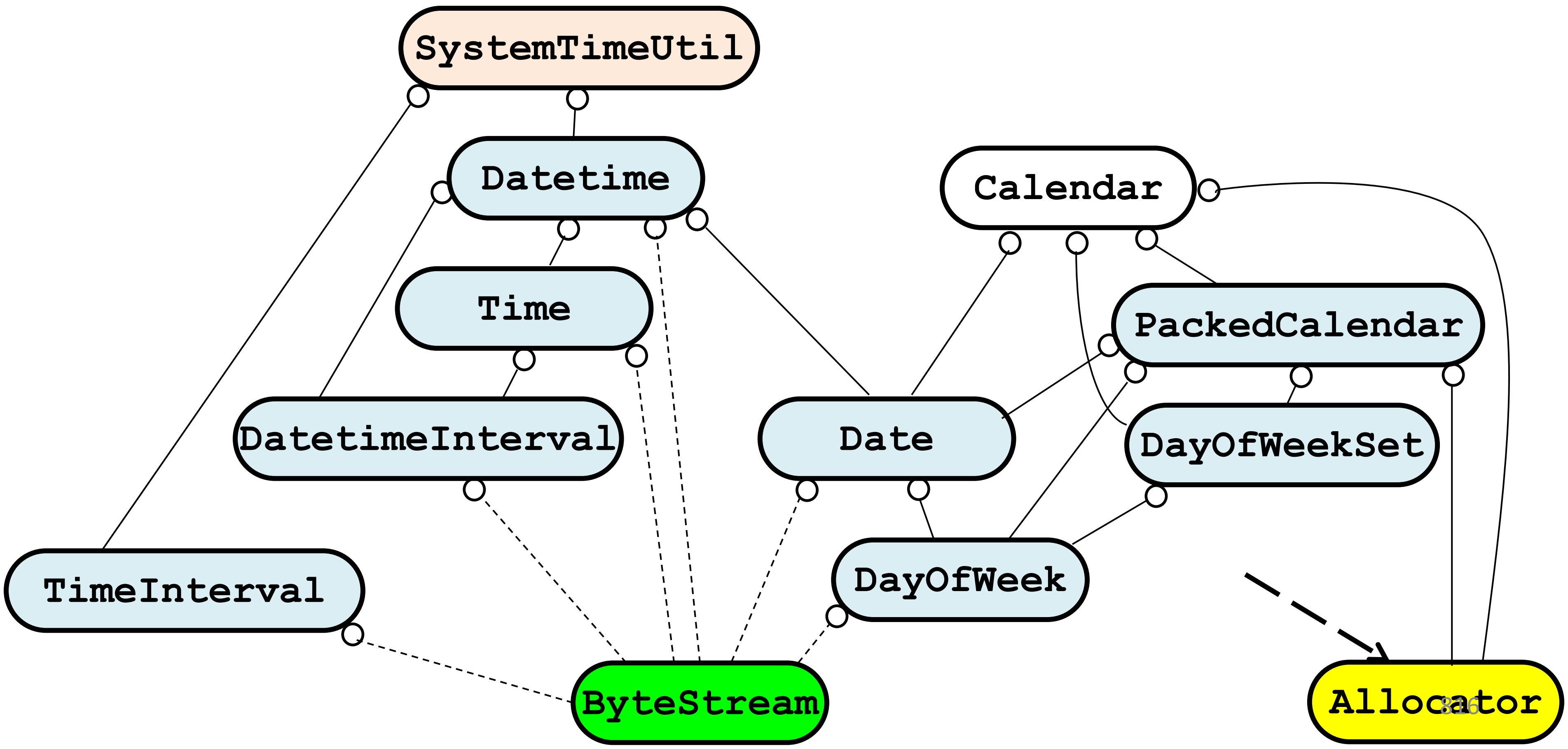
4. Bloomberg Development Environment

Determine if a Date Value is a *Business Day*



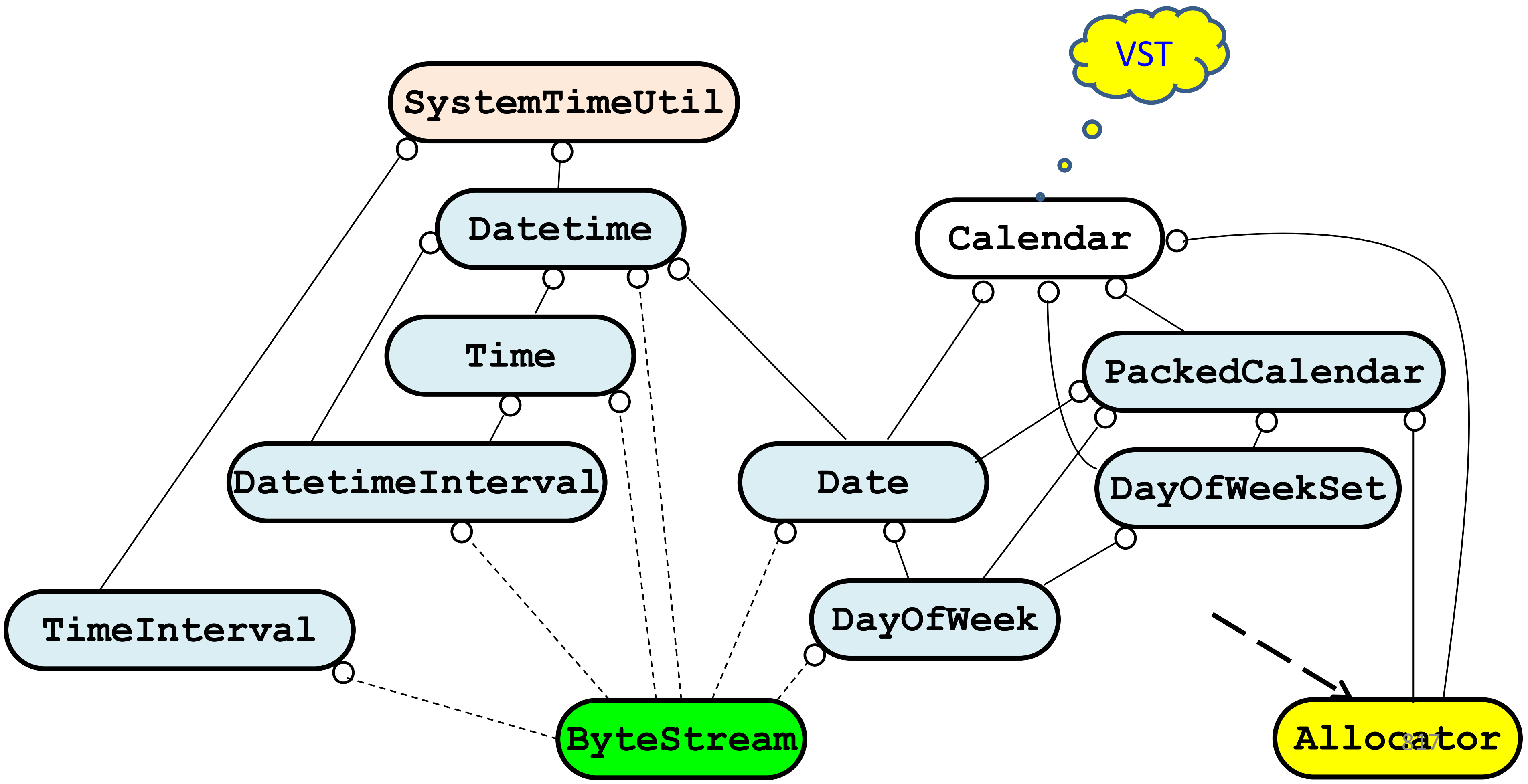
4. Bloomberg Development Environment

Determine if a Date Value is a *Business Day*



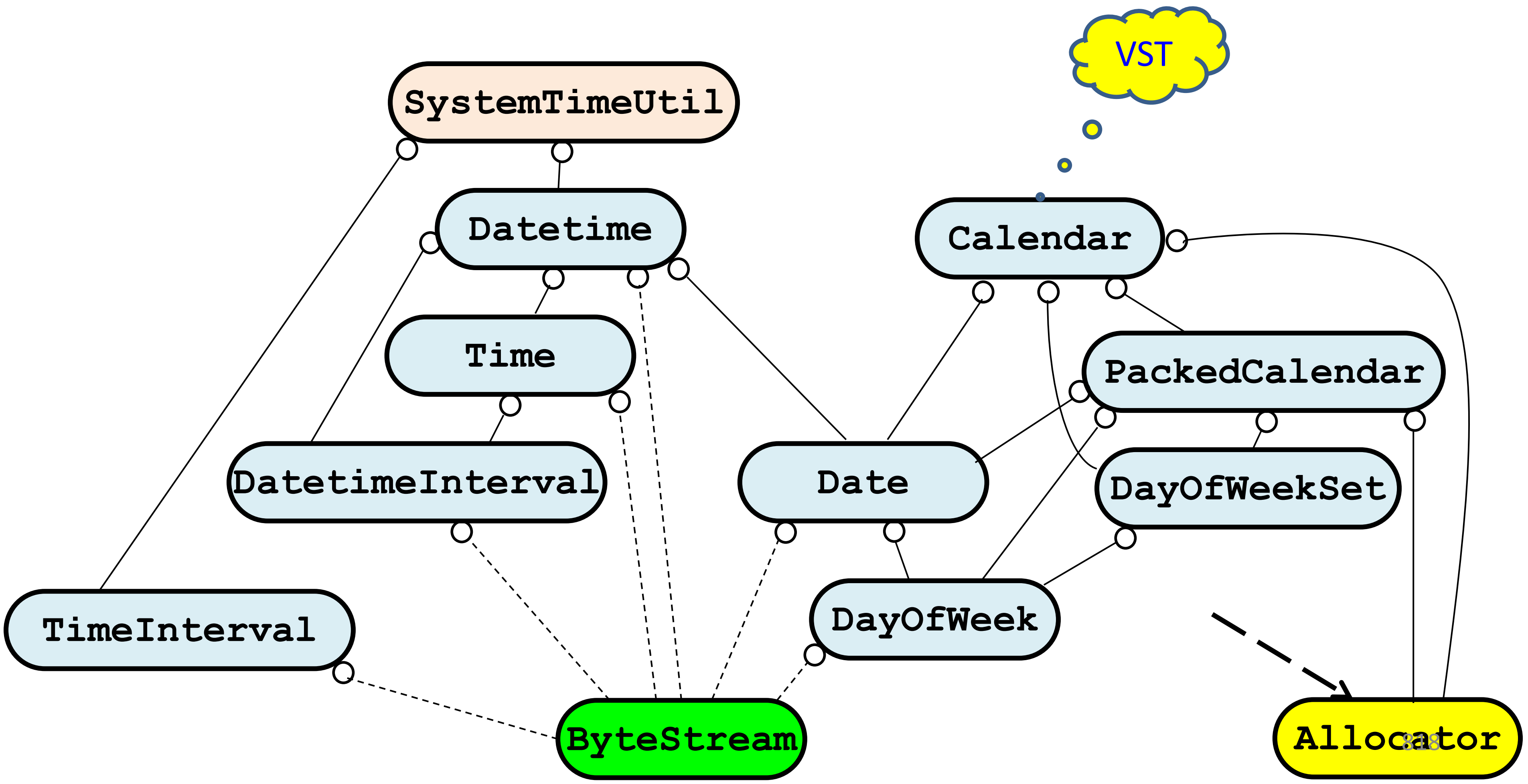
4. Bloomberg Development Environment

Determine if a Date Value is a *Business Day*



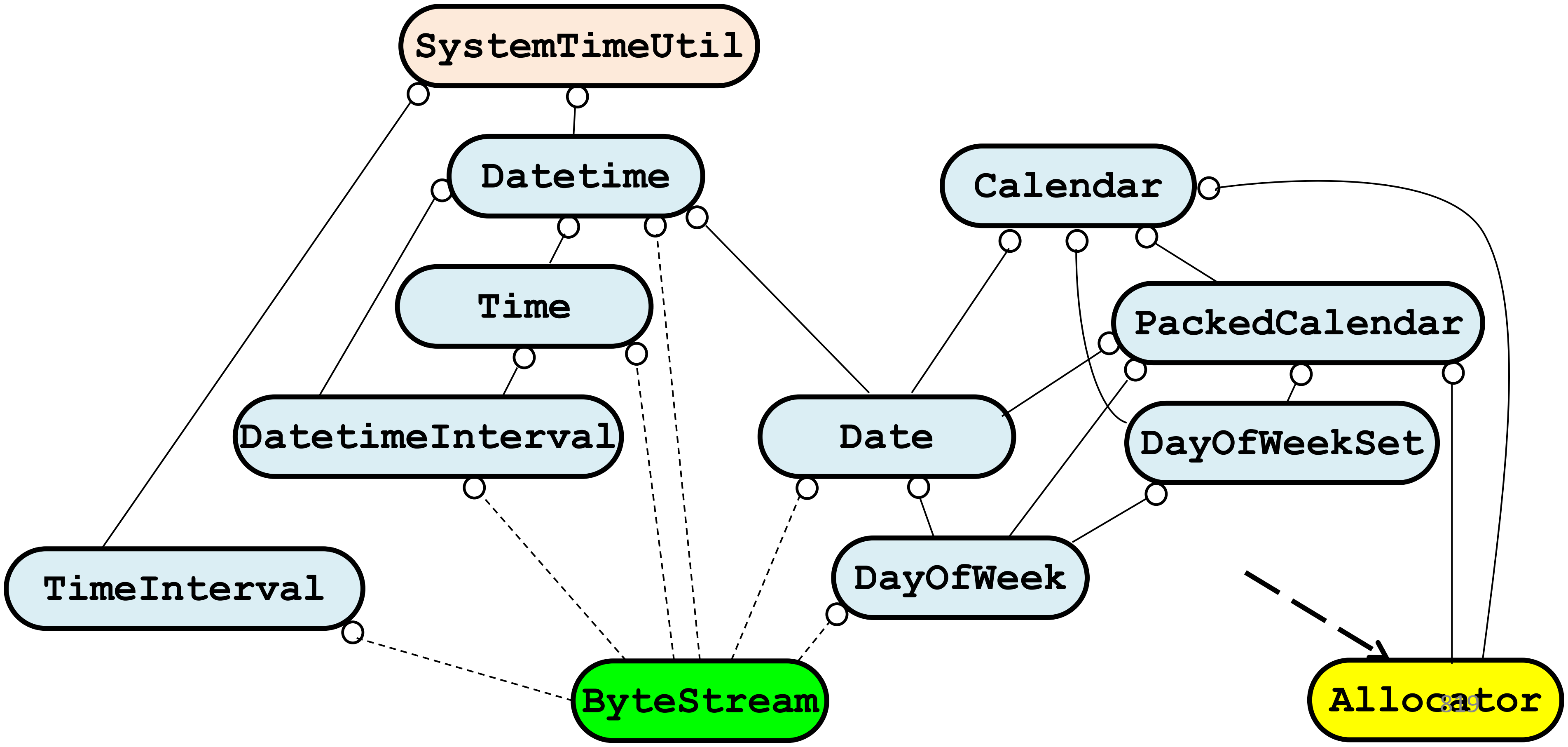
4. Bloomberg Development Environment

Determine if a Date Value is a *Business Day*



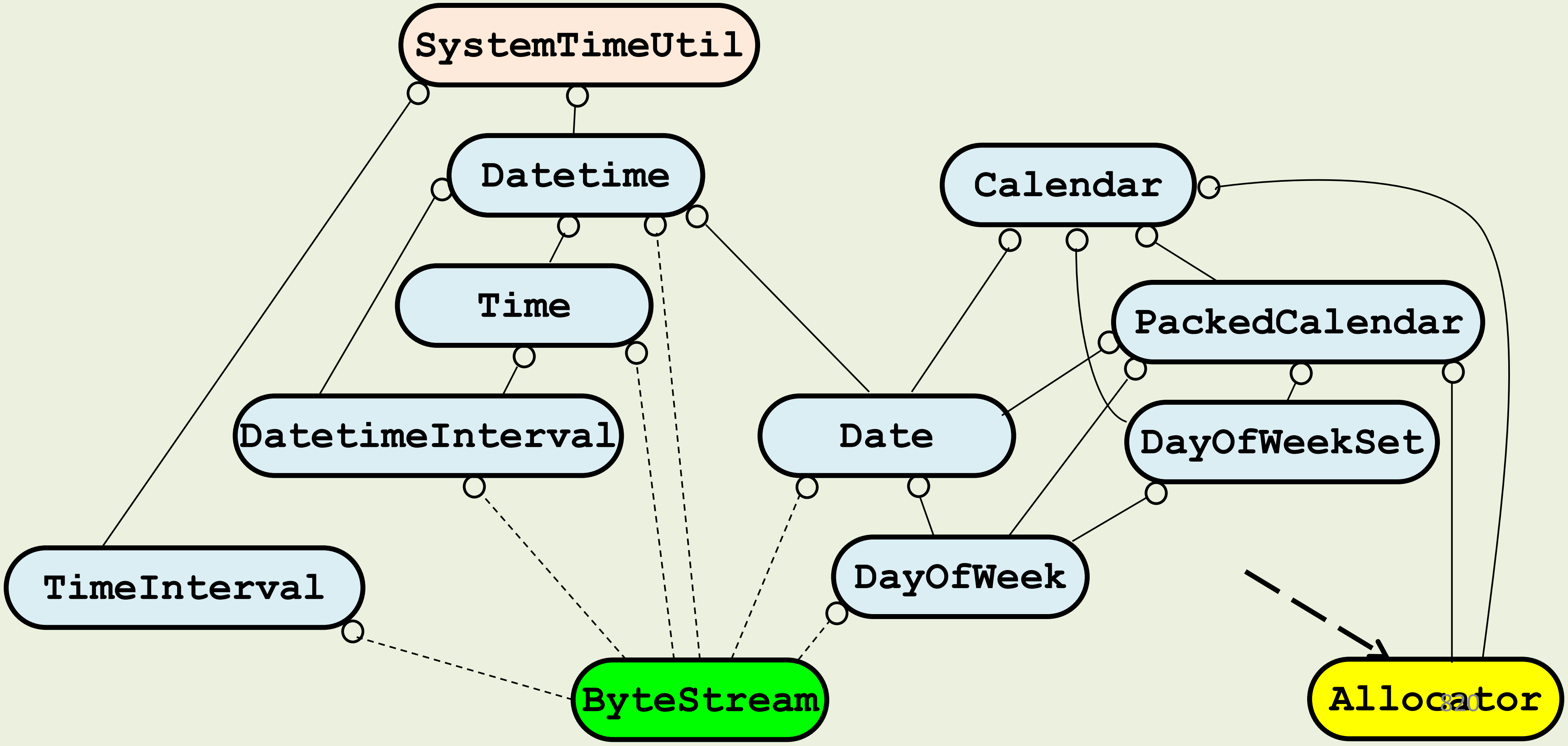
4. Bloomberg Development Environment

Determine if a Date Value is a *Business Day*

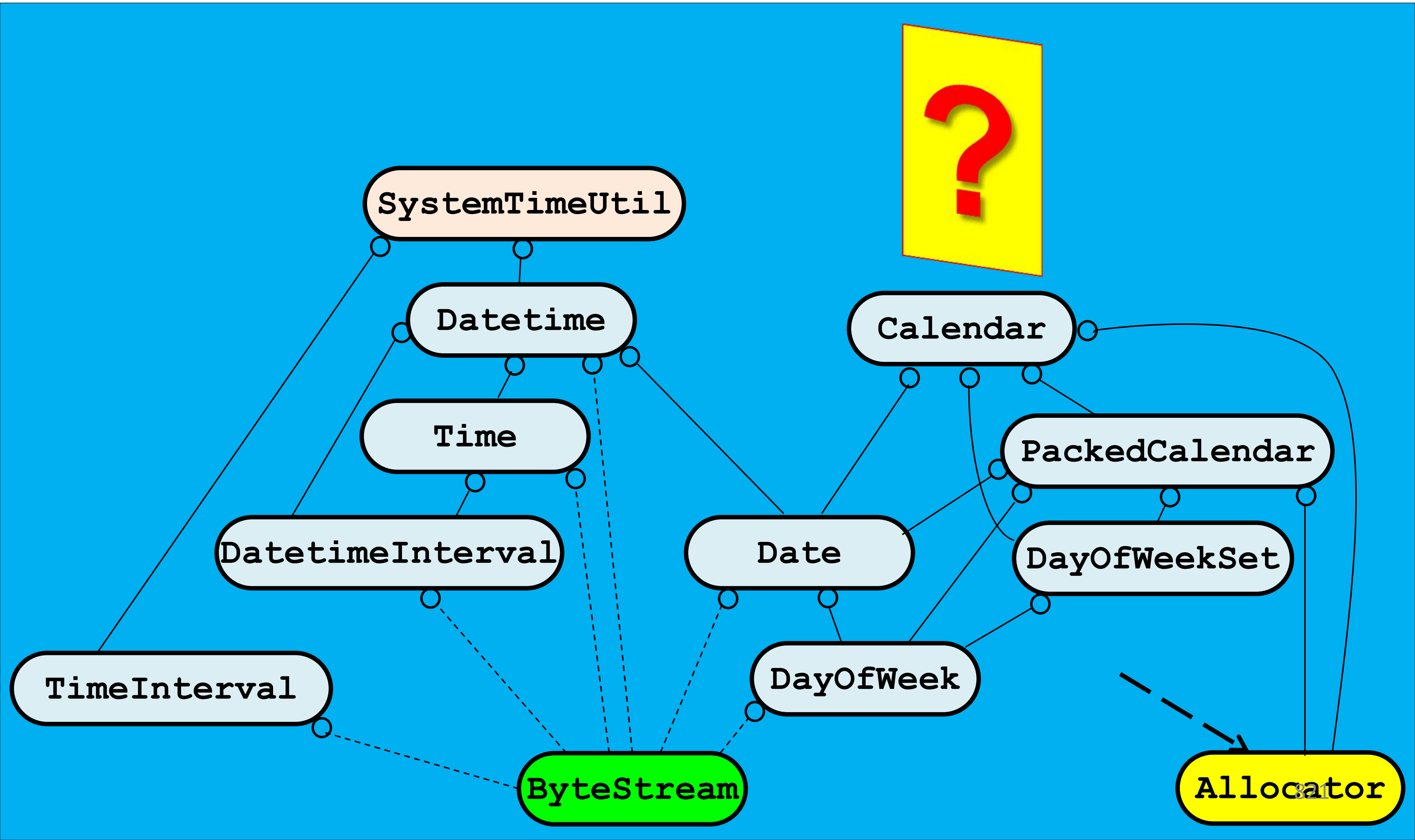


4. Bloomberg Development Environment

Determine if a Date Value is a *Business Day*

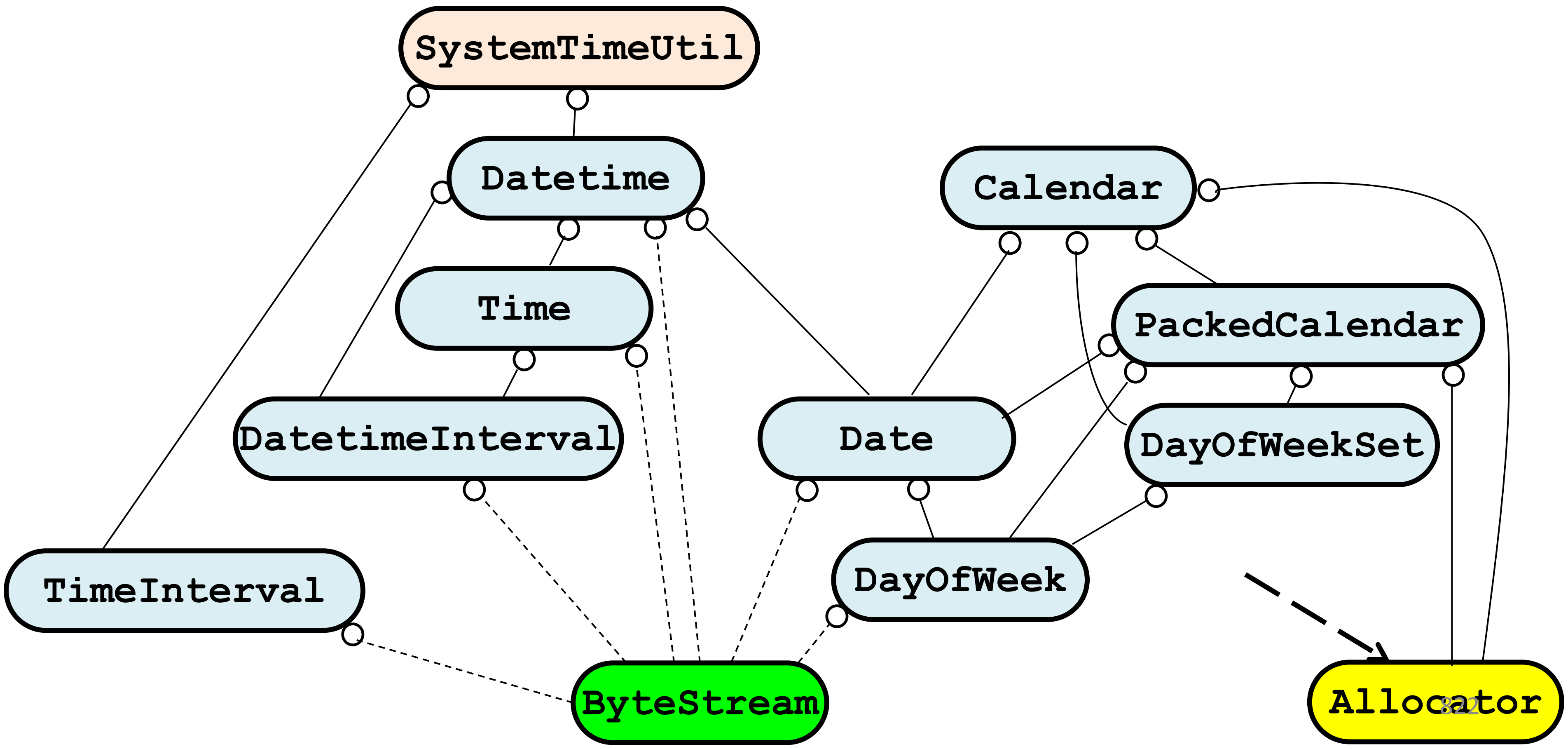


Wait a Minute: Where is the Data Source?

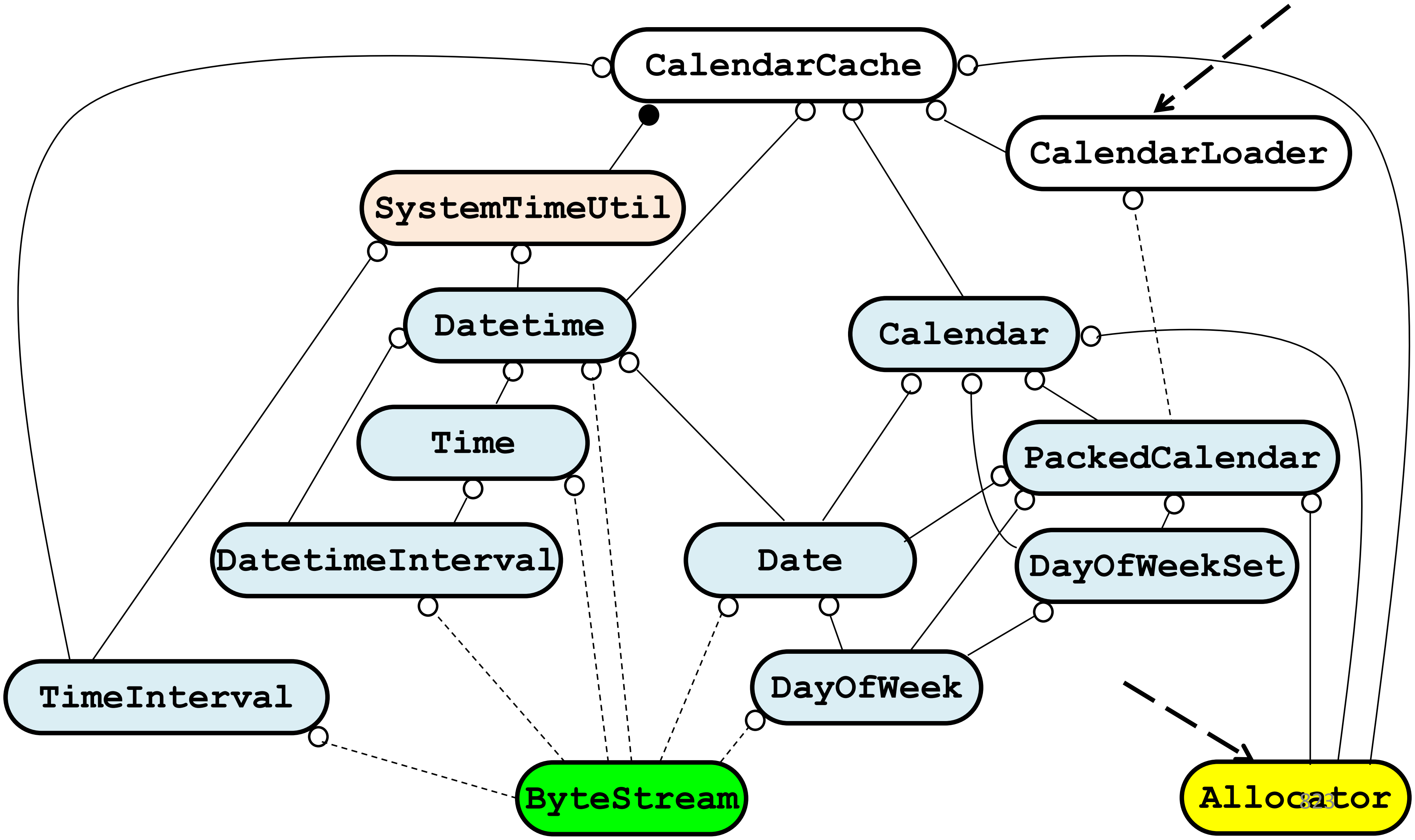


4. Bloomberg Development Environment

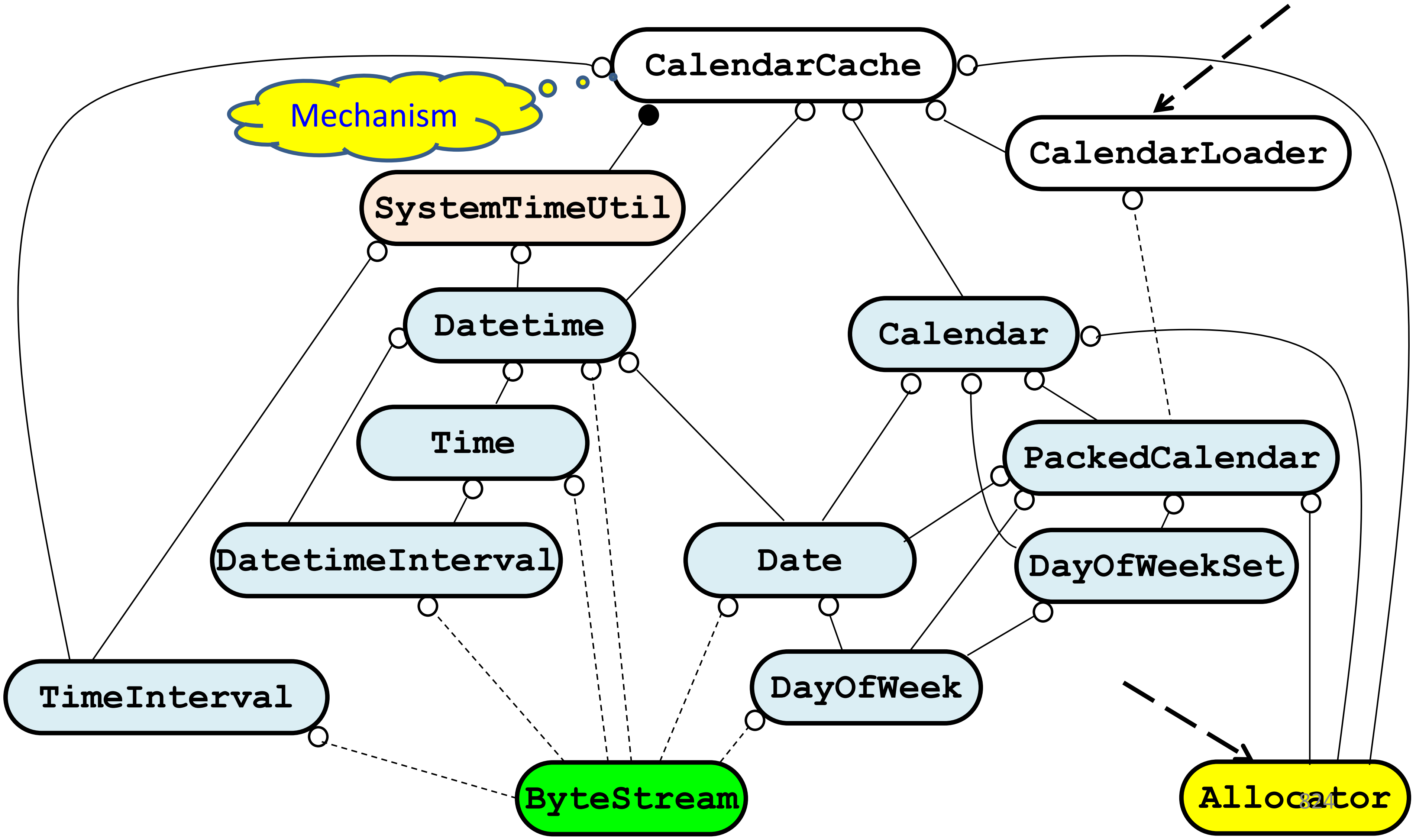
Wait a Minute: Where is the Data Source?



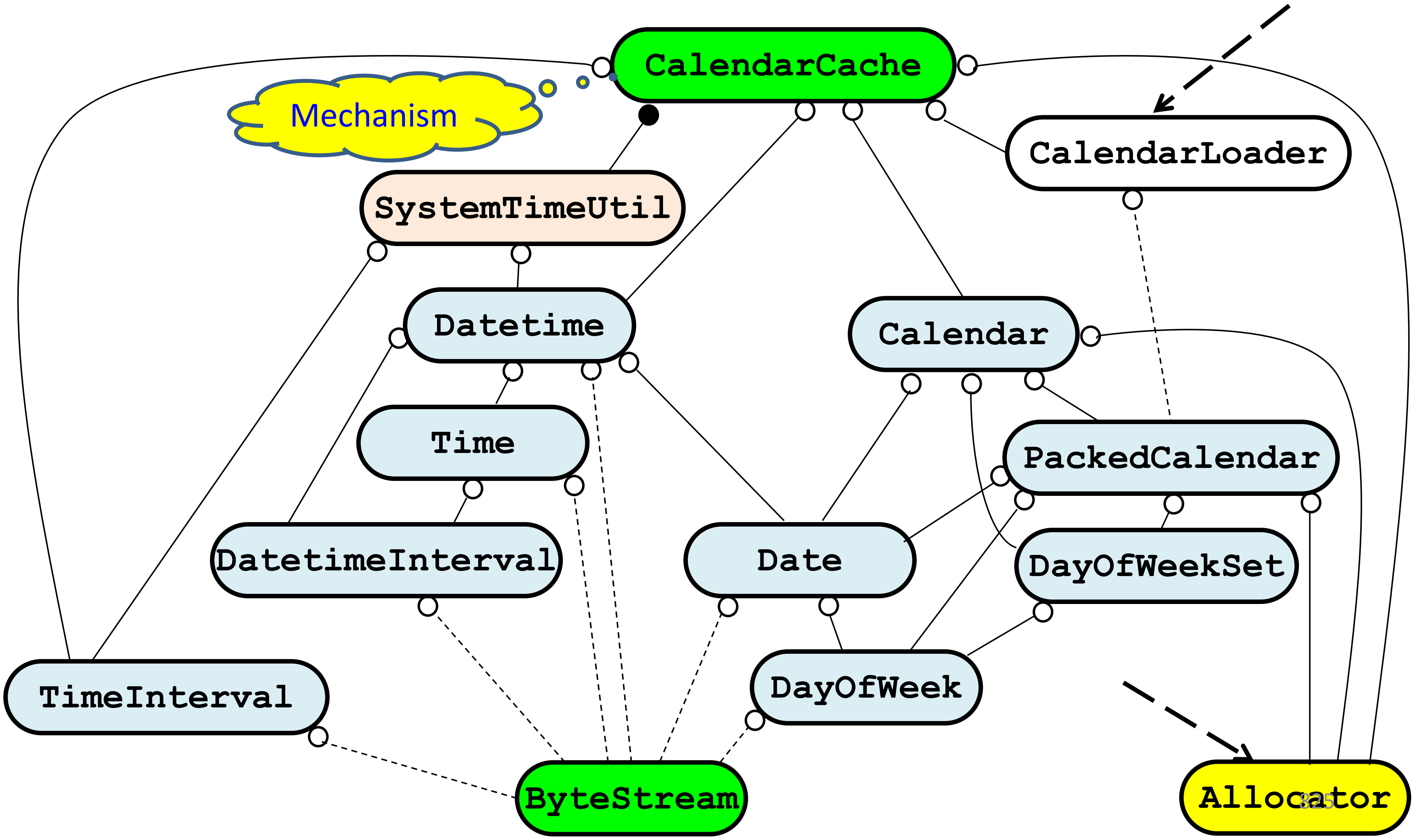
Wait a Minute: Where is the Data Source?



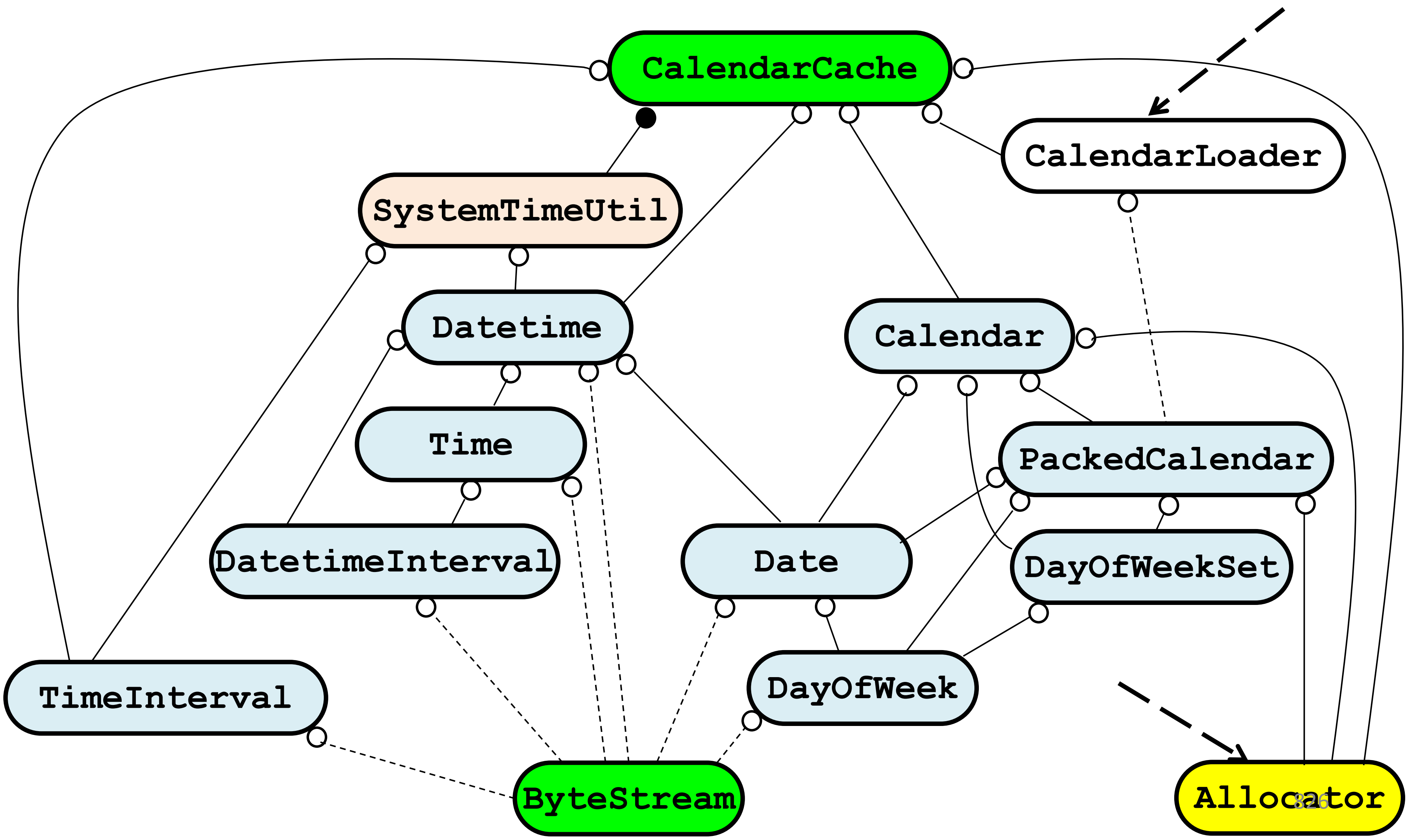
Wait a Minute: Where is the Data Source?



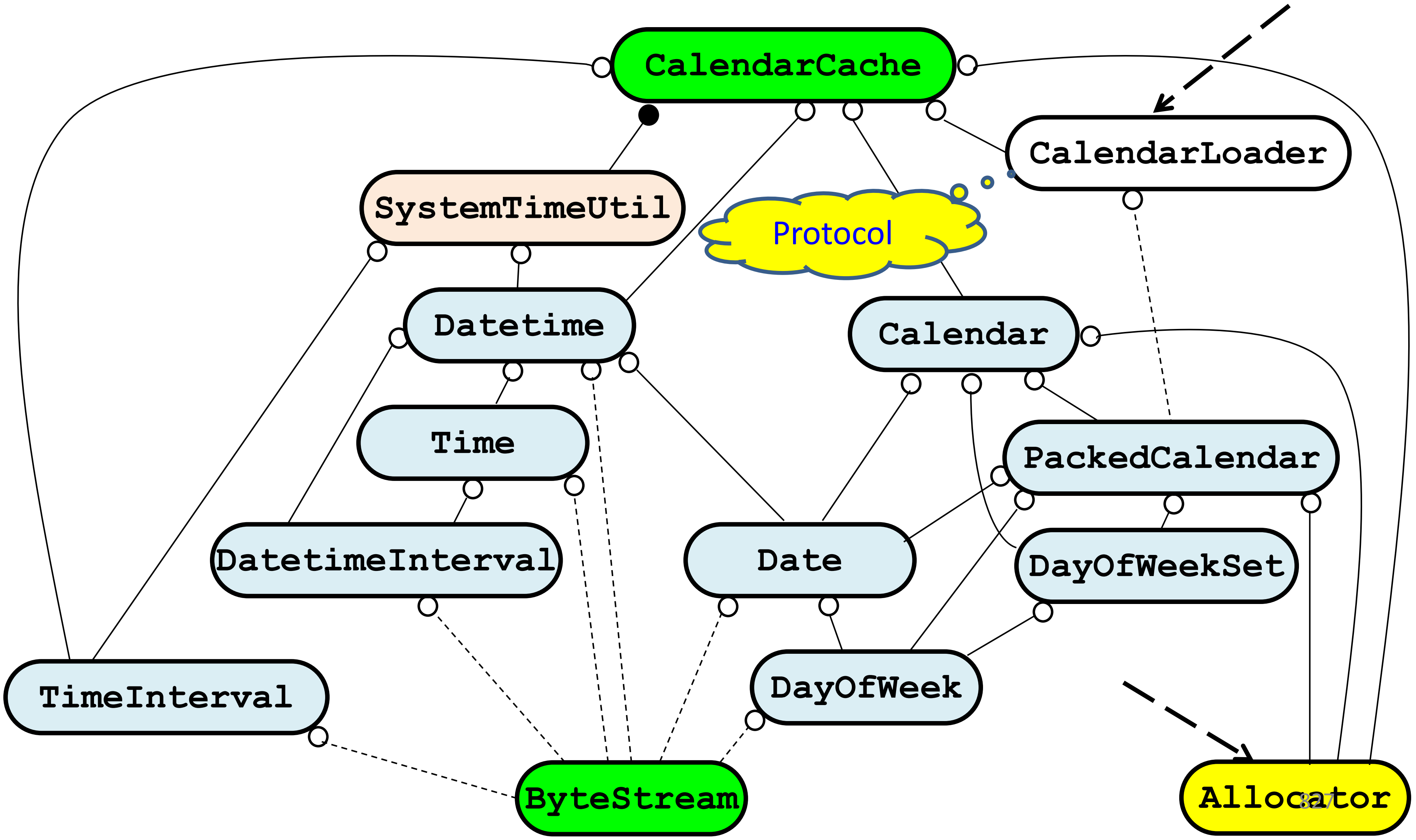
Wait a Minute: Where is the Data Source?



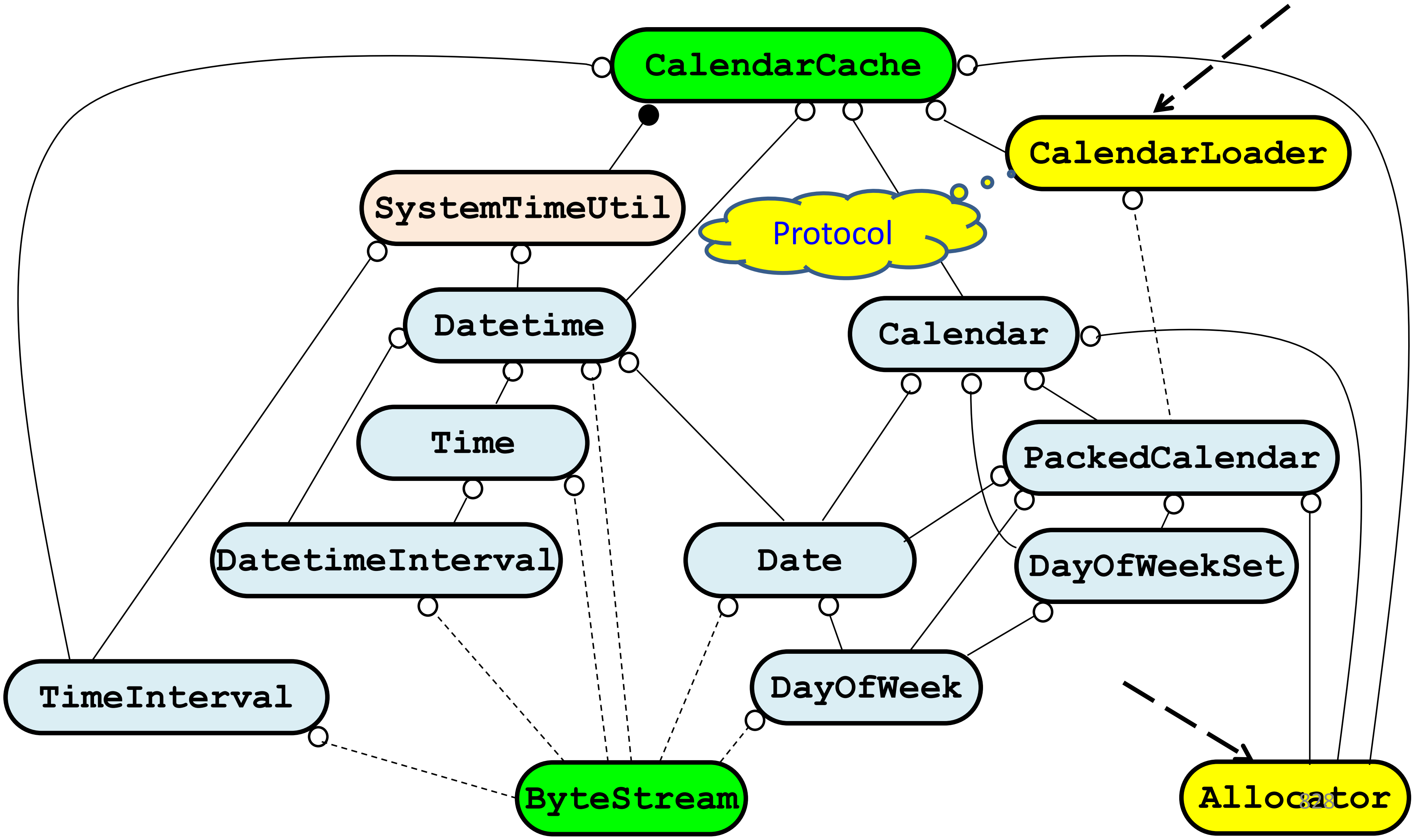
Wait a Minute: Where is the Data Source?



Wait a Minute: Where is the Data Source?

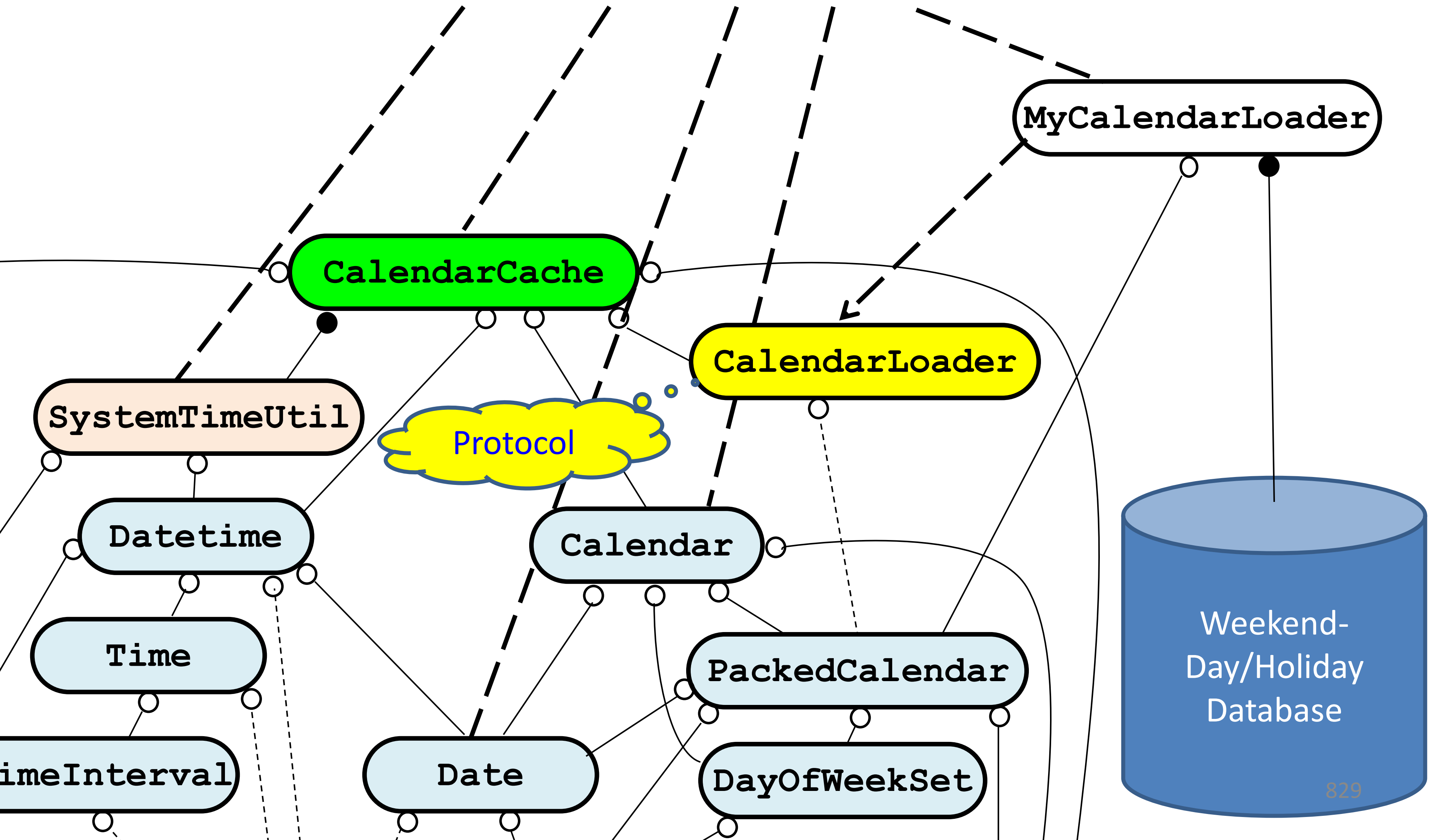


Wait a Minute: Where is the Data Source?

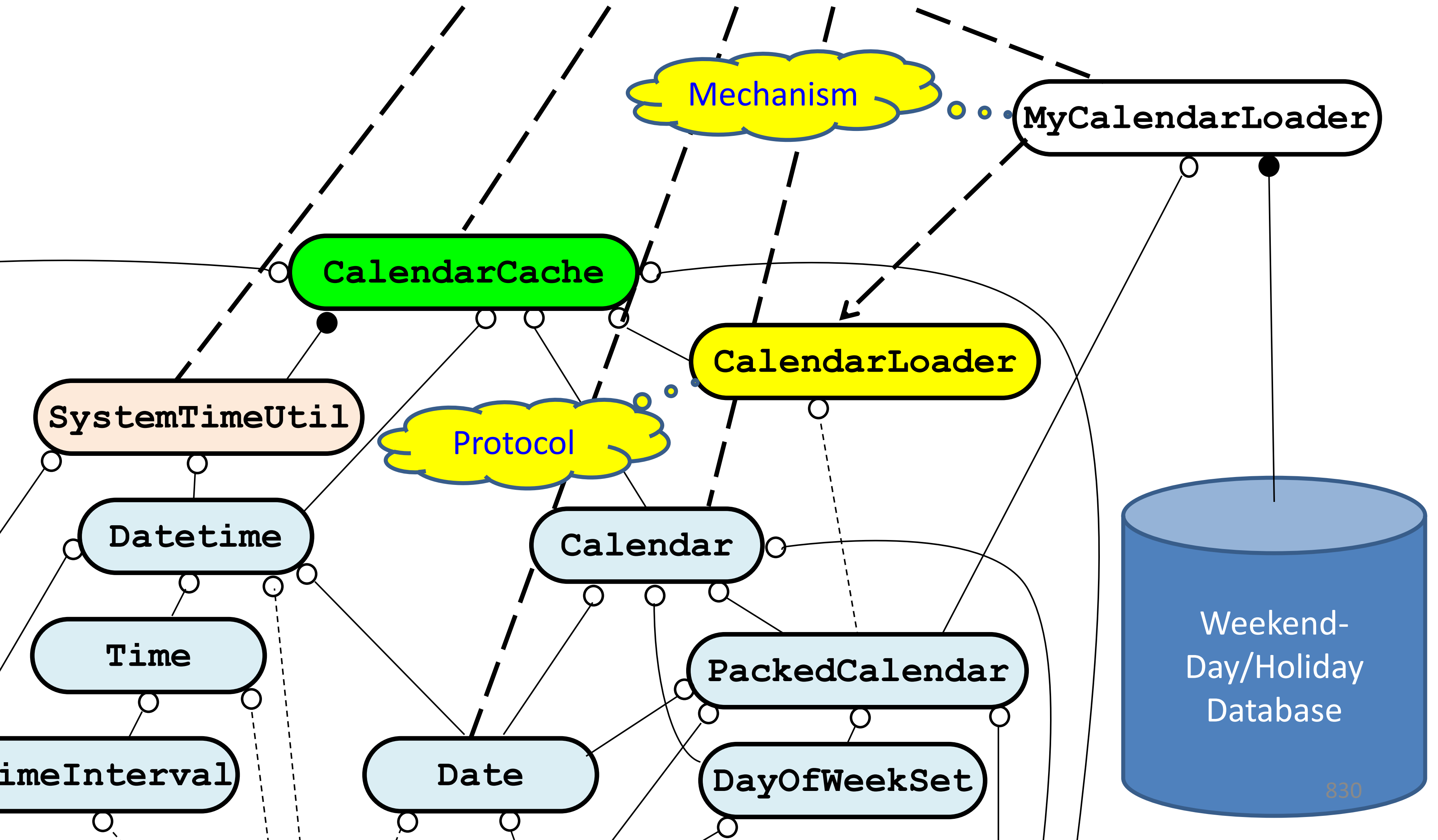


4. Bloomberg Development Environment

Wait a Minute: Where is the Data Source?

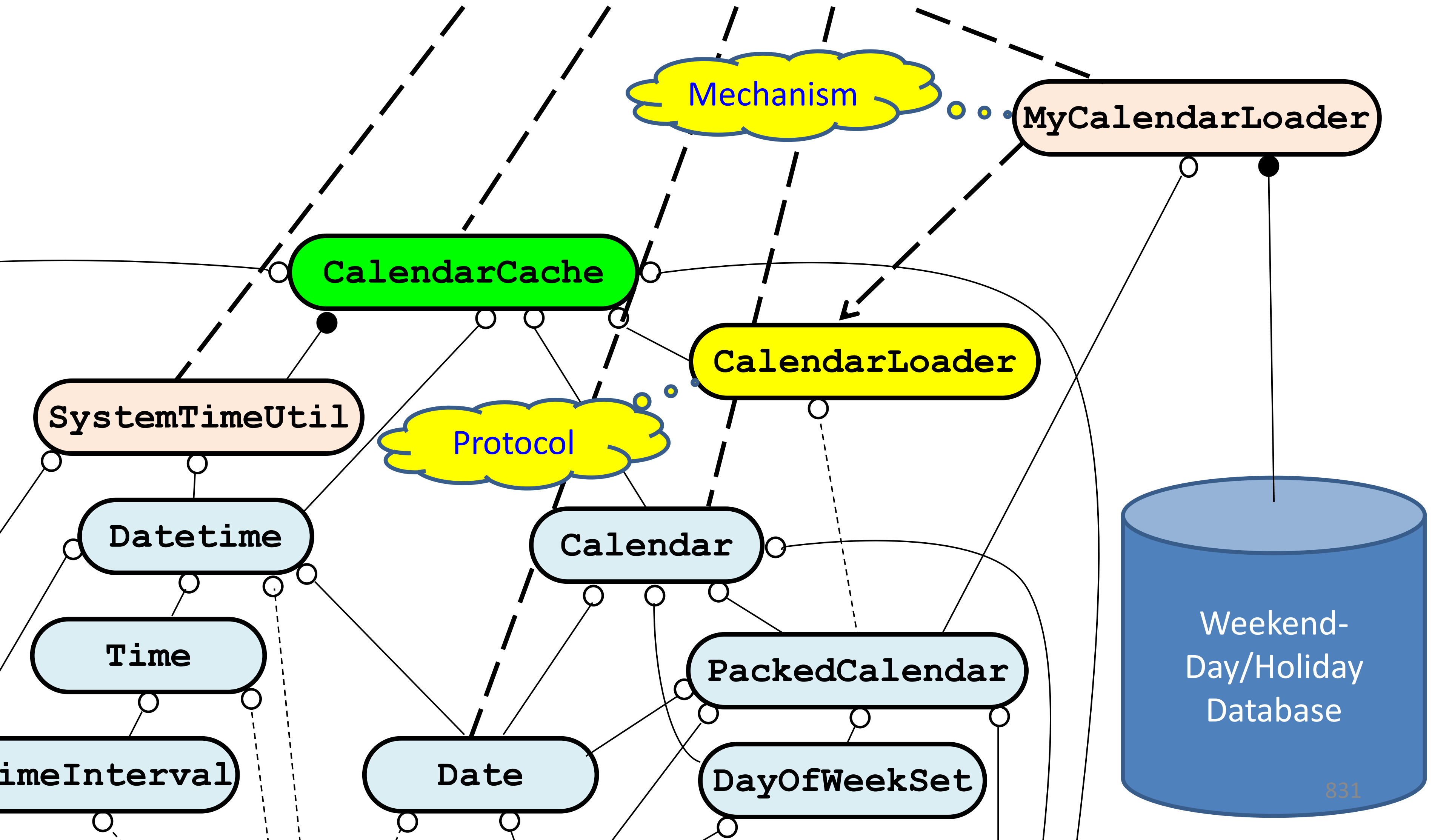


Wait a Minute: Where is the Data Source?



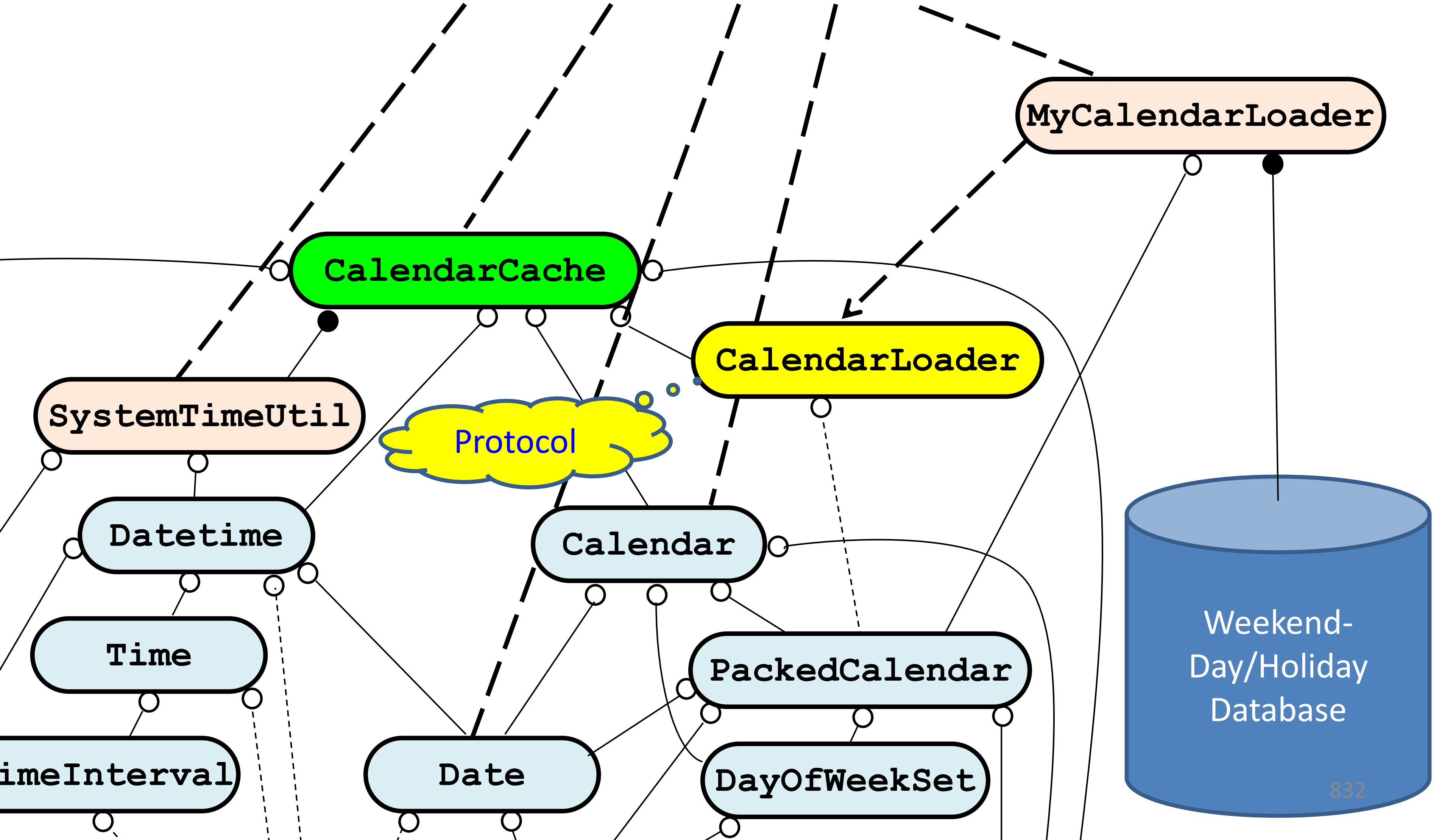
4. Bloomberg Development Environment

Wait a Minute: Where is the Data Source?

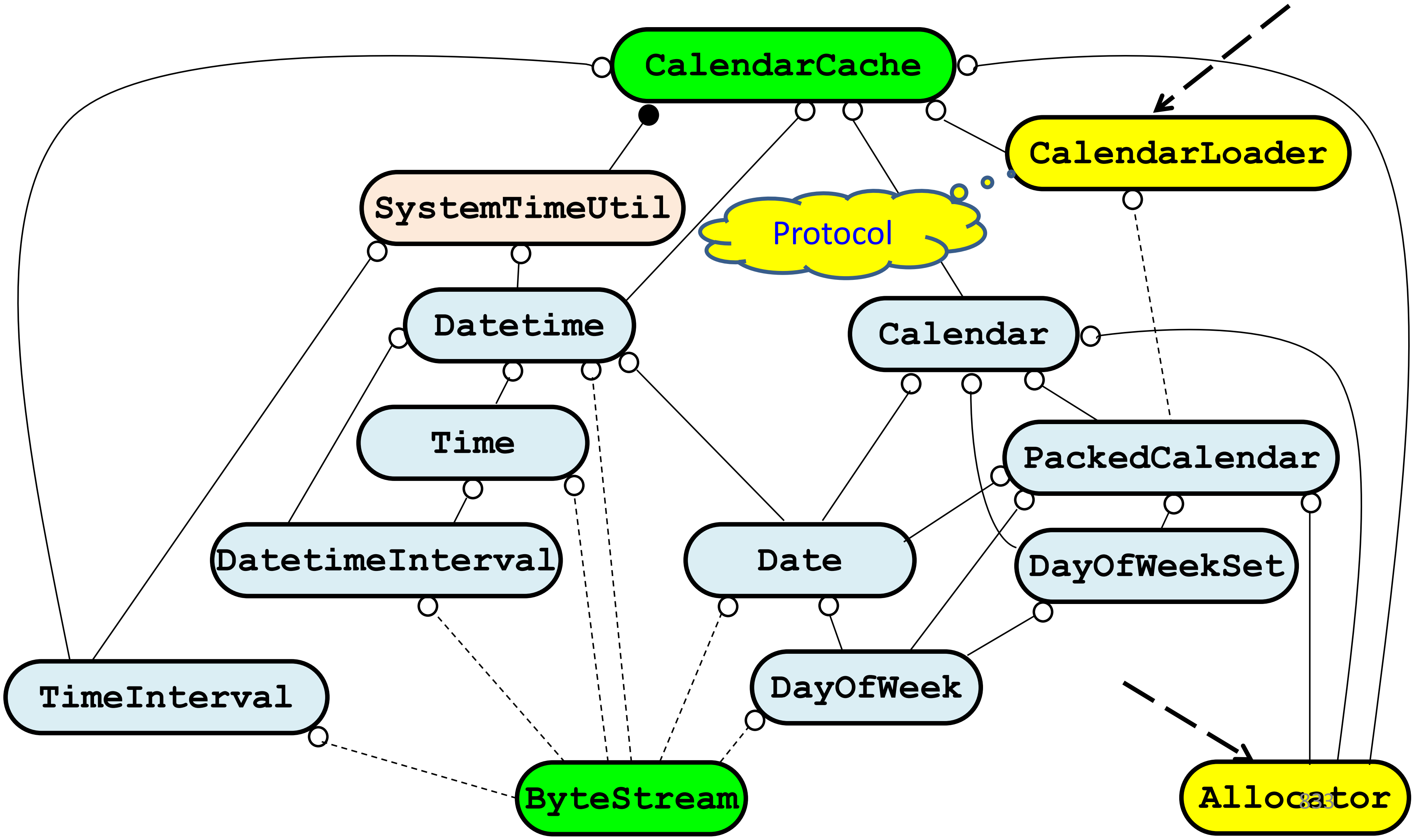


4. Bloomberg Development Environment

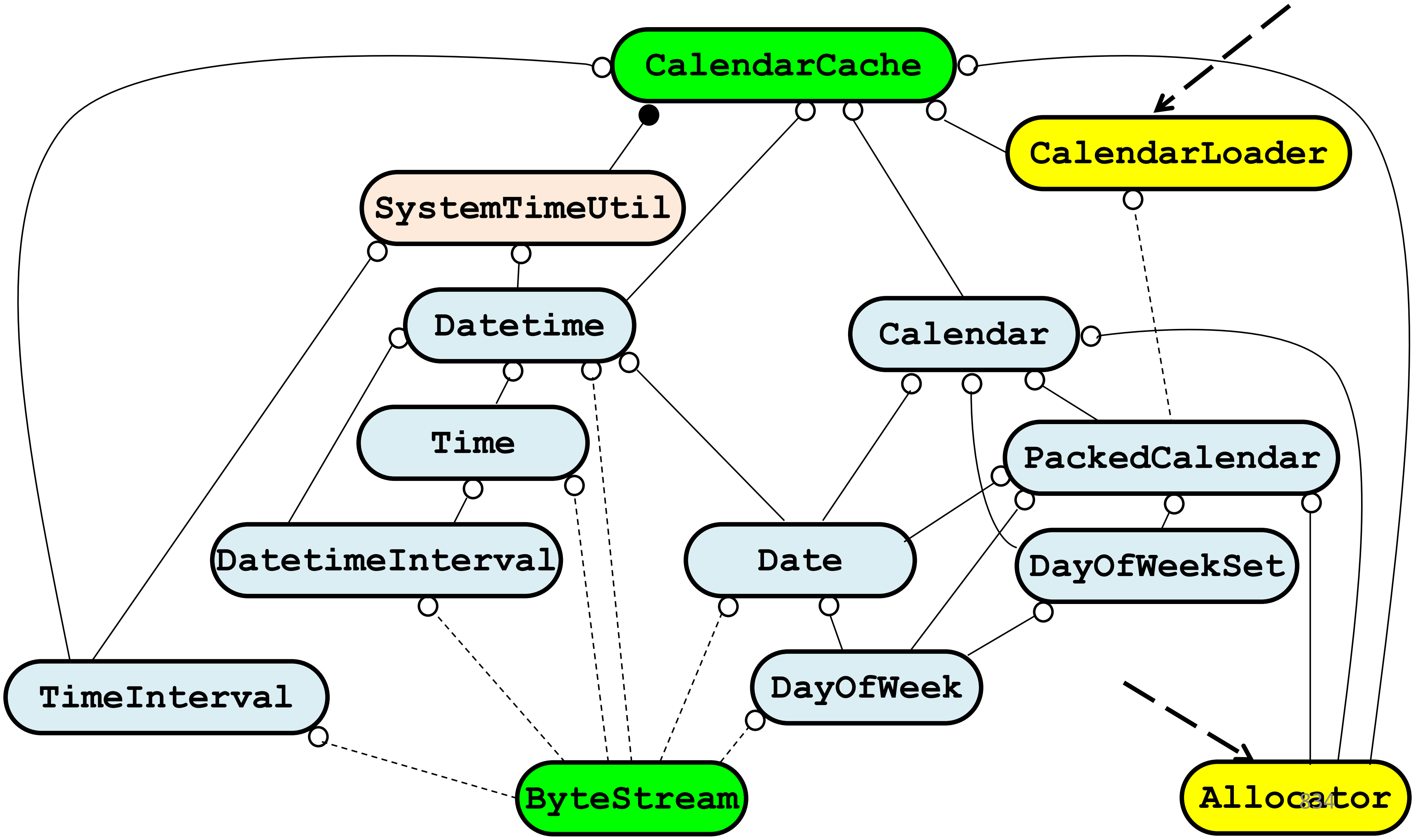
Wait a Minute: Where is the Data Source?



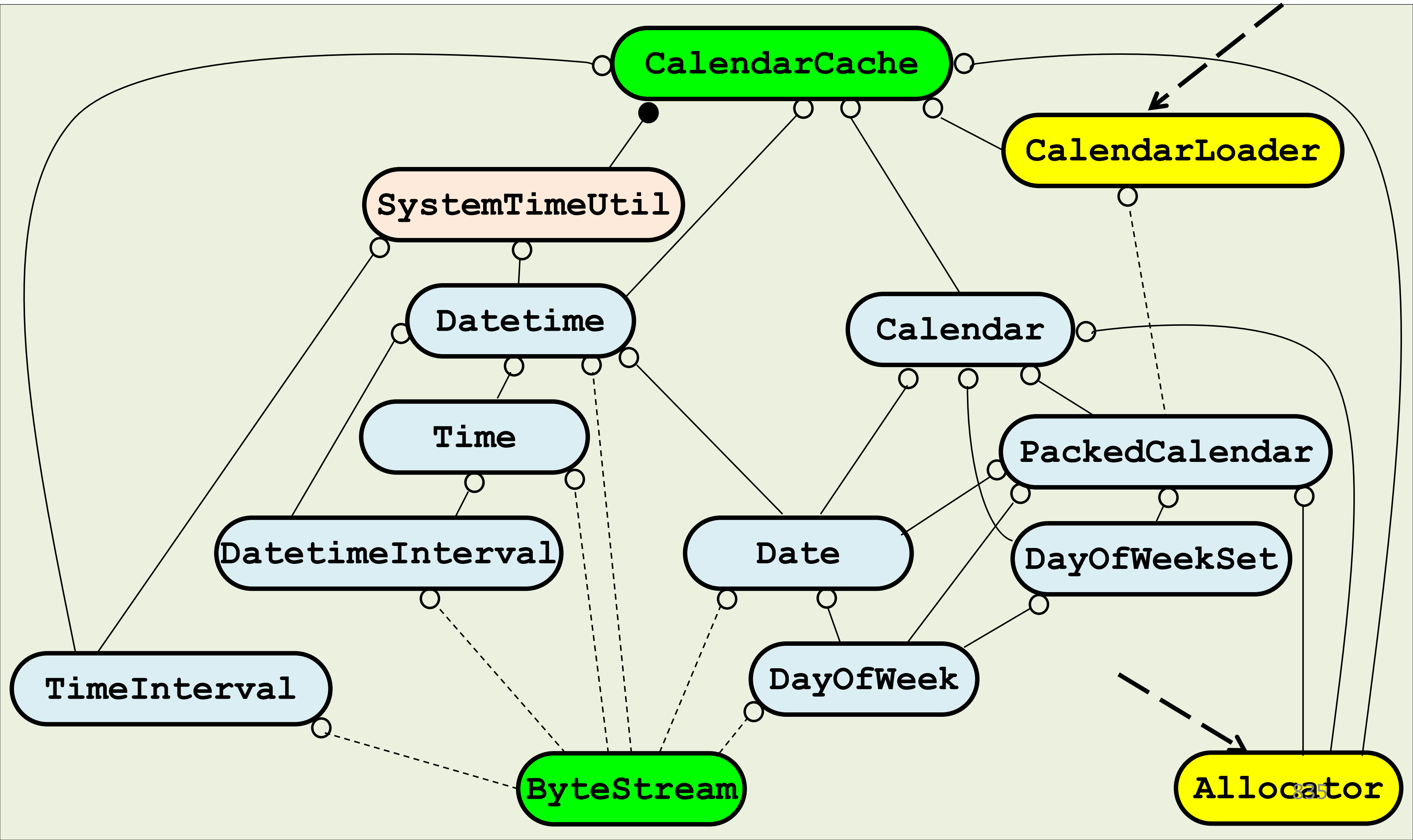
Wait a Minute: Where is the Data Source?



Wait a Minute: Where is the Data Source?



Solution 3: Is `Date` a Business Day?



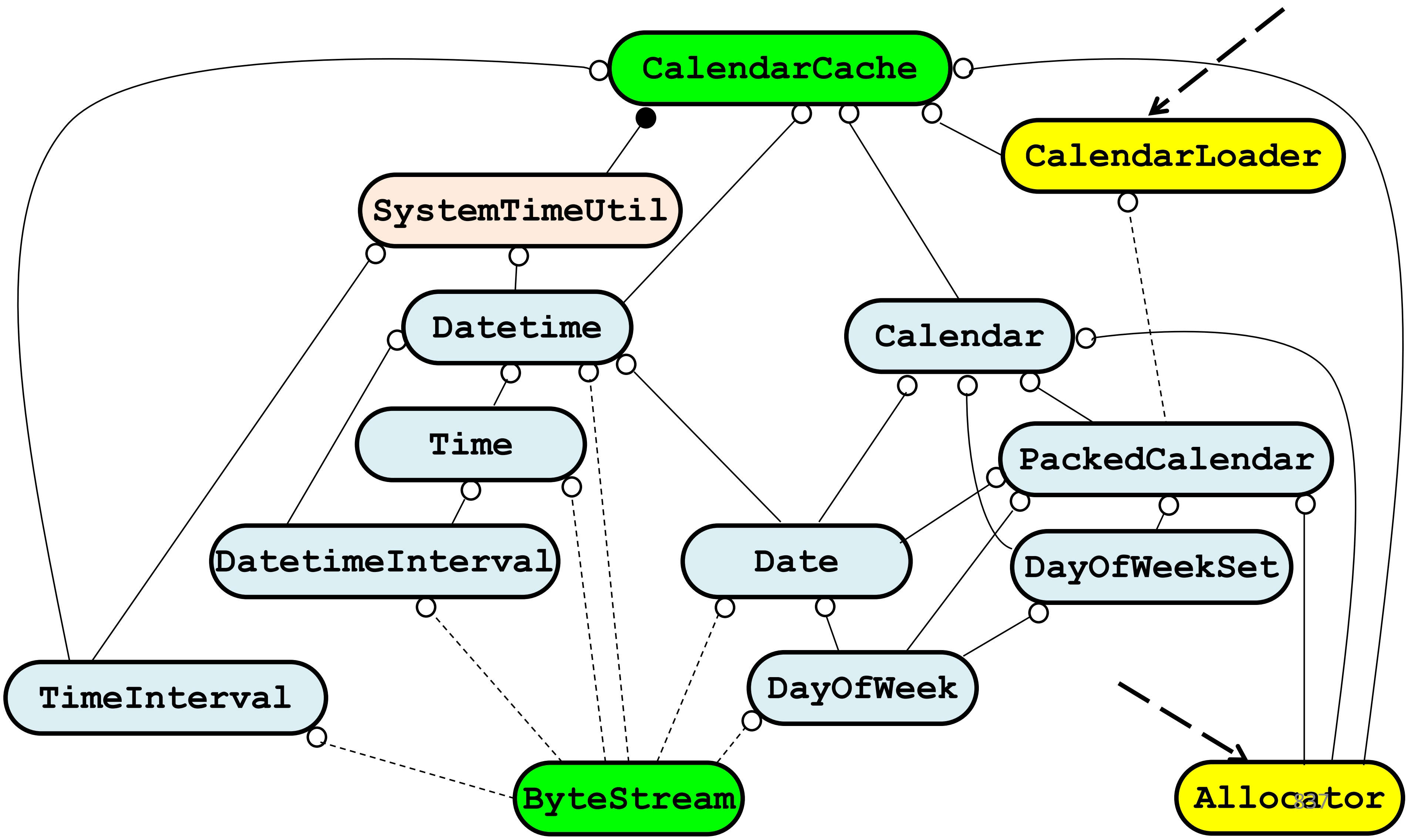
The Original Request

"Write me a 'Date' class that tells me whether today is a business day."

What are the *real* requirements?

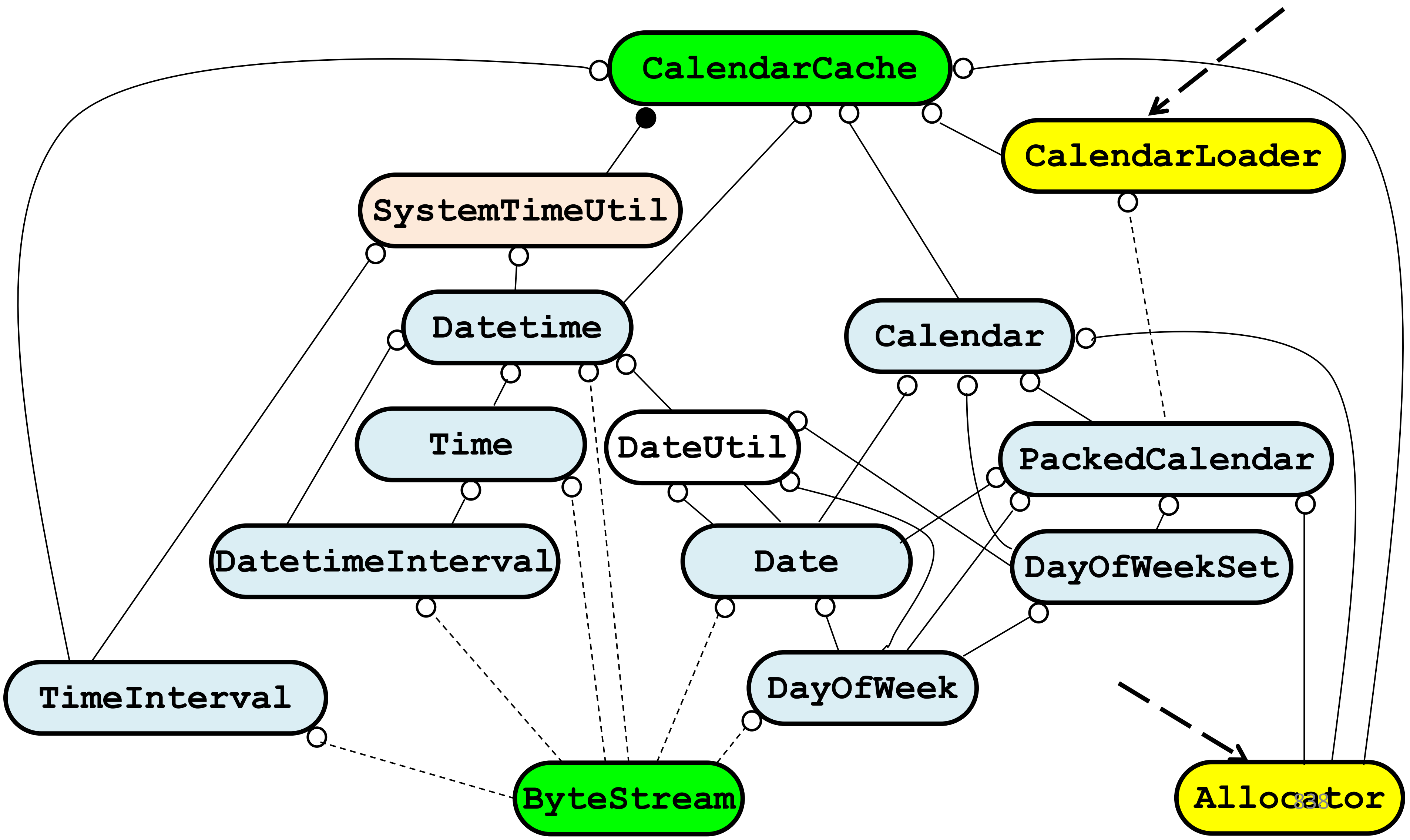
1. Represent a *date value* as a C++ Type.
2. Determine what date value *today* is.
3. Determine if a date value is a *business day*.
4. **Provide well-factored useful components that we'll need over and over again!**

Non-Primitive Functionality



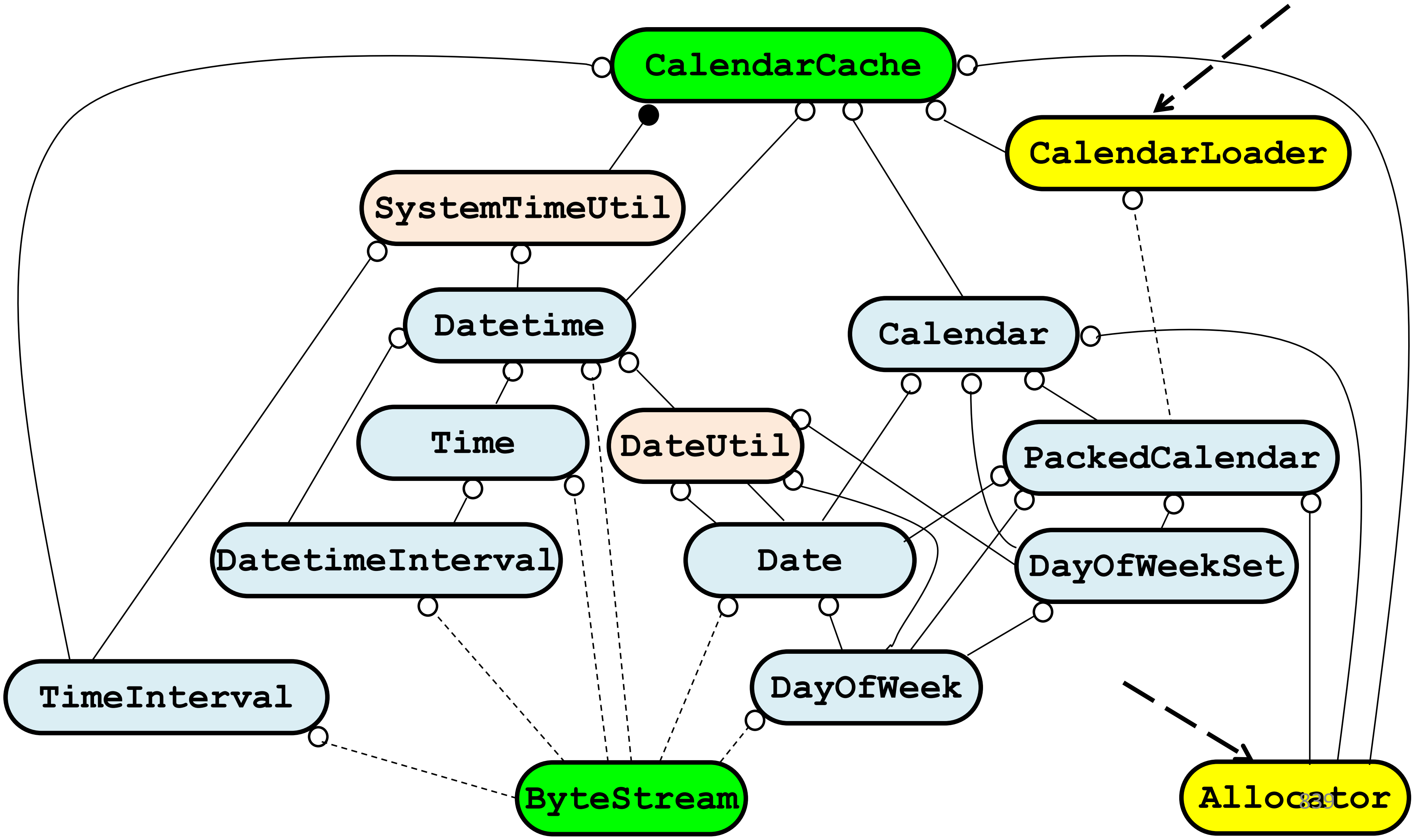
4. Bloomberg Development Environment

Non-Primitive Functionality



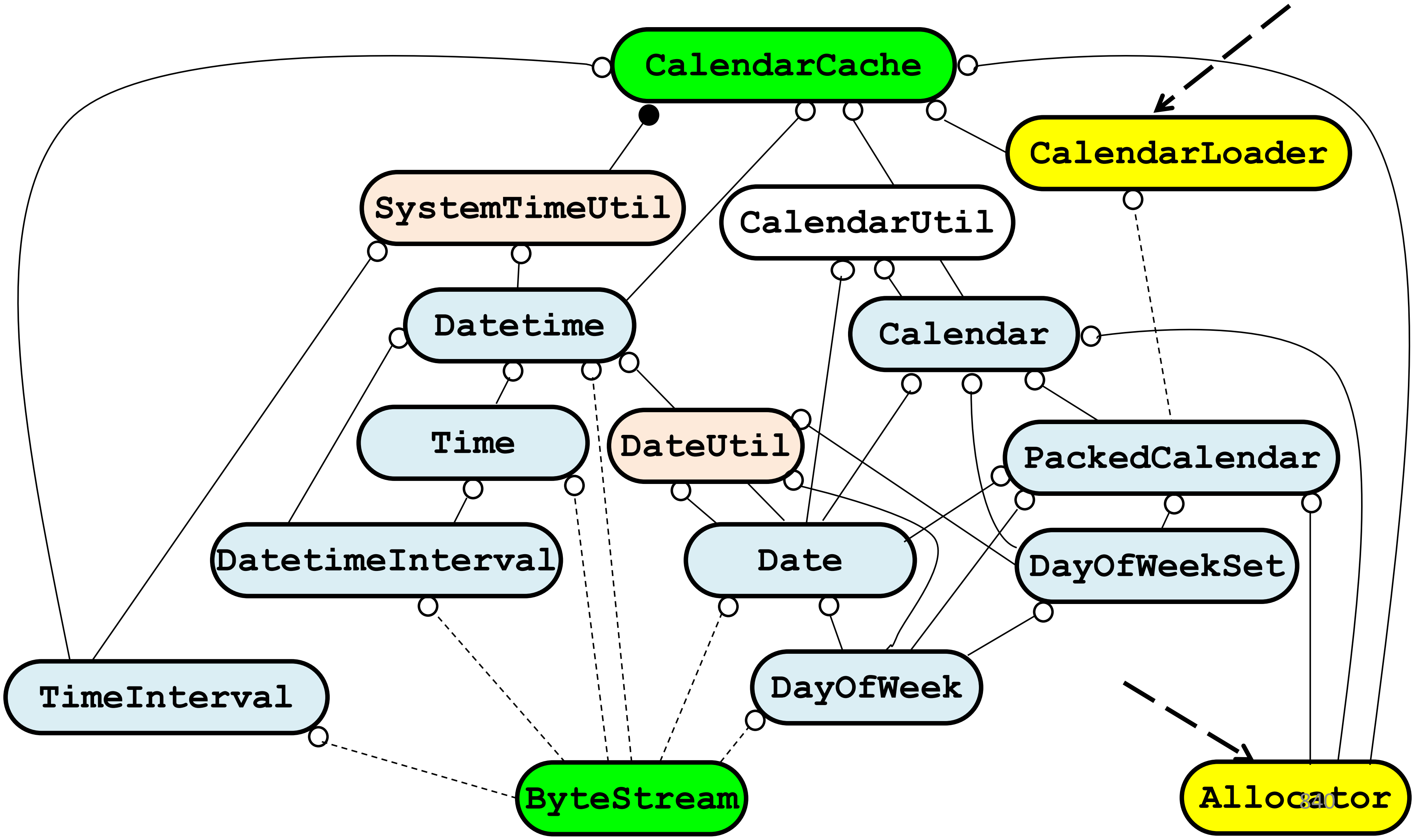
4. Bloomberg Development Environment

Non-Primitive Functionality

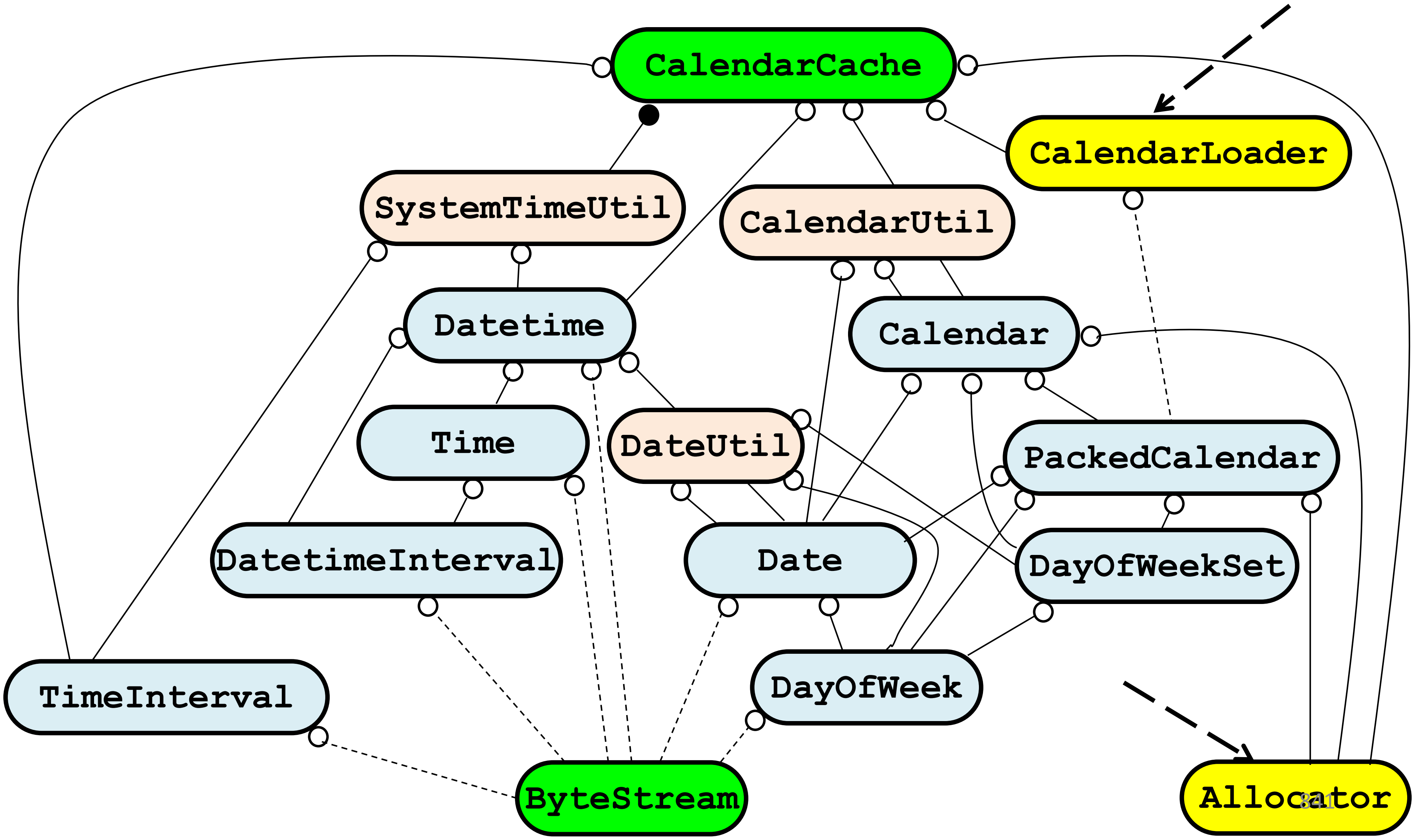


4. Bloomberg Development Environment

Non-Primitive Functionality

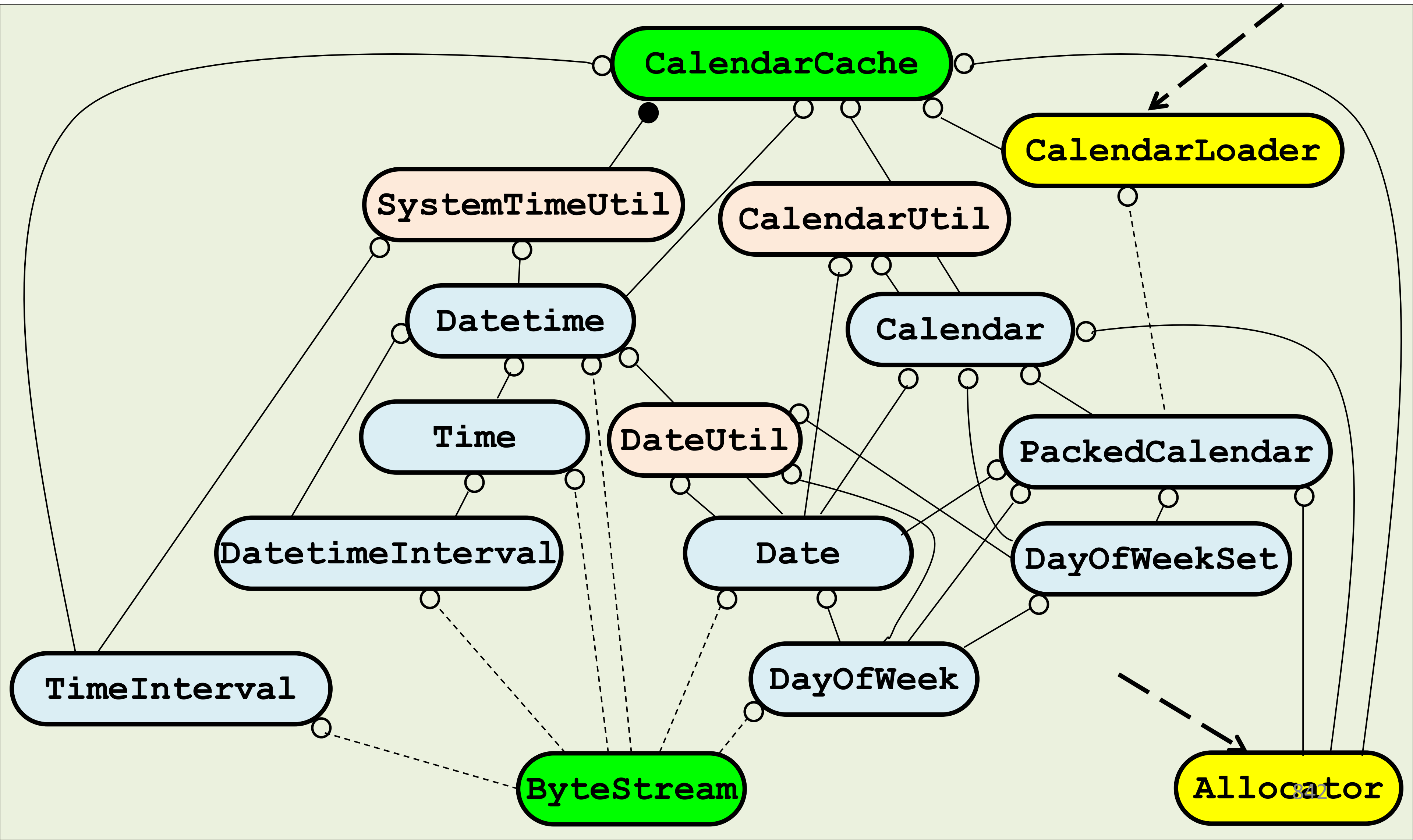


Non-Primitive Functionality



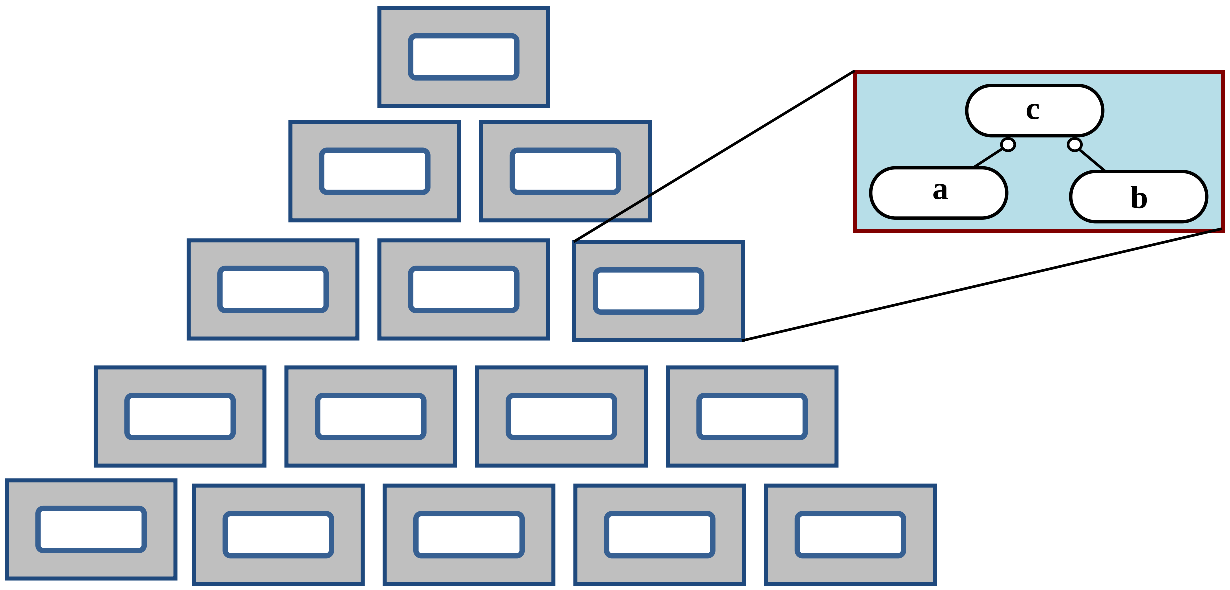
4. Bloomberg Development Environment

Fine-Grained Reusable Class Design



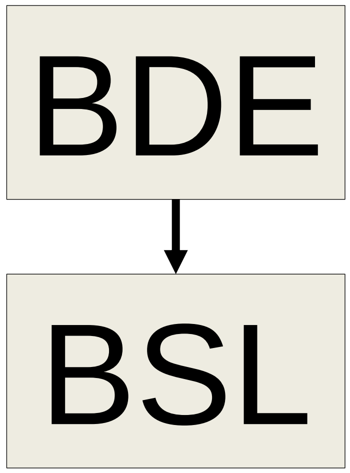
Rendering Software as Components

Logical content aggregated into a *Physical* hierarchy of ***components***



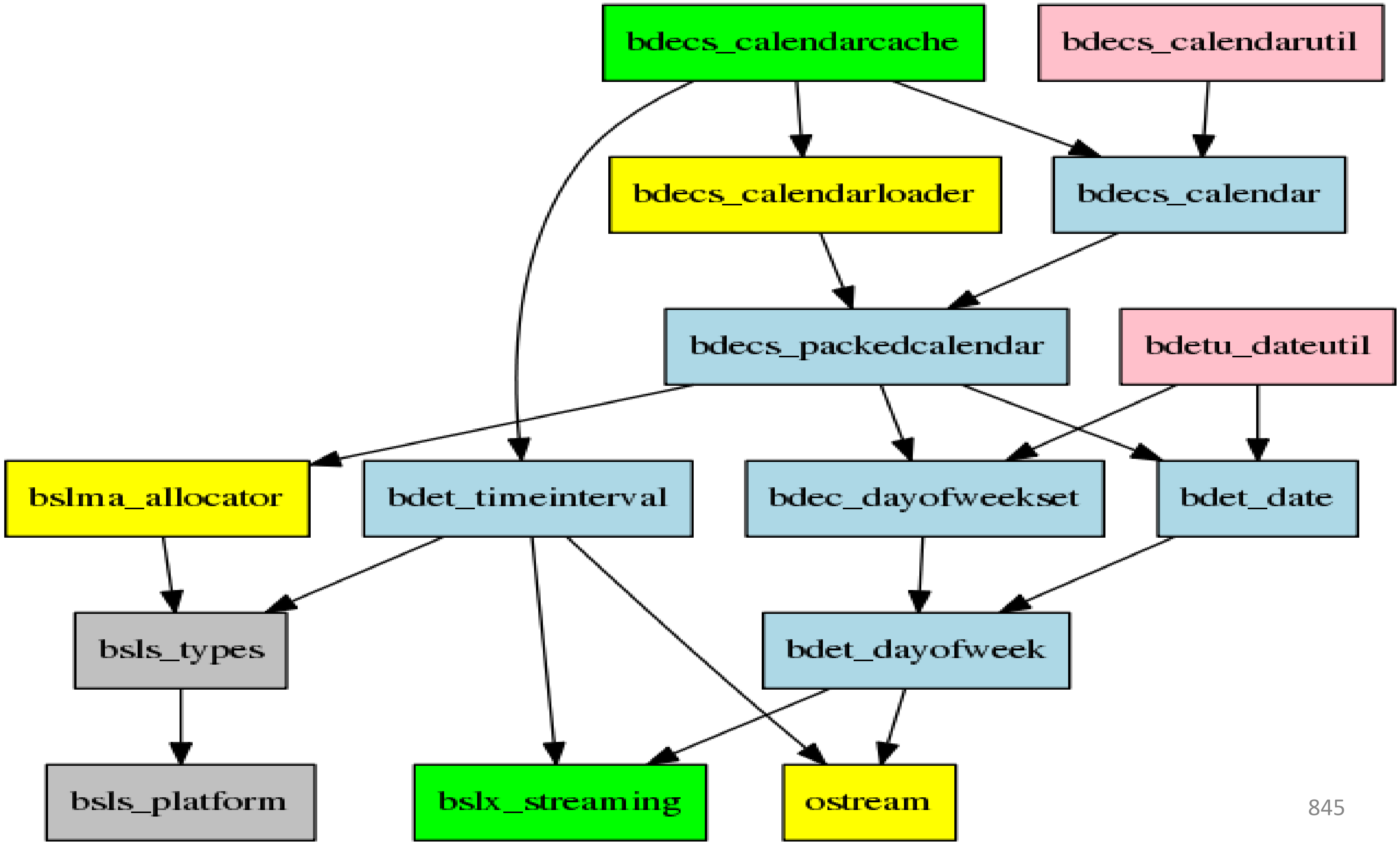
4. Bloomberg Development Environment

Package Group Dependencies



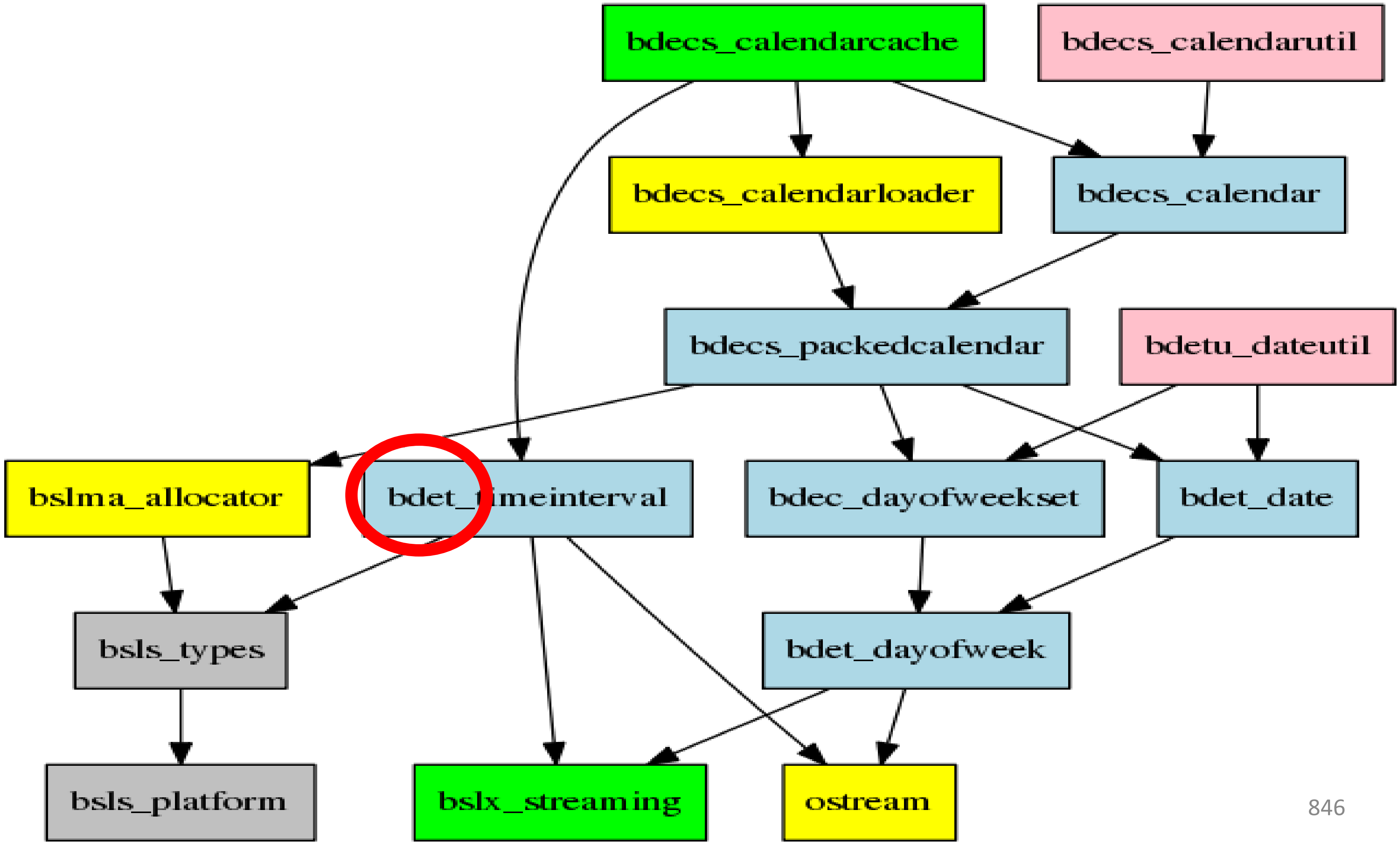
4. Bloomberg Development Environment

Client-Facing Component Diagram



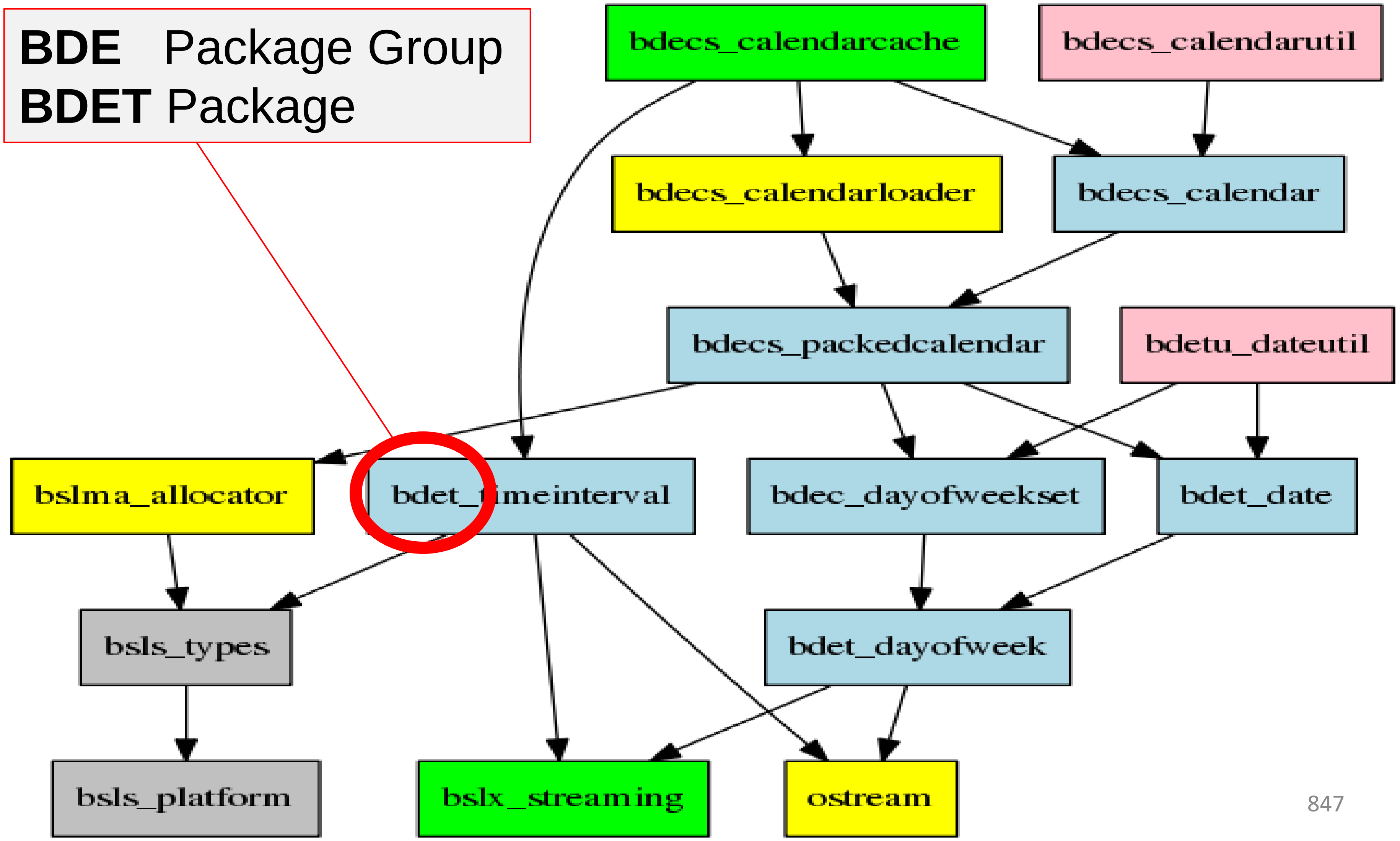
4. Bloomberg Development Environment

Client-Facing Component Diagram



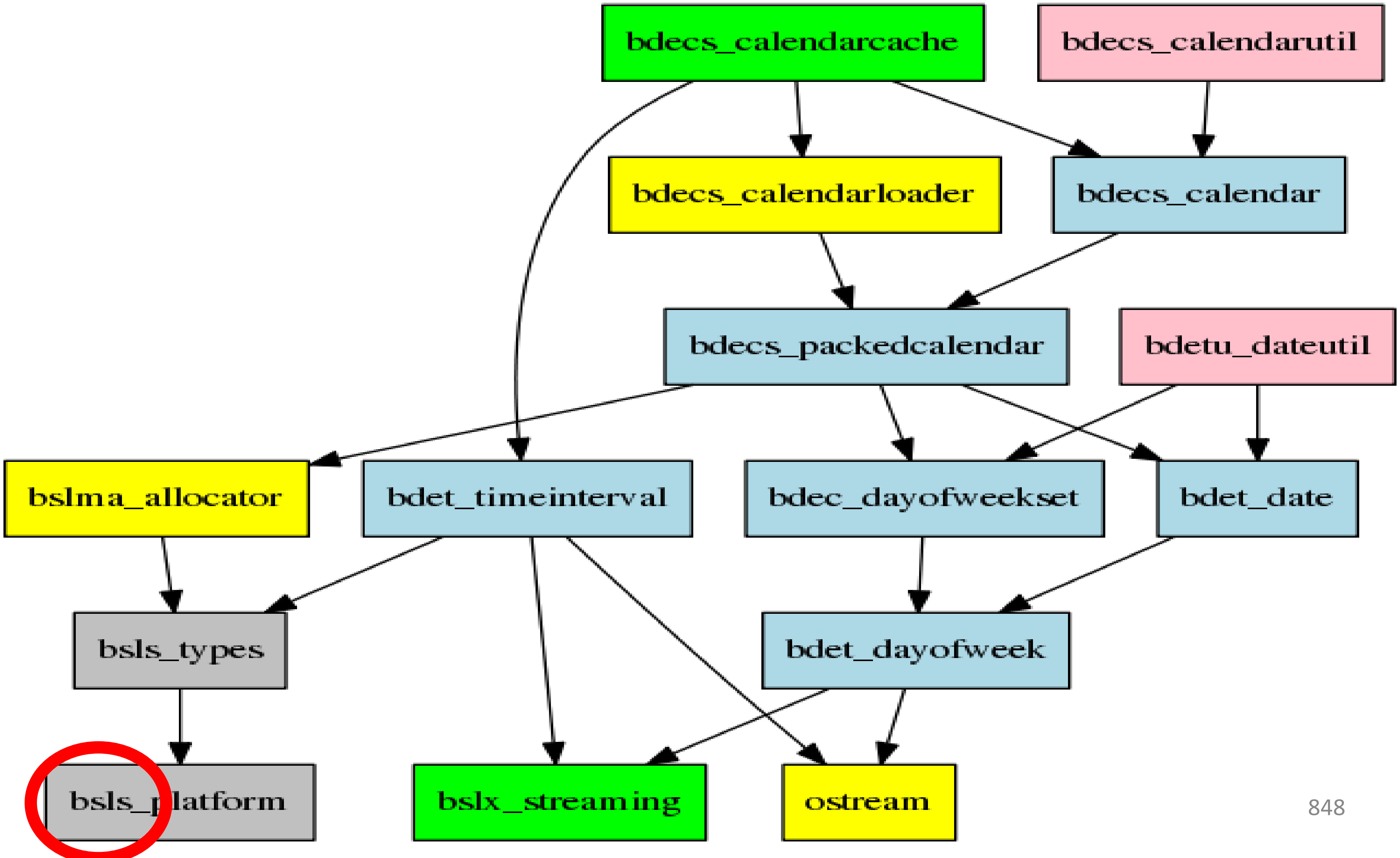
Client-Facing Component Diagram

BDE Package Group
BDET Package



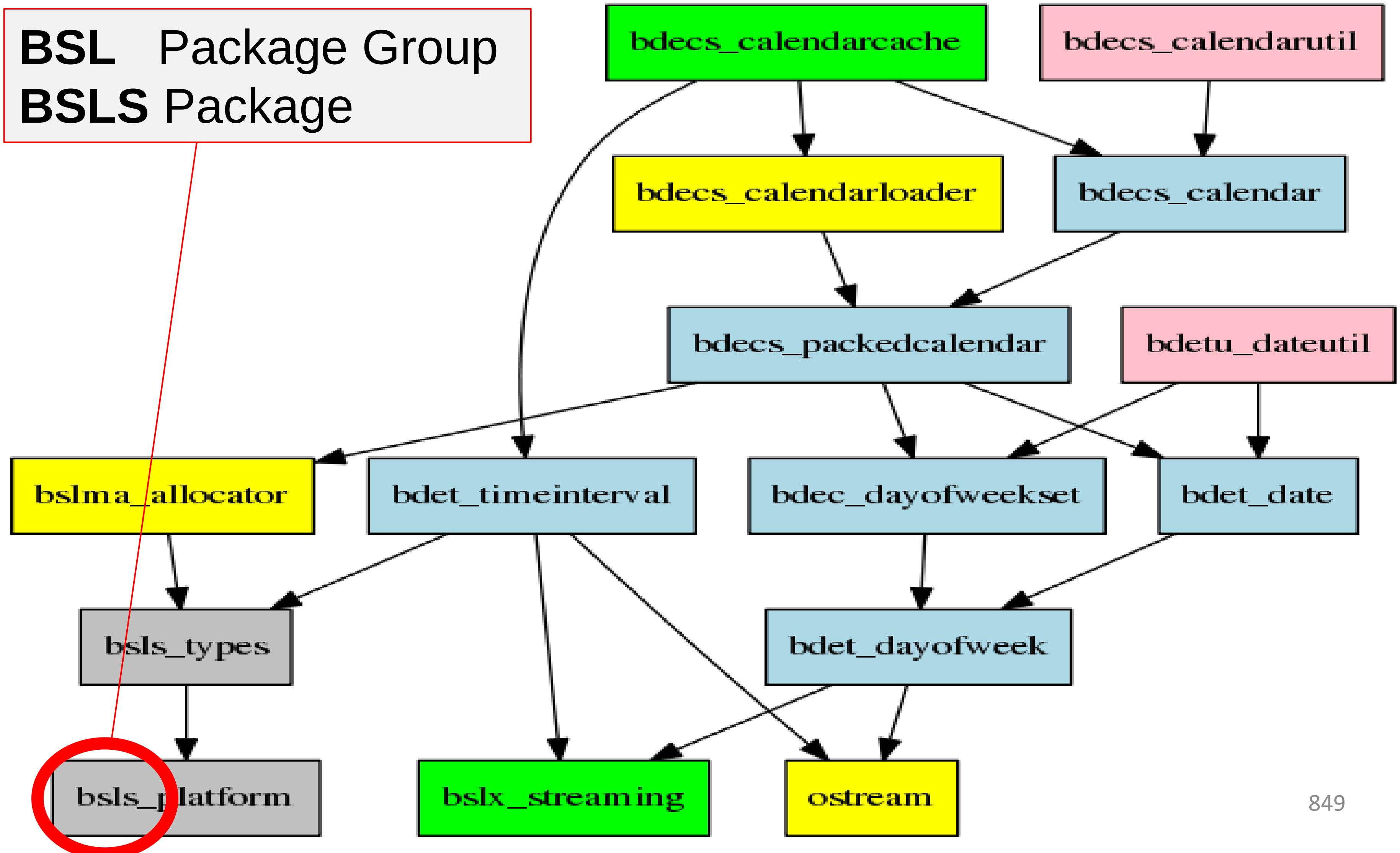
4. Bloomberg Development Environment

Client-Facing Component Diagram



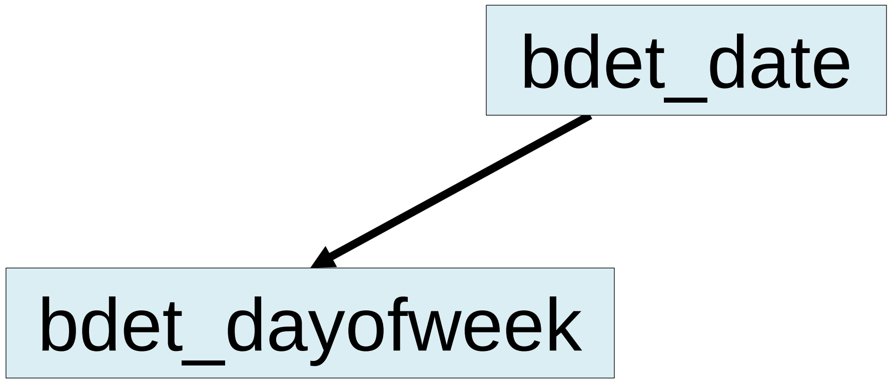
4. Bloomberg Development Environment

Client-Facing Component Diagram

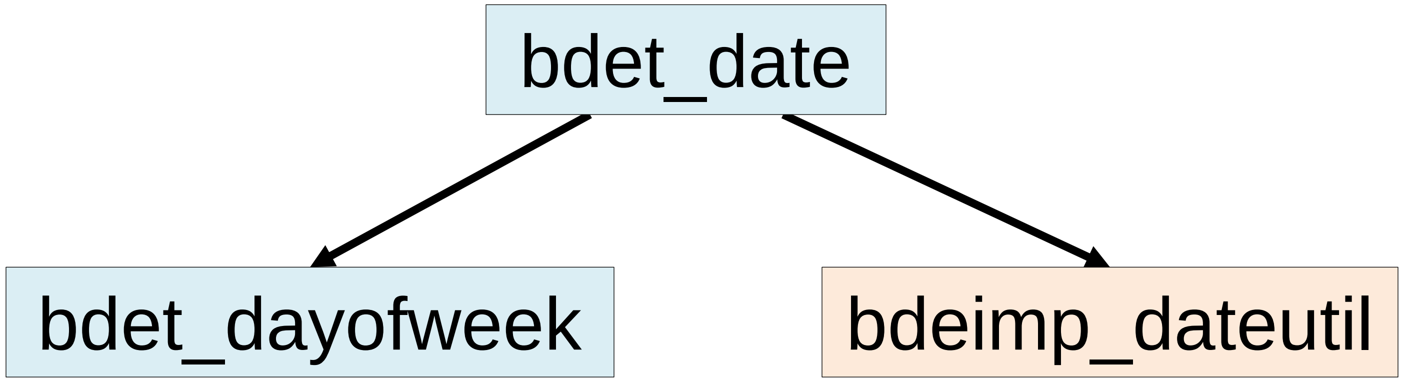


4. Bloomberg Development Environment

Implementing `bdet_date`

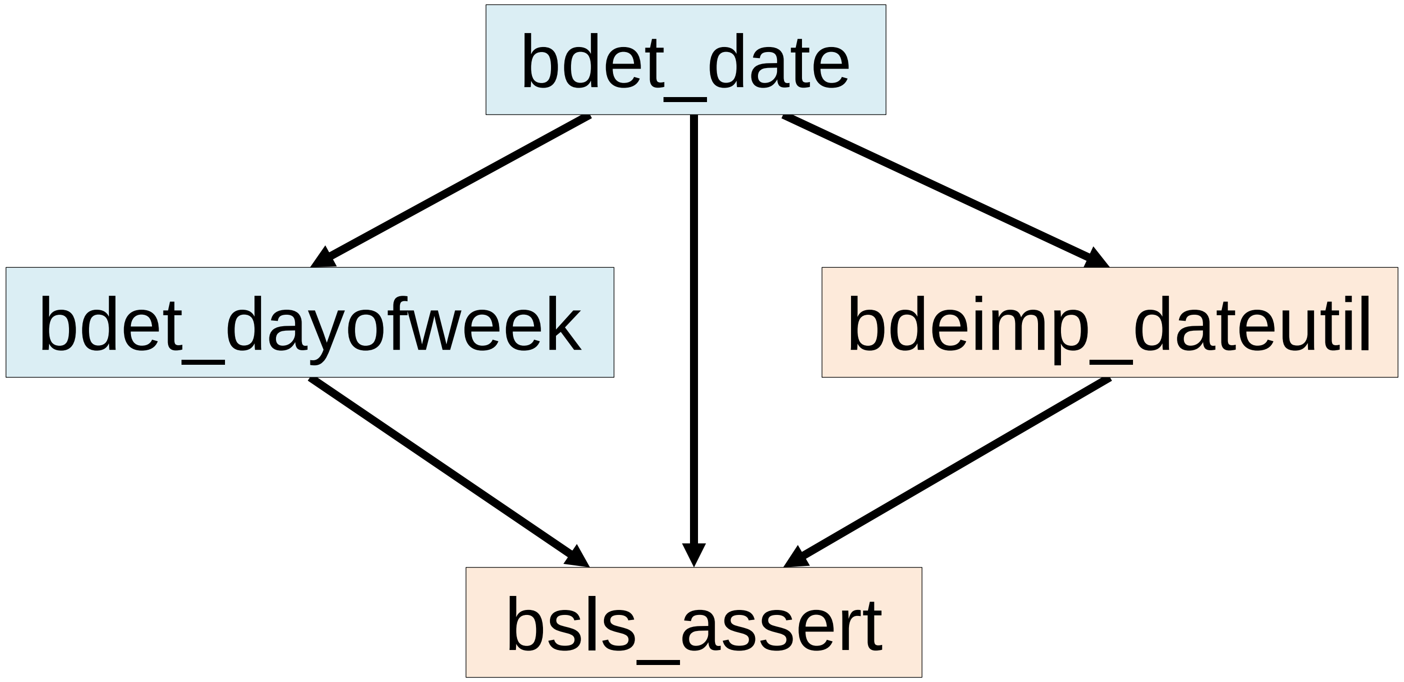


Implementing `bdet_date`



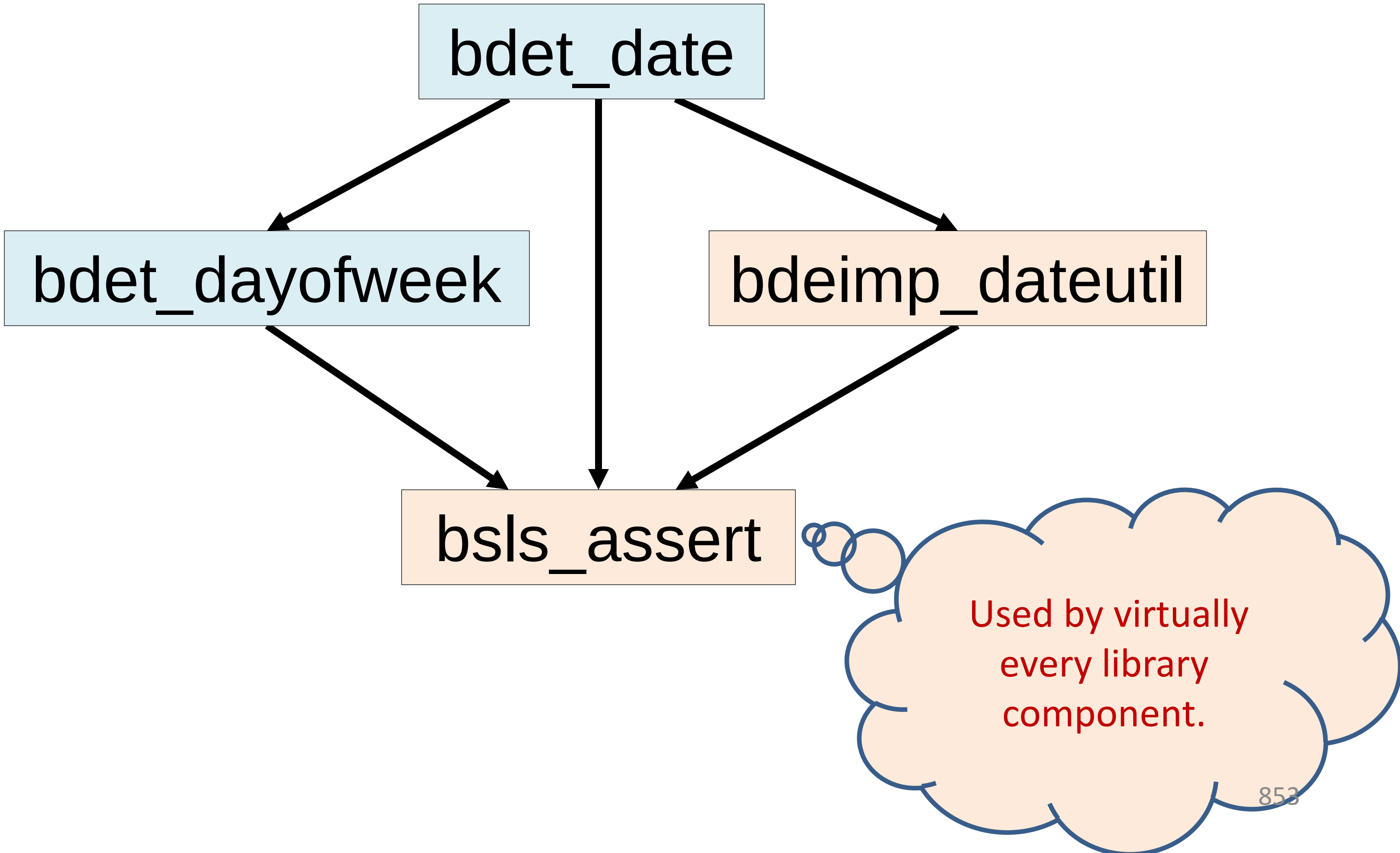
4. Bloomberg Development Environment

Implementing `bdet_date`

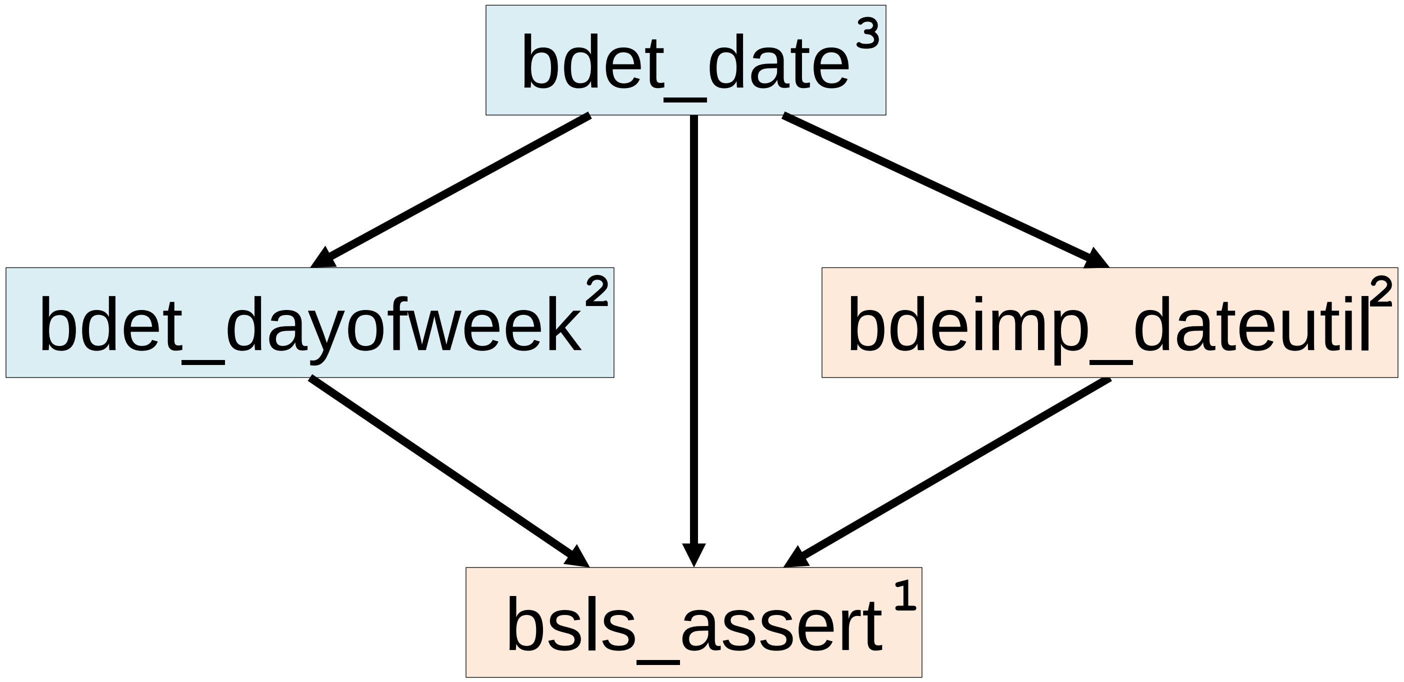


4. Bloomberg Development Environment

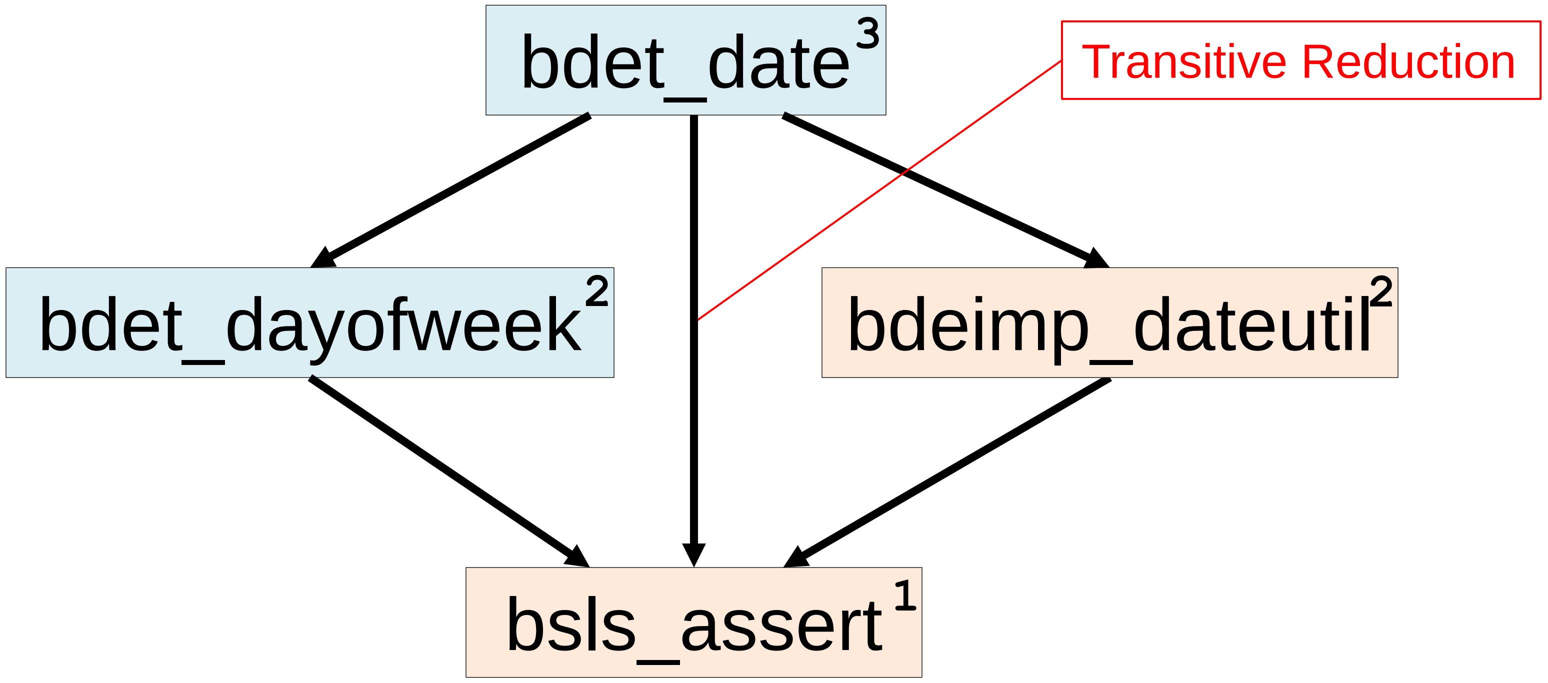
Implementing `bdet_date`



Implementing `bdet_date`

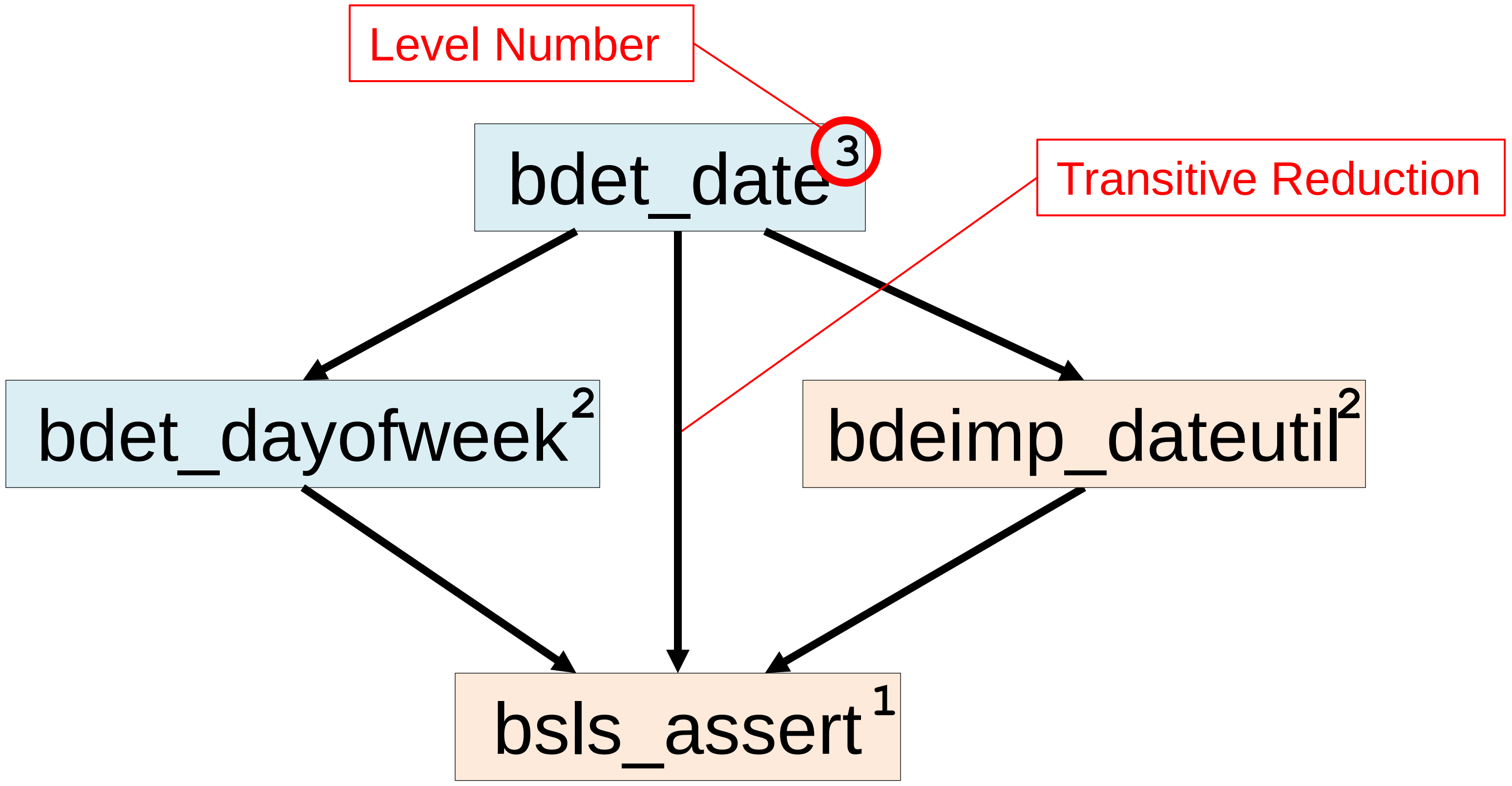


Implementing `bdet_date`



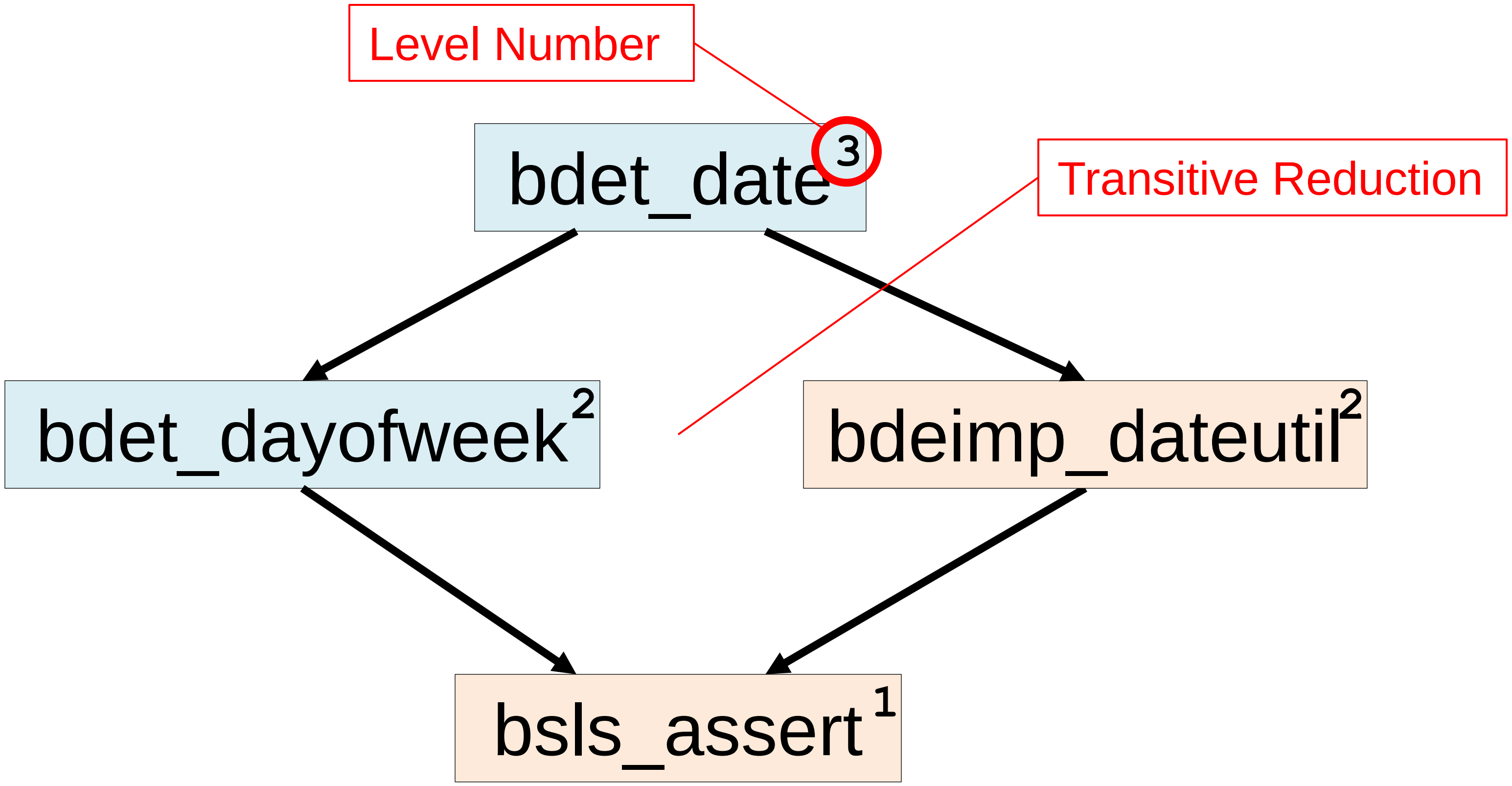
4. Bloomberg Development Environment

Implementing `bdet_date`



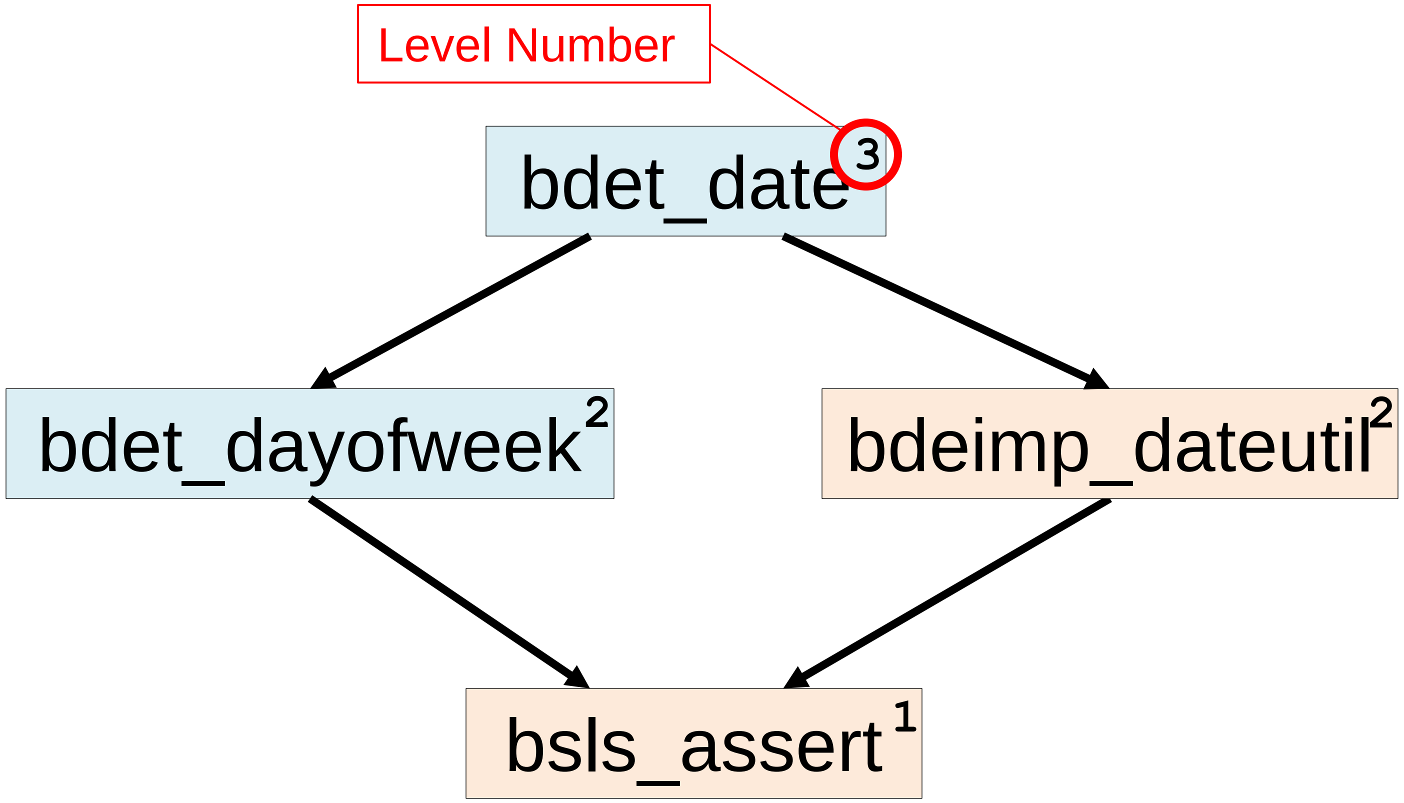
4. Bloomberg Development Environment

Implementing `bdet_date`

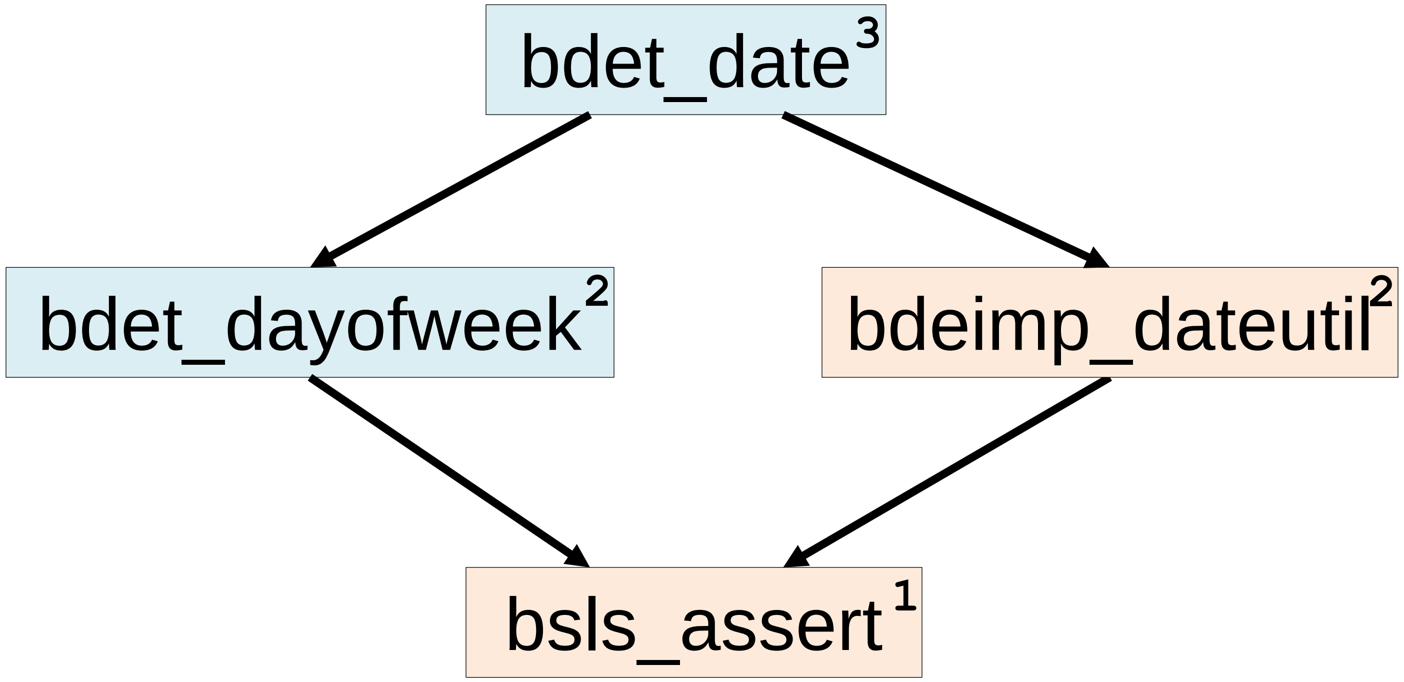


4. Bloomberg Development Environment

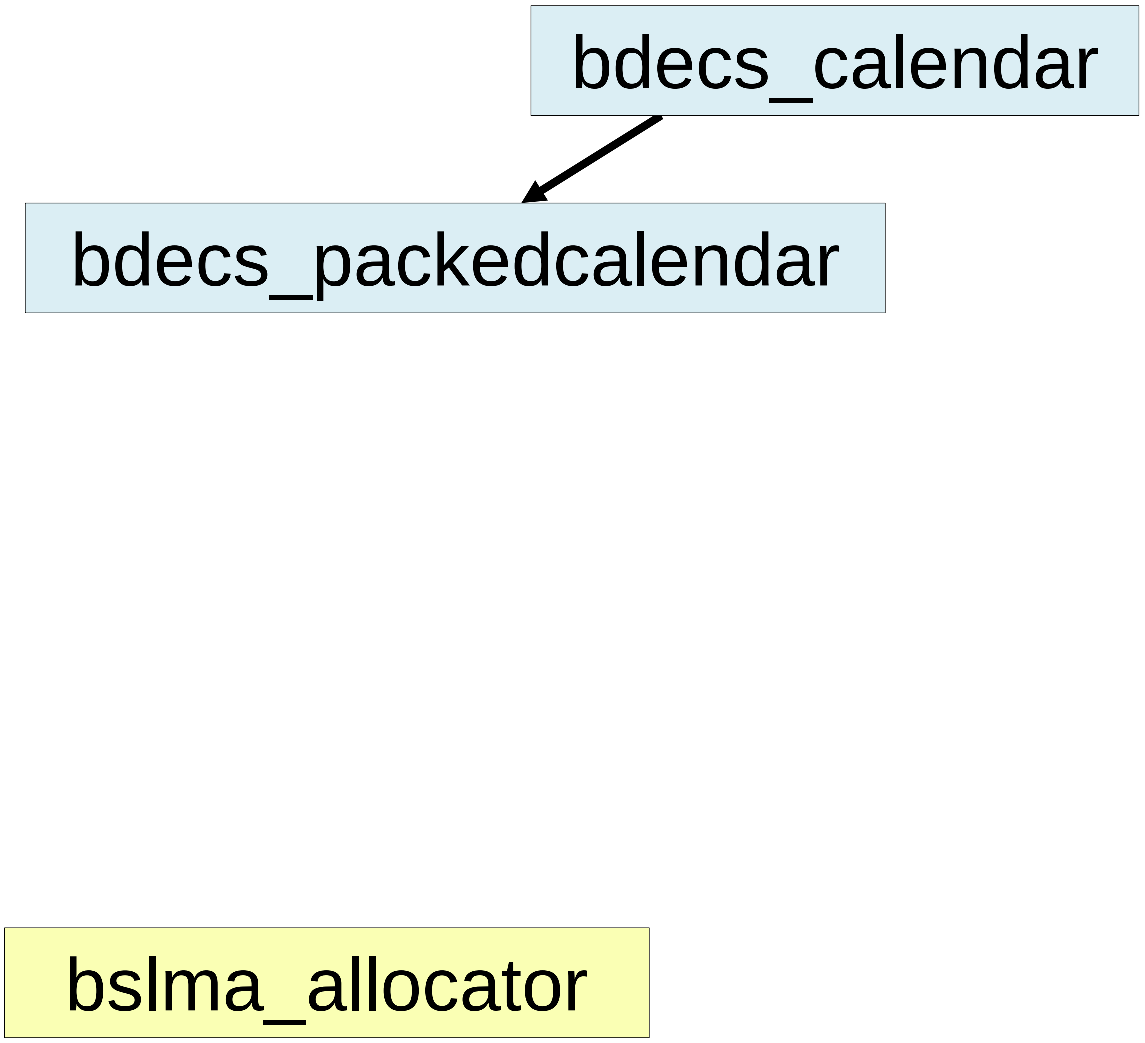
Implementing `bdet_date`



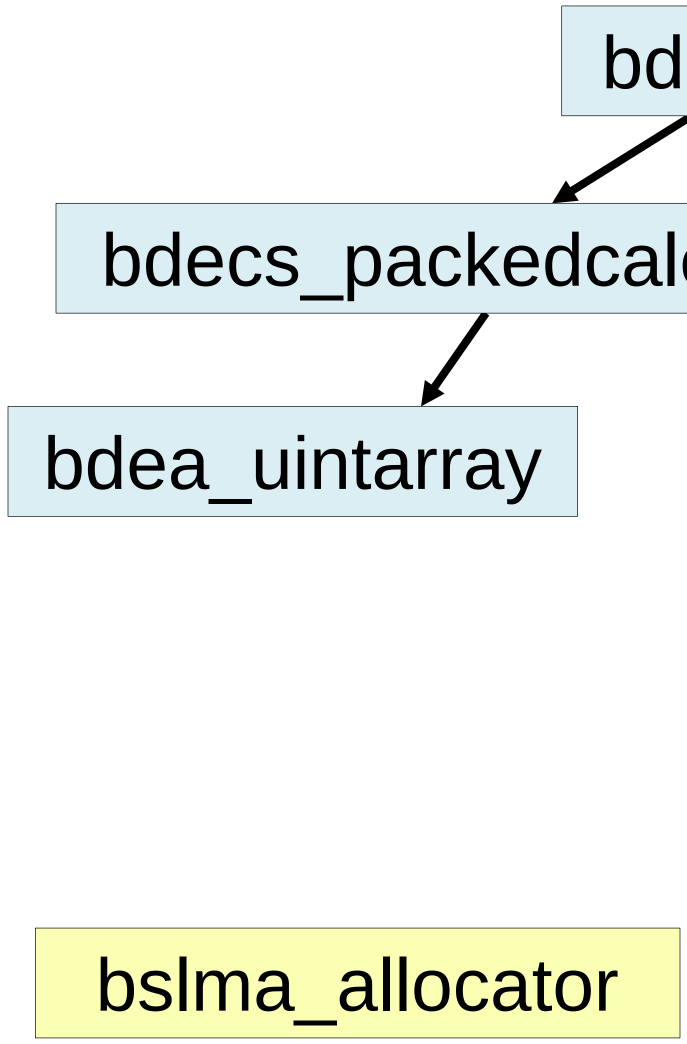
Implementing `bdet_date`



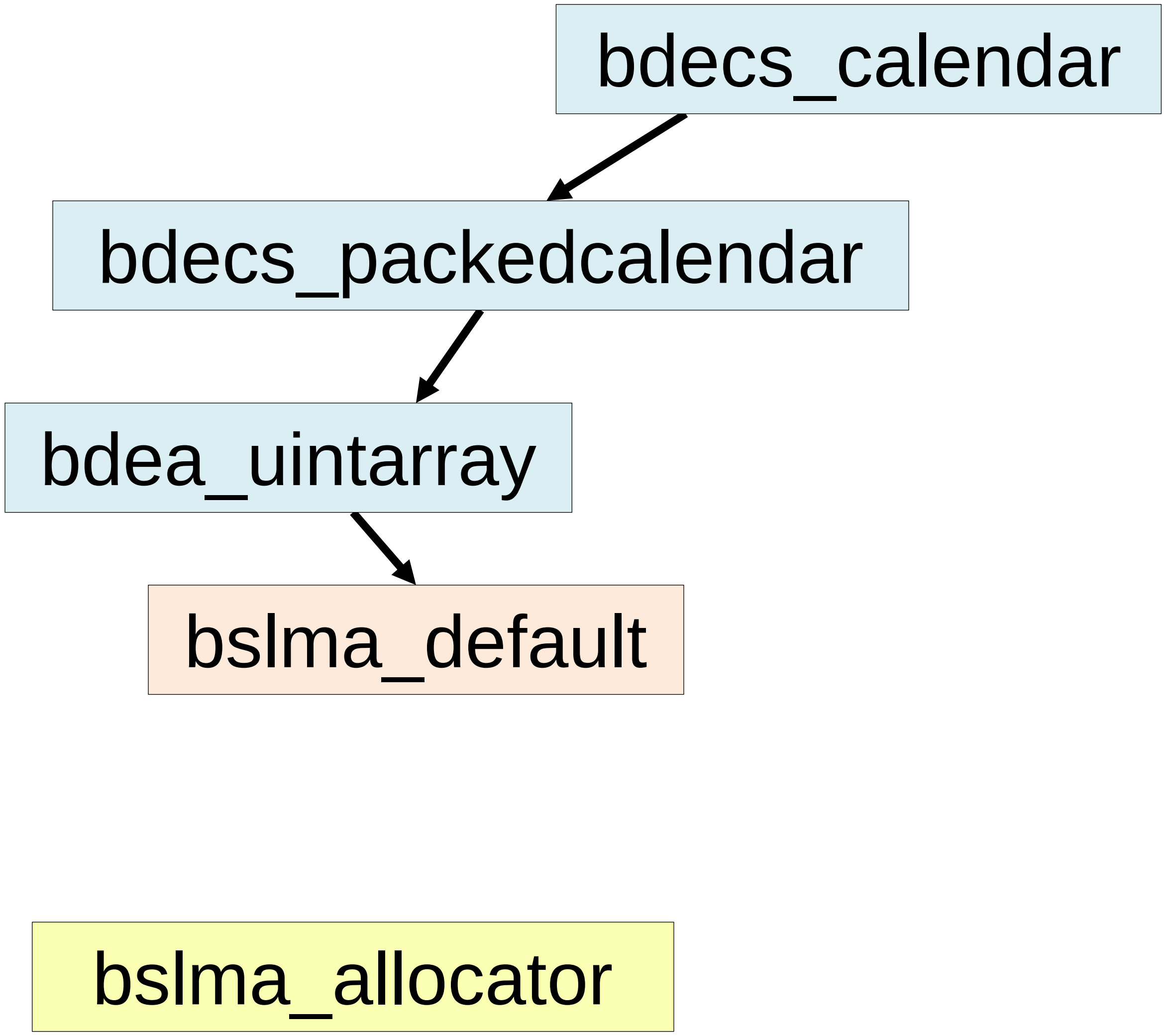
Implementing `bdecs_calendar`



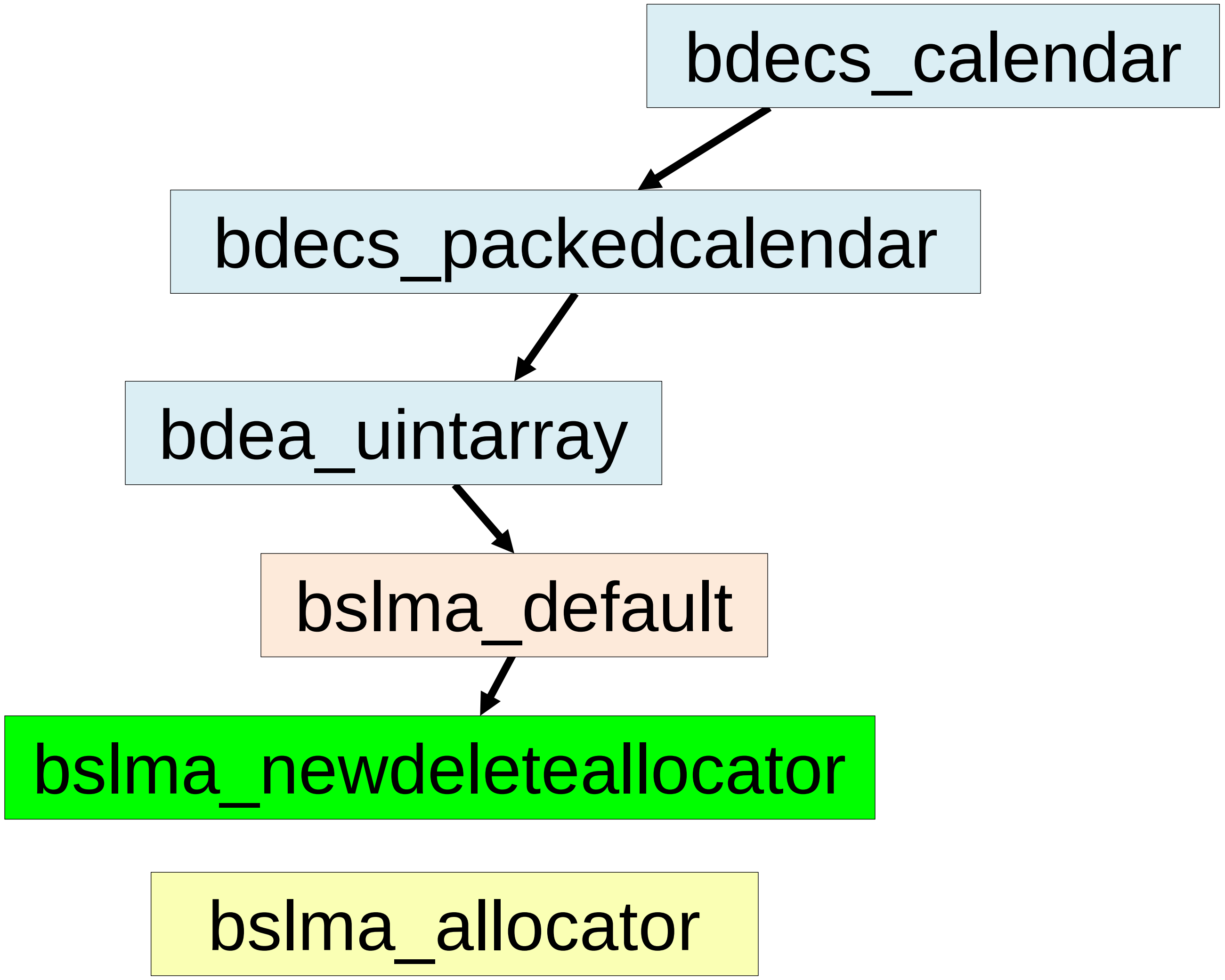
Implementing `bdecs_calendar`



Implementing `bdecs_calendar`

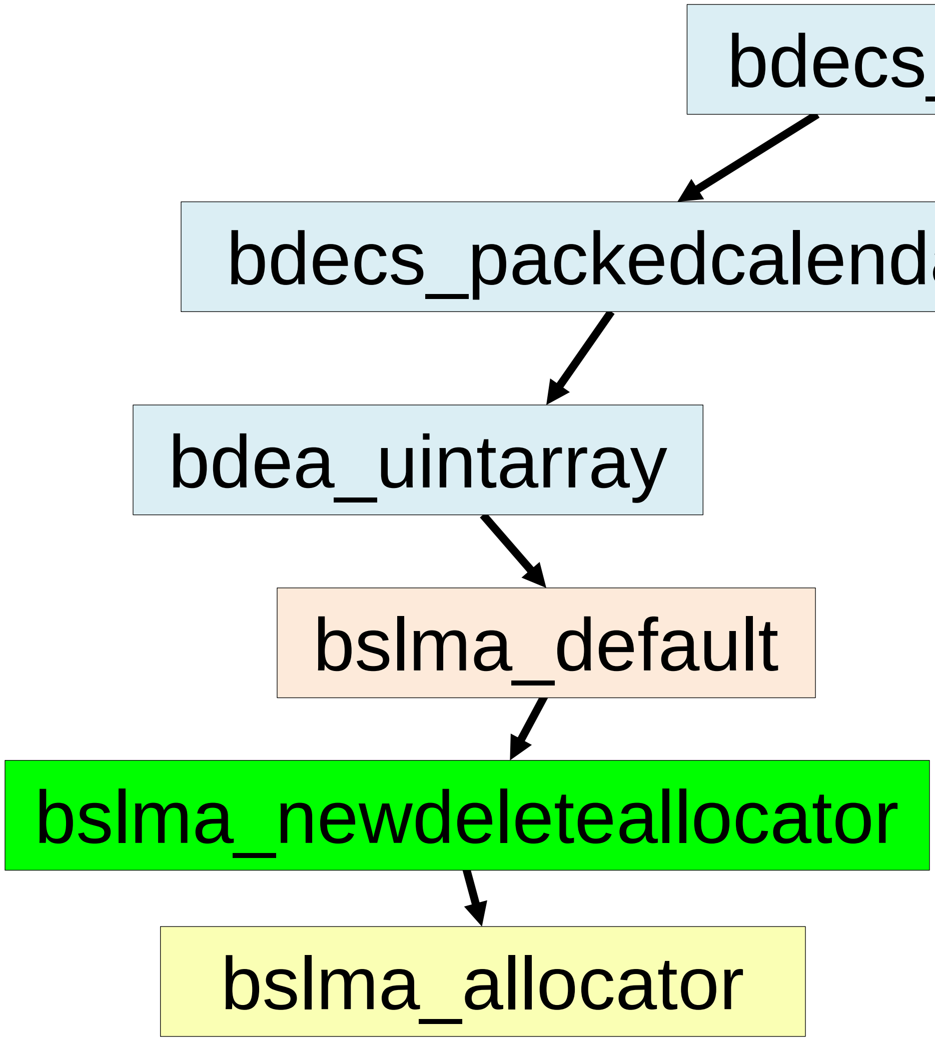


Implementing `bdecs_calendar`

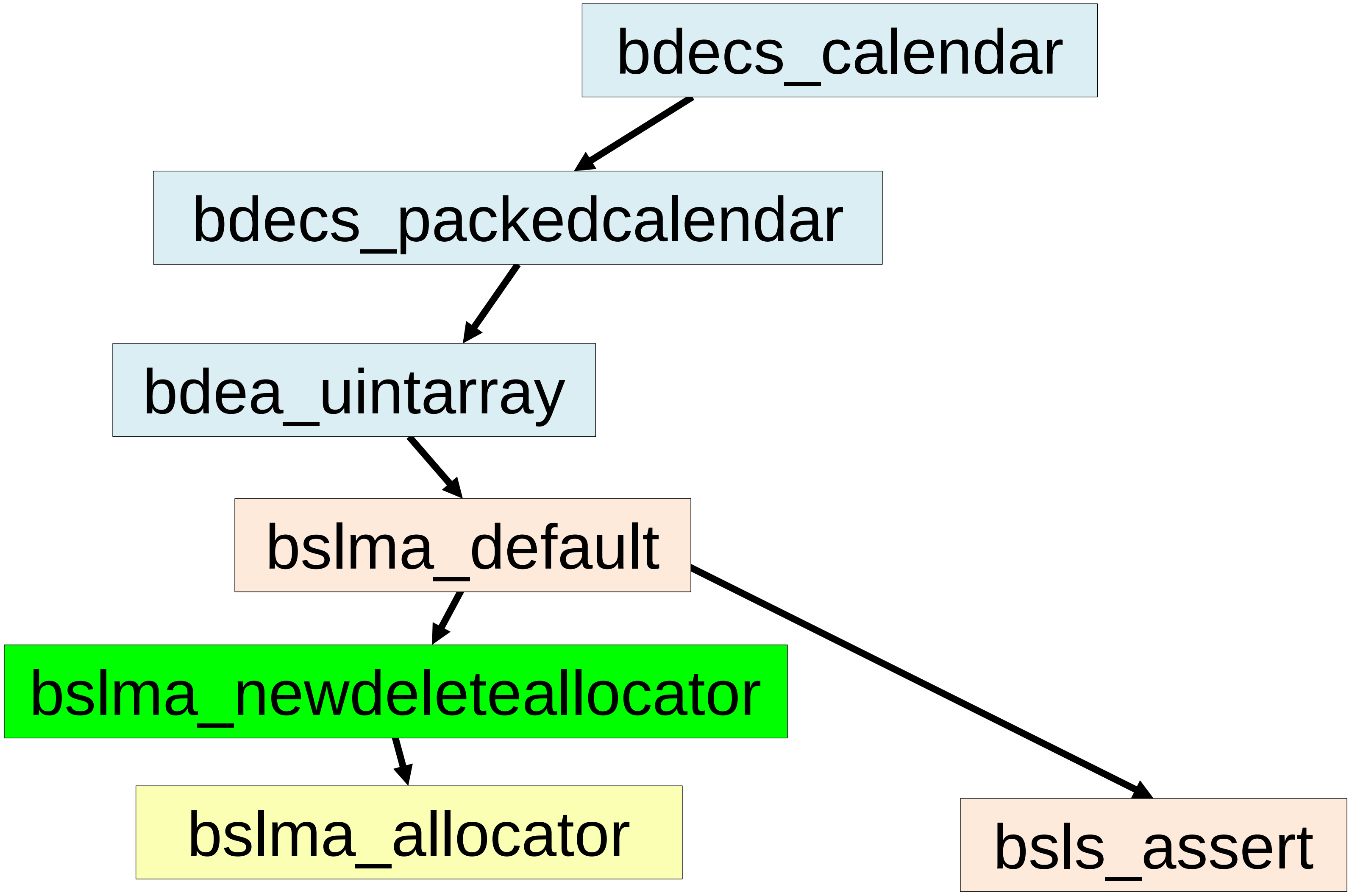


4. Bloomberg Development Environment

Implementing `bdecs_calendar`

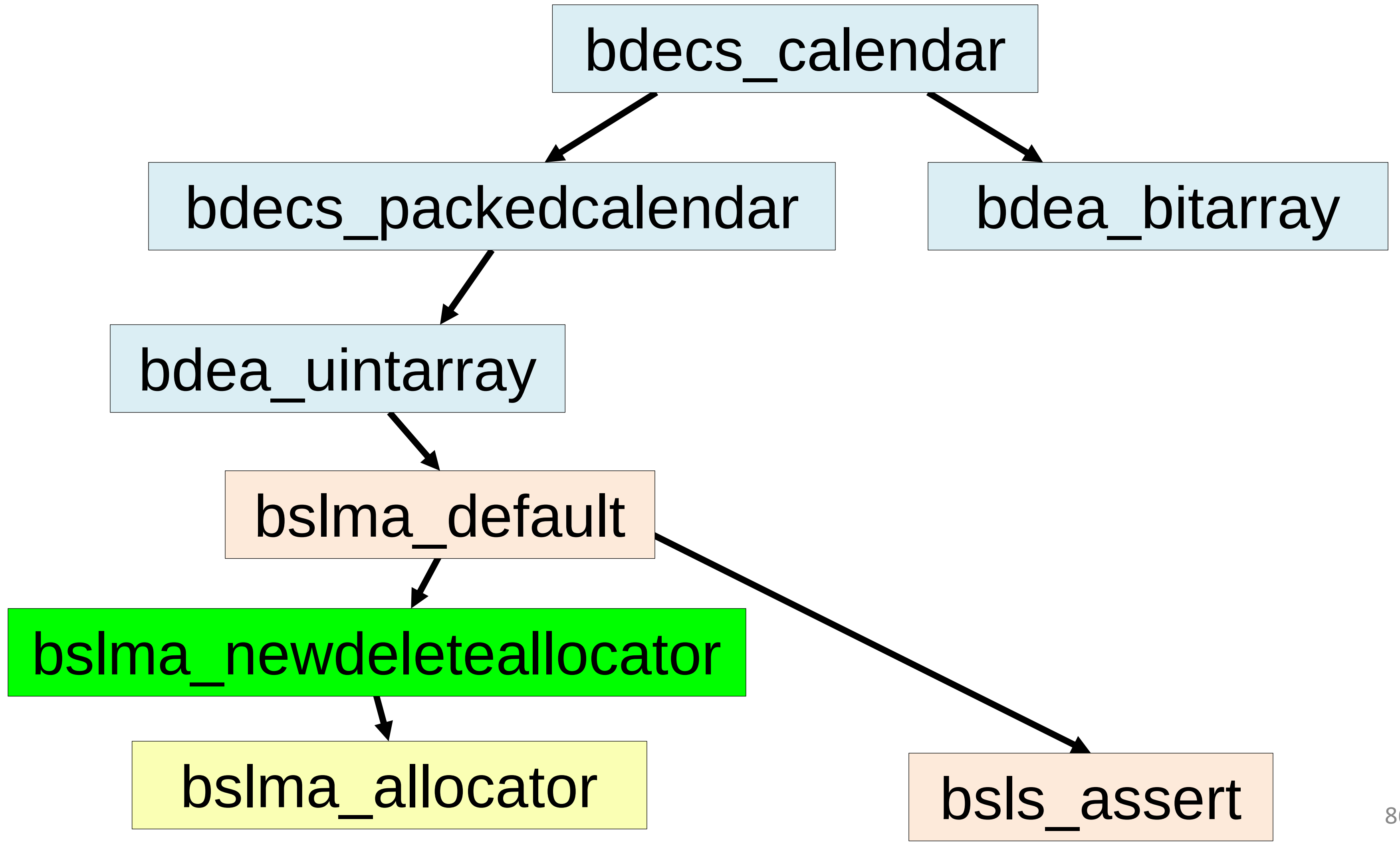


Implementing `bdecs_calendar`



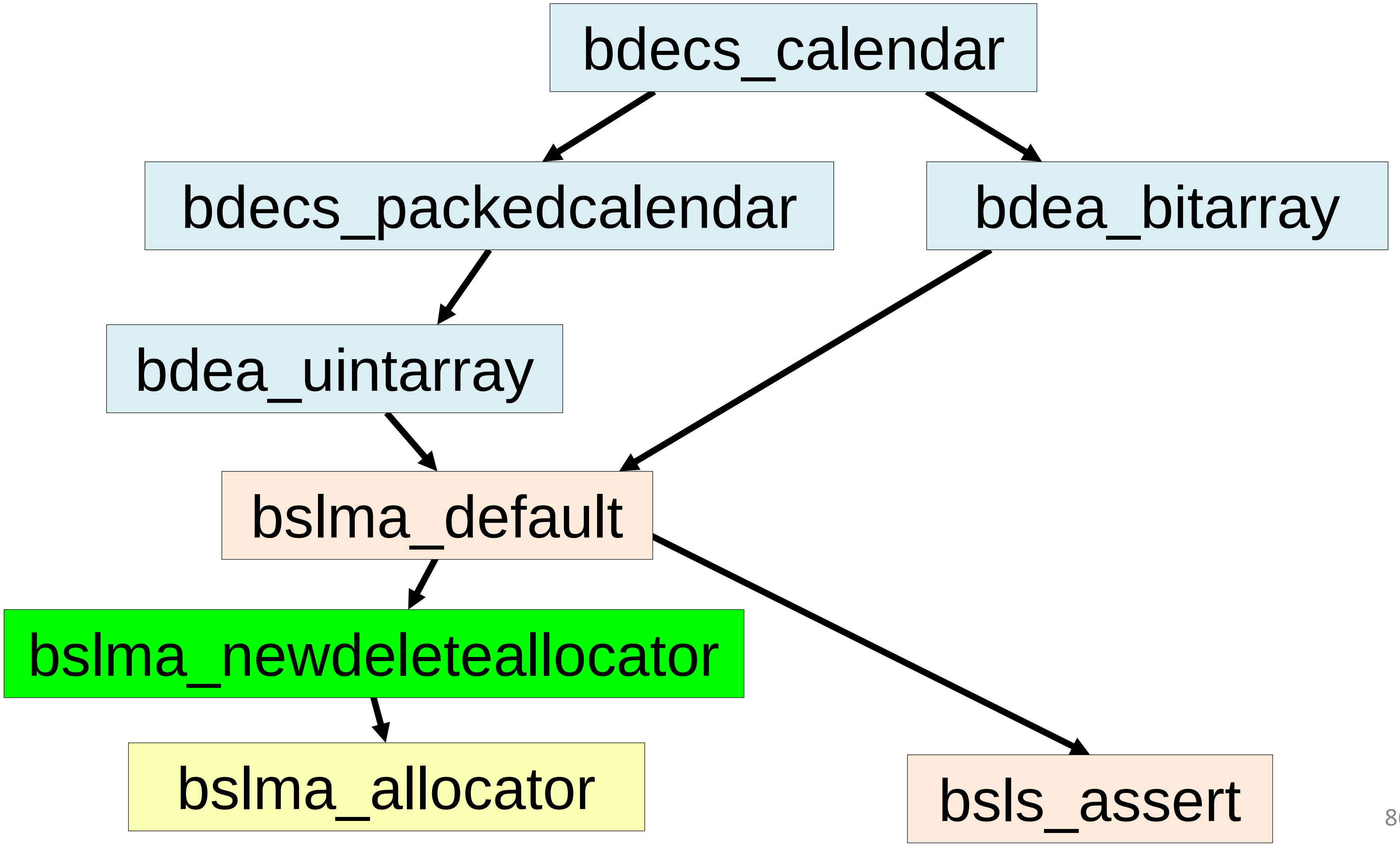
4. Bloomberg Development Environment

Implementing `bdecs_calendar`



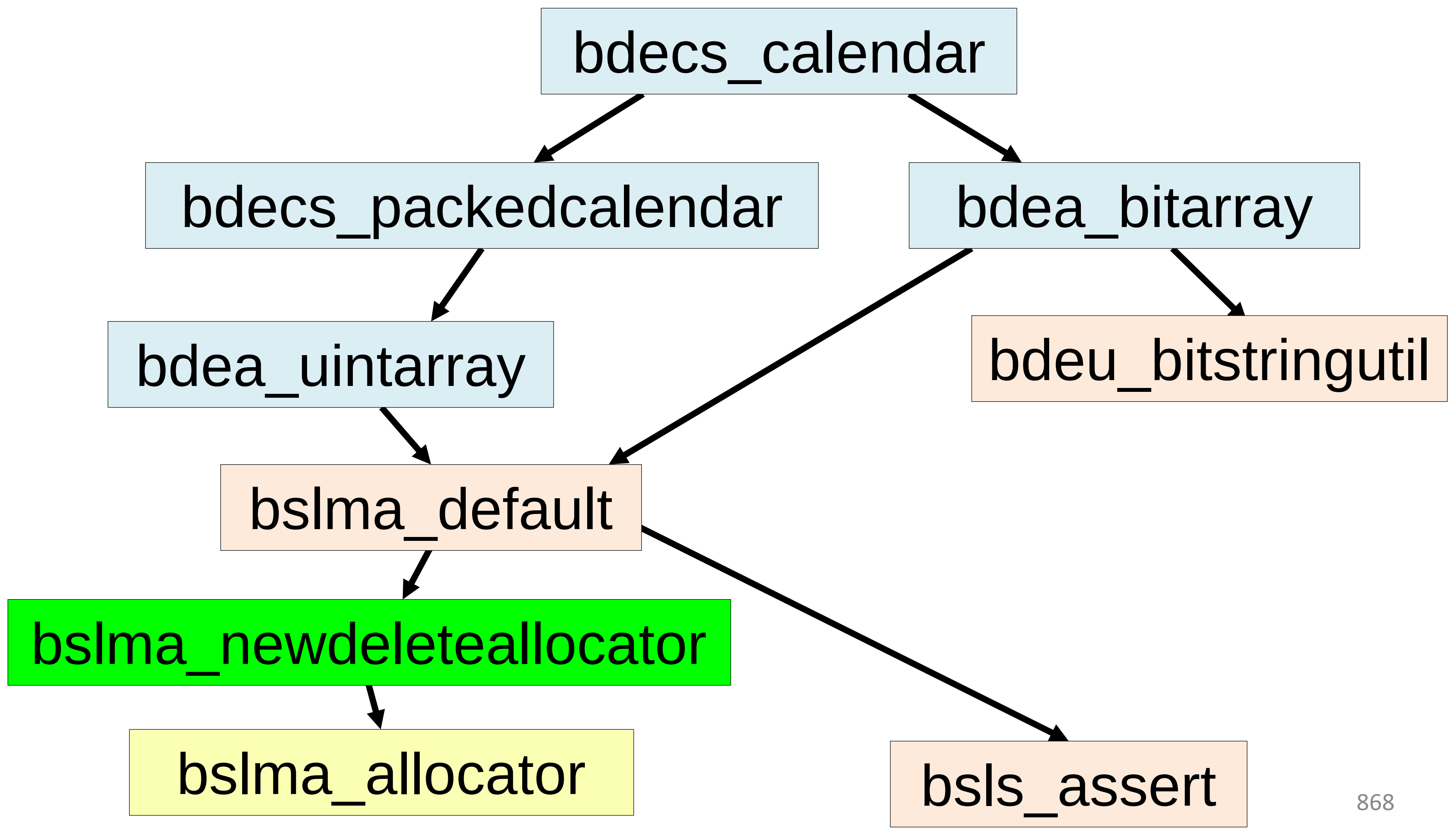
4. Bloomberg Development Environment

Implementing `bdecs_calendar`

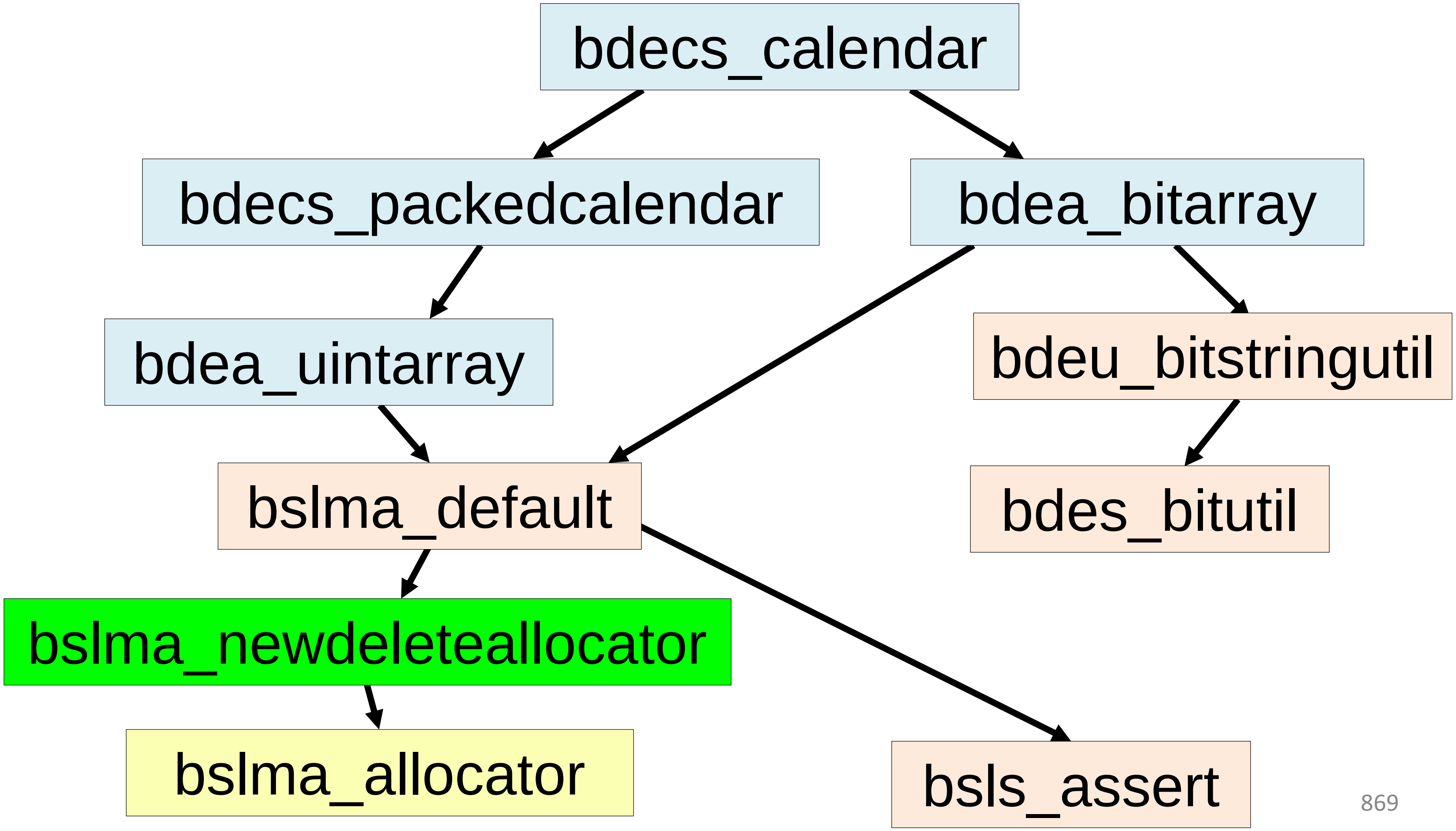


4. Bloomberg Development Environment

Implementing `bdecs_calendar`

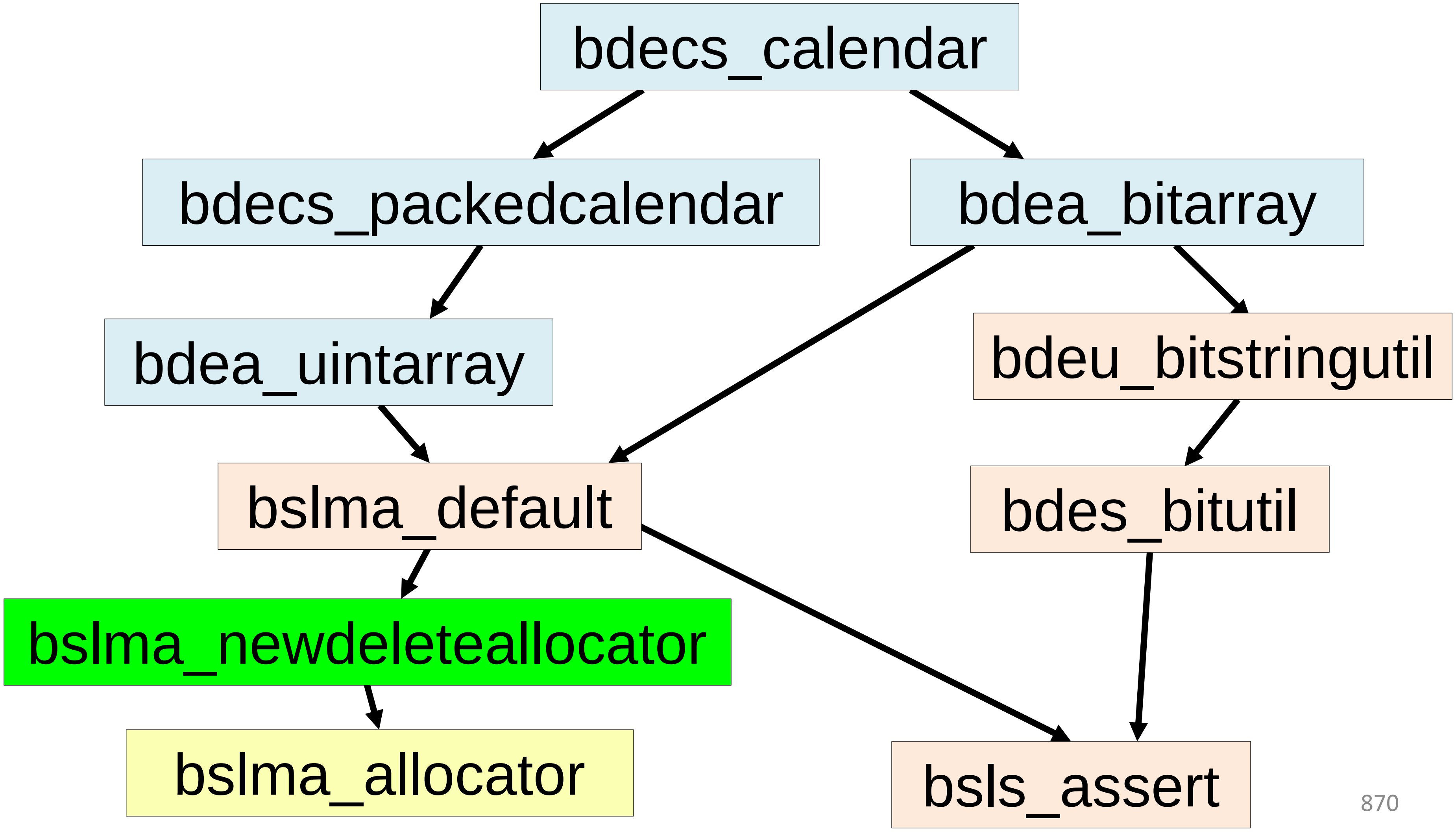


Implementing `bdecs_calendar`



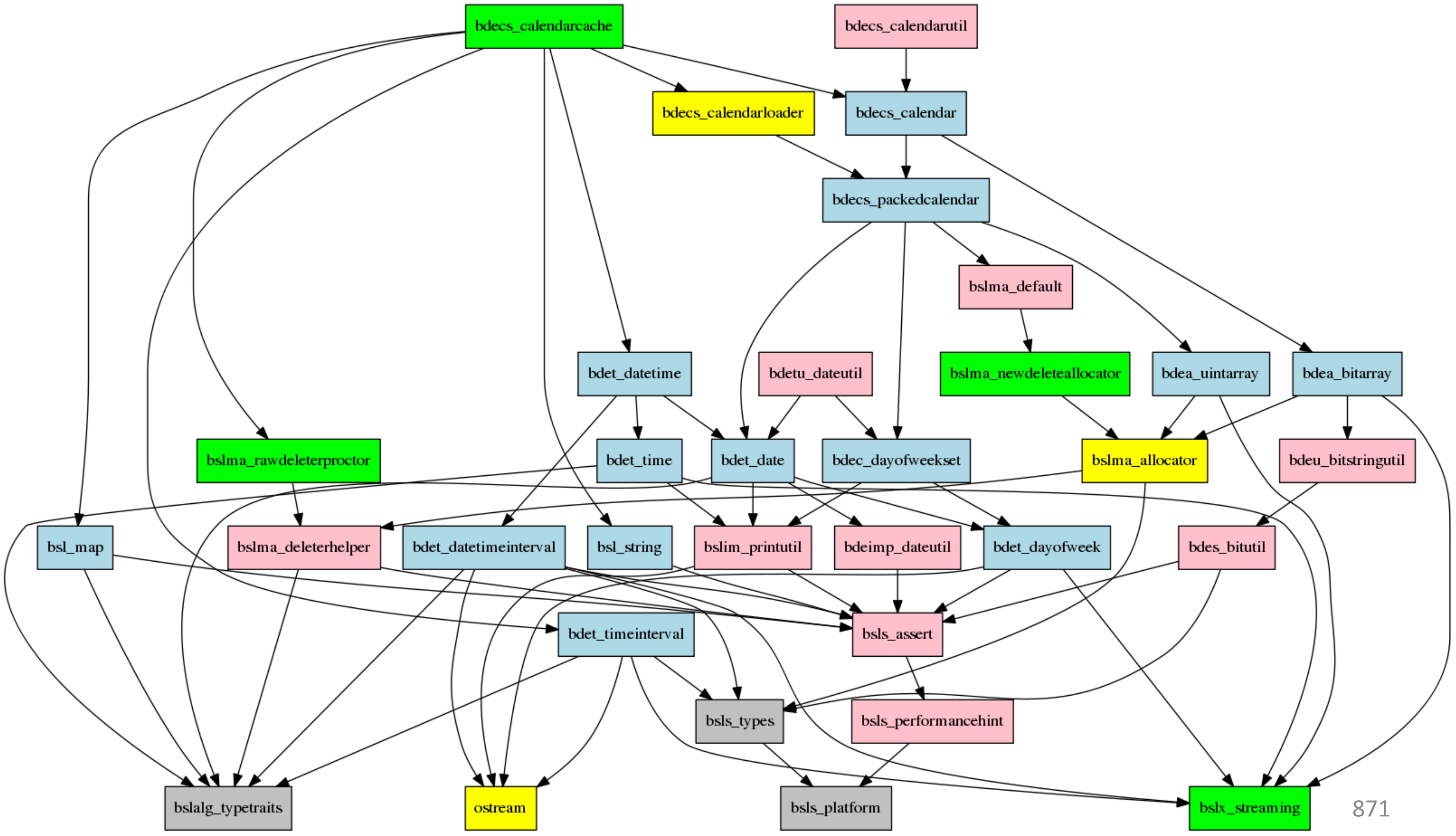
4. Bloomberg Development Environment

Implementing bdecs_calendar



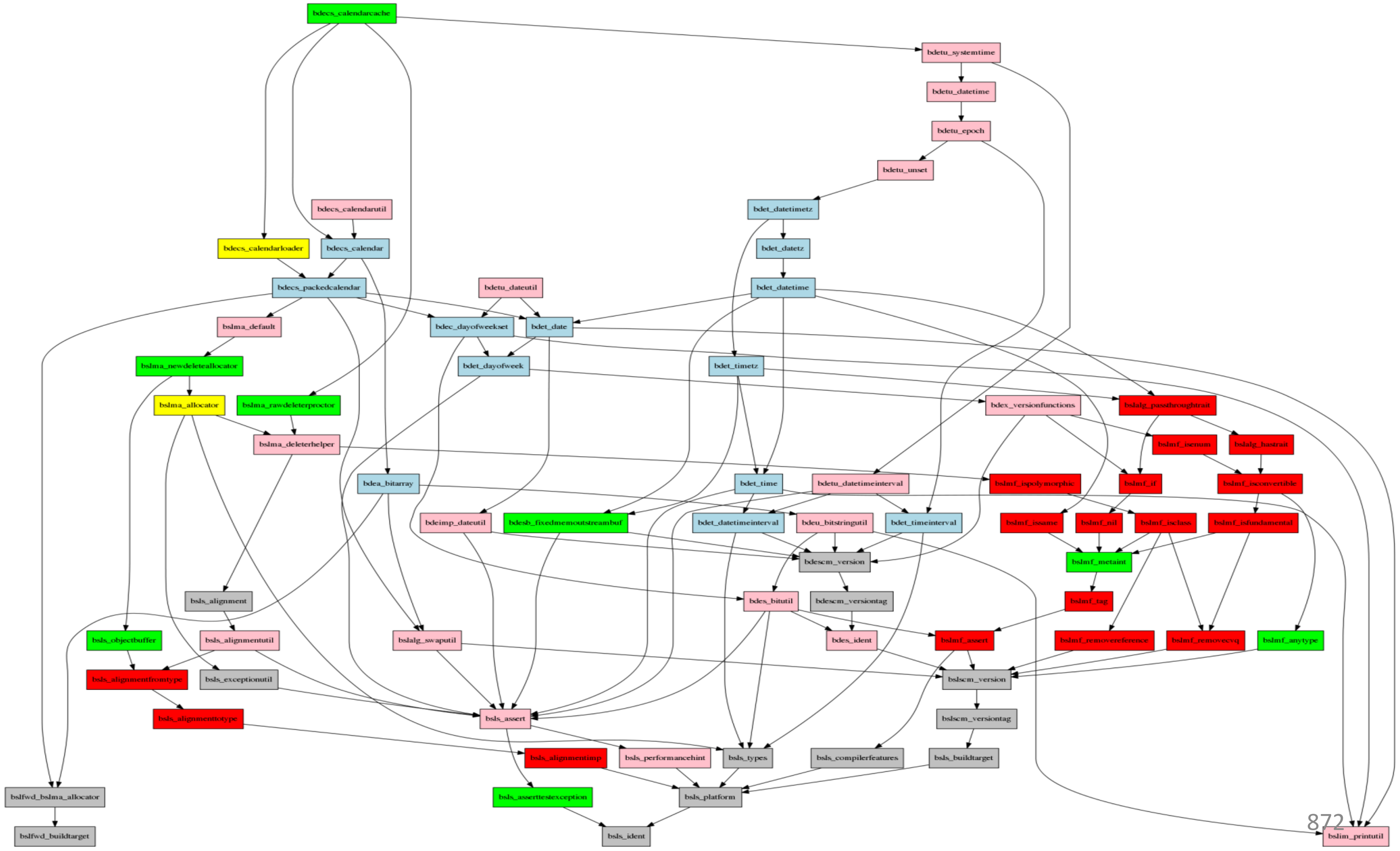
4. Bloomberg Development Environment

Hierarchically Reusable Implementation



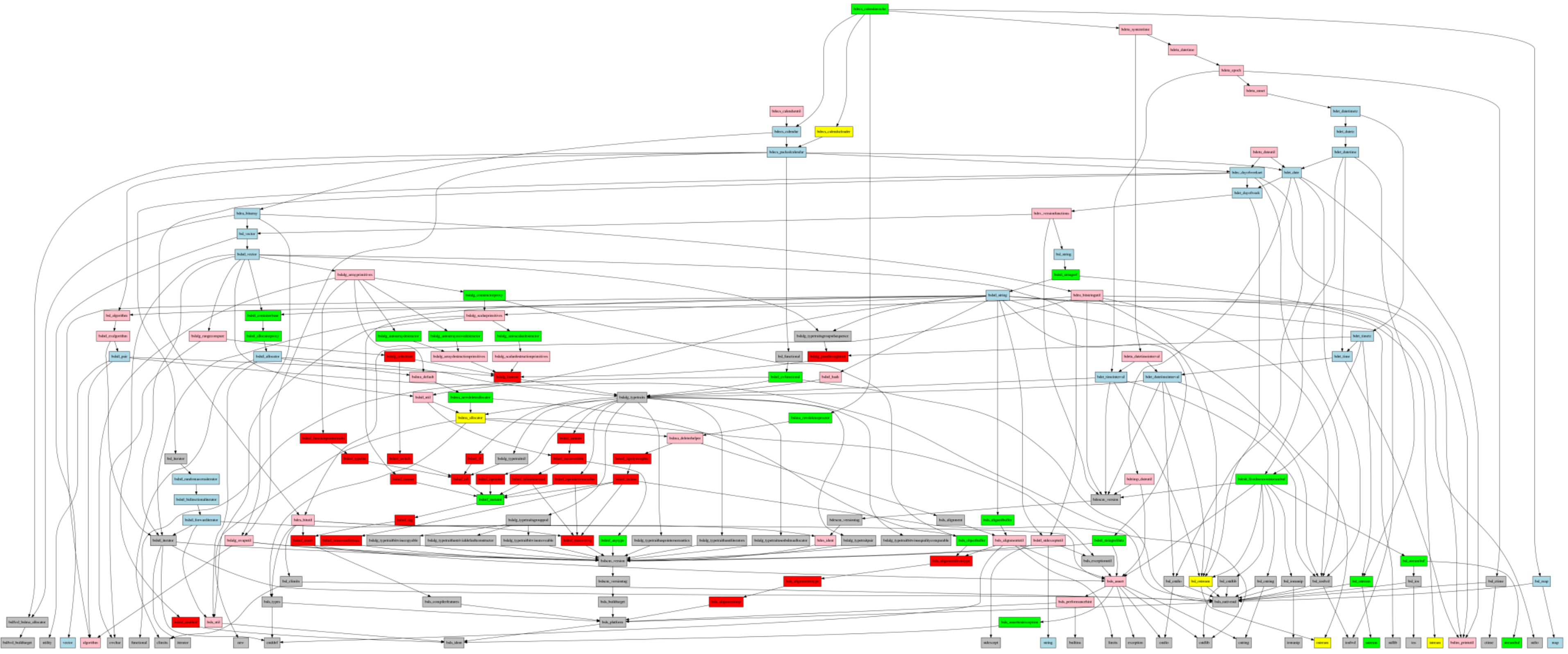
4. Bloomberg Development Environment

Hierarchically Reusable Implementation



4. Bloomberg Development Environment

Hierarchically Reusable Implementation



4. Bloomberg Development Environment

Foundation “Package-Group” Libraries

$$\begin{tikzcd}
 & & \boxed{BAE} \arrow[d] & \boxed{BTE} \arrow[dl] \\
\boxed{BBE} \arrow[dr] & & \boxed{BCE} \arrow[dl] & \\
 & \boxed{BDE} \arrow[d] & & \\
 & \boxed{BSL} & &
\end{tikzcd}$$

Outline

0. Goals

   What we are trying to do, for whom, and how.

1. Process & Architecture

   Organizing Software as Components, Packages, & Package Groups.

2. Design & Implementation

   Using Class Categories, Value Semantics, & Vocabulary Types.

3. Verification & Testing

   Component-Level Test Drivers, Peer Review, & Defensive Checks.

4. Bloomberg Development Environment (BDE)

   Rendered as Fine-Grained *Hierarchically* *Reusable* Components.

Outline

0. Goals

   What we are trying to do, for whom, and how.

1. Process & Architecture

   Organizing Software as Components, Packages, & Package Groups.

2. Design & Implementation

   Using Class Categories, Value Semantics, & Vocabulary Types.

3. Verification & Testing

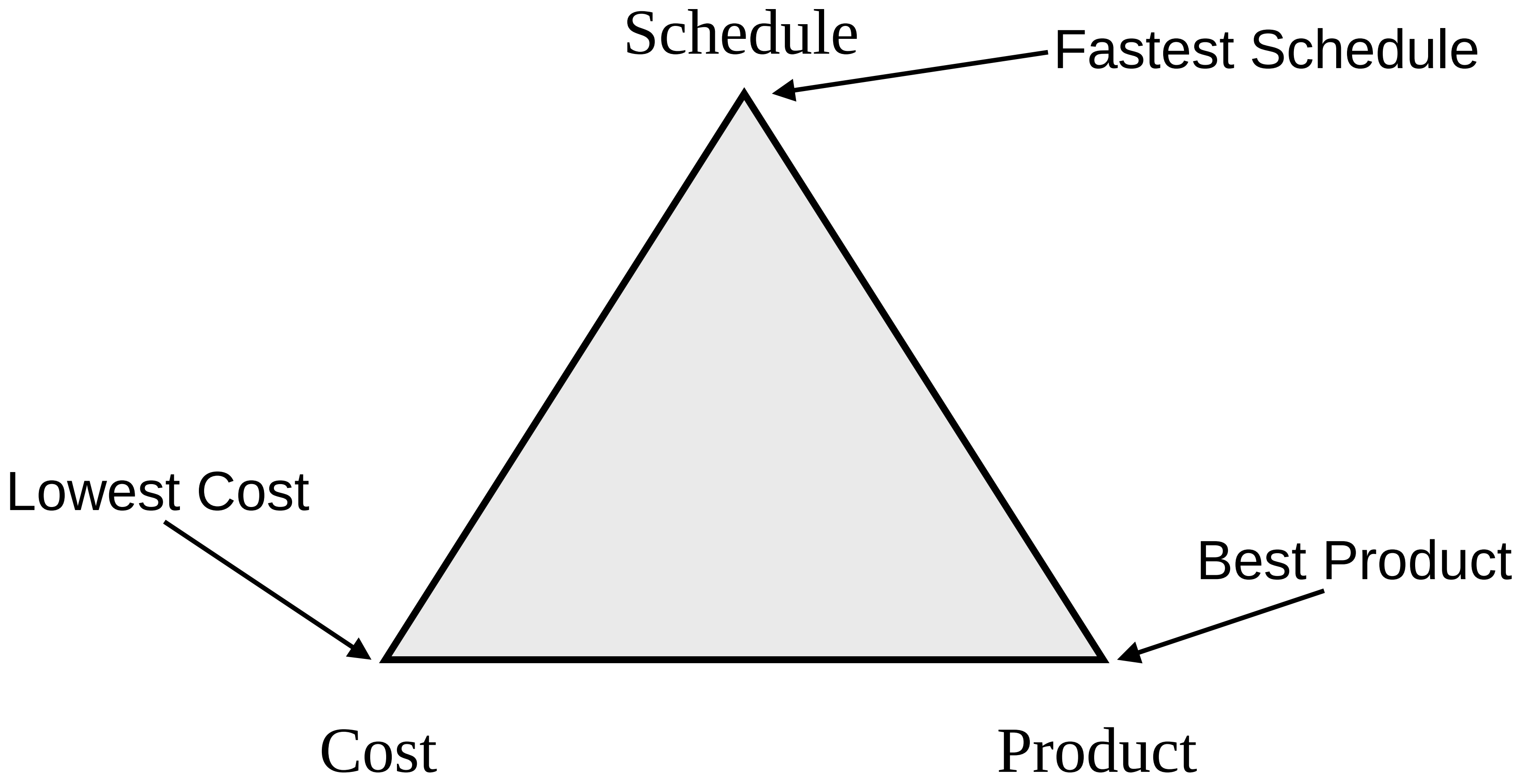
   Component-Level Test Drivers, Peer Review, & Defensive Checks.

4. Bloomberg Development Environment (BDE)

   Rendered as Fine-Grained *Hierarchically* *Reusable* Components.

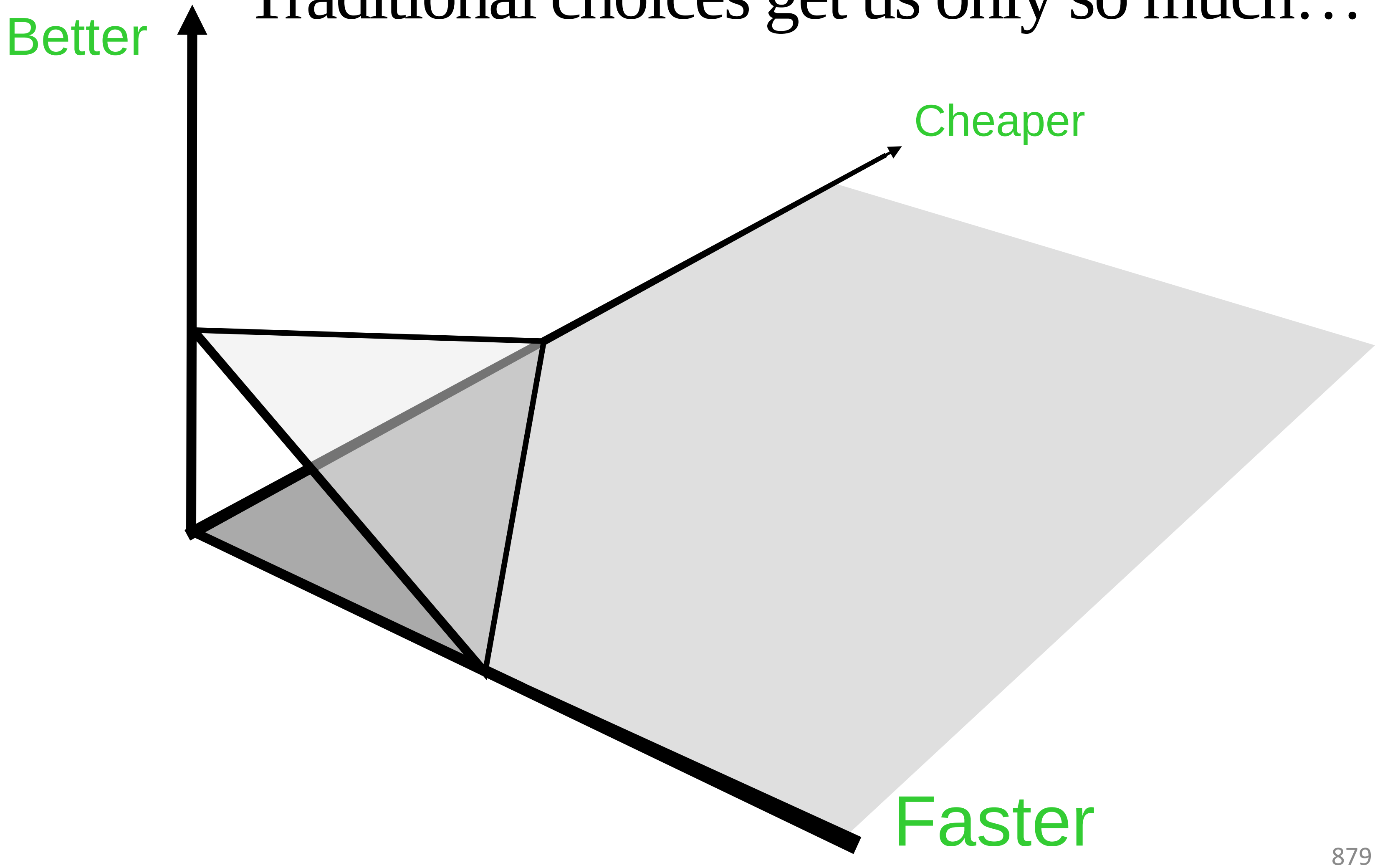
Conclusion

Conclusion
The Goal: Faster, Better, Cheaper!



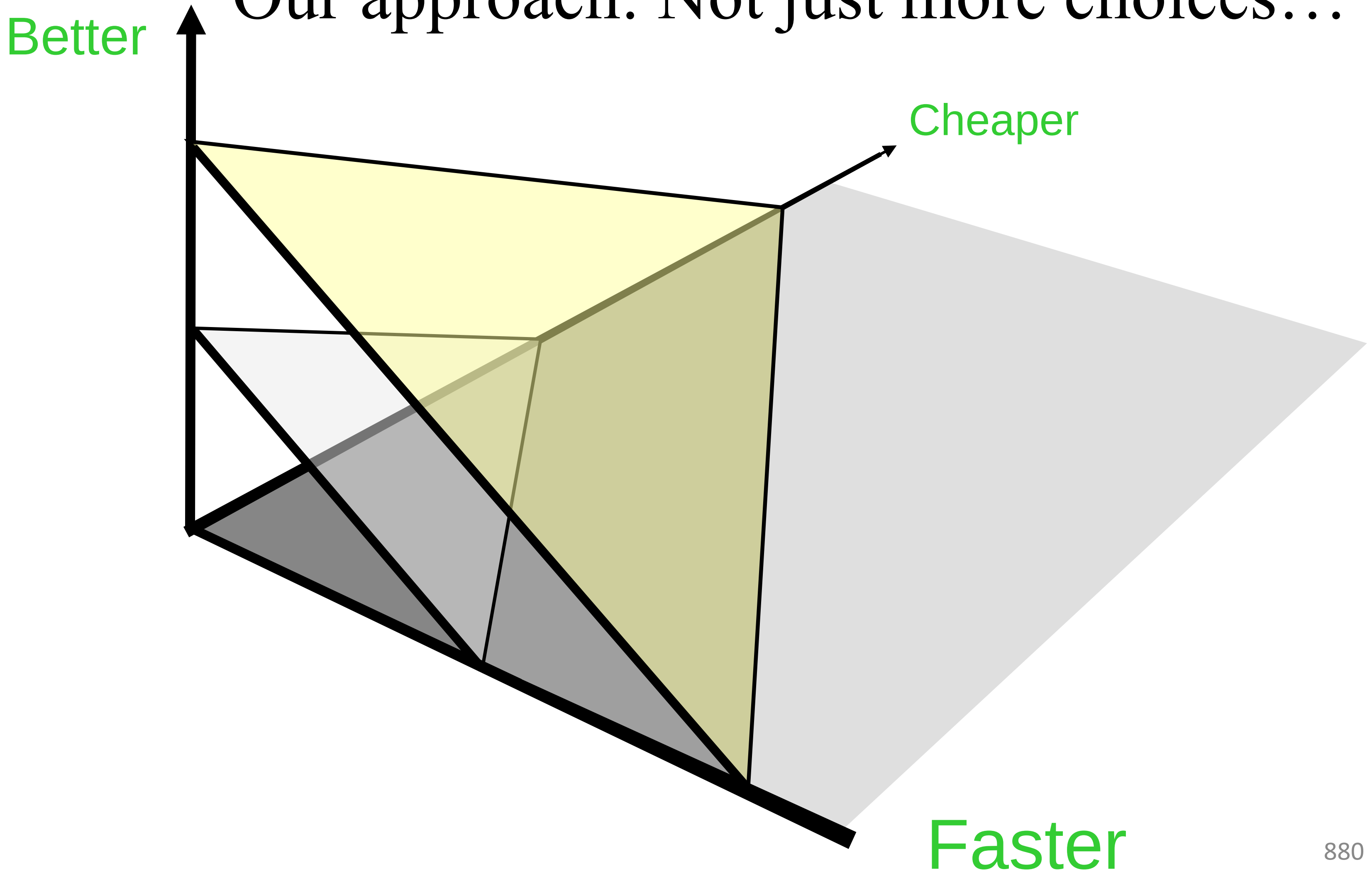
Conclusion

“Traditional choices get us only so much…”



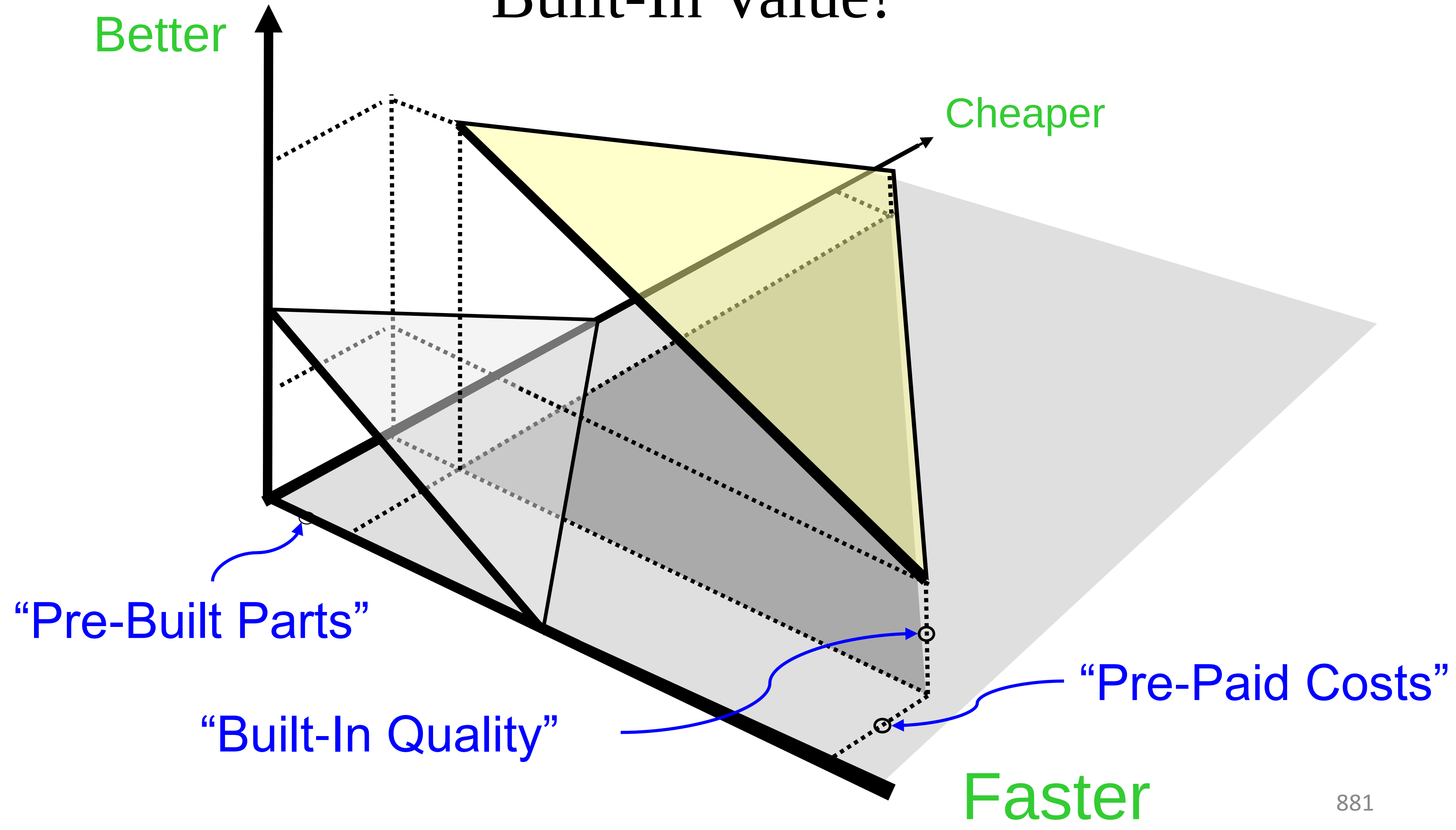
Conclusion

Our approach: Not just more choices…



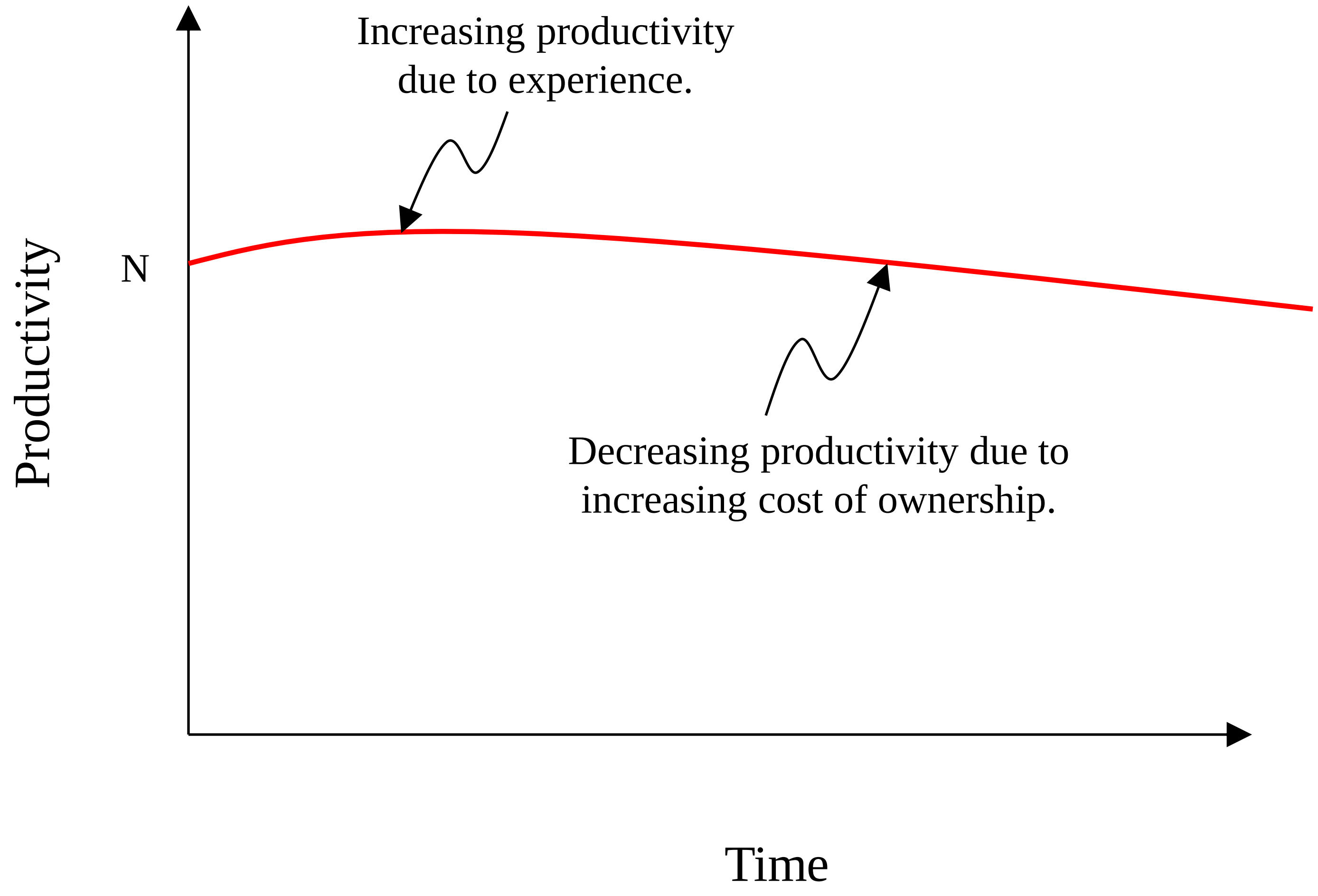
Conclusion

Built-In Value!



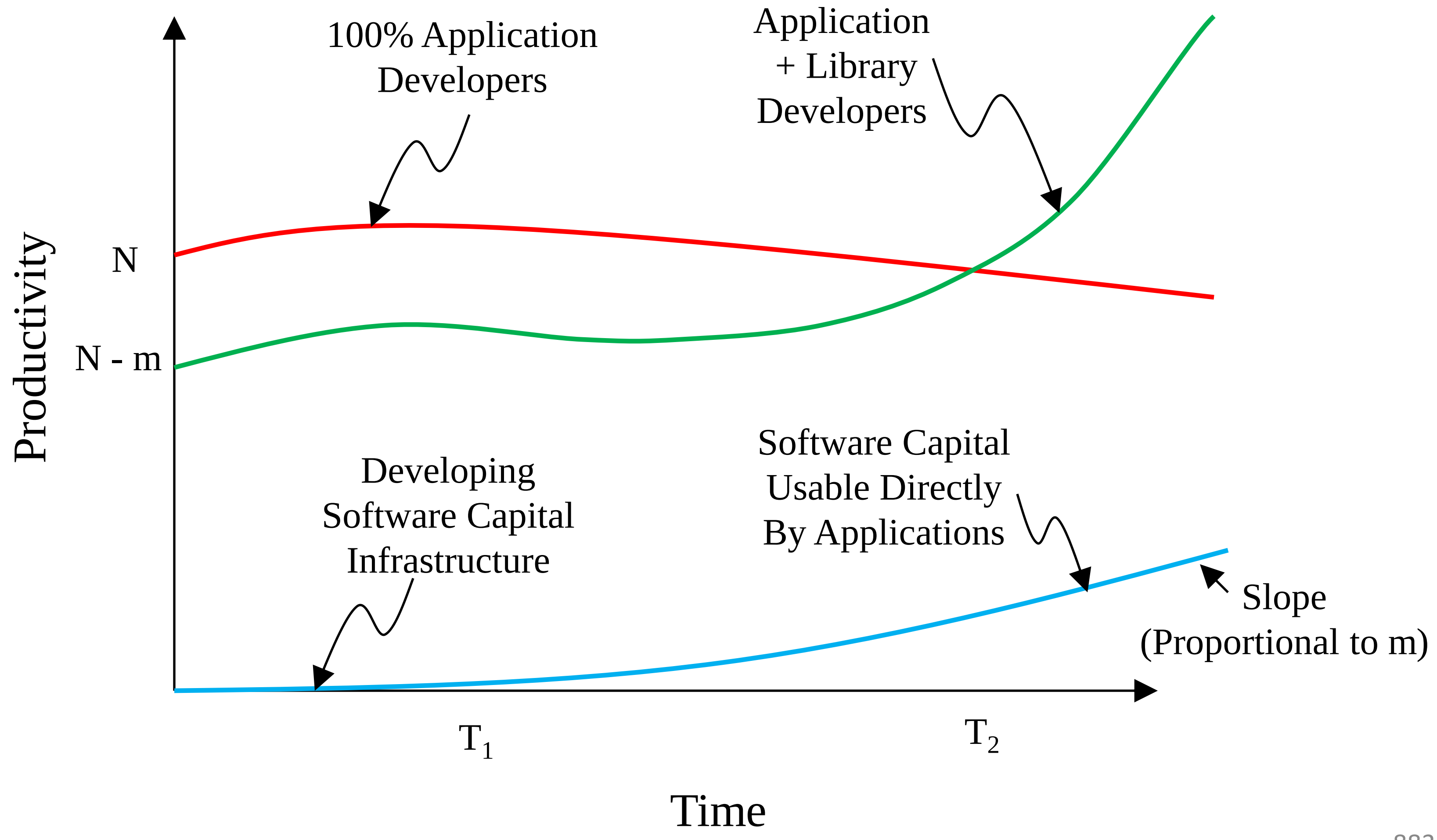
Conclusion

Productivity: Homogeneous Development Team



Conclusion

Productivity: Heterogeneous Development Team



Conclusion

So what are the take-aways?

Conclusion

So what are the take-aways?

- We have exhibited a proven methodology that yields hierarchically reusable libraries.

Conclusion

So what are the take-aways?

- We have exhibited a proven methodology that yields hierarchically reusable libraries.
- We are open-sourcing the root of such a hierarchy as a framework and to demonstrate how it is done.

Conclusion

- Find our open-source distribution at: **`http://www.openbloomberg.com/bsl`**
- Moderator: **`kpfleming@bloomberg.net`**
- How to contribute? *See our site.*
- All comments and criticisms welcome...

Conclusion

- Find our open-source distribution at: **`http://www.openbloomberg.com/bsl`**
- Moderator: **`kpfleming@bloomberg.net`**
- How to contribute? *See our site.*
- All comments and criticisms welcome...

The End