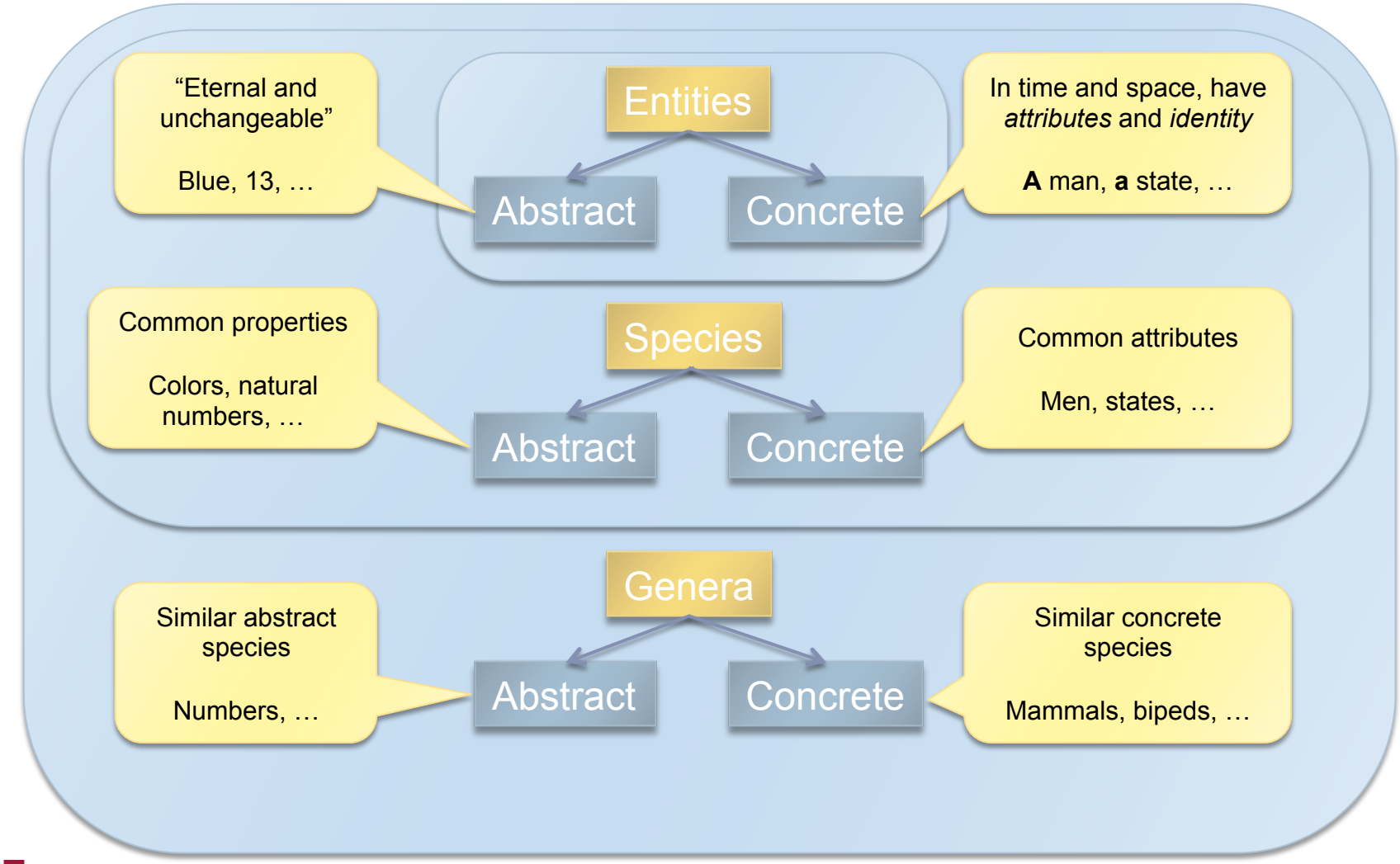
Entities, Species, Genera, Value Types, Computational Bases, and Concepts

What is the Best Way to Glue Code Together?

What is this talk about?

- Part I
  - Introduce a taxonomy of ideas in programming
    - From “Elements of Programming” by Stepanov and McJones
- Part II
  - Consider concepts
    - Are concepts **the** abstraction we want and need?
    - What concerns do concepts combine?
  - Punch line: Concepts may be a wrong solution
    - I only hint what we could try to do
    - No, I don’t have a language and an implementation

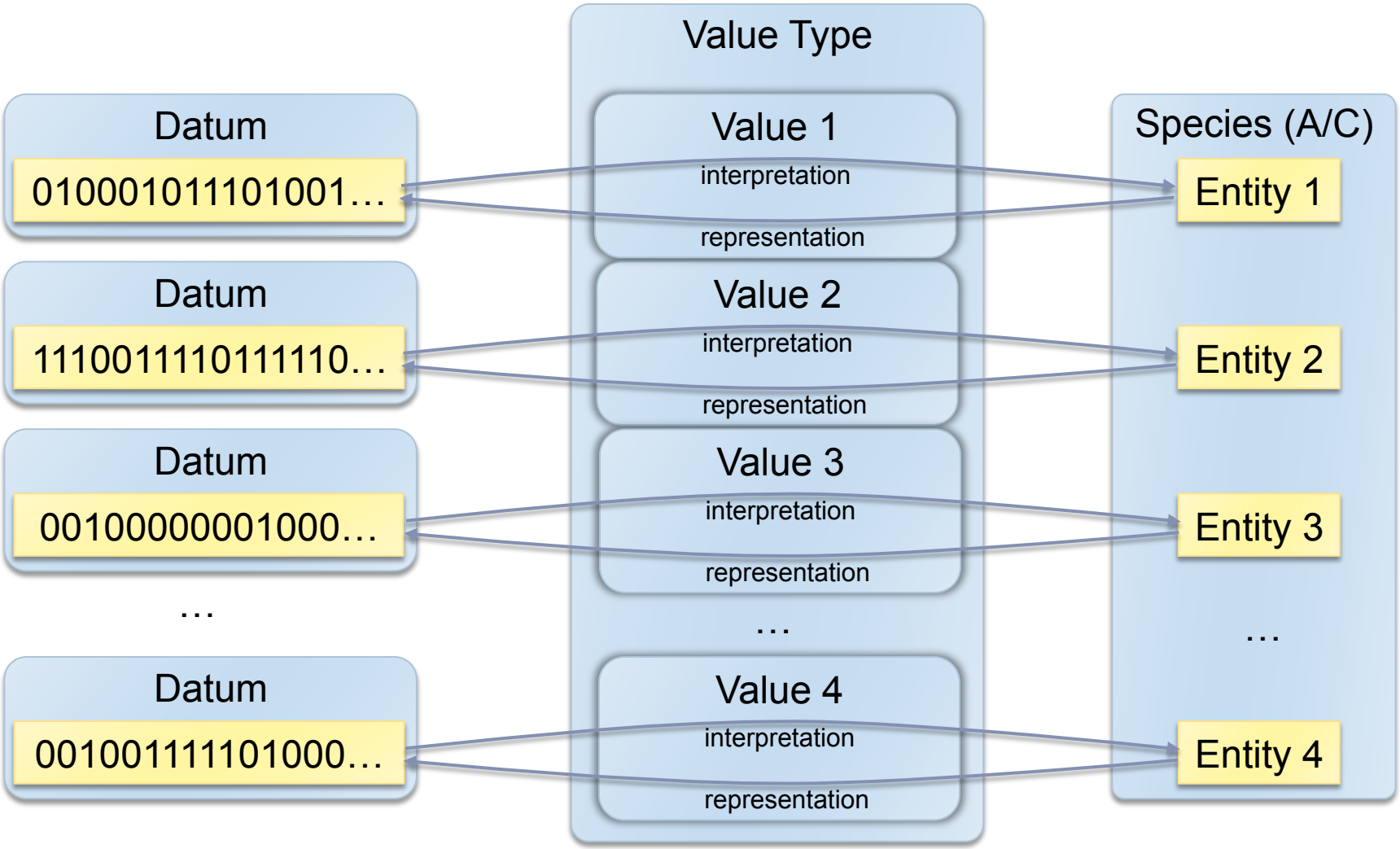
Category of Ideas: Entity, Species, Genus



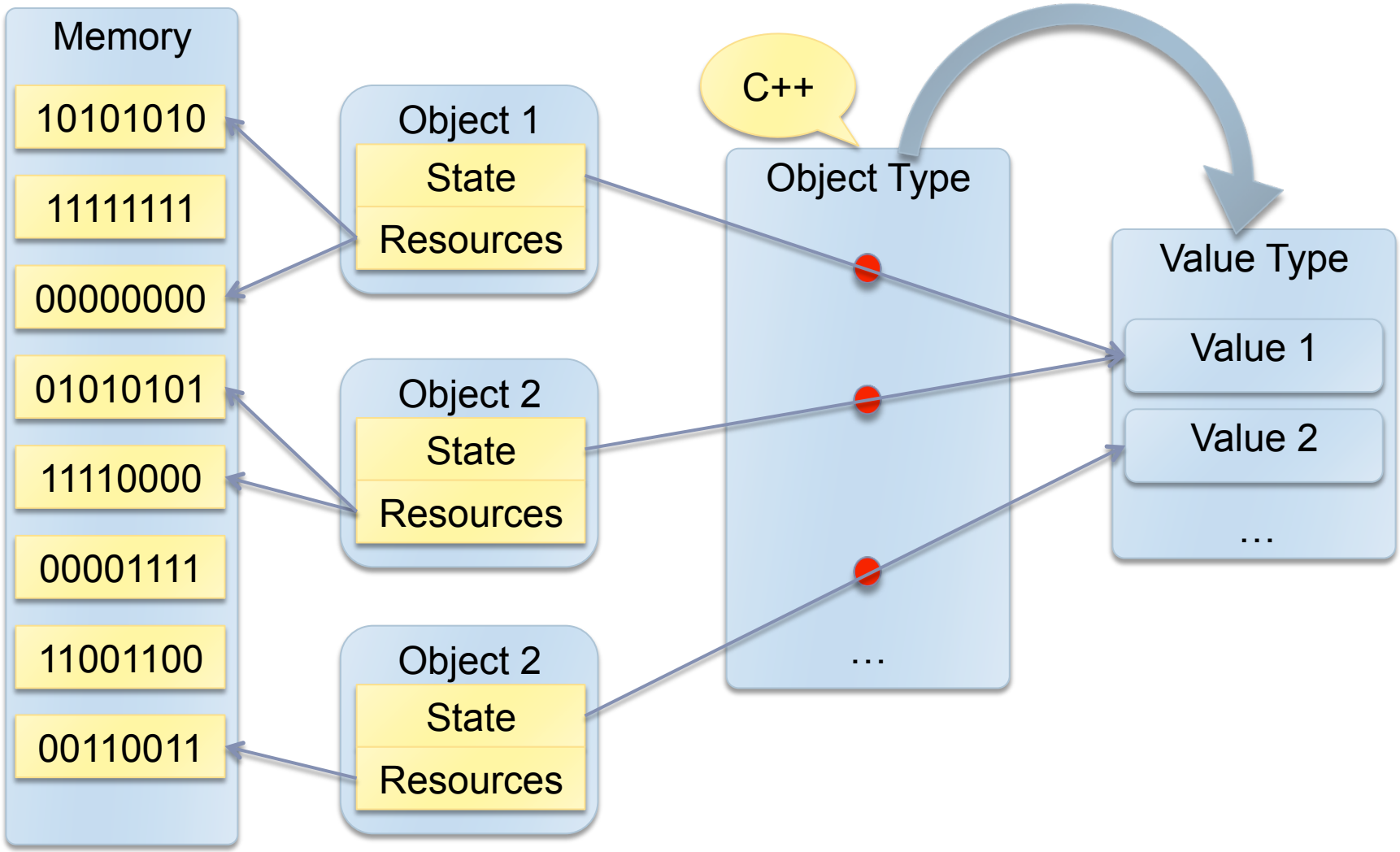
Category of Ideas: Entity, Species, Genus

- These categories are what we think
  - We use these in discussions
  - We write these on whiteboards
- A good library attempts:
  - to provide abstractions that mimic these categories
  - provide “implementations” of these abstractions
- But, what language mechanisms do we have to help us with the task?
  - The rest of the talk

Values

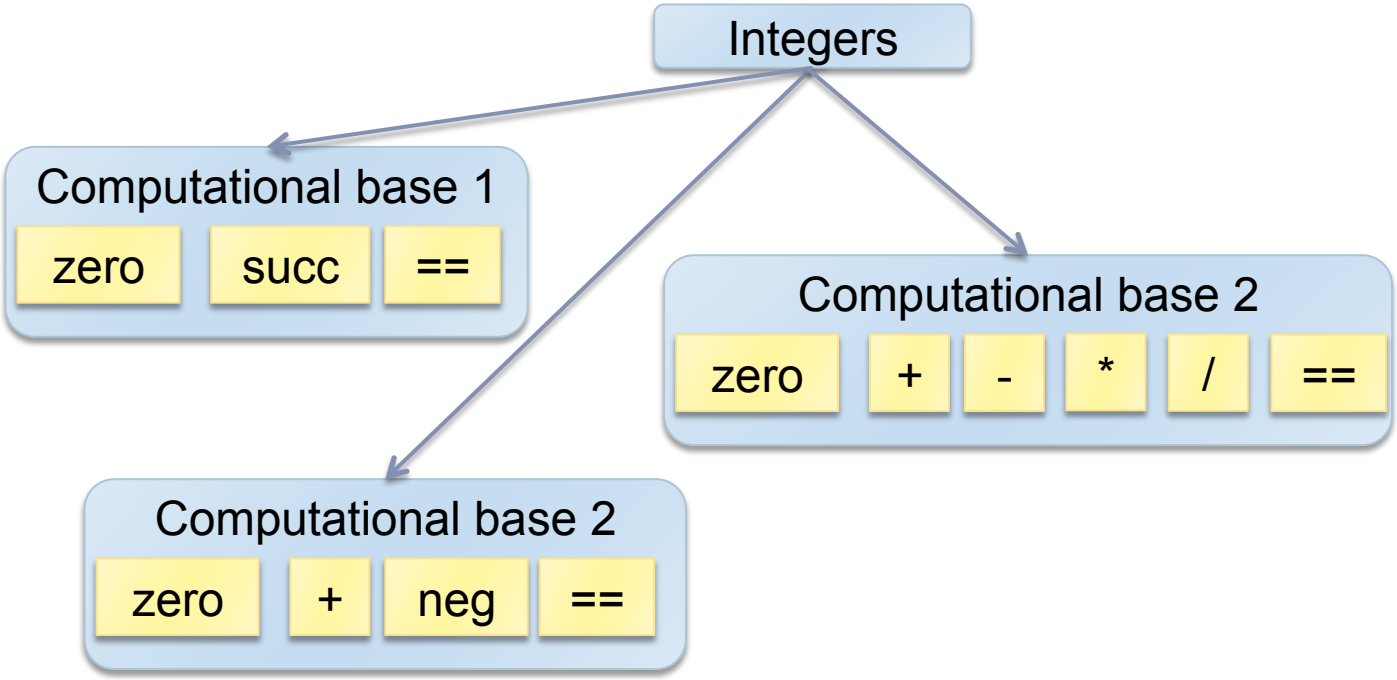

Value Type
Datum
0100010111010010…
Datum
1110011110111110…
Datum
0010000001000…
…
Datum
0010011111010000…
Value 1
interpretation
representation
Value 2
interpretation
representation
Value 3
interpretation
representation
…
Value 4
interpretation
representation
Species (A/C)
Entity 1
Entity 2
Entity 3
…
Entity 4

Objects


Memory
10101010
11111111
00000000
01010101
11110000
00001111
11001100
00110011
Object 1
State
Resources
Object 2
State
Resources
Object 2
State
Resources
C++
Object Type
…
Value Type
Value 1
Value 2
…

Computational Bases

- Procedures modify, construct, or destroy objects
- Each object type has one or more *computational bases*



. . .

Concepts: Abstraction Level and Concerns

- We have seen all that rich development of ideas
- Concepts do away with all of it and:

  A concept is a description of requirements on one or more (object) types stated in terms of the existence and properties of procedures, type attributes, and type functions defined on types.

- Concept concerns
  - Procedure-level requirements
  - Performance
  - Species

Problems with Concepts

- Concepts end up being too specific
  - Trying to match particular procedures
  - Building hierarchies of performance requirements
- Concepts do not correspond to entities, value types, or to computational bases
- Concepts result in rigid hierarchies and high coupling
- Concepts work well for C++, but what about next generation of “programming systems”
  - More of programming should be automated
  - A “programming system” could help us assemble code
  - Rigid interfaces would stand in the way

Topics for Discussion

- Separate concerns
  - Species

```
species Number;
species NaturalNumber refines Number;
```

  - Computational bases (why only for object types?), “concept maps”

```
comp_base<species NaturalNumber> {
  NaturalNumber zero();
  NaturalNumber succ(NaturalNumber);
}
```

or

```
NaturalNumber operator+(NaturalNumber, NaturalNumber);
NaturalNumber operator-(NaturalNumber, NaturalNumber);
...
```

  - Implicit requirements, performance requirements, property requirements

```
template<species NaturalNumber>
void f(NaturalNumber nat) {
  add(nat, nat); [[Constant]]
}
```

```
Matrix mult(Matrix m, Matrix n)
require Diagonal(m), Sparse(m) {
  m.mult(n); // default implementation
}
```

  - Interface enforcement
    - Chunks of computational bases
    - Computed by an IDE, versioned, overloaded on …

Credits

- Andreas Priesnitz
  - “Multistage Algorithms in C++”