A Practitioner's Categorical Guide to Complex Data

Mike Caruso

Background

- *Vision* is a temporal object-oriented analytic database platform that's been powering large-scale production analytics and decision support for investment managers (and others) for over 30 years.
- A technology whose foundations and insights I look forward to sharing with you now.

A Simple Hard Problem

A mutual fund manager needs to measure holdings overlap with other funds

```
Account defineMethod: [ | getHoldingsOverlap |
!lowerPct <- pctEq * 0.8; !upperPct <- ... ;
holdings send: [security].
  select: [type isEquity].
  collectListElementsFrom: [holdings].
  groupedBy: [account].
  select: [pctEq >= ^my lowerPct
    && pctEq <= ^my upperPct].
  extendBy: [
    !xref <- ^my holdings;
    !ofactor <- groupList total: [
      percentOfPort min: (^my xref at: security.
        percentOfPort)
    ]
  ].
  sortDown: [ofactor]
];
```

A Simple Hard Problem

A mutual fund manager needs to measure holdings overlap with other funds

Is this an application program or a database query?

```
Account defineMethod: [ | getHoldingsOverlap |
!lowerPct <- pctEq * 0.8; !upperPct <- ... ;
holdings send: [security].
  select: [type isEquity].
  collectListElementsFrom: [holdings].
  groupedBy: [account].
  select: [pctEq >= ^my lowerPct
   && pctEq <= ^my upperPct].
  extendBy: [
   !xref <- ^my holdings;
   !ofactor <- groupList total: [
     percentOfPort min: (^my xref at: security.
       percentOfPort)
   ]
  ].
  sortDown: [ofactor]
];
```

A Simple Hard Problem

A mutual fund manager needs to measure holdings overlap with other funds

Is this an application program or a database query?

This product space query can easily touch large parts of a large database.

```
Account defineMethod: [ | getHoldingsOverlap |
!lowerPct <- pctEq * 0.8; !upperPct <- ... ;
holdings send: [security].
  select: [type isEquity].
  collectListElementsFrom: [holdings].
  groupedBy: [account].
  select: [pctEq >= ^my lowerPct
   && pctEq <= ^my upperPct].
  extendBy: [
   !xref <- ^my holdings;
   !ofactor <- groupList total: [
     percentOfPort min: (^my xref at: security.
       percentOfPort)
   ]
  ].
  sortDown: [ofactor]
];
```

A Simple Hard Problem

A mutual fund manager needs to measure holdings overlap with other funds

Is this an application program or a database query?

This product space query can easily touch large parts of a large database.

Analysis routinely crosses and combines levels of abstraction and aggregation.

```
Account defineMethod: [ | getHoldingsOverlap |
!lowerPct <- pctEq * 0.8; !upperPct <- ... ;
holdings send: [security].
  select: [type isEquity].
  collectListElementsFrom: [holdings].
  groupedBy: [account].
  select: [pctEq >= ^my lowerPct
    && pctEq <= ^my upperPct].
  extendBy: [
    !xref <- ^my holdings;
    !ofactor <- groupList total: [
      percentOfPort min: (^my xref at: security.
        percentOfPort)
    ]
  ].
  sortDown: [ofactor]
];
```

A Simple Hard Problem

A mutual fund manager needs to measure holdings overlap with other funds

Is this an application program or a database query?

This product space query can easily touch large parts of a large database.

Analysis routinely crosses and combines levels of abstraction and aggregation.

What insights can be found to help this scale?

```
Account defineMethod: [ | getHoldingsOverlap |
!lowerPct <- pctEq * 0.8; !upperPct <- ... ;
holdings send: [security].
  select: [type isEquity].
  collectListElementsFrom: [holdings].
  groupedBy: [account].
  select: [pctEq >= ^my lowerPct
   && pctEq <= ^my upperPct].
  extendBy: [
   !xref <- ^my holdings;
   !ofactor <- groupList total: [
     percentOfPort min: (^my xref at: security.
       percentOfPort)
   ]
  ].
  sortDown: [ofactor]
];
```

A Simple Hard Problem

A mutual fund manager needs to measure holdings overlap with other funds

Is this an application program or a database query?

This product space query can easily touch large parts of a large database.

Analysis routinely crosses and combines levels of abstraction and aggregation.

What insights can be found to help this scale?

```
Account defineMethod: [ | getHoldingsOverlap |
!lowerPct <- pctEq * 0.8; !upperPct <- ... ;
holdings send: [security].
  select: [type isEquity].
  collectListElementsFrom: [holdings].
  groupedBy: [account].
  select: [pctEq >= ^my lowerPct
   && pctEq <= ^my upperPct].
  extendBy: [
   !xref <- ^my holdings;
   !ofactor <- groupList total: [
     percentOfPort min: (^my xref at: security.
       percentOfPort)
   ]
  ].
  sortDown: [ofactor]
];
```

What if this application is run for a collection of accounts?

```
FundUniverse do: [
    ^self getHoldingsOverlap do: […]
]
```

A Simple Hard Problem

A mutual fund manager needs to measure holdings overlap with other funds

Is this an application program or a database query?

This product space query can easily touch large parts of a large database.

Analysis routinely crosses and combines levels of abstraction and aggregation.

What insights can be found to help this scale?

```
Account defineMethod: [ | getHoldingsOverlap |
!lowerPct <- pctEq * 0.8; !upperPct <- ... ;
holdings send: [security].
  select: [type isEquity].
  collectListElementsFrom: [holdings].
  groupedBy: [account].
  select: [pctEq >= ^my lowerPct
   && pctEq <= ^my upperPct].
  extendBy: [
   !xref <- ^my holdings;
   !ofactor <- groupList total: [
     percentOfPort min: (^my xref at: security.
       percentOfPort)
   ]
  ].
  sortDown: [ofactor]
];
```

What if this application is run for a collection of accounts?

```
FundUniverse do: [
    ^self getHoldingsOverlap do: […]
]
```

Or depends on the context dependent *fabric of relationships* in the data itself?

```
2 yearsAgo evaluate: [MyFund getHoldingsOverlap ... ]

^today - 1 monthEnds to: ^today -12 monthEnds by: 1 monthEnds.
  evaluate: [ MyFund getHoldingsOverlap … ];
```

Time and Context

Time dependency is encoded in relationships…

mike bloodType	*… not time dependent*
Security named IBM price	*… time dependent*

Time and Context

Time dependency is encoded in relationships…

mike bloodType … *not time dependent*

Security named IBM price … *time dependent*

Security named IBM :price … *but seldom directly*
[prices adjustedPrice]

Security named IBM :prices
TimesSeries of PriceRecord

Security named IBM prices :adjustedPrice
[rawPrice / adjustmentFactor]

Security named IBM prices :adjustmentFactor
[security adjustmentRelativeTo: adjustmentDate]

Time and Context

Time dependency is encoded in relationships…

mike bloodType … *not time dependent*

Security named IBM price … *time dependent*

Security named IBM :price … *but seldom directly*
[prices adjustedPrice]

Security named IBM :prices
TimesSeries of PriceRecord

Security named IBM prices :adjustedPrice
[rawPrice / adjustmentFactor]

Security named IBM prices :adjustmentFactor
[security adjustmentRelativeTo: adjustmentDate]

Resolved by query context…

```
Security named IBM price

20170430 evaluate: [
    Security named IBM price
]
```

Time and Context

Time dependency is encoded in relationships…

mike bloodType … *not time dependent*

Security named IBM price … *time dependent*

Security named IBM :price … *but seldom directly*
[prices adjustedPrice]

Security named IBM :prices
TimesSeries of PriceRecord

Security named IBM prices :adjustedPrice
[rawPrice / adjustmentFactor]

Security named IBM prices :adjustmentFactor
[security adjustmentRelativeTo: adjustmentDate]

Resolved by query context…

Security named IBM price

20170430 evaluate: [
Security named IBM price
]

Context applies to more than just time…

Currency conversion
User entitlements
Model scenario

Design Retrospective

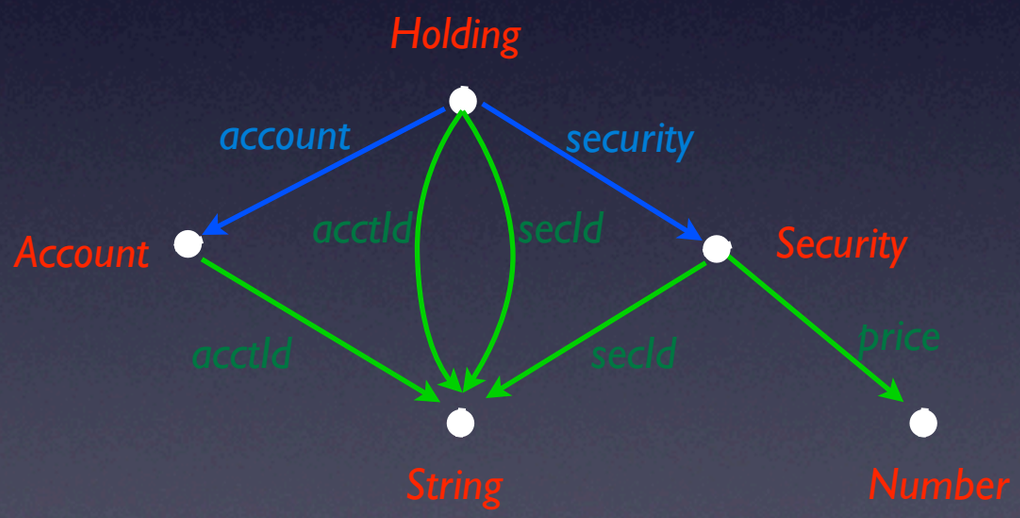
- Algebraic (matrix) Programming Languages *(APL)*
- Functional Programming Languages
- Object-Oriented Programming Languages
- Relational Algebras and Languages

```
Account defineMethod: [ | getHoldingsOverlap |
!lowerPct <- pctEq * 0.8; !upperPct <- ... ;
holdings send: [security].
  select: [type isEquity].
  collectListElementsFrom: [holdings].
  groupedBy: [account].
  select: [pctEq >= ^my lowerPct
    && pctEq <= ^my upperPct].
  extendBy: [
    !xref <- ^my holdings;
    !ofactor <- groupList total: [
      percentOfPort min: (^my xref at: security.
        percentOfPort)
    ]
  ].
  sortDown: [ofactor]
];
```

Exploring the Structure of Data

Considering…

- the cross-sectional nature of analytic applications (columns, not rows)
- the need for performance
- the unique nature of temporal data

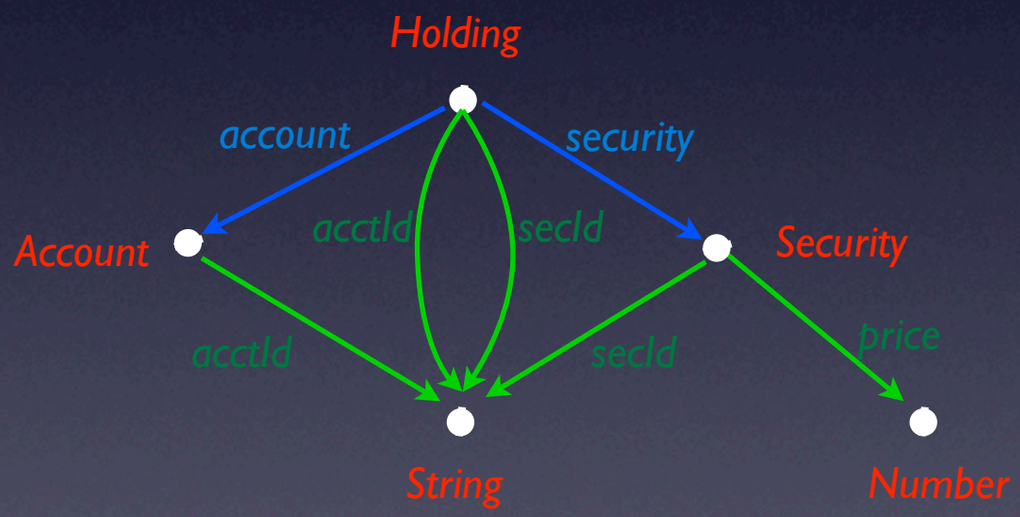


Exploring the Structure of Data

Considering…

- the cross-sectional nature of analytic applications (columns, not rows)
- the need for performance
- the unique nature of temporal data

Naturally led us to relationship centric view of the world...



Exploring the Structure of Data

Considering…

- the cross-sectional nature of analytic applications (columns, not rows)
- the need for performance
- the unique nature of temporal data

Naturally led us to relationship centric view of the world...

... in which objects (nodes, sets) are structure-*less*

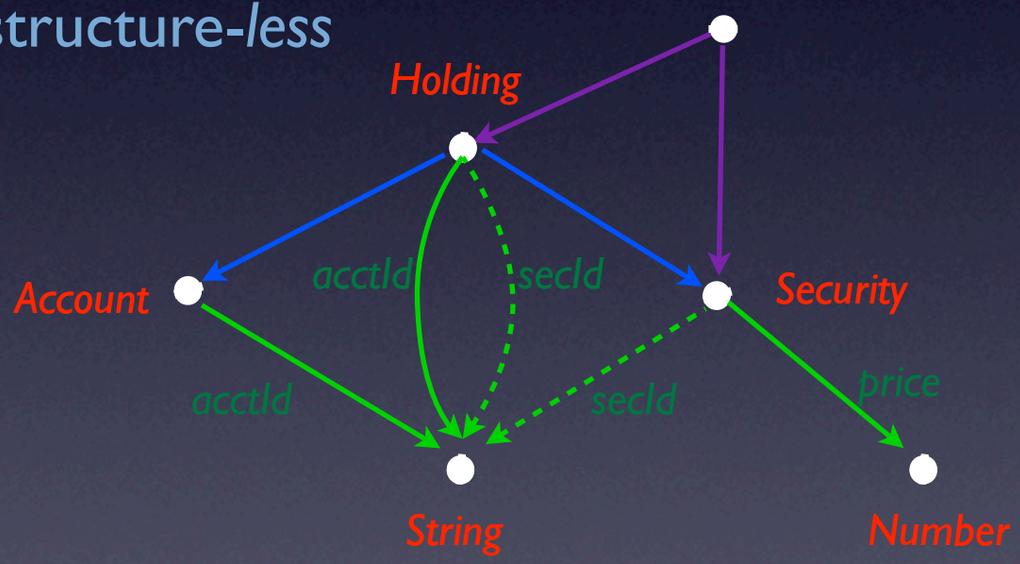
Exploring the Structure of Data

Considering…

- the cross-sectional nature of analytic applications (columns, not rows)
- the need for performance
- the unique nature of temporal data

Naturally led us to relationship centric view of the world...

... in which objects (nodes, sets) are structure-*less*



... and relationships (arrows, functions) are structure-*rich*

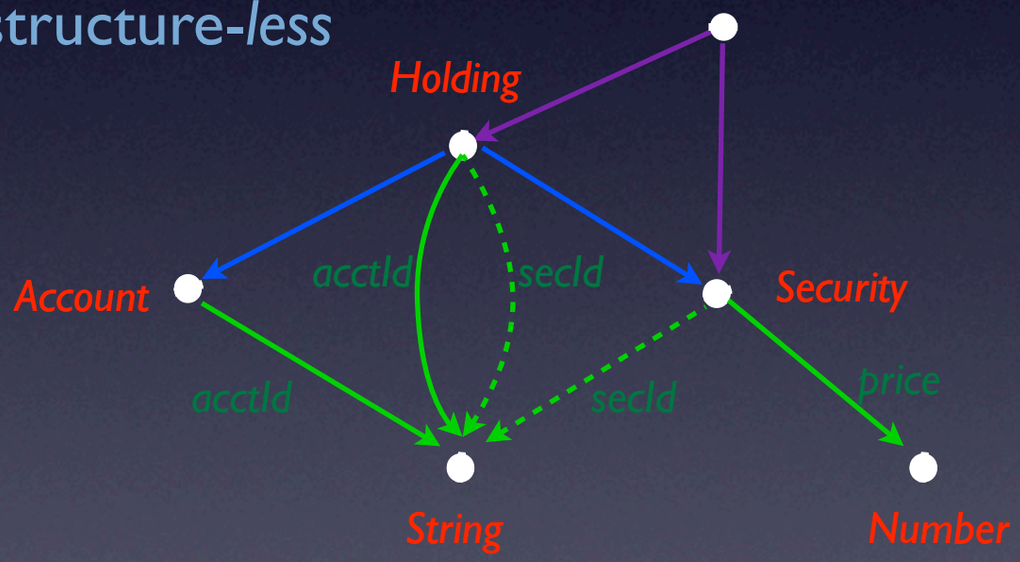
Exploring the Structure of Data

Considering…

- the cross-sectional nature of analytic applications (columns, not rows)
- the need for performance
- the unique nature of temporal data

Naturally led us to relationship centric view of the world...

... in which objects (nodes, sets) are structure-*less*



... and relationships (arrows, functions) are structure-*rich*

For some of you , that may sound a lot like category theory…

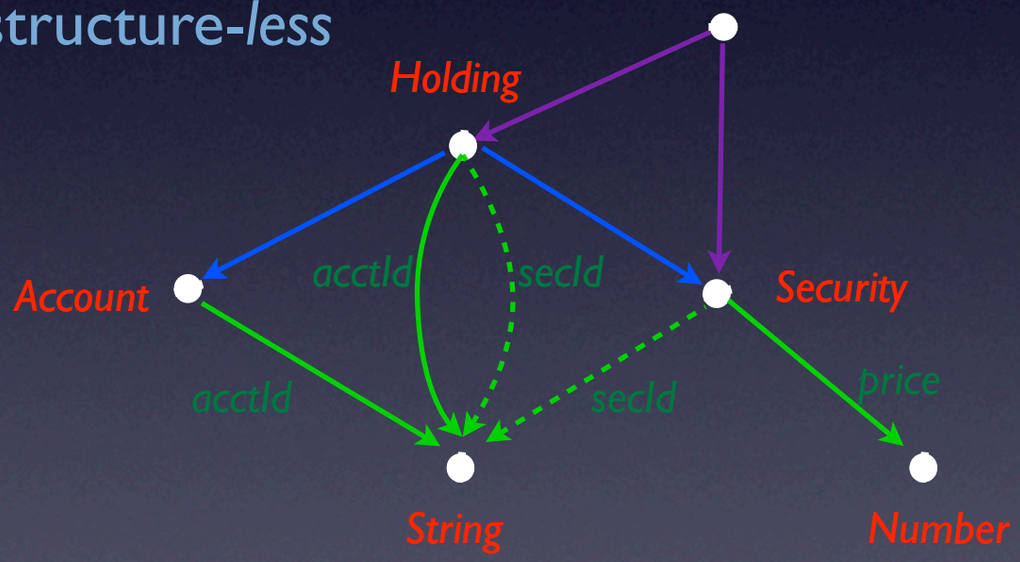
Exploring the Structure of Data

Considering…

- the cross-sectional nature of analytic applications (columns, not rows)
- the need for performance
- the unique nature of temporal data

Naturally led us to relationship centric view of the world...

... in which objects (nodes, sets) are structure-*less*



... and relationships (arrows, functions) are structure-*rich*

For some of you , that may sound a lot like category theory…

… and you'd be right.

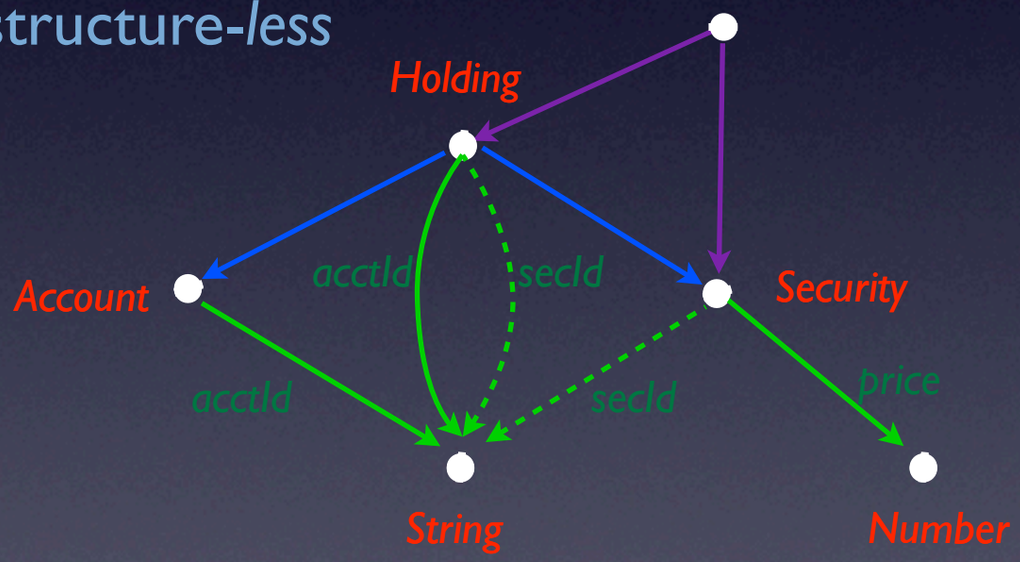
Exploring the Structure of Data

Considering…

- the cross-sectional nature of analytic applications (columns, not rows)
- the need for performance
- the unique nature of temporal data

Naturally led us to relationship centric view of the world...

... in which objects (nodes, sets) are structure-*less*



... and relationships (arrows, functions) are structure-*rich*

For some of you , that may sound a lot like category theory…

… and you'd be right.

… Subsumes and generalizes other algebraic models such as the relational model

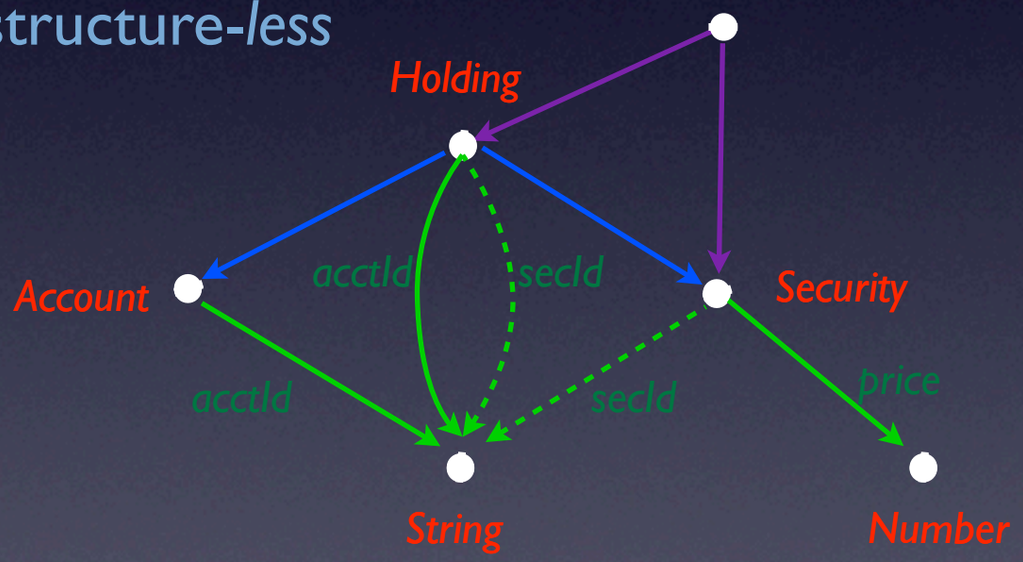
Exploring the Structure of Data

Considering…

- the cross-sectional nature of analytic applications (columns, not rows)
- the need for performance
- the unique nature of temporal data

Naturally led us to relationship centric view of the world...

... in which objects (nodes, sets) are structure-*less*



... and relationships (arrows, functions) are structure-*rich*

For some of you , that may sound a lot like category theory…

… and you'd be right.

… Subsumes and generalizes other algebraic models such as the relational model

… Provides a practical, algebraically sound, collection-centric, *implementation* model

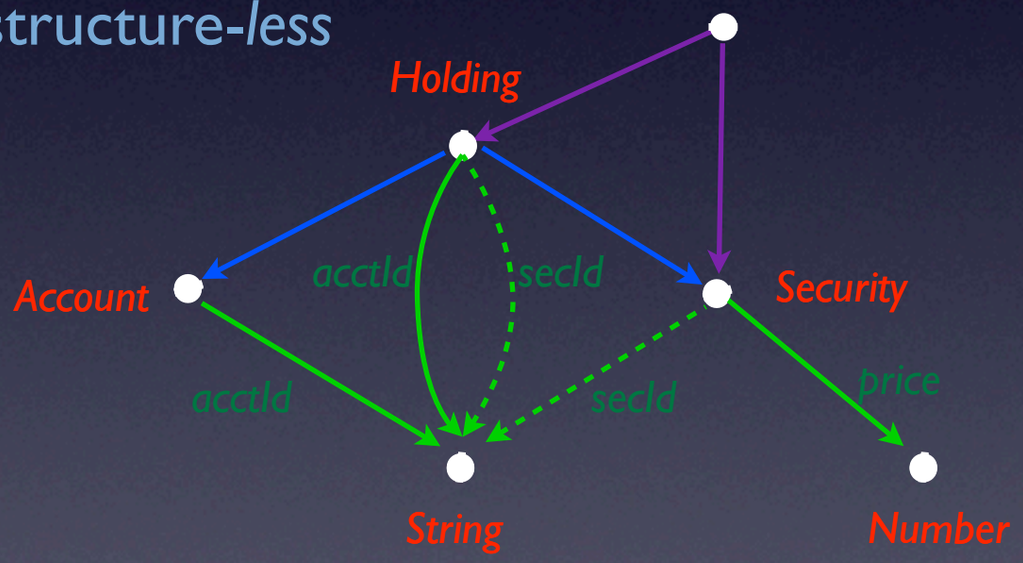
Exploring the Structure of Data

Considering…

- the cross-sectional nature of analytic applications (columns, not rows)
- the need for performance
- the unique nature of temporal data

Naturally led us to relationship centric view of the world...

... in which objects (nodes, sets) are structure-*less*



... and relationships (arrows, functions) are structure-*rich*

For some of you , that may sound a lot like category theory…

… and you'd be right.

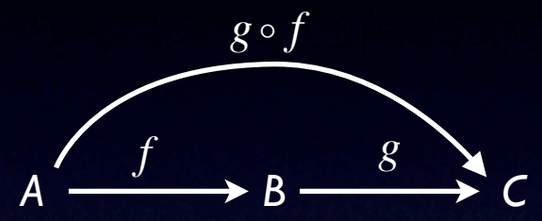
… Subsumes and generalizes other algebraic models such as the relational model

… Provides a practical, algebraically sound, collection-centric, *implementation* model

… Replaces *behavioral* characteristics *(Set, MultiSet, Bag)* with algebraic map properties *(injective, surjective, bijective)*

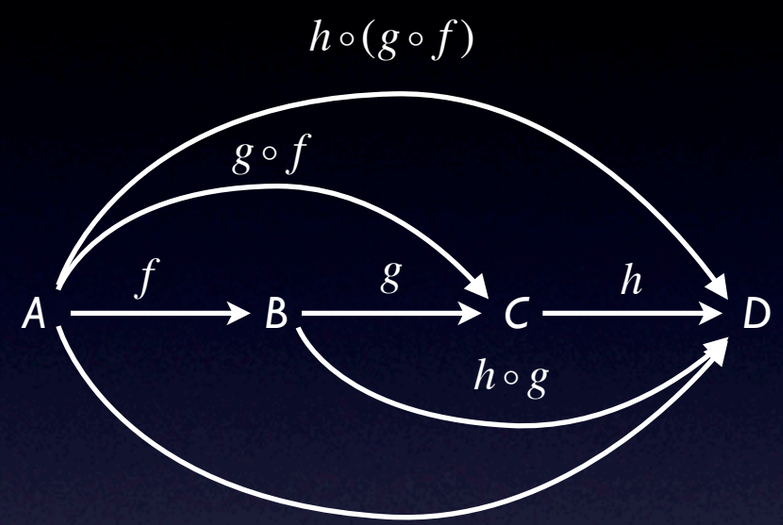
(In)Formal Preliminaries

- Arrows (maps) compose in the natural way



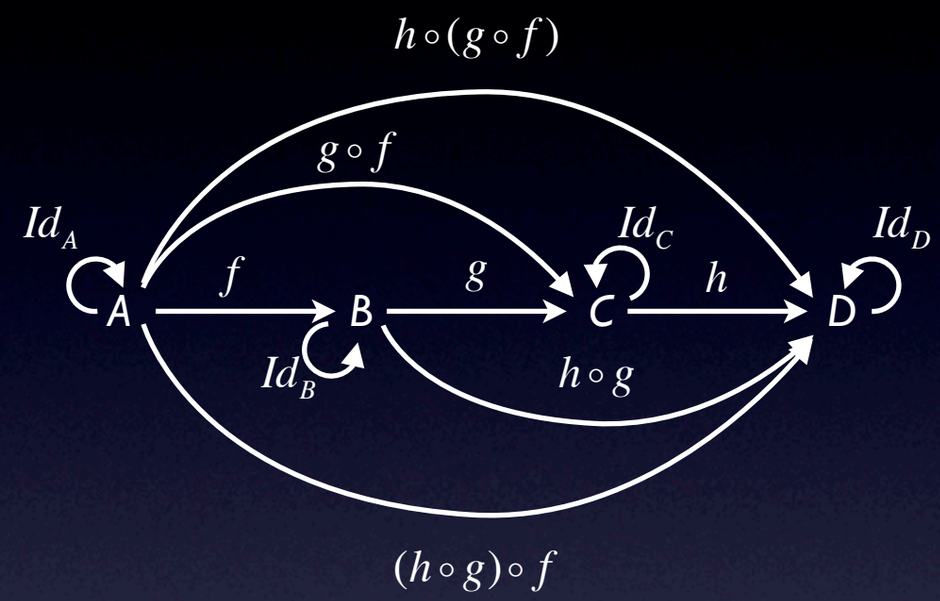
(In)Formal Preliminaries

- Arrows (maps) compose in the natural way
- Composition is associative



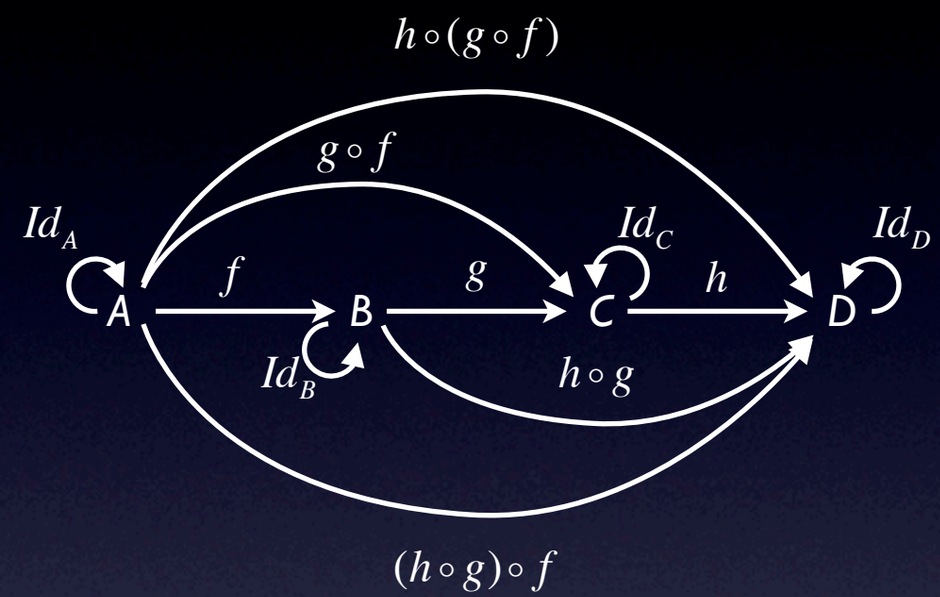
(In)Formal Preliminaries

- Arrows (maps) compose in the natural way
- Composition is associative
- An identity map exists for every object



(In)Formal Preliminaries

- Arrows (maps) compose in the natural way
- Composition is associative
- An identity map exists for every object
- Maps are strongly typed…



$g \circ f$ exists *iff* $cod(f) = dom(g)$

Objects and Structure

- Note that objects in a *Category* aren't totally structure-*less*. Instead, they have the structure required by their category.
- In the category of *(finite)(countable)Sets* and *(total)Functions*, object structure can be viewed as a collection of points.
- With that in mind…

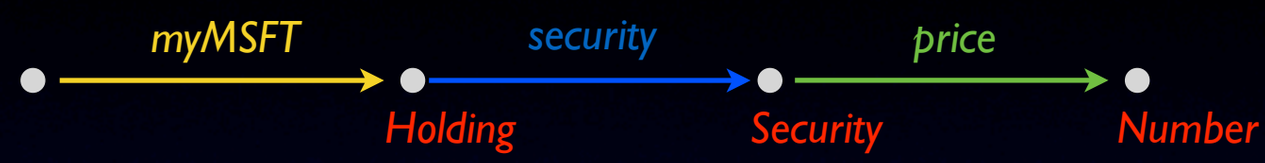
Maps From The Outside In

myMSFT security price

Maps From The Outside In

myMSFT security price

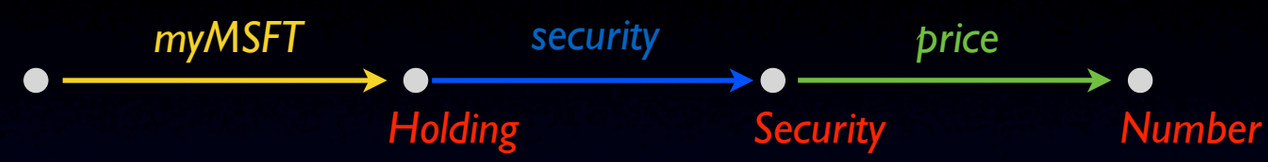
…external diagram



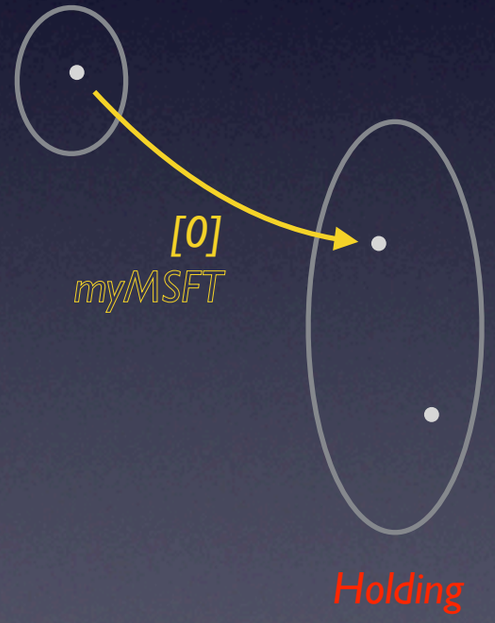
Maps From The Outside In

myMSFT security price

…external diagram



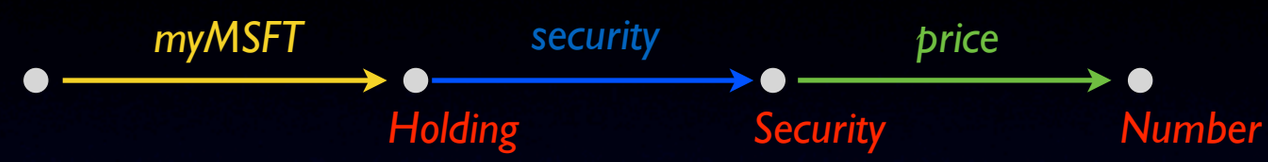
…internal diagram



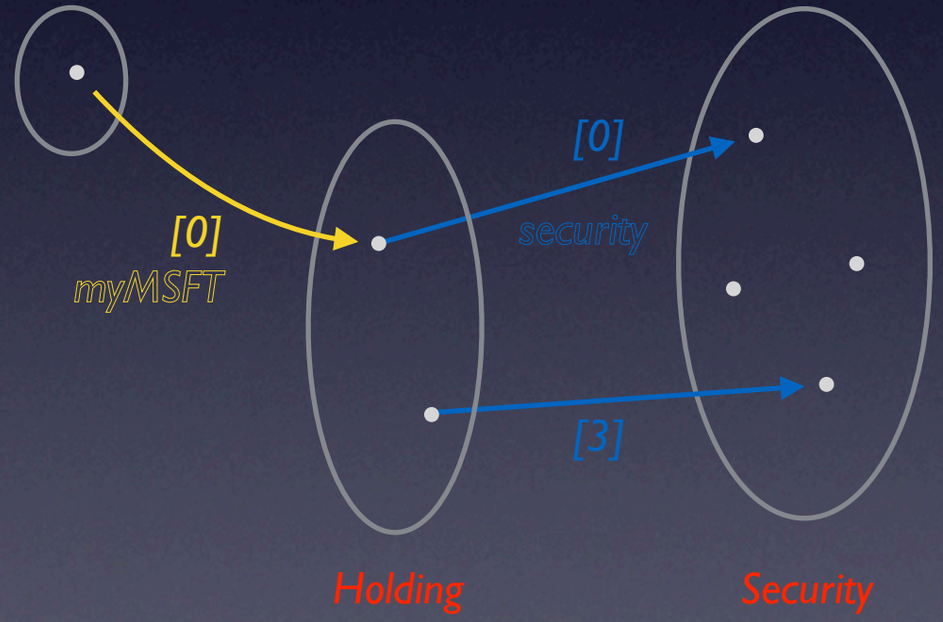
Maps From The Outside In

myMSFT security price

…external diagram



…internal diagram

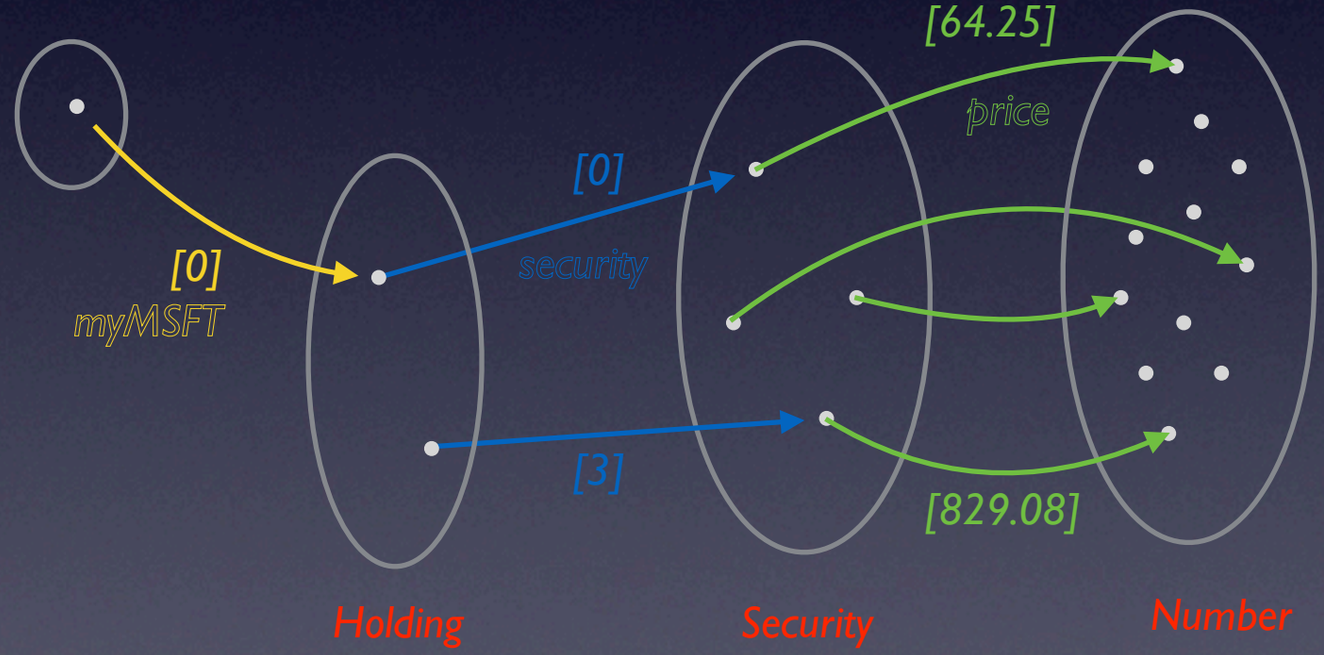


Maps From The Outside In

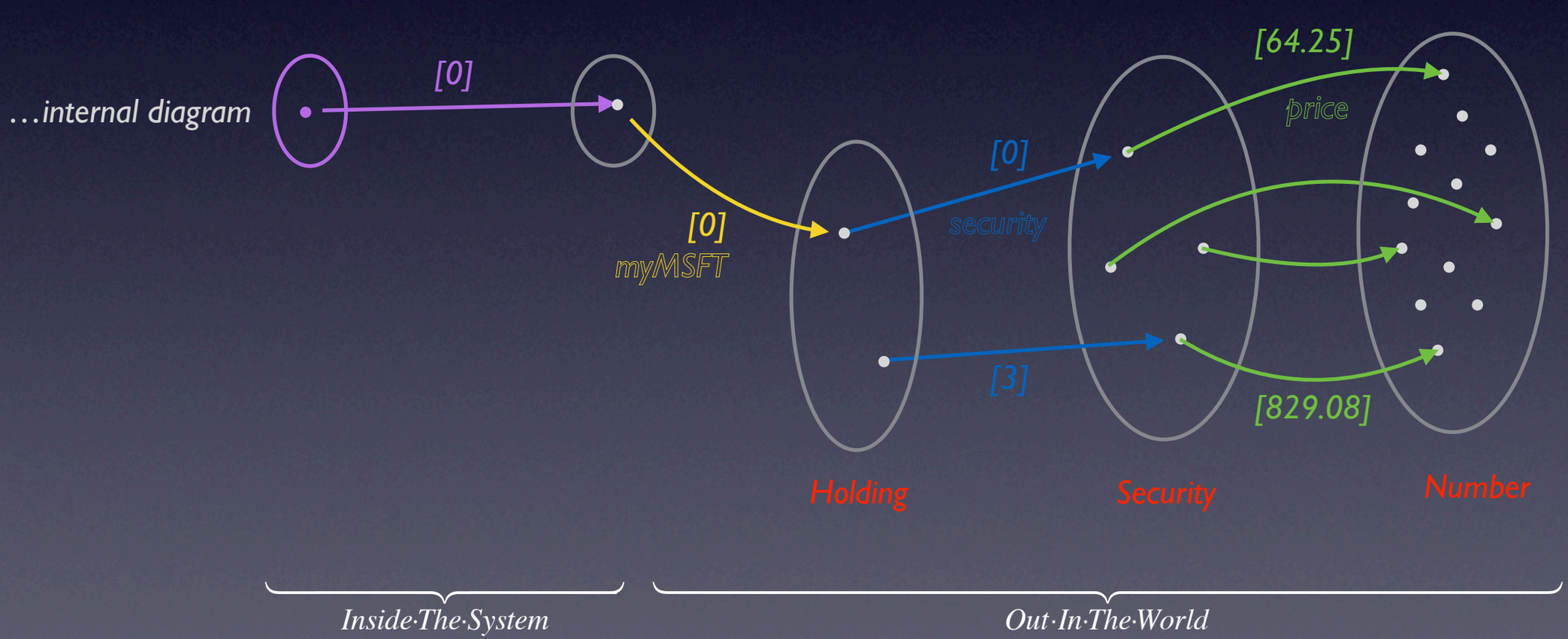
myMSFT security price

…external diagram

…internal diagram

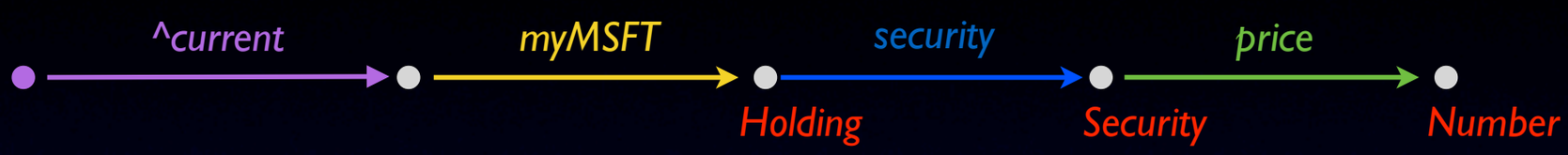


Maps From The Outside In

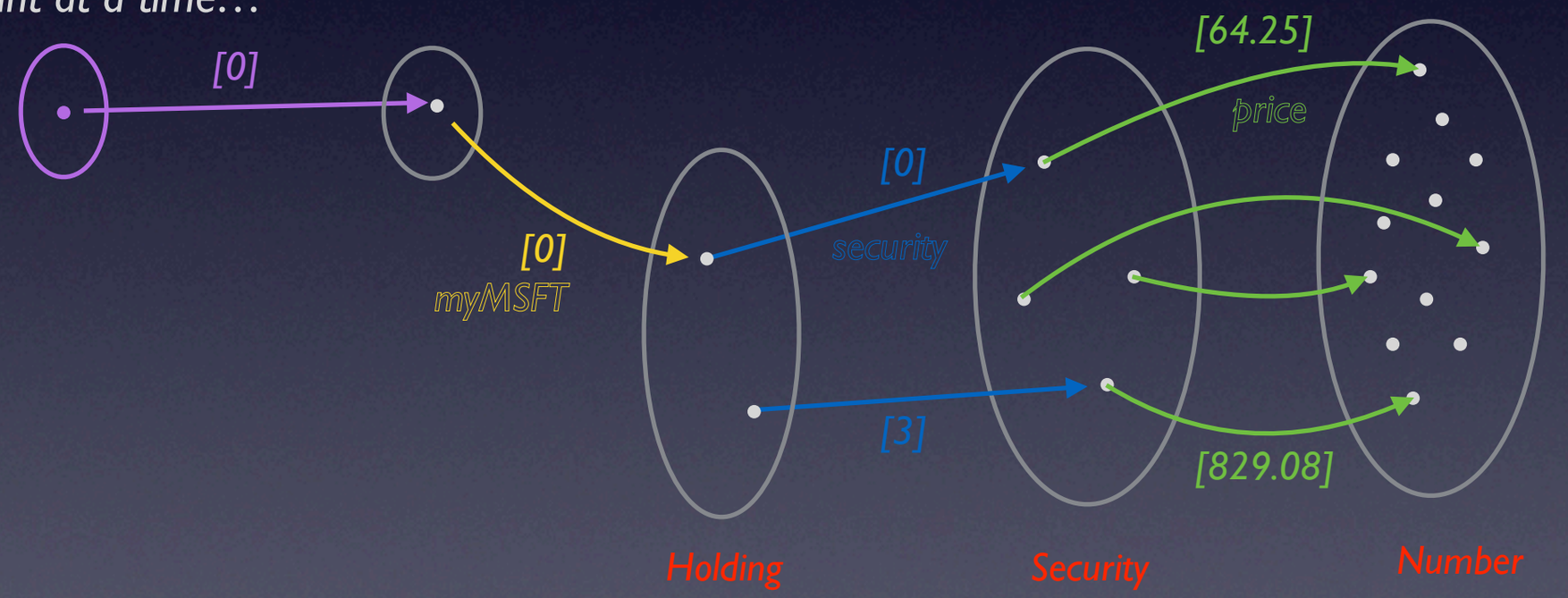


Images, Points, and Parallelism

myMSFT security price



We can exploring these maps using a single point at a time…

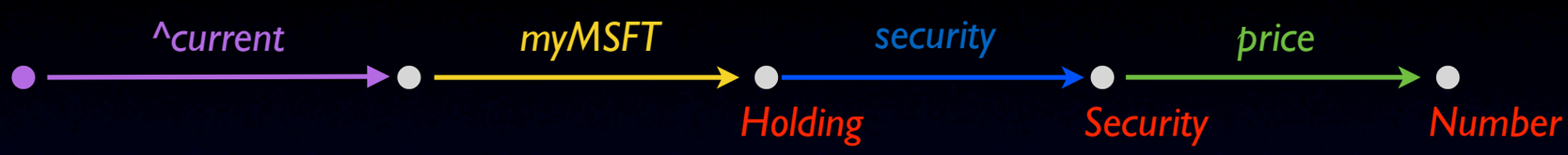


Inside·The·System

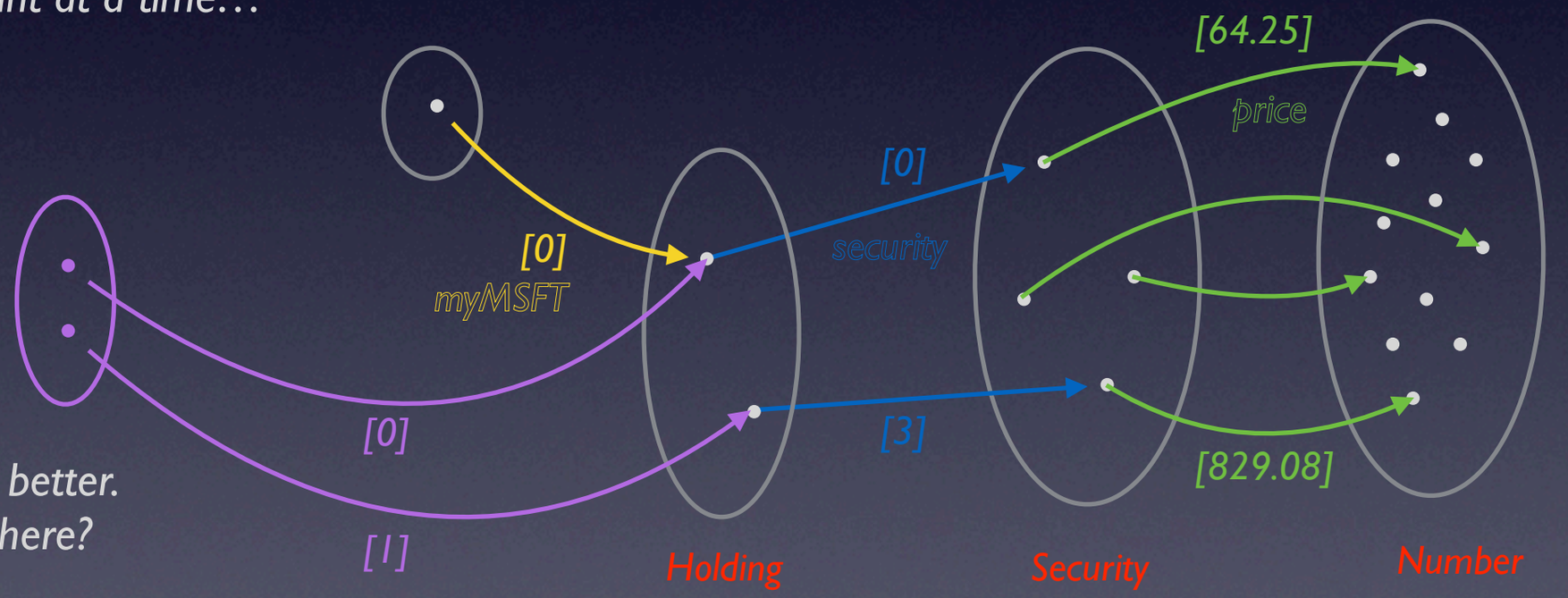
Out·In·The·World

Images, Points, and Parallelism

myMSFT security price



We can exploring these maps using a single point at a time…



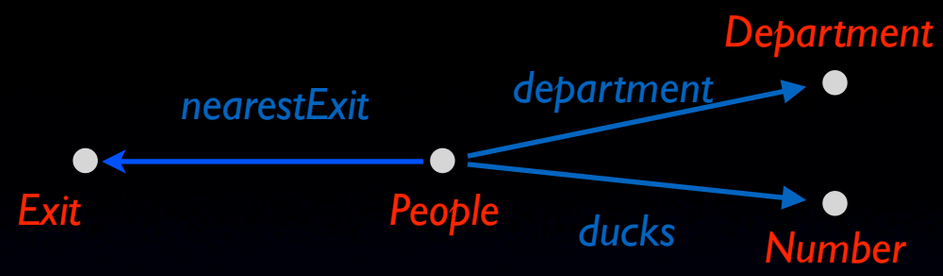
… or we can do better. How do we get there?

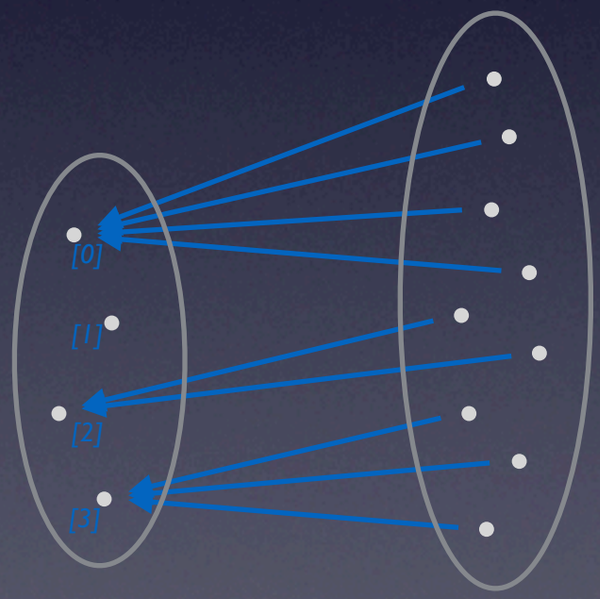
Inside·The·System

Out·In·The·World

The Structure Of Collections

A simple example about people, exits, and the nearest exit…

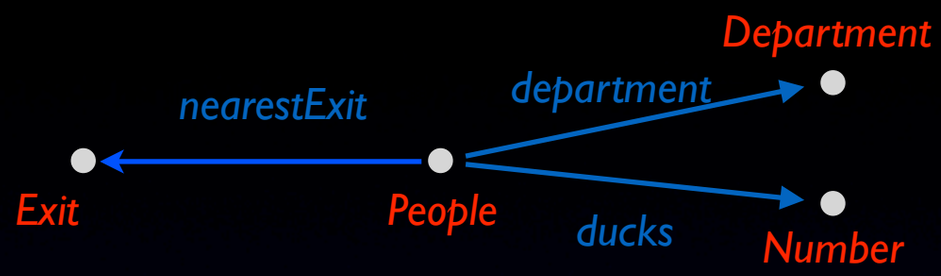


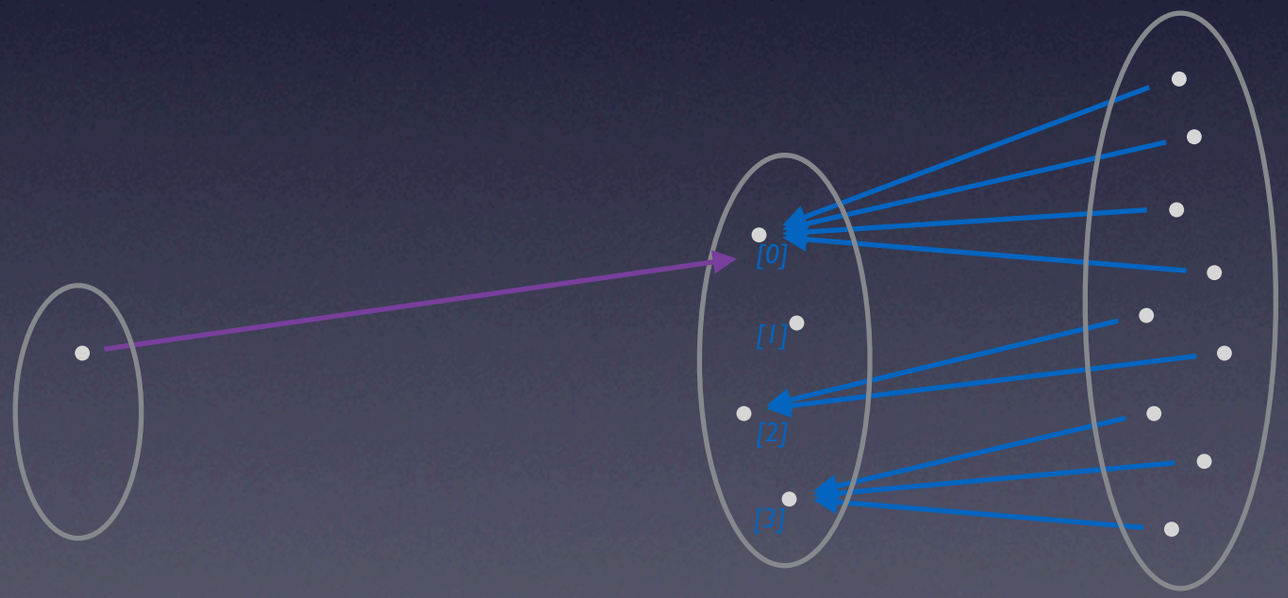


A map assigning people to the nearest exit

The Structure Of Collections

A simple example about people, exits, and the nearest exit…



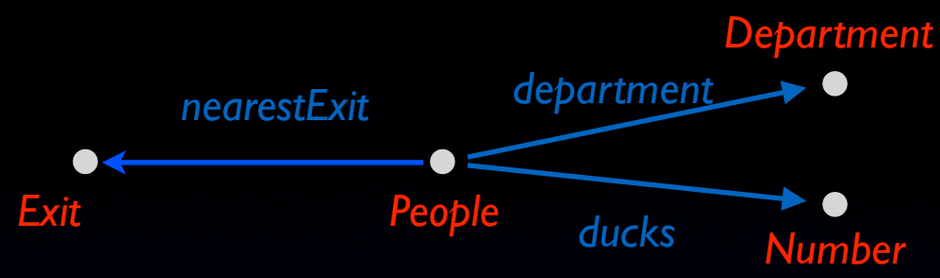


Q:Which people are nearest exit [0]?

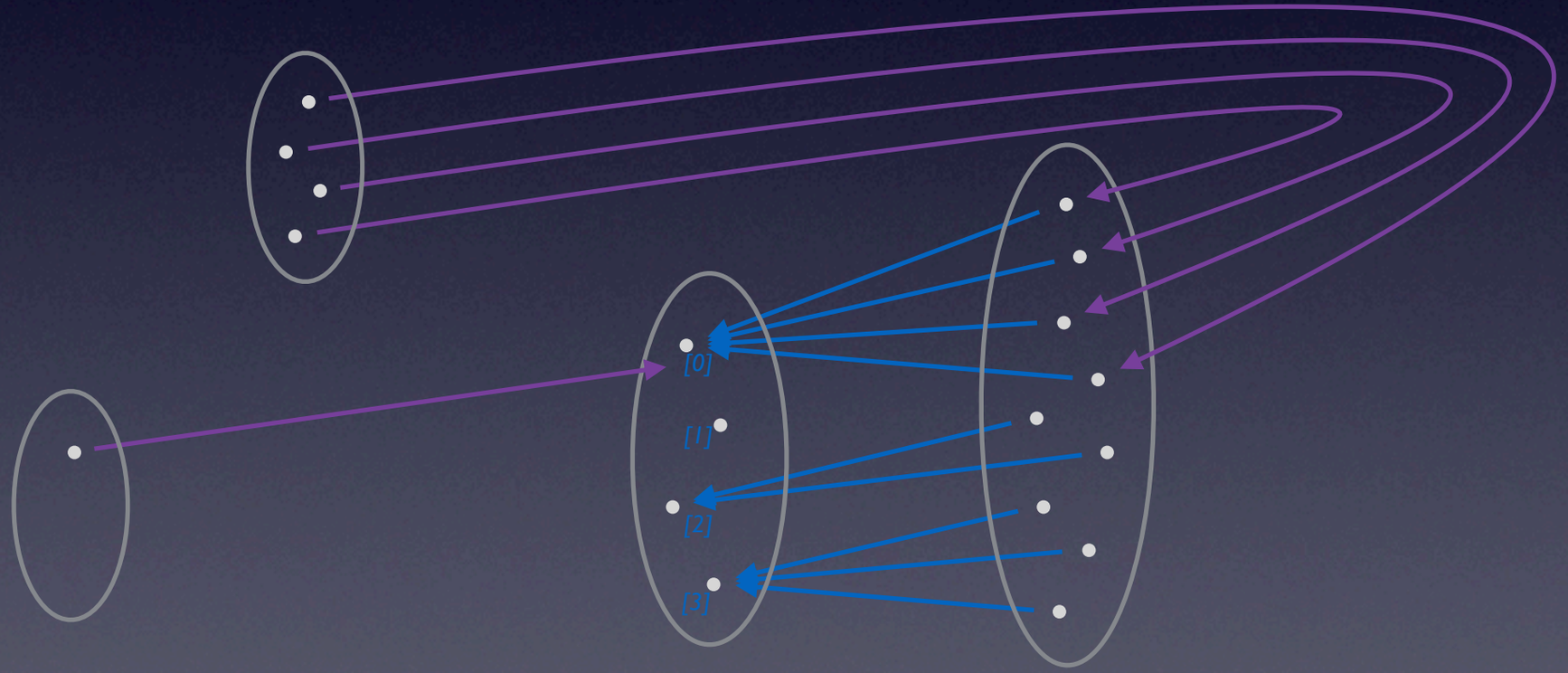
A map assigning people to the nearest exit

The Structure Of Collections

A simple example about people, exits, and the nearest exit…



A:The collection of people nearest exit [0]

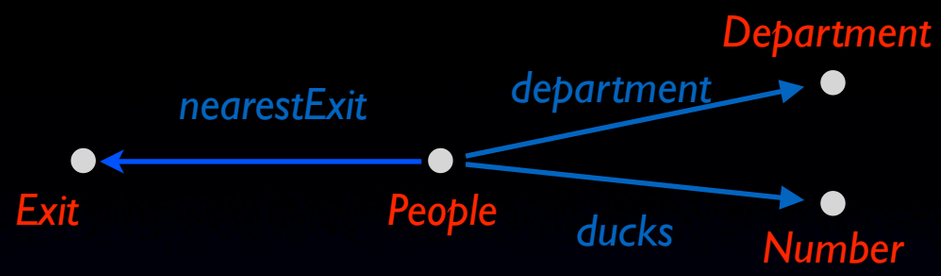


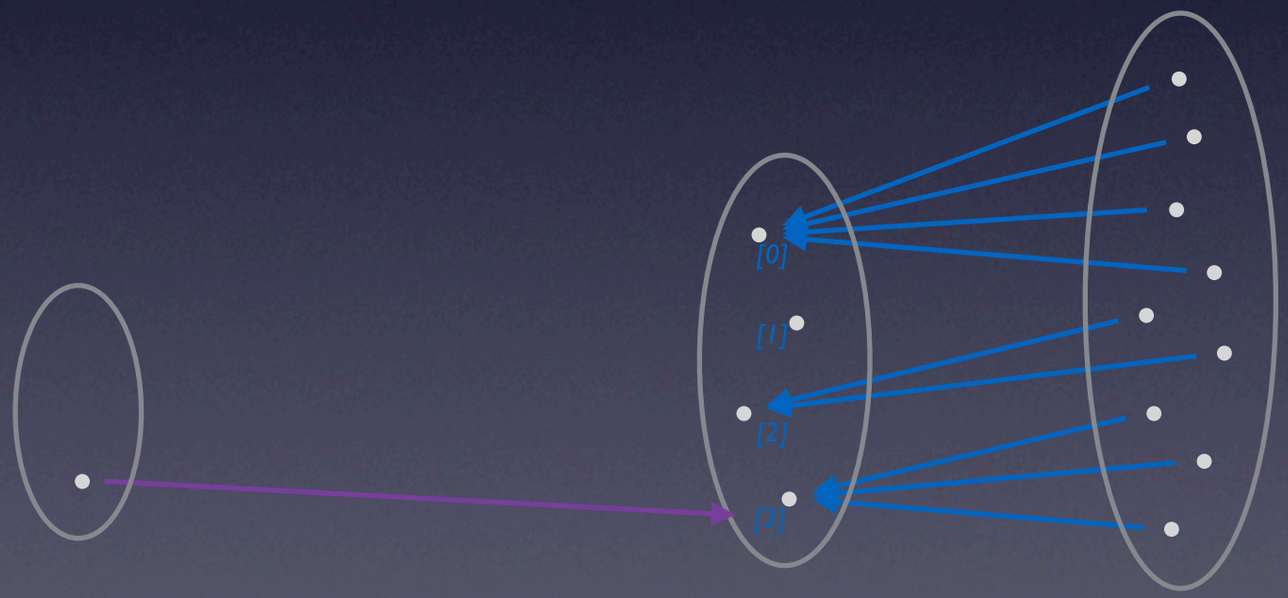
Q:Which people are nearest exit [0]?

A map assigning people to the nearest exit

The Structure Of Collections

A simple example about people, exits, and the nearest exit…



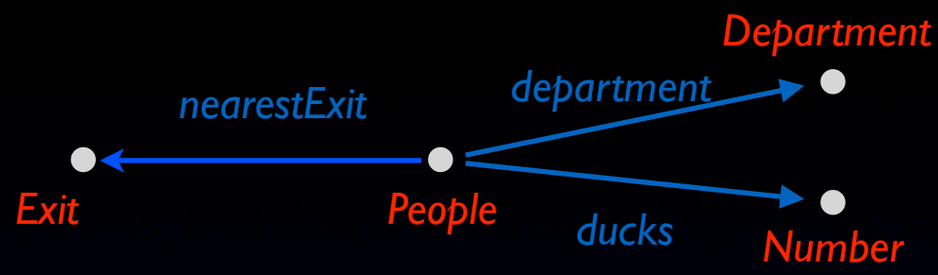


Q: How about exit [3]?

A map assigning people to the nearest exit

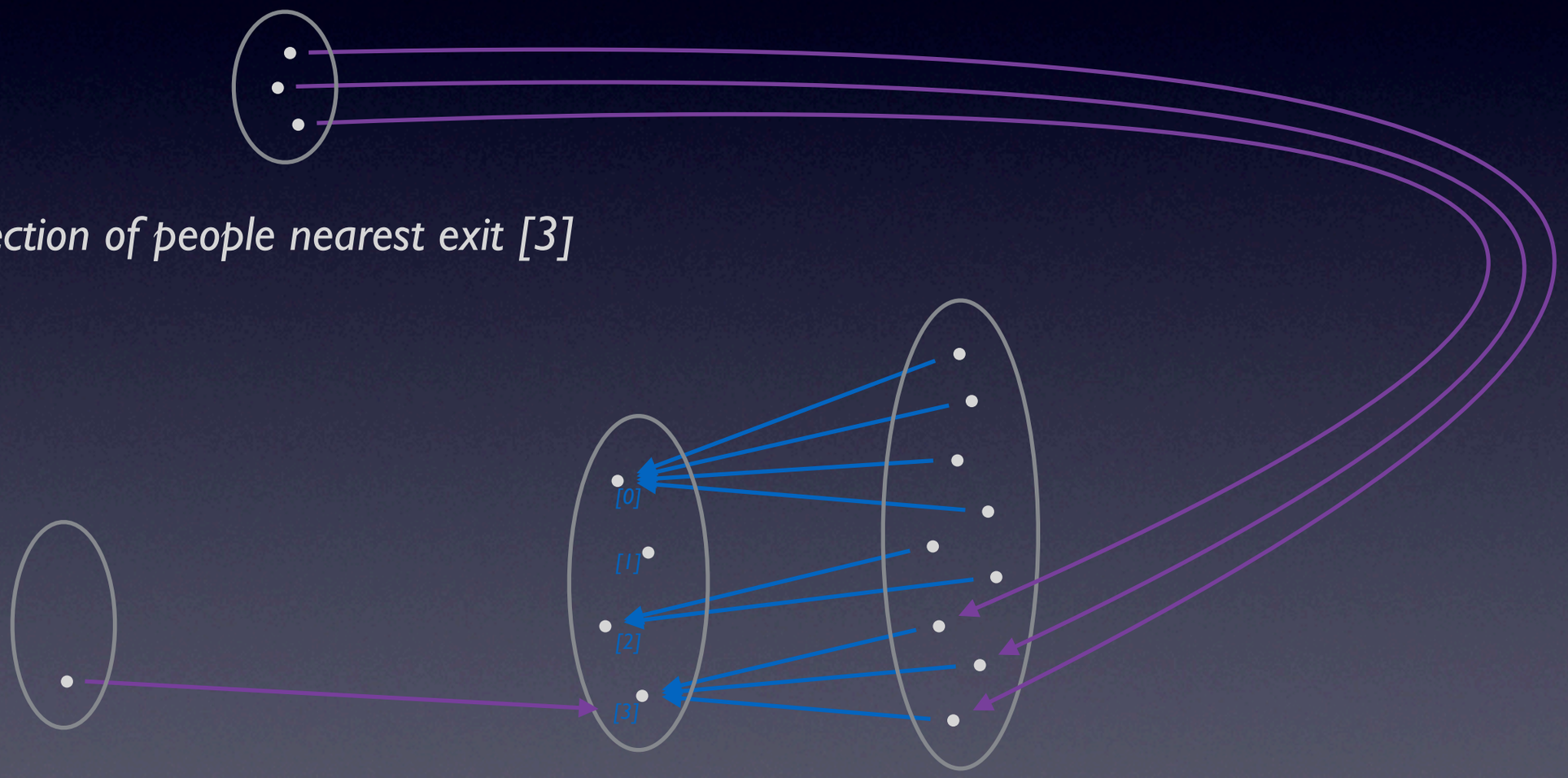
The Structure Of Collections

A simple example about people, exits, and the nearest exit…



A: The collection of people nearest exit [3]

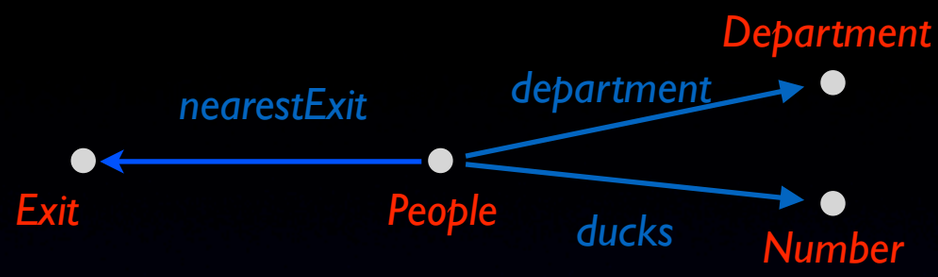
Q: How about exit [3]?

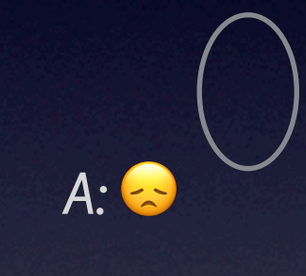


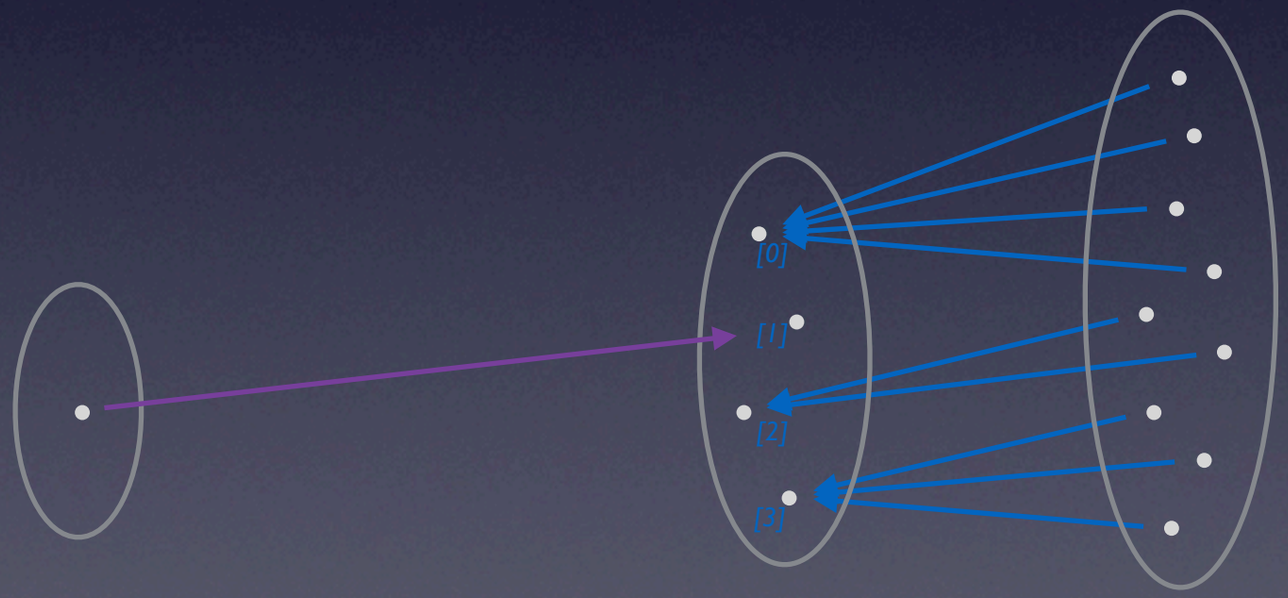
A map assigning people to the nearest exit

The Structure Of Collections

A simple example about people, exits, and the nearest exit…







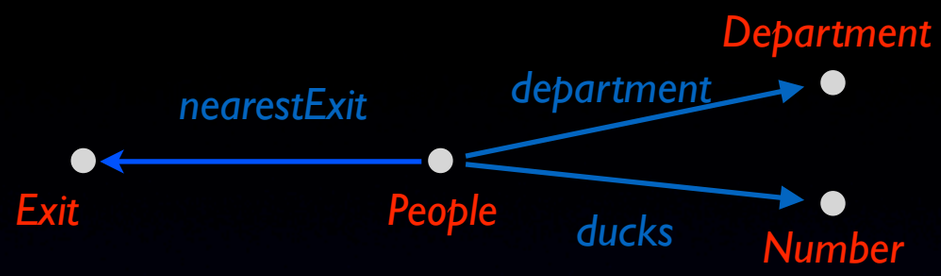
Q: And exit [1]?

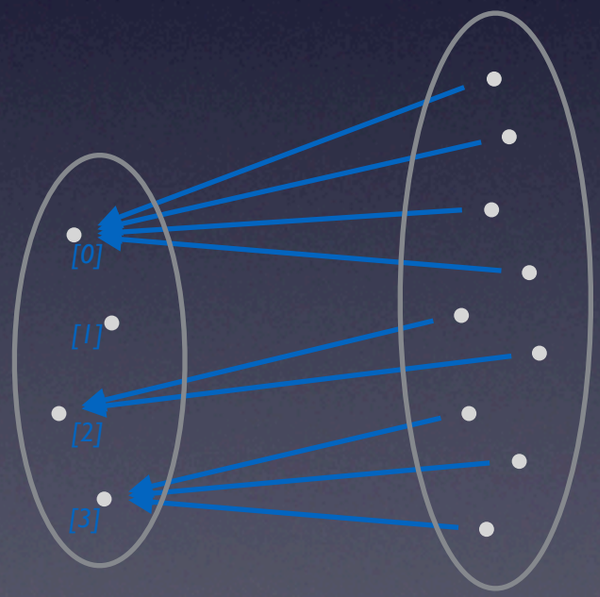
A map assigning people to the nearest exit

The Structure Of Collections

A simple example about people, exits, and the nearest exit…

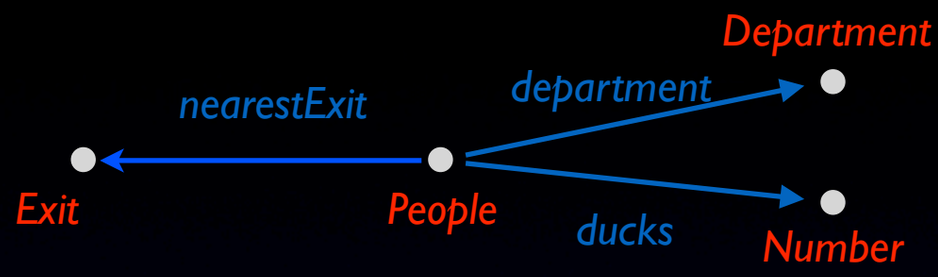
There's a pattern here…





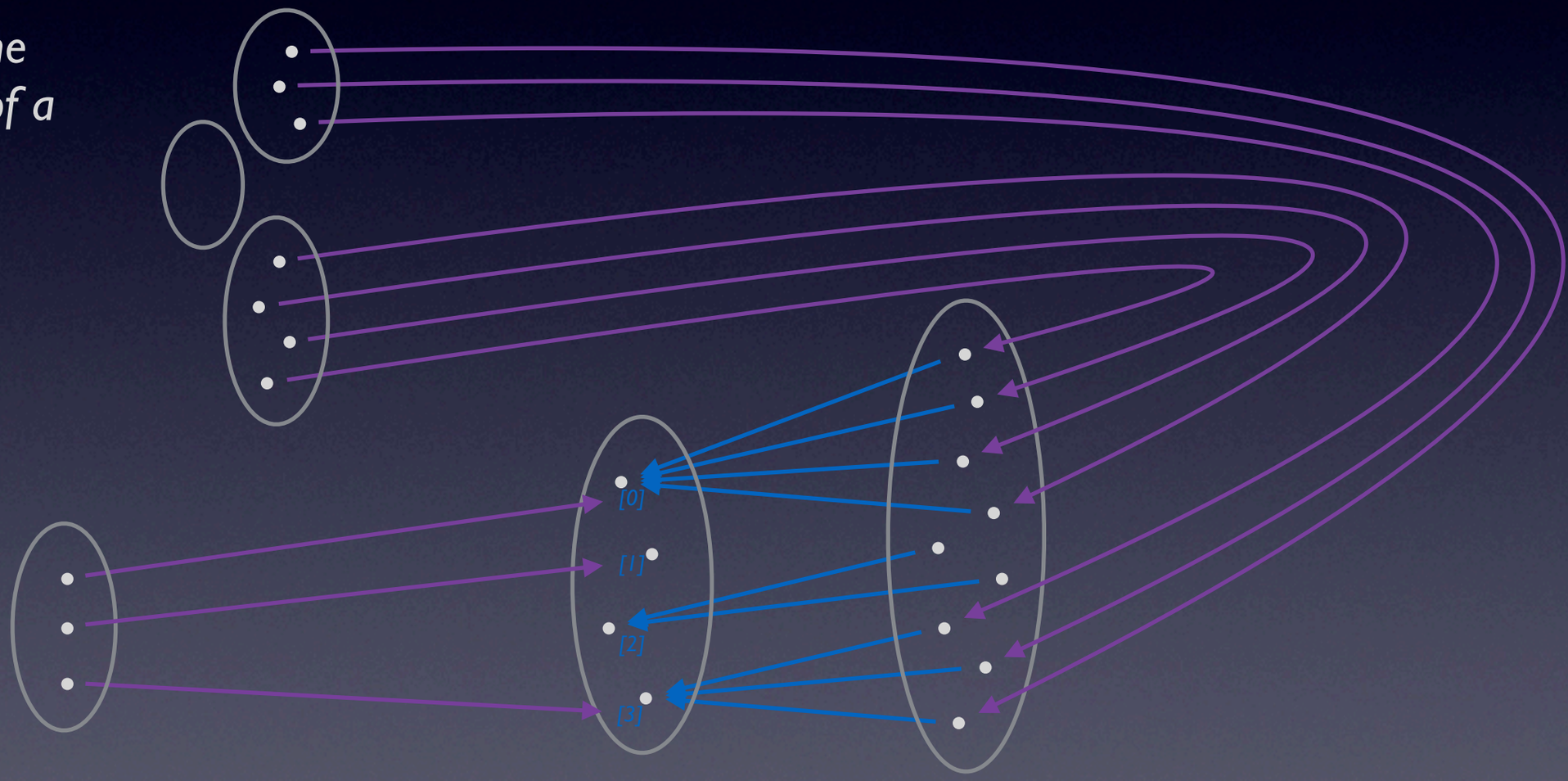
The Structure Of Collections

A simple example about people, exits, and the nearest exit…



There's a pattern here…

… generated by the internal structure of a single map on its domain.



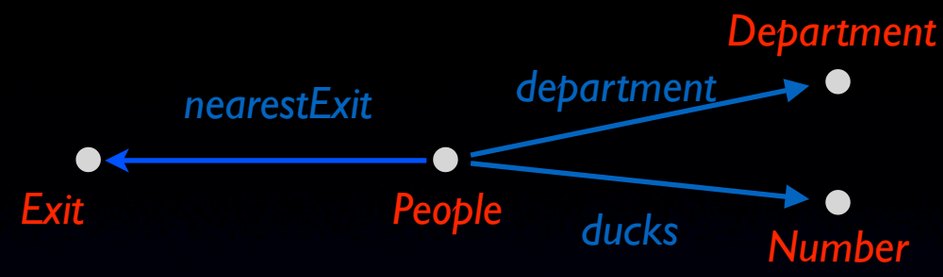
A map assigning people to the nearest exit

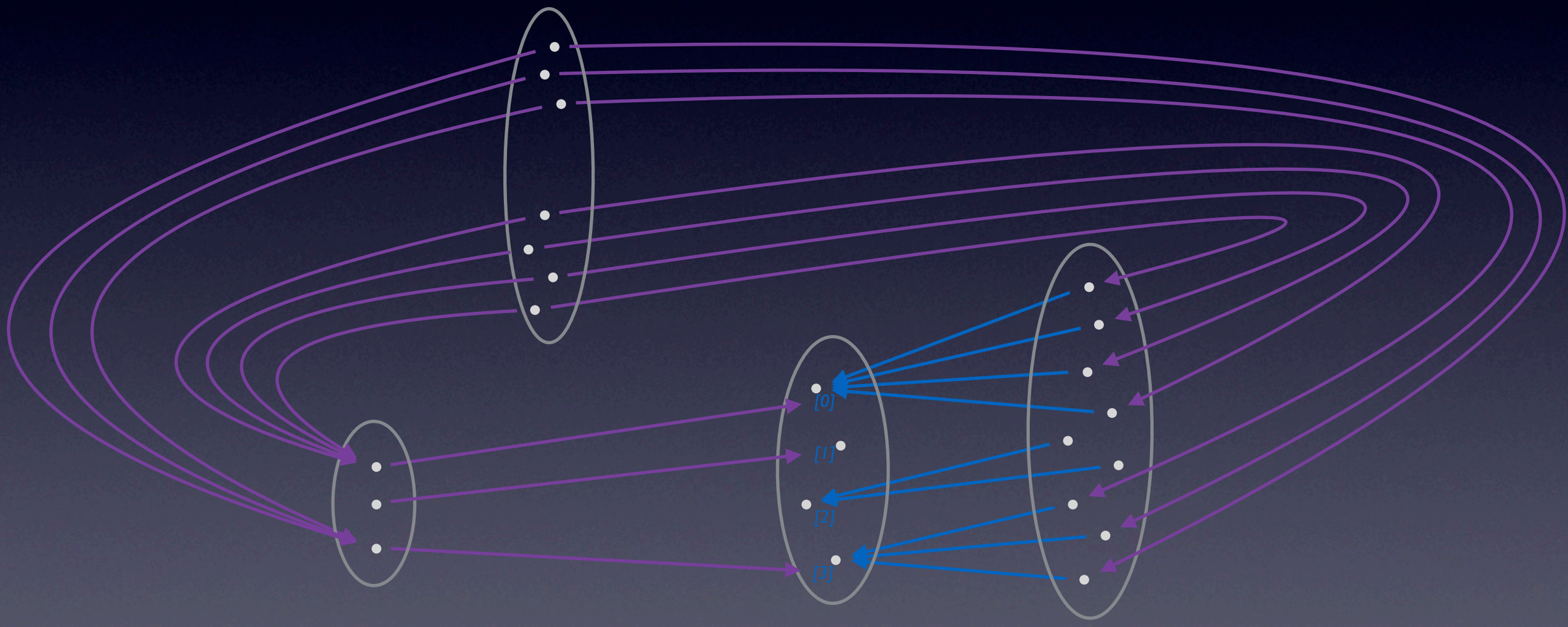
The Structure Of Collections

People, Exits, and The Nearest Exit

Inherently parallel…

… pullback, fibered product, join



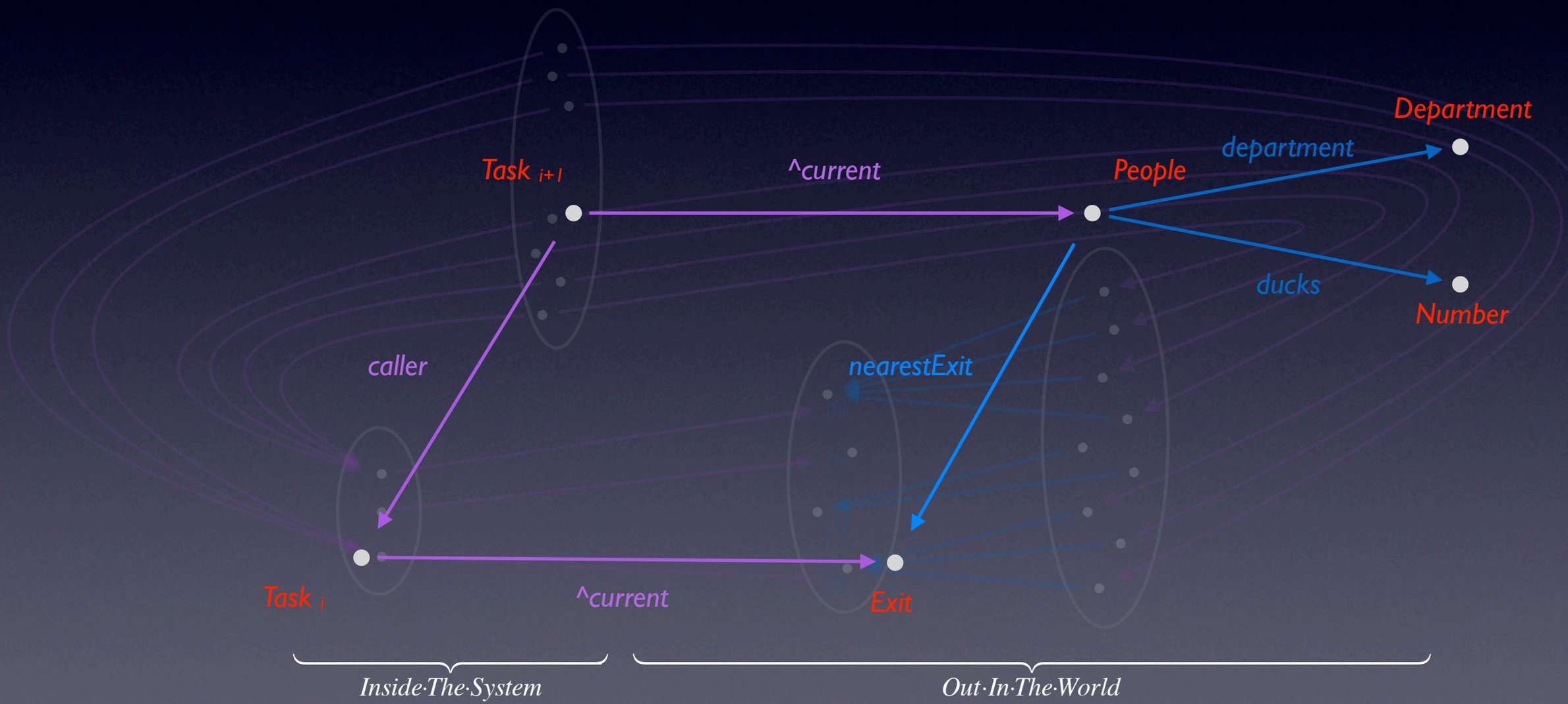


Inside·The·System

Out·In·The·World

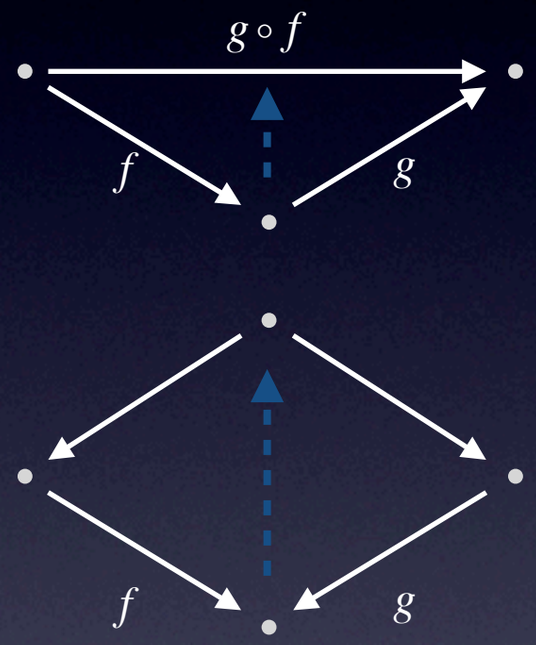
The Structure Of Enumeration

Exits, People, and The Nearest Exit



Computational Patterns
(so far)

Composition



Pullback

Up Next…

Disjoint Union

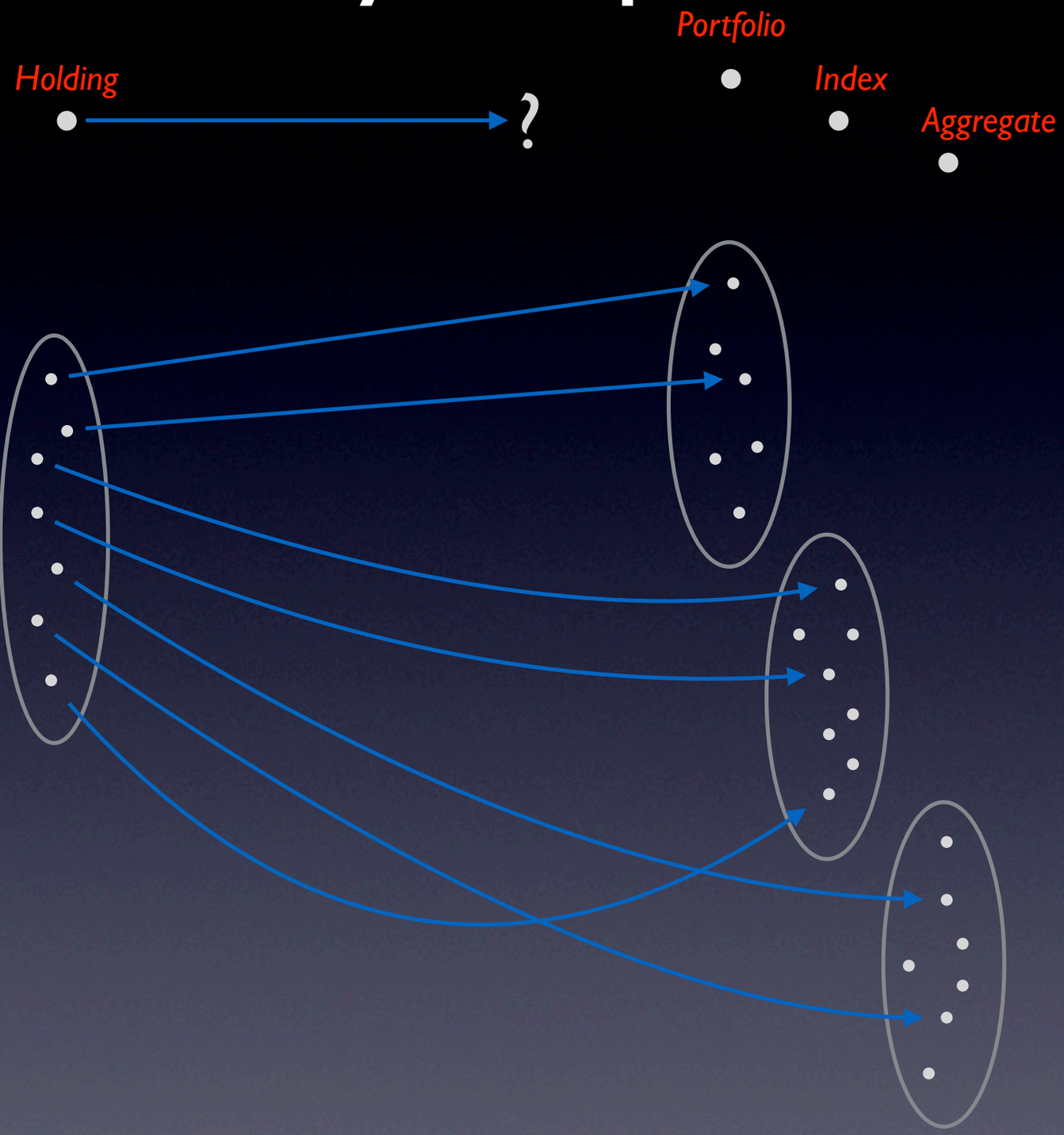
Disjoint Union & Polymorphism

Map don't always have simple co-domains…

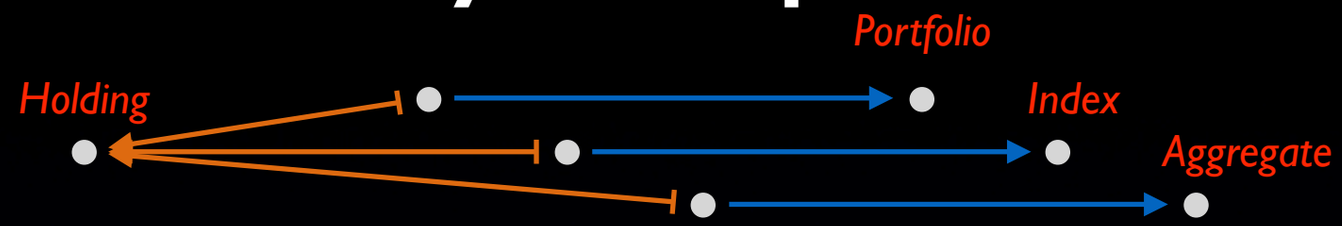
- *A map that names both Integers and Real numbers.*
- *A map that names objects with similar Interfaces (Account) but very different Implementations (Portfolio, IndexedAccount, AggregateAccount).*
- *Occurrences of this sort of situation can be moved but they cannot be avoided.*

Disjoint Union & Polymorphism

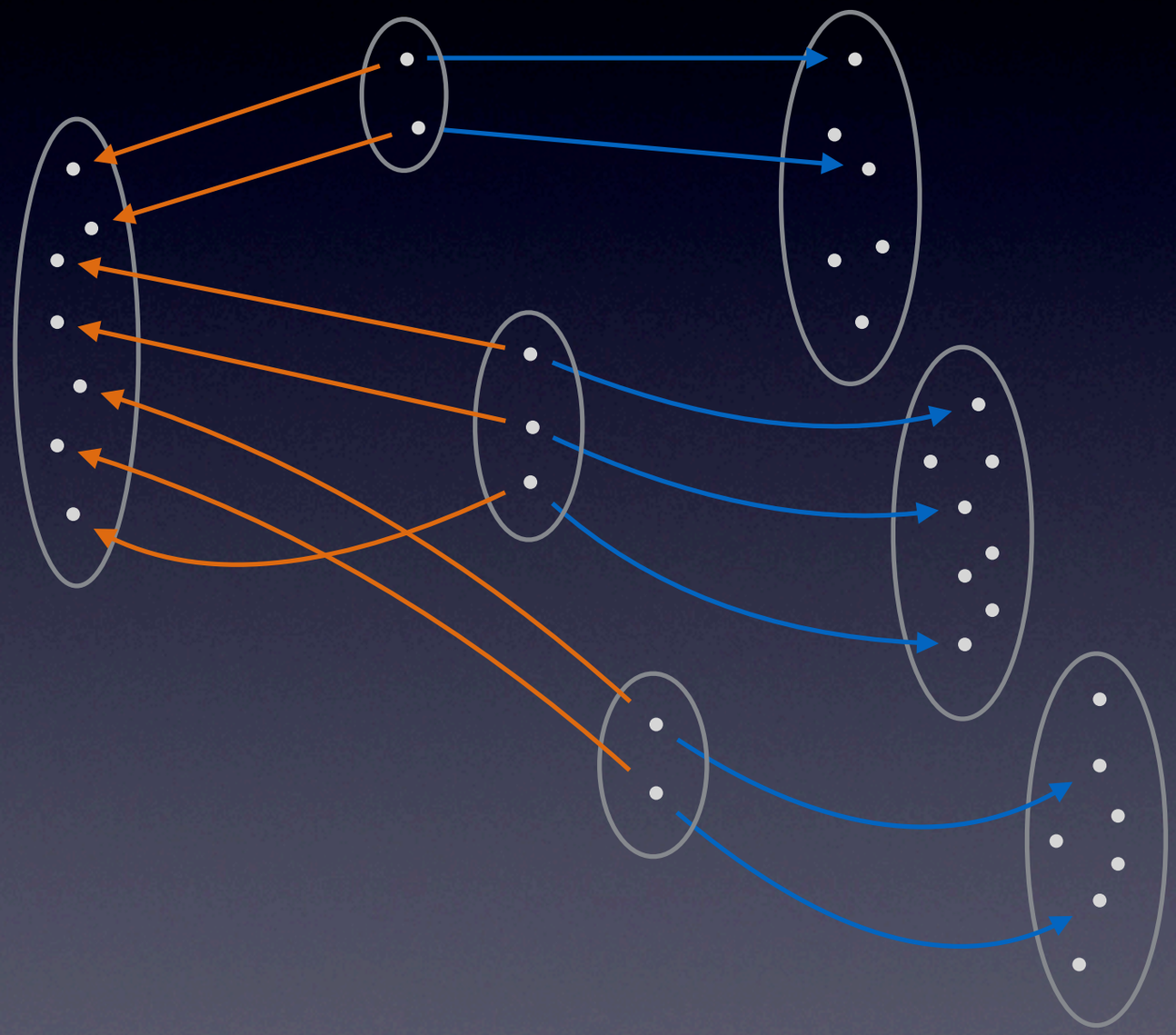
How do we build maps that look like this?



Disjoint Union & Polymorphism



How do we build maps that look like this?



Disjoint Union

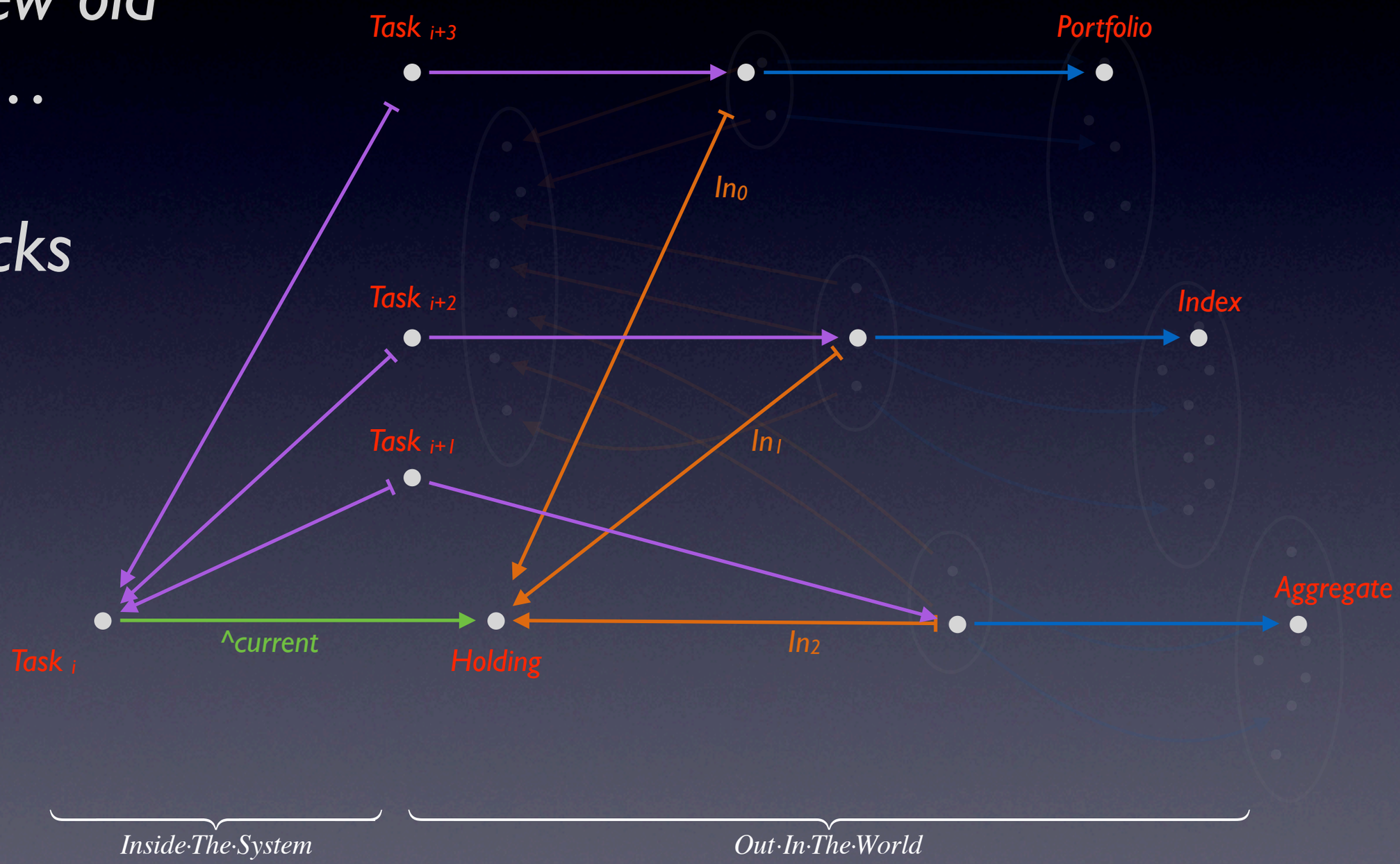
Algebraic addition defined in terms of inclusion maps

Injective in Set

Disjoint Union and Composition

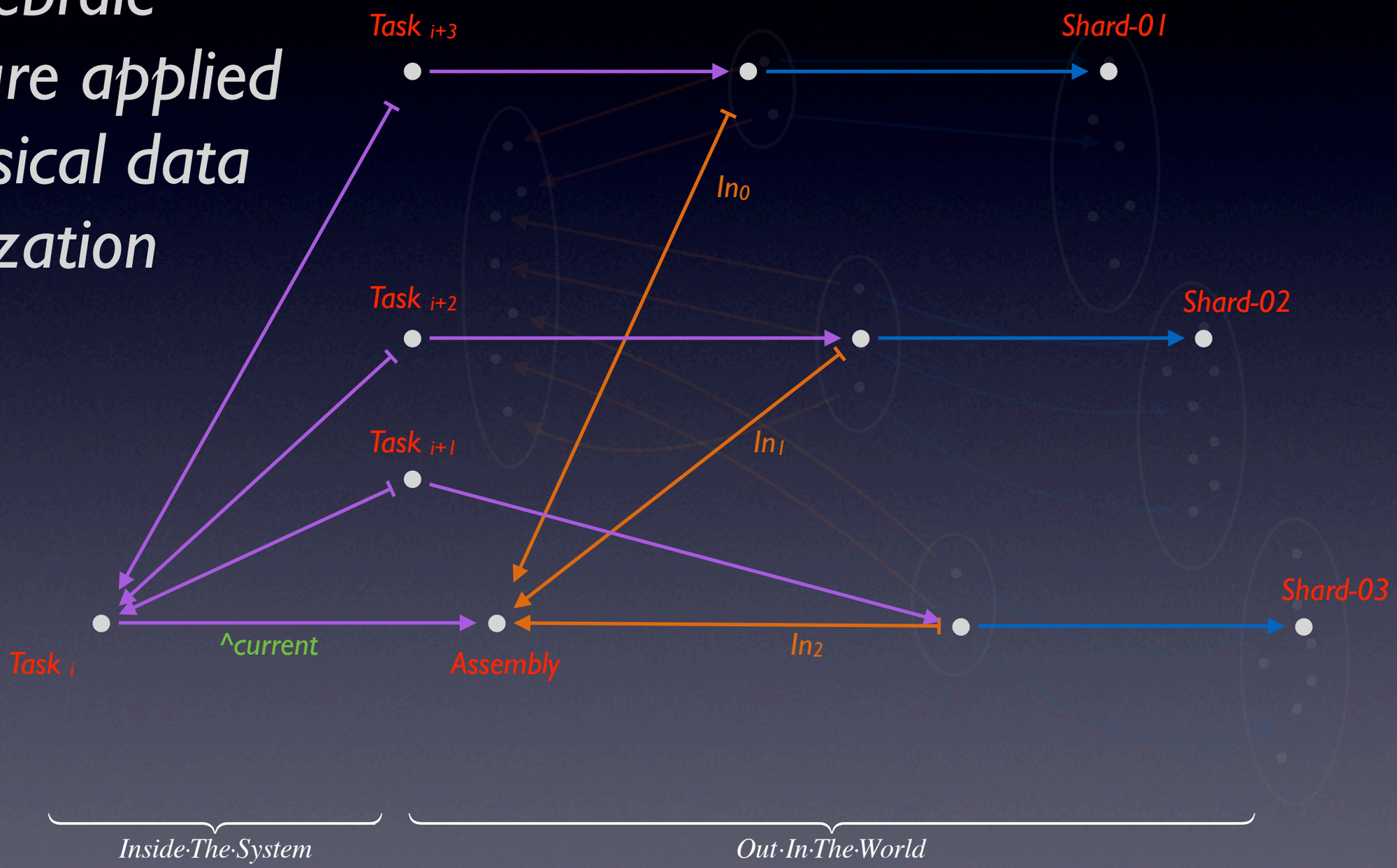
Our new old friend…

Pullbacks



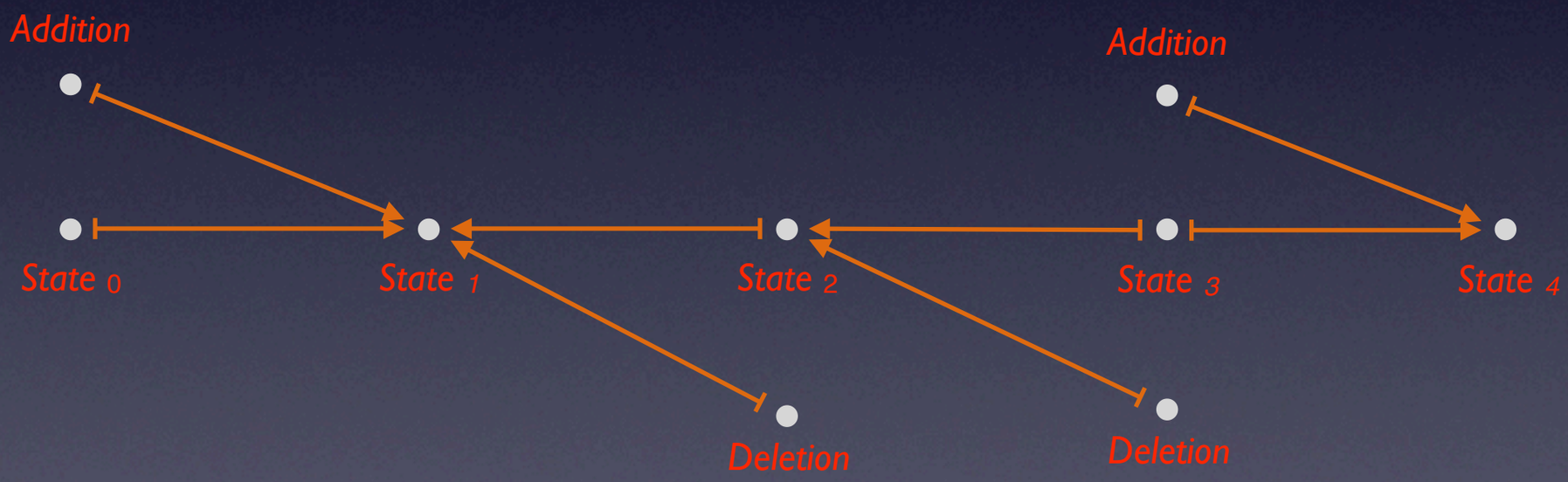
Disjoint Union and Partitioning

An algebraic structure applied to physical data organization



Disjoint Union and Update

Here's an algebraic pattern encoding a sequence of 4 updates consisting of an addition, two deletions, and an addition.



Implementing Maps

- They're not as hard as they might look
- Arrays offer one simple natural way
- Encoding and compression
- Ordering
- Factoring and rewriting

Re-collections

- Relationships matter more than things.
- They induce collections, guide explorations and direct transformations.
- They provide the theory that makes all this practical.

```
Account defineMethod: [ | getHoldingsOverlap |
!lowerPct <- pctEq * 0.8; !upperPct <- ... ;
holdings send: [security].
  select: [type isEquity].
  collectListElementsFrom: [holdings].
  groupedBy: [account].
  select: [pctEq >= ^my lowerPct
    && pctEq <= ^my upperPct].
  extendBy: [
    !xref <- ^my holdings;
    !ofactor <- groupList total: [
      percentOfPort min: (^my xref at: security.
        percentOfPort)
    ]
  ].
  sortDown: [ofactor]
];
```

```
^today - 1 monthEnds to: ^today -12 monthEnds by: 1 monthEnds.
  evaluate: [ MyFund getHoldingsOverlap … ];
```

Re-collections

- Relationships matter more than things.
- They induce collections, guide explorations and direct transformations.
- They provide the theory that makes all this practical.

```
Account defineMethod: [ | getHoldingsOverlap |
!lowerPct <- pctEq * 0.8; !upperPct <- ... ;
holdings send: [security].
  select: [type isEquity].
  collectListElementsFrom: [holdings].
  groupedBy: [account].
  select: [pctEq >= ^my lowerPct
    && pctEq <= ^my upperPct].
  extendBy: [
    !xref <- ^my holdings;
    !ofactor <- groupList total: [
      percentOfPort min: (^my xref at: security.
        percentOfPort)
    ]
  ].
  sortDown: [ofactor]
];
```

```
FundUniverse do: [
    ^self getHoldingsOverlap do: […]
]
```

```
2 yearsAgo evaluate: [MyFund getHoldingsOverlap ... ]
```

```
^today - 1 monthEnds to: ^today -12 monthEnds by: 1 monthEnds.
    evaluate: [ MyFund getHoldingsOverlap … ];
```

Left Unsaid

- Sparse associative product spaces (matrices, time-series, multi-dimensional indices)
- Context, indeterminacy, and provenance
- Much more about updates
- Optimization techniques
- *A Whole Lot More*

Questions?