K3: Language Design for Building Multi-Platform Domain-Specific Runtimes

P.C. Shyamshankar
with Zachary Palmer and Yanif Ahmad

Department of Computer Science,
The Johns Hopkins University

First International Workshop on Cross-Model Language Design and Implementation

What are Domain-Specific Runtimes?

Runtimes: Systems that underlie an application's execution.

- Data Management
- Execution Management
- Integrity Management

Domain Specific Runtimes:

- Hadoop
- Pregel
- LINQ

A Language for Building Domain-Specific Runtimes

Translate high-level domain-specific information into low-level implementation decisions.

- Describe application logic flexibly.
- Represent domain-specific information at a high level.
- Recognize existing runtime patterns.
- Revisit implementation decisions over time.

Applications

- DBToaster (SQL) <http://www.dbtoaster.org/>
- Dyna (Weighted Logic Programming) <http://www.dyna.org/>
- BLOG (Probabilistic Graphical Models) <http://bayesianlogic.cs.berkeley.edu/>

Building Domain Specific Runtimes

Language Design

Annotations: Exploiting Domain Specific Information

Closing

Simple Control Flow

Triggers carry out small step computation. They:

- Perform side-effecting functional style computation.
- Only contain acyclic control flow.
- Can send messages to other triggers.

```
trigger fibonacci(n:int, a:int, b:int) {} =
    if n == 1 then send(sink, a)
    else send(fibonacci, n - 1, b, a + b)
```

Complex Control Flow

Large step computation is done using *message* passing.

- Triggers are invoked on receiving a message.
- Message passing is asynchronous.
- Message processing is governed by a scheduler.
- Flexible enough to capture most execution patterns.

Collection Management

The K3 collection model is based on structural recursion.

- Basic collection transformers provide bounded iteration.
- More complex transformations are provided through annotations, and are subject to depth-based analyses.
- Collection access operators provide the ability to mutate all or parts of the collection.

Mutable State

K3 maintains a deep value-based semantics of mutability by default.

- Particular implementations can choose which approaches to use (copy-on-write, etc.), to provide this mutability.
- Pointer-based semantics are available on demand, for annotation writers, etc.
- Mutability of collections is determined at multiple granularities:
  - The entire collection,
  - Parts of the collection (restructurability),
  - Individual elements,
- Mutation operations ensure that the relevant integrity constraints are satisfied.

Building Domain Specific Runtimes

Language Design

Annotations: Exploiting Domain Specific Information

Closing

Exploiting Domain Specific Information

K3 uses a system of *annotations* to encode, and make use of domain specific information. Annotations can:

- Be attached to any part of a K3 program.
- Be acted upon by any part of the toolchain.

Categorization of Annotations

- Data structure annotations specify properties about a collection, and facilitate *declarative data structures*.
  - `Sorted`, `Layout*`, ...
- Control annotations specify properties of a piece of code, and facilitate adaptive execution.
  - `Logging`, `Profiling`, ...
- *Hint Annotations* describe possible optimizations.
  - `Layout*`, `Locking`, ...
- *Constraint Annotations* describe correctness properties of the program, and require code to be generated to check them.
  - `FunDep`, `Unique`, ...

Data		Control and Execution	
Integrity (Constraint)	**Efficiency (Hint)**	**Assurances (Constraint)**	**Scalability (Hint)**
Functional dependencies	Layout, and compression	Fault tolerance, checkpointing	Degrees of parallelism
Sortedness	Indexes, views	Service-level agreements	Vectorization
Orderedness	Allocation, GC		Scheduling
Referential integrity	Data placement and replication	Auditing and compliance	Autotuning heuristics
Concurrency	Lock granularity	Access control	Profiling

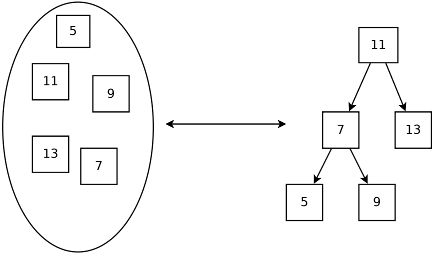
Components of a Data Structure Annotation

A user-defined data structure annotation should contain specifications of:

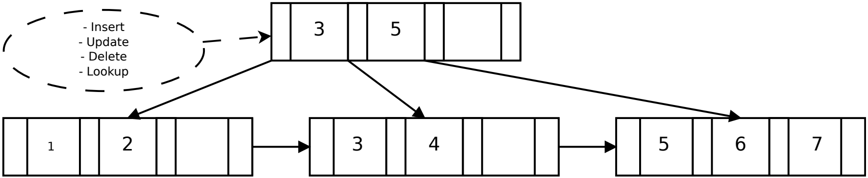
- Requirements from other annotations on the collection.
- Per-collection data structures.
- Schema extensions.
- Method definitions.
- Method hooks (`method.pre`, `method.post`, ...).

A Simple Data Structure Annotation: `Index`

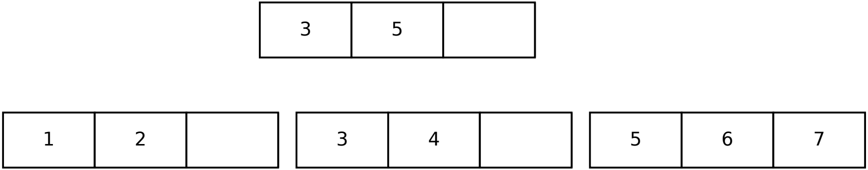
- Other required annotations: None
- Per-collection data: An auxiliary lookup data structure.
- Schema extensions: None
- Method definitions: `lookup`
- Method hooks: Post hooks for the maintenance of the auxiliary data structure.



Composing Annotations: B+Trees

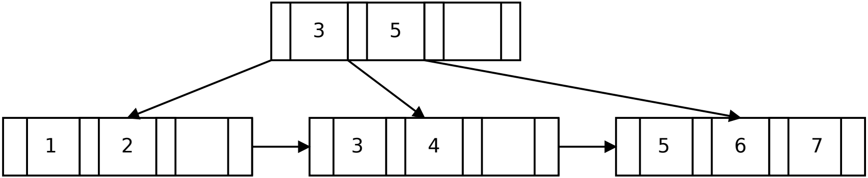


A Collection Of Blocks



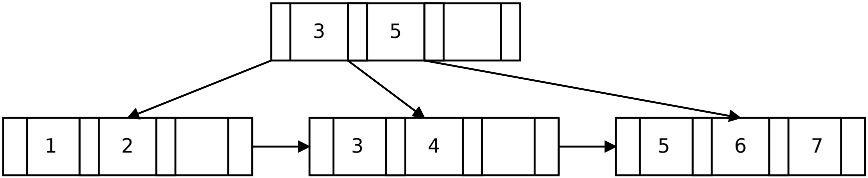

```
declare b : Collection(Collection(t))
```

Adding Tree Linkage



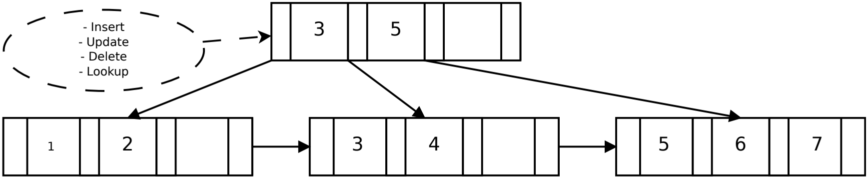

```
declare b : Collection(Collection(t)) @ { Tree }
```

Managing Overflow and Underflow




```
declare b :
    Collection(
        Collection(t) @ {
            Capacity(k), Fill(k),
            OverflowHandler, UnderflowHandler
        }
    ) @ { Tree(Capacity(k)) }
```

Providing a B+Tree Interface




```
declare b :
    Collection(
        Collection(t) @ {
            Capacity(k), Fill(k),
            OverflowHandler, UnderflowHandler
        }
    ) @ { Tree(Capacity(k)), BPTree }
```

Extending the B+Tree

We can extend the existing B+Tree with other behaviors, such as:

- Cache consciousness, with an annotation describing fractal layouts of collections.
- Concurrency, through annotations providing logging or locking.

```
declare b :
  Collection(
    Collection(t) @ {
      Capacity(k), Fill(k),
      OverflowHandler,
      UnderflowHandler
    }
  ) @ {
    Tree(Capacity(k)), BPTree
    FractalLayout, Logged
  }
```

Building Domain Specific Runtimes

Language Design

Annotations: Exploiting Domain Specific Information

Closing

Implementation Status

K3 currently has:

- A functional core, with value-based mutation.
- A simple distributed execution model.
- An initial model of data structure and control annotations.

Next Steps

- Language Features:
  - Effect System - Guiding parallelization decisions.
  - Depth analysis of annotation methods - User-defined collection transformations.
- Scalability and Performance:
  - Optimizer Model.
  - Eventually-consistent distributed data structures.

The End

- <http://damsl.cs.jhu.edu/>
- <http://cs.jhu.edu/~shyam/>