project:Einstein

Project Sponsors

Paremus

Mangala Solutions
www.mangala.co.uk

The Vision

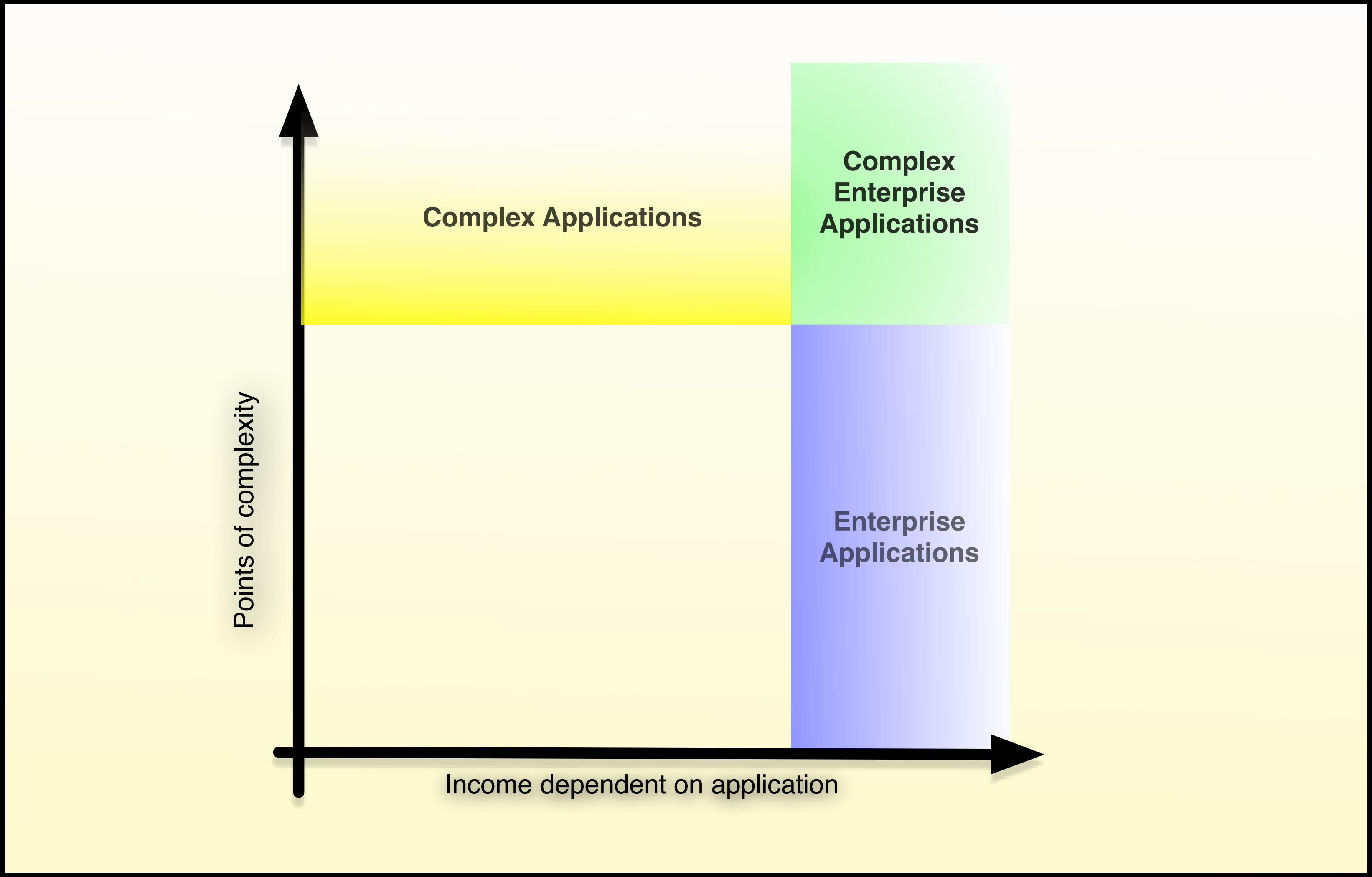
To provide a language and development environment for the rapid production of complex and distributed enterprise applications.

Complex Applications

Complexity Indicators

- Protocols used (HTTP, SMTP, RMI etc.)
- Data transformations applied
- Data representations (CSV, XML etc)
- Transactional boundaries
- Security boundaries
- Physical nodes
- Failure modes (no. of exceptional paths)
- Data paths
- System states
- Long running conversations
- Live versions of services
- Transactions processed per second

Complexity is orthogonal to the 'Enterprise' nature of an application



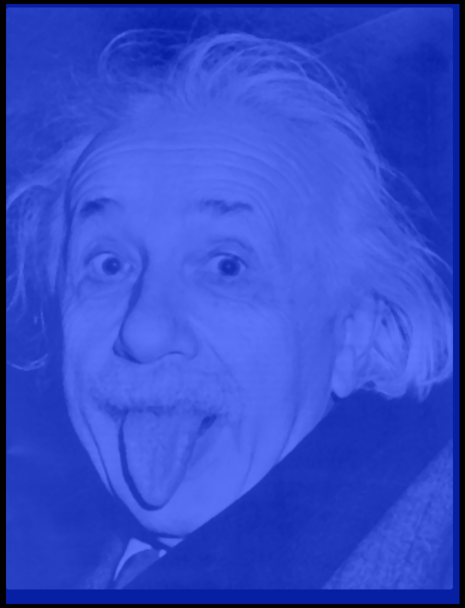
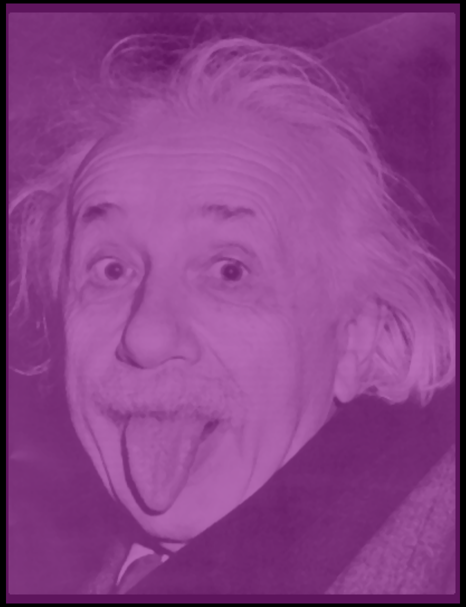
The majority of Enterprise Applications are not Complex Applications

Enterprise technologies are designed for Enterprise Applications but rarely for Complex Applications

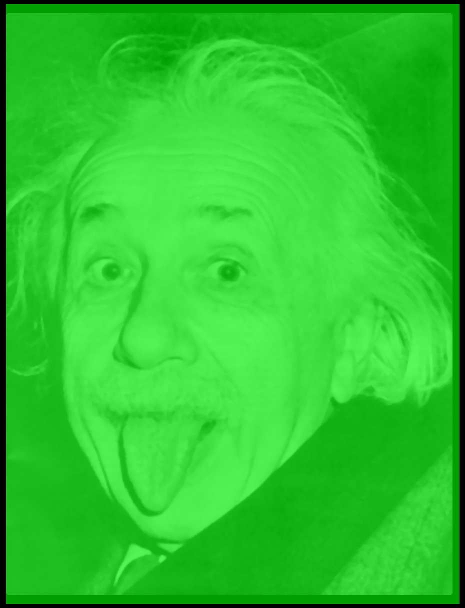
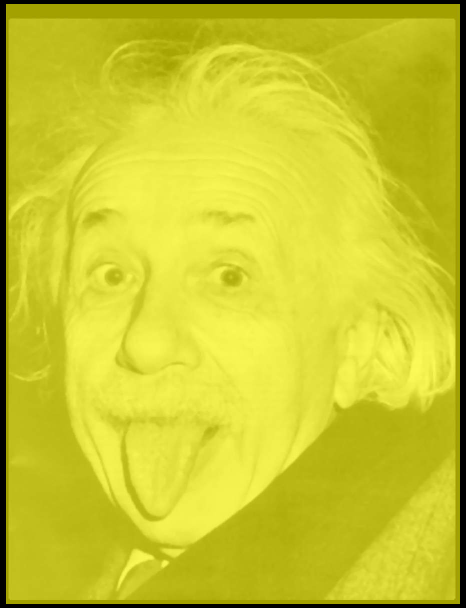
Einstein helps you develop complex applications ...

Easily!

Overview

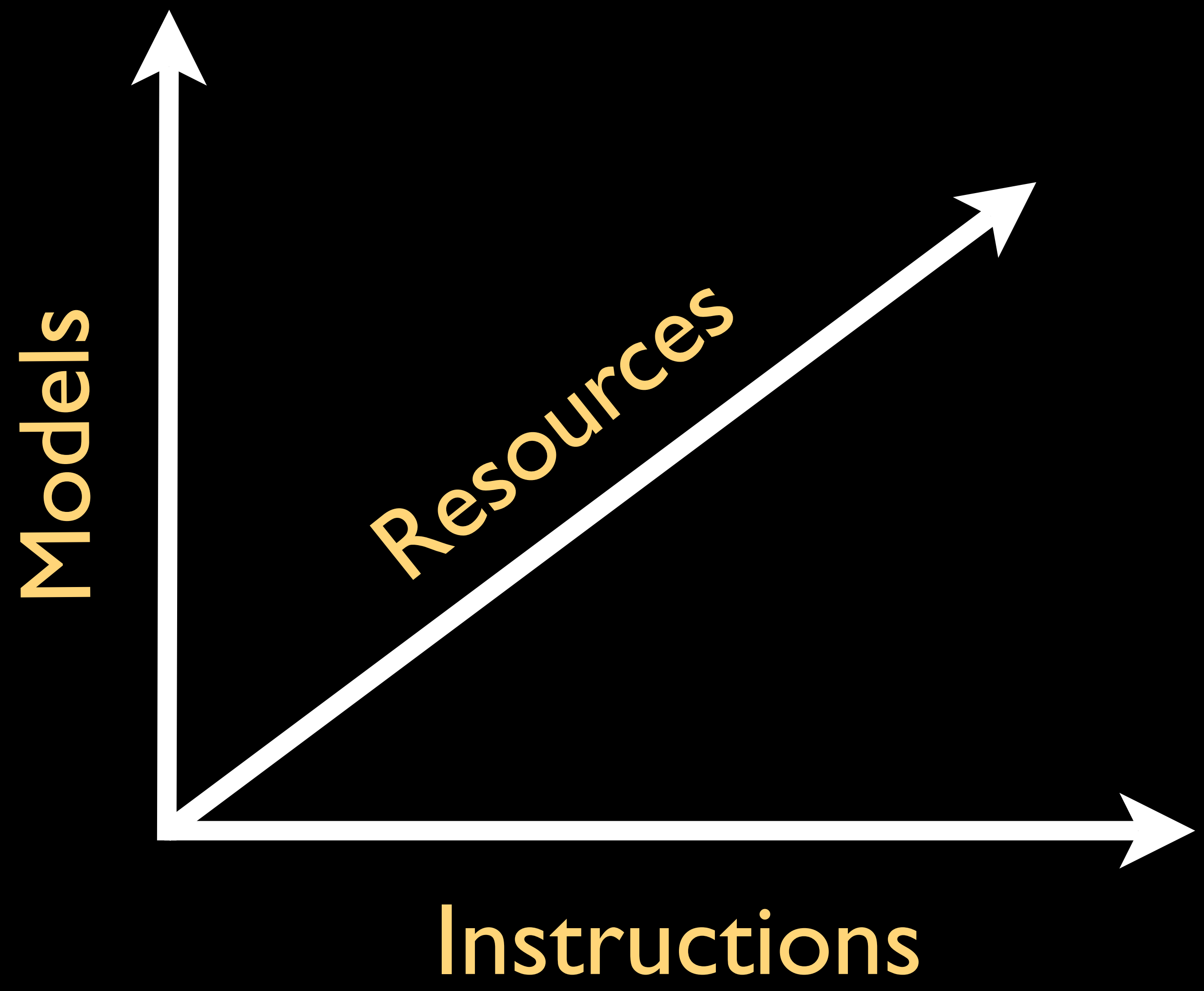
What is Einstein?

A Powerful 4GL

Einstein is a language designed from the bottom up to support: distributed execution, enterprise integration, message flows, type safety across multiple domains, heterogeneous data types and autonomic systems.

The 3 Axis of Einstein



Simplifying Complexity

- The Three Axis of Einstein: Models, Resources and Instructions reduce the number of possible operations required to implement an Enterprise system.

- For example a distributed join and a local join vary only by the Model not the Instruction. Sending a message to a JMS queue or a HTTP server varies by the Resource not the Instruction.

- The power of the Three Axis can be seen as the cartesian product of the Three Axis. 4 Models, 4 Resources and 4 Instructions gives 64 operations.

Resources

Resources

- Resources can <u>best</u> be compared to variables in Java. URIs can <u>best</u> be compared to Classes in Java they are (mostly) static, refer to actual resources and can provide functionality.
- Best, because Einstein is not Java, similarities are often superficial, Einstein is a 4GL and therefore has abstractions that don't map easily onto Java's language constructs.

URIs

- URIs in Einstein are almost the same as standard URIs (with URLs as a subset), except they support two extra features.
- Metadata - that is information which relates to a given provider (or scheme in URL/URI parlance).
- Nesting - a URI can have an indefinite amount of nesting.

“cache(timeout=2000):dynamic(lang=groovy):jms(i
mpl=RabbitMQ): $payload.destinationAddress”

URI DSL

- Writing a URI entirely as a String can be confusing.
- A DSL for URIs will be added to the language to support complex URIs.
- This will also reduce potential format errors.

```
uri::{
        camel (thing=thang) {
                    http://neil : password@thing.server.com
                        : 8000
                        /Place/ "another%20Place" ? (a=1, b=2)
        }
}
```

Type Safety

Type Safety

- Type Safety is as important when programming in the large, as programming in the small.
- Adding type safety to heterogeneous data types is not easy!

Guarantees

- A Guarantee is an executable assertion that can be placed on a message. The guarantee is made by a programmer but can be checked (as a textual assertion) by either the compiler or at runtime.
- Guarantees can also be executed at runtime to assure validity.
- Guarantees form the 'Typing' of Einstein .

Types

- Resources can be typed.
- The type is actually a collection of guarantees.
- The compiler can check for the existence of expected guarantees.

type **SmallWidgetType** ("groovy: payload instanceof Widget", "xpath://widgetSize < 100");

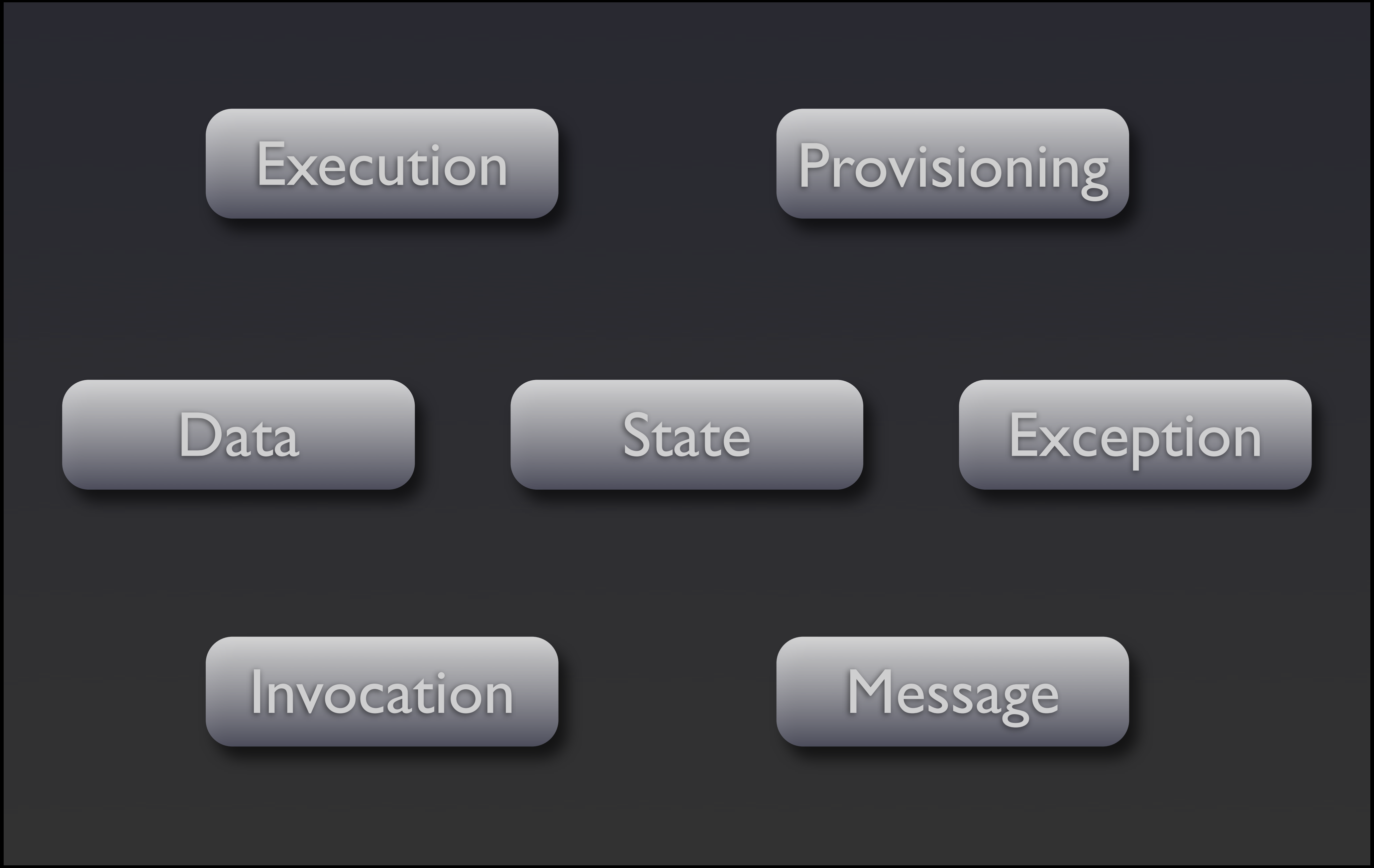
type **SmallBlueWidgetType** constrains **SmallWidgetType** with ("xpath://widgetColor = 'blue'");

The Models

Introduction

Models in Einstein are mostly non-functional requirements of the system. They express how things interact, not the interaction itself. They are often cross cutting in nature, one model can be applied to multiple interactions.

The Models


Execution
Provisioning
Data
State
Exception
Invocation
Message

Execution Model

This determines how instructions are executed.

Some Execution Models

- Direct
  - Single Thread
  - Synchronous
- SEDA
  - Multi Thread
  - Asynchronous
- Pipeline
  - Multi Thread
  - Synchronous

Data Model

This determines how the payload data is represented.

Data Models

- Data Models determine how payload objects are queried, split, joined etc.
- Einstein is agnostic to specific data formats, it uses a combination of query providers (like XPath) and Data Models to act upon Data.
- Data Models can be switched explicitly at any time in Einstein.

Some Data Models

- Object
  - Object Graph
  - Stream
- XML
  - DOM
  - StAX
  - String
- Other
  - CSV
  - Numeric

Message Model

This determines what type of object carries the payload.

Message Models

- Messages can be self-auditing.
- Messages can make their history available.
- Messages can be lightweight for performance.
- The Message Model applies to any Messages created in it’s scope.

Exception Model

This determines how exceptions will be handled.

Exception Models

- Einstein supports retry blocks, transactional blocks and try/catch blocks.
- An Exception Model determines what action to take in all these cases.

Distributed State Model

This determines how Systems, and other state aware components distribute state across nodes.

Distributed State Models

- Local
- Cached
- Replicated
- Stored
- QoS - Transactionality, Guarantees

Invocation Model

How a Service will be invoked

Invocation Models

- Native
  - The Service provided by the Implementation Model implements an Einstein Interface.
- Reflective
  - The appropriate method on the Service is selected based on the Message’s payload.
- Container
  - The Implementation Model is passed the message and it decides.

Provisioning Model

This determines how prepared Service instances are obtained

Provisioning Models

- Pooled
  - Built instances are pooled
- Singleton
  - A single shared instance
- On Demand
  - New instance created each time an instance is requested

Instructions

Tuple Operators

- Join - Join two tuples to form a third.
- Split - create a tuple from a single item tuple.
- Merge - convert multiple item tuple to a single item tuple.
- Give - pass a data tuple to an instruction tuple.
- Flatten - flatten tuple hierachy to a single tuple.

Message Operators

- Insert - insert data into a message payload.
- Extract - create a new message from part of another message’s payload.

Resource Operators

- Get - obtain a message from a resource.
- Send/Dispatch - pass a message from a resource.
- Listen - react to messages becoming availab le on a resource.
- Execute - treat the resource as a service and execute as such.

Conditionals

- If - execute if a resource query returns true.
- While - execute while a resource query returns true.

Tuple Conditionals

- Any - any of a tuple of conditions is true.
- None - none of a tuple of conditions is true.
- One - one of a tuple of conditions is true.
- Some - more than one of a tuple of conditions is true.
- All - all of a tuple of conditions is true.

Iteration

- Each - iterate over a tuple.

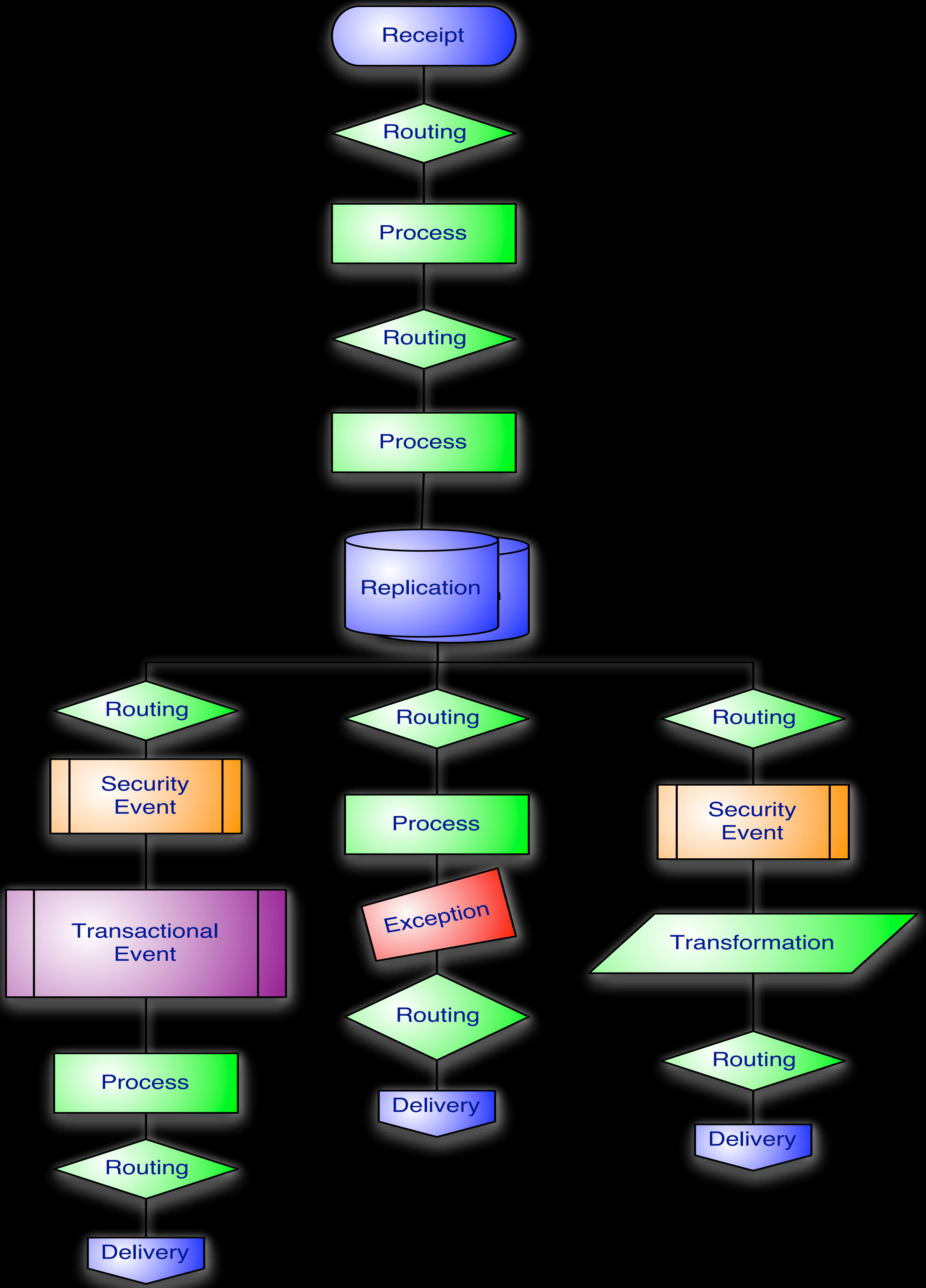
Instruction Groups

- Flow - each instruction in a flow group is executed sequentially with the result of one being the input to the next.
- Unsequenced (Parallel) - a set of instructions whose value is the result of the first to complete execution.
- Tuple - a collection of instructions which when executed form a tuple.
- Map - named tuples.

Getting the Message?

Messaging Events

Messaging events provide controlled state changes to a message and can be used for auditing/tracking/debugging purposes.


Receipt
Routing
Process
Routing
Process
Replication
Routing
Routing
Routing
Security Event
Process
Security Event
Transactional Event
Exception
Transformation
Routing
Process
Routing
Delivery
Routing
Delivery
Delivery

Providers

The Provider Framework

Provider Model

- Each provider is a collection of Facades, each Facade relates to an activity (e.g. Query, Get, Send, Listen, Execute) that the resource can participate in.
- Simple, but not simplistic means to extend Einstein.
- Isolates incompatibly licensed code.

Example URLs

- camel:http://rss.news.yahoo.com/rss/topstories
- xpath://*[local-name()='content']/@url
- java:org.cauldron.einstein.ri.examples.MontageMaker

Profiles

Profiles

Profiles provide a means of connecting multiple models to describe a particular scenario. A profile is used to provide a set of default models for a system.

Example Profiles

- Lightweight High Volume
- Low Volume High QoS
- Formal Audited

Custom Profiles

Custom profiles can be designed to describe any development scenario and then be applied to multiple Services/Systems.

An Example

```
einstein.asm :: {
                    @xmlProfile;
                    while "bool:true" {
                        get "camel:http://rss.news.yahoo.com/rss/topstories";
                        split "xpath://*[local-name()='content']/@url" {
                            @pojoProfile;
                            send "java:org.cauldron.einstein.ri.examples.ImageMaker";
                            send "java:org.cauldron.einstein.ri.examples.MontageMaker";
                        };
                    };
      };
```

Thank You!