Leveraging ETS Effectively

Evadne Wu

ev@radi.ws
twitter.com/evadne

REVISED
8 April 2019

Outline

📚 **Background:** Historical Context

🛠️ **Using ETS:** C/R/U/D • Match Specifications

🤯 **Use Cases:** When & Why • Libraries & Applications

🚀 **Adopting ETS:** Ways to Add ETS to Your Application

self()

Elixir Enthusiast • ~~Bug Farmer~~ • Bodger of Applications

Inquiries: ev@radi.ws

Arguments: twitter.com/evadne

1

Background

Historical Context ⅓

A History of Erlang

Joe Armstrong
Ericsson AB
joe.armstrong@ericsson.com

Abstract

Erlang was designed for writing concurrent programs that "run forever." Erlang uses concurrent processes to structure the program. These processes have no shared memory and communicate by asynchronous message passing. Erlang processes are lightweight and belong to the language, not the operating system. Erlang has mechanisms to allow programs to change code "on the fly" so that programs can evolve and change as they run. These mechanisms simplify the construction of software for implementing non-stop systems.

This paper describes the history of Erlang. Material for the paper comes from a number of different sources. These include personal recollections, discussions with colleagues, old newspaper articles and scanned copies of Erlang manuals, photos and computer

operations occur. Telephony software must also operate in the "soft real-time" domain, with stringent timing requirements for some operations, but with a more relaxed view of timing for other classes of operation.

When Erlang started in 1986, requirements for virtually zero down-time and for in-service upgrade were limited to rather small and obscure problem domains. The rise in popularity of the Internet and the need for non-interrupted availability of services has extended the class of problems that Erlang can solve. For example, building a non-stop web server, with dynamic code upgrade, handling millions of requests per day is very similar to building the software to control a telephone exchange. So similar, that Erlang and its environment provide a very attractive set of tools and libraries for building non-stop interactive distributed services.

Historical Context ⅔

> In developing large-scale telecommunications applications it soon became apparent that the "pure" approach of storing data could not cope with the demands of a large project and that some kind of real-time database was needed.

> This realisation resulted in a DBMS called Mnesia.

Historical Context ⅔

> At the highest level of abstraction was a new query language called Mnemosyne (developed by Hans Nilsson) and at the lowest level were a set of primitives in Erlang with which Mnesia could be written.

J. Armstrong. A History of Erlang. 2009.

Summary

ETS was created as an Erlang module in order to meet higher level requirements.

It provides an **escape hatch** from the functional world, and is useful when / where the best solution requires destructive data structures.

Also See: C. Okasaki. Purely Functional Data Structures. 1996.

2

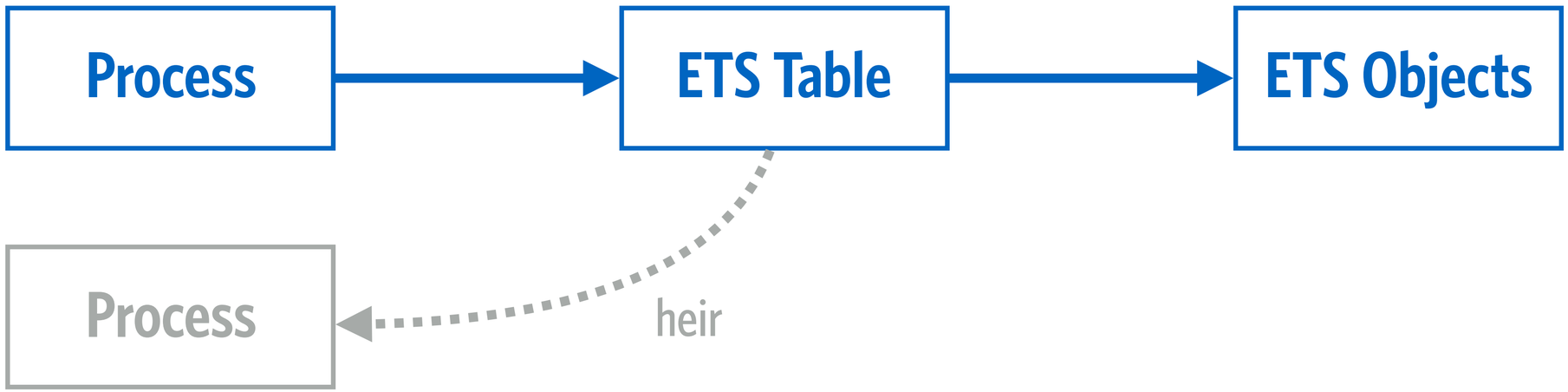
Using ETS

Conceptual Model ½

Each ETS table holds a group of Objects that are represented as Tuples. Each Tuple holds one or more Elements.

Tables are held by processes that created them, but can be given away (or transferred to heir in case the parent process crashes).

Conceptual Model ²⁄₂



Quick Start

Example: Create & Populate an ETS Table

```
table_reference = :ets.new(:table_name, [:set])
:ets.insert(table_reference, {:a, 1})
:ets.insert(table_reference, [{:b, 2}, {:c, 3}])
:ets.lookup(table_reference, :a) # -> [{:a, 1}]
:ets.lookup(table_reference, :d) # -> []
```

Table Types

Type	Objects Per Key	Duplicates Allowed?
Set	One	No
Ordered Set	One	No
Bag	Many	No
Duplicate Bag	Many	Yes

Access Types

Type	Access Control
Public	R/W for any process
Protected	R/W for owner; R for others
Private	R/W for owner; no access for others

Key Functions

new/2, delete/1

select/2, select/3, match_object/2, match/2

insert/2, insert_new/2

update_element/3, update_counter/3, update_counter/4

delete/2, delete_all_objects/1, delete_object/2

fun2ms/1, tab2list/1

Match Specifications

You can use a Match Specification to represent an ETS Query. Each Match Specification has 3 parts:

Match Head, representing shape of the desired objects

Match Conditions, representing desired filters

Match Body, representing format of returned values

ets:match_object/2

Example: Matching Objects in ETS Table

```
table_ref = :ets.new(table_name, [:set])
_ = :ets.insert(table_ref, [{:a, 1}, {:b, 2}])
[_, _] = :ets.match_object(table_ref, {:"$1", :"$2"})
[_] = :ets.match_object(table_ref, {:"$1", 2})
```

ets:match/2

Example: Matching Objects in ETS Table

```
table_ref = :ets.new(table_name, [:set])
data = [{:a, 1, 2, 3}, {:b, 4, 5, 6}]
pattern = {:a, :"$1", :"$2", :"$3"}
_ = :ets.insert(table_ref, data)
[{:a, 1, 2, 3}] = :ets.match_object(table_ref, pattern)
[[1, 2, 3]] = :ets.match(table_ref, pattern)
```

ets:select/2

Example: Matching Objects in ETS Table

```
table_ref = :ets.new(table_name, [:set])
data = [{:a, 1, 2, 3}, {:b, 4, 5, 6}]
match_spec = {{:a, :"$1", :"$2", :"$3"}, [], [[:"$1", :"$3"]]}
_ = :ets.insert(table_ref, data)
[[1, 3]] = :ets.select(table_ref, [match_spec])
```

ets:fun2ms/1

Example: Generating ETS Match Specification with ets:fun2ms/1

```
> :ets.fun2ms(fn {:a, x, c, x} when x > 5 -> [:a, x, c, x] end)

[{
  {:a, :"$1", :"$2", :"$1"},
  [{:>, :"$1", 5}],
  [[:a, :"$1", :"$2", :"$1"]]
}]
```

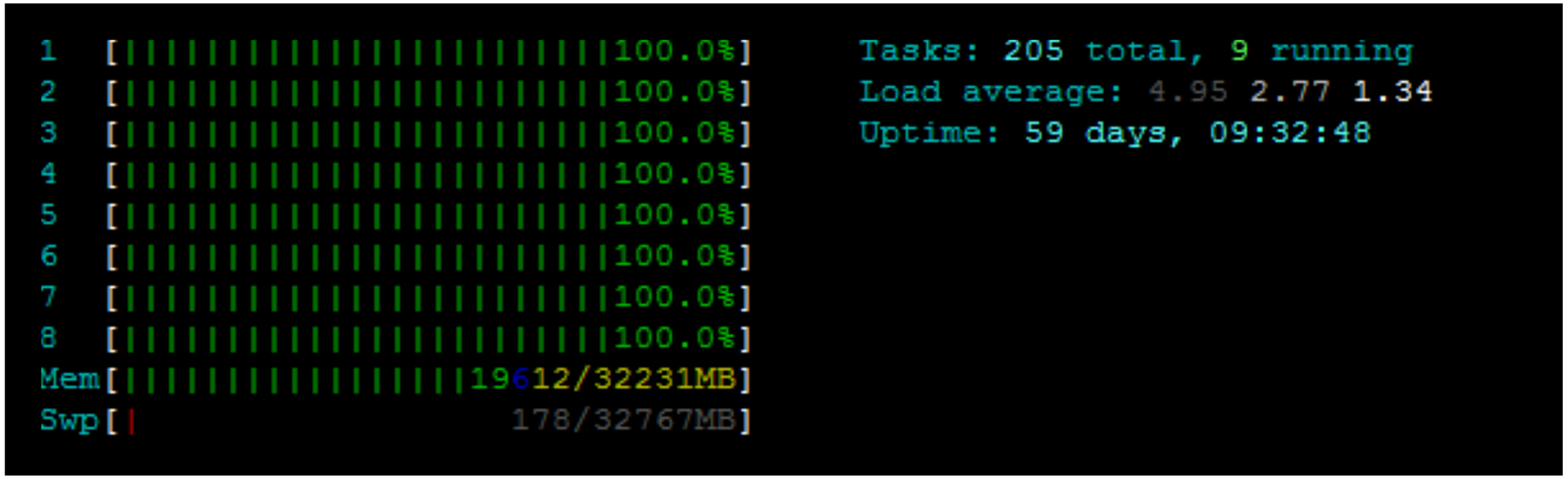
3

Use Cases

Consider ETS When...

...your purely functional code is not fast enough!

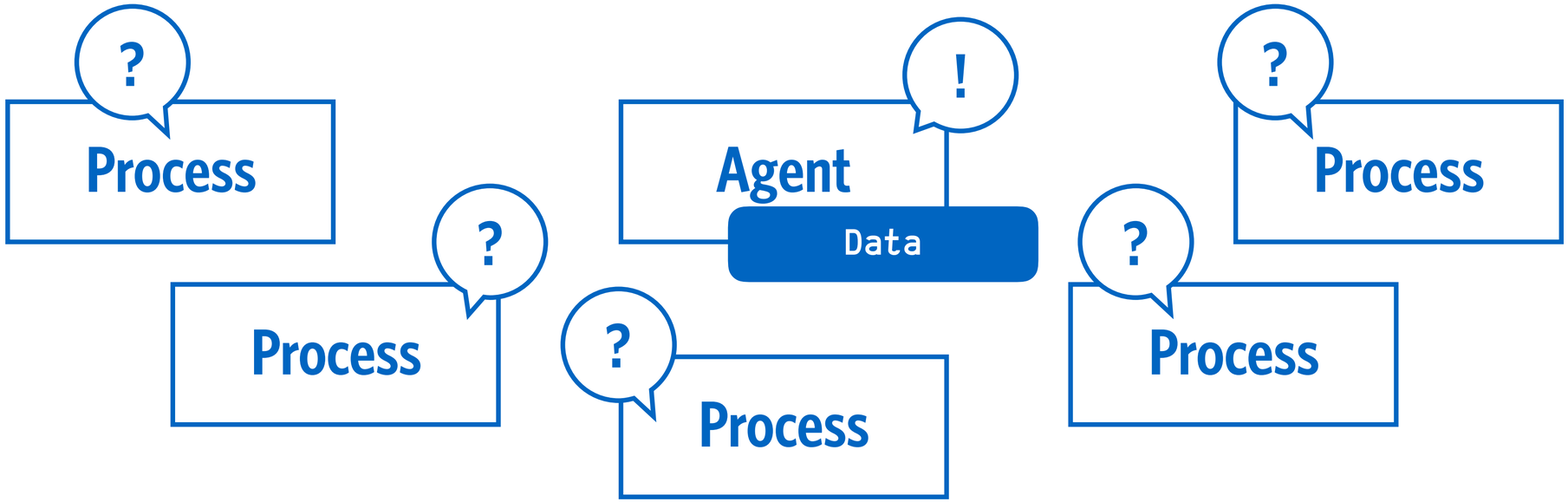
This will usually become apparent when you profile your application.



Consider ETS When...

...your application has concurrency bottlenecks!

Each Process has one mailbox and all messages are processed serially. This can cause bottlenecks to form when you have many actors accessing the data.



Deciding When to Use ETS

Consider: Data Source, Volume, Frequency of Changes, Frequency of Reads and Writes... this is, in itself, a broad and interesting domain.

See: Martin Kleppmann. Designing Data-Intensive Applications (2017).

Demo: Agent vs. ETS

Perhaps a demonstration will help you make the right decision for your application.

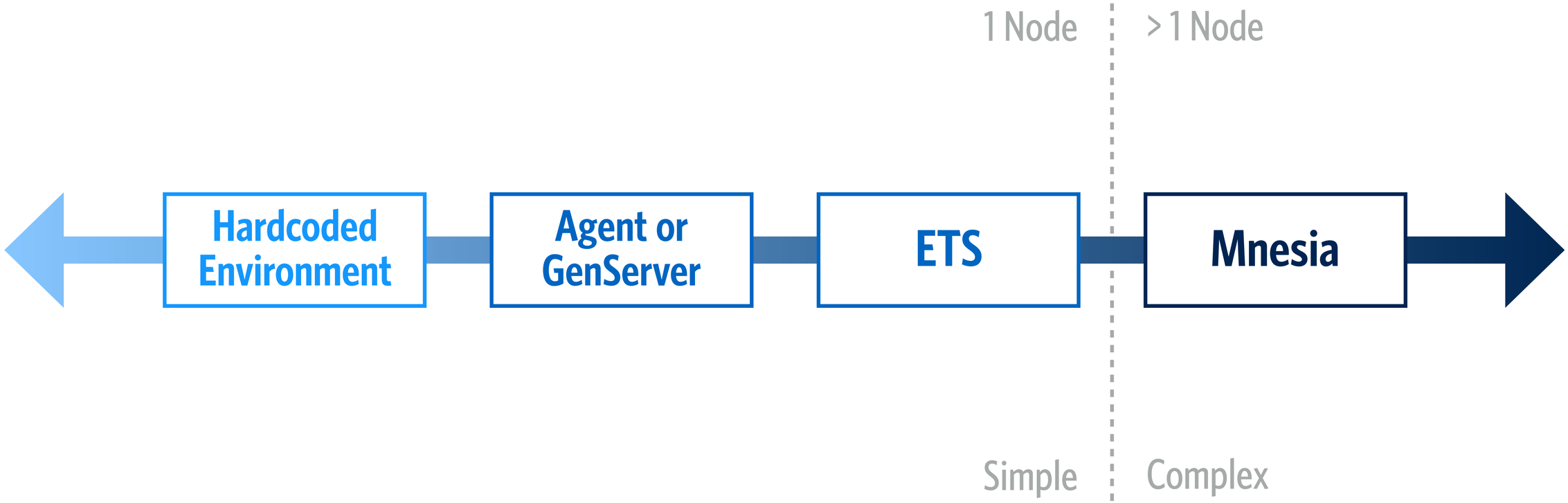
Module: Playground.Scenario.{ETS, Agent}
Dimensions: Sequential vs. Concurrent • Ordered vs. Unordered • Tasks vs. One-Shot

Recap: Agent vs. ETS

Agents: are single-threaded; updating a value held by an Agent requires rewriting the whole data structure.

ETS: allows greater concurrency via **destructive** updates; updates are more performant as well.

Mental Model: Complexity Gradient



ETS Use Cases: 4 Examples

- Persistent Shared State i.e. Presets
- Ephemeral Shared State i.e. Cached Data
- Inter-Process Communication i.e. Message Buffer
- Concurrency Control i.e. Atomic Counters

Use Case 1: Sharing Preset Data

If you have a large amount of data which either changes infrequently or does not need to be fully consistent, you can still use Presets and eliminate the database.

By eliminating the Database, you would have removed a bottleneck for scaling.

Use Case 1: Sharing Preset Data

Example 1: The **tzdata** library.

Goal: Make Time Zone information available.

Approach: Bundle a version of the data with the library. On startup, (optionally) refresh it and load the ETS tables, which are public and can be read with convenience functions.

Use Case 1: Sharing Preset Data

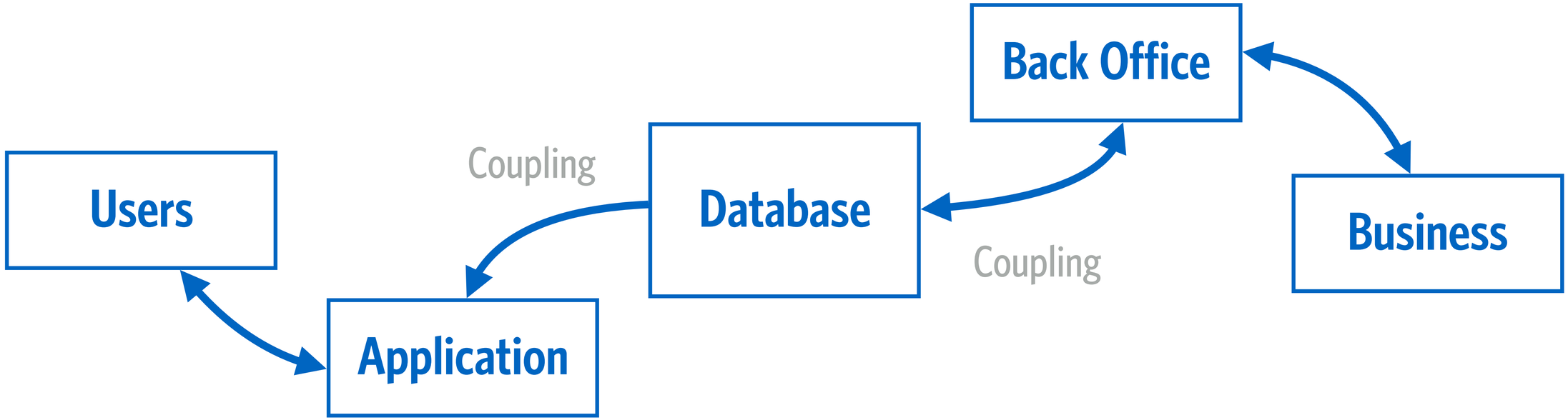
Example 2: Hotelicopter (Roomkey)

Goal: Fix scaling problems.

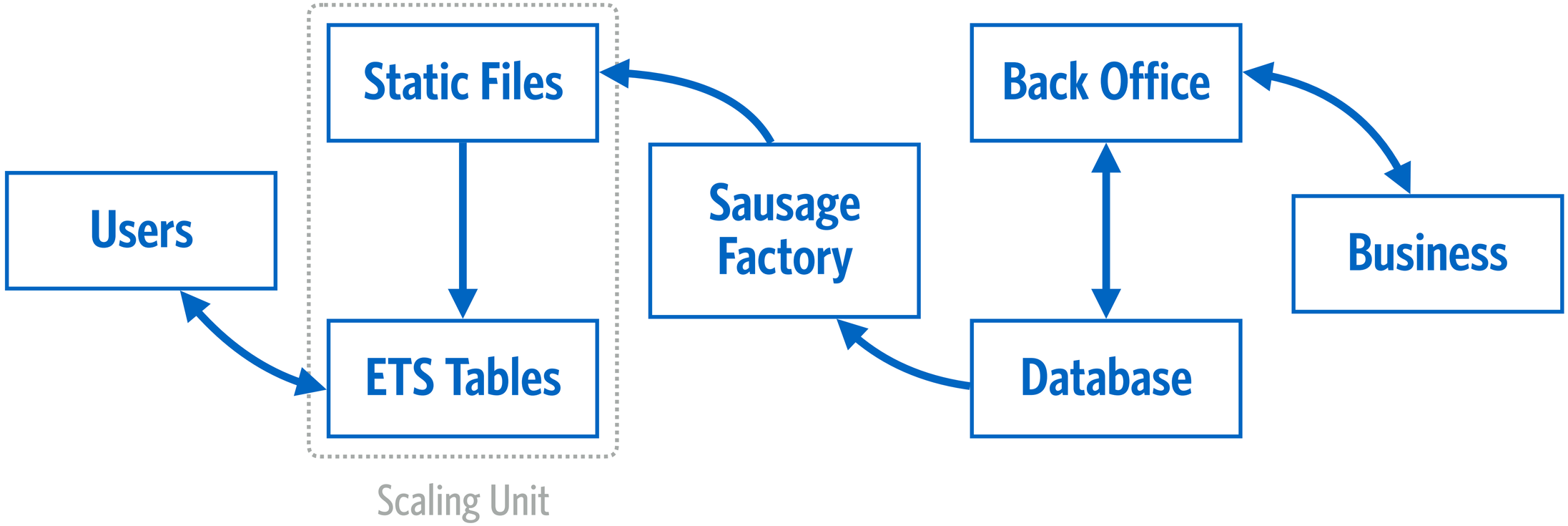
Approach: Ditch shared state service, bundle state within application. Reconfirm live state only when transactions takes place.

See: Colin Steele. Against the Grain.

Without Presets: OLTP All The Way



With Presets: Easy to Scale




ONE NANOSECOND!
d = 1ft
c = 983571056
≈ 10^-9 s
= 1ns
d=ct
t=d/c
comic.browserling.com

×1,000

Use Case 2: Storing Cached State

Example 1: The **cache_tab** Library.

Goal: To reduce the number of backend operations.

Approach: Start N shards, where each shard holds an ETS table, and N = number of schedulers. Use consistent hashing to determine which shard holds each object. Handle expiration on a per-object basis.

Use Case 2: Storing Cached State

Example 2: The **cxy_cache** module in **epocxy.**

Goal: To be able to bulk-evict stale objects from cache.

Approach: Use 2 ETS tables, one for the New Generation and another for the Old Generation. Constantly promote objects to the New Generation. Evict objects in the Old generation by dropping the ETS table.

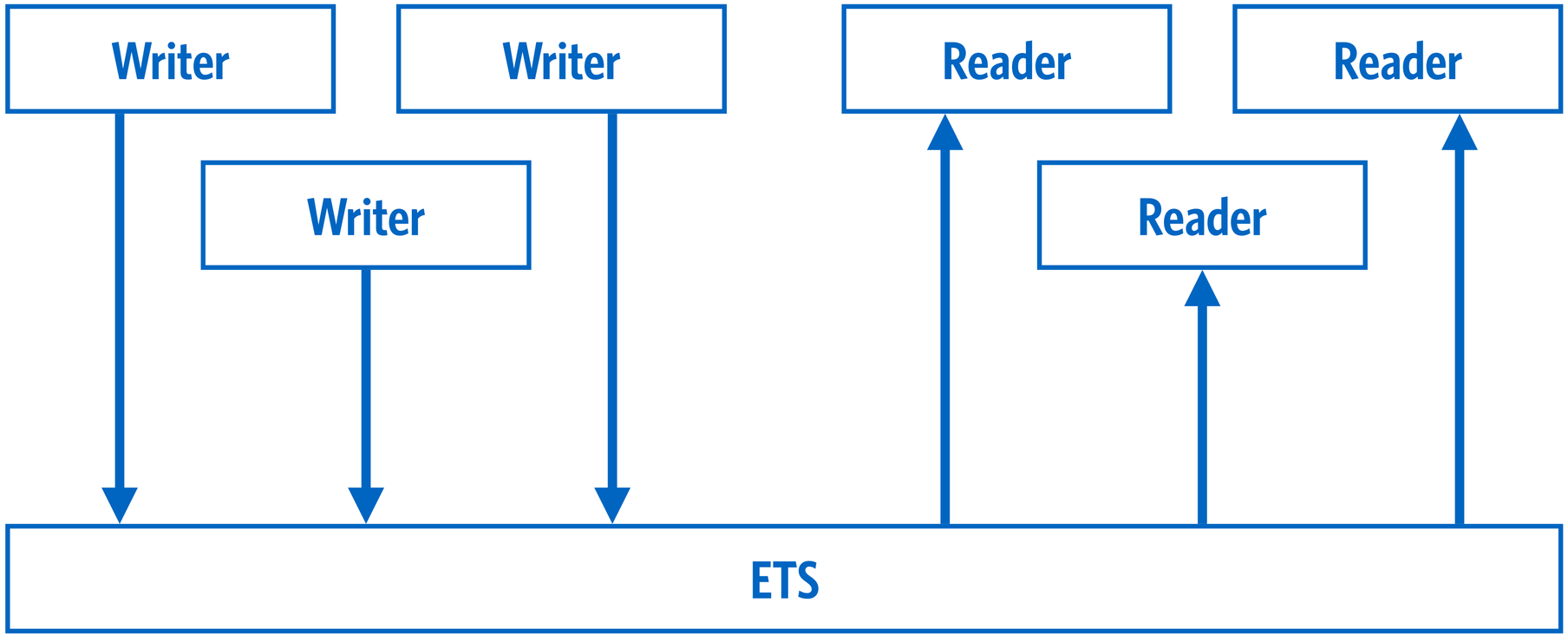
Use Case 3: Facilitating IPC

Example: The **ets_buffer** module in **epocxy.**

Goal: Have a way to support multiple writers and readers, without hitting constraints of a single process.

Approach: Use the “Ordered Set” table type to impose ordering and build FIFO / LIFO / Ring Buffers. Make the ETS tables public.

Use Case 3: Facilitating IPC



Use Case 4: Concurrency Control

Example: The **cxy_ctl** module in **epocxy.**

Goal: To regulate system usage, and avoid overload.

Solution: Use ETS to count the number of in-flight processes for every Process Type, and enforce quotas this way to prevent running new Processes that already have too many of their type running.

Sidebar: Counting in ETS vs. Atomics

When using ETS, you can increase/decrease a counter by using either update_element/3 or update_counter/3.

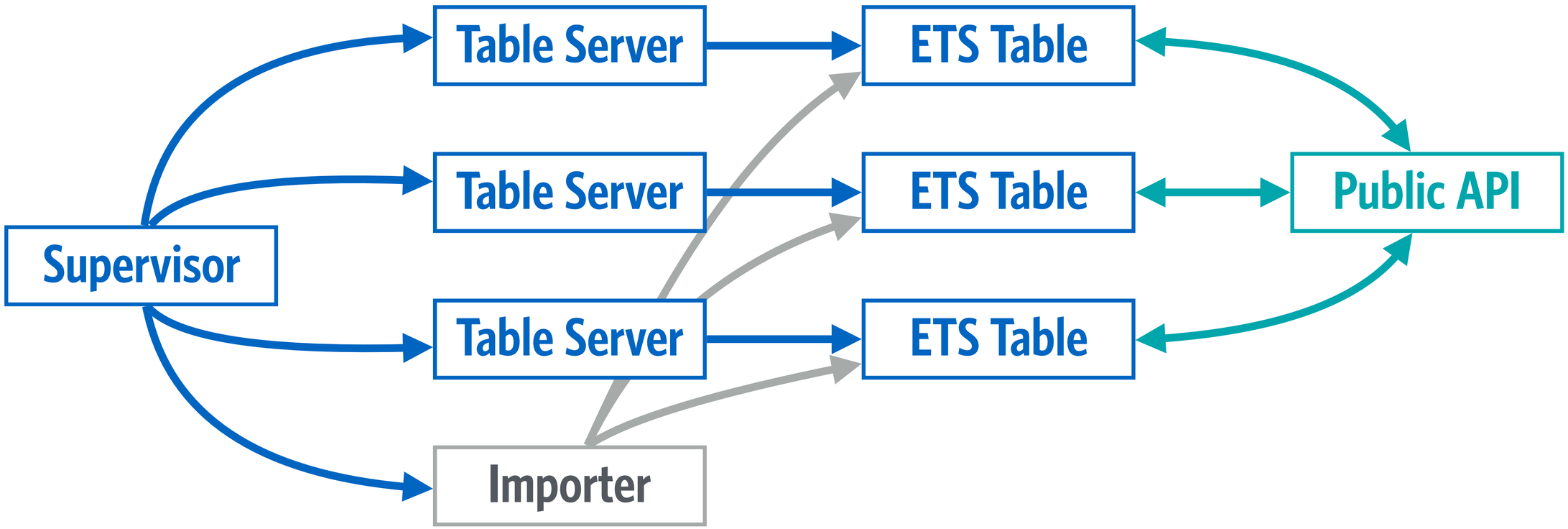
The Atomics module has been made available in OTP 21.2, which provides another great way of counting.

Module: Playground.Scenario.Counters.{Many, One, Atomics.Many, Atomics.One}
Concurrent & Sequential Access / 1-Arity Atomics / n-Arity Atomics

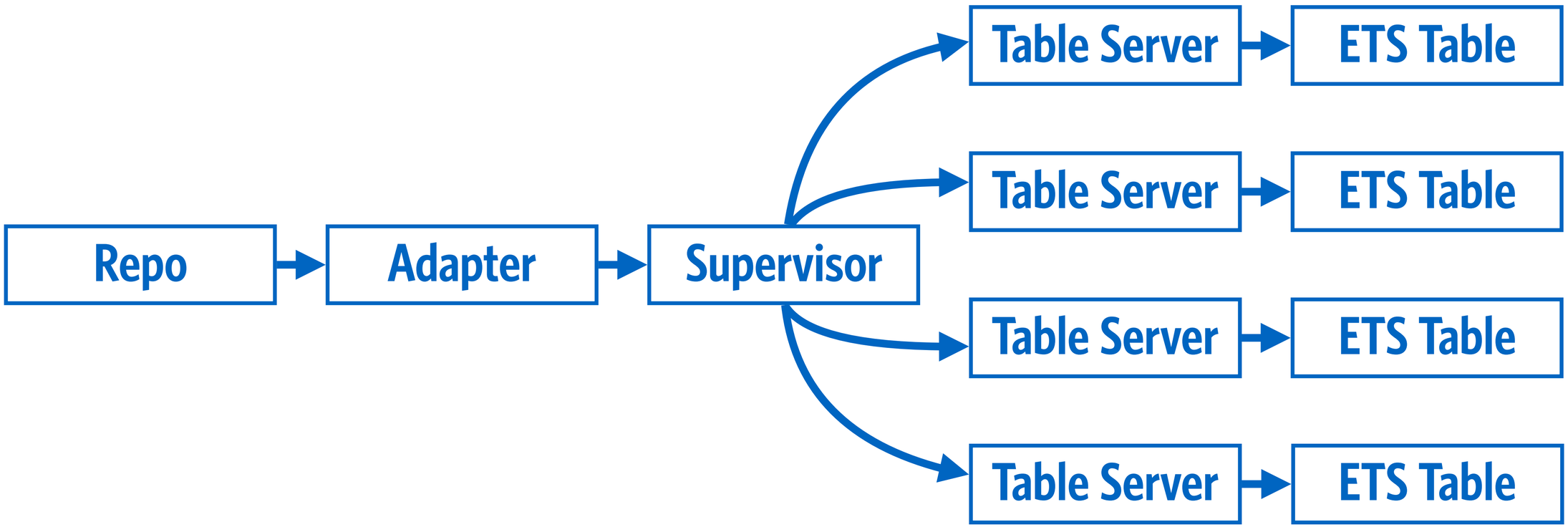
4

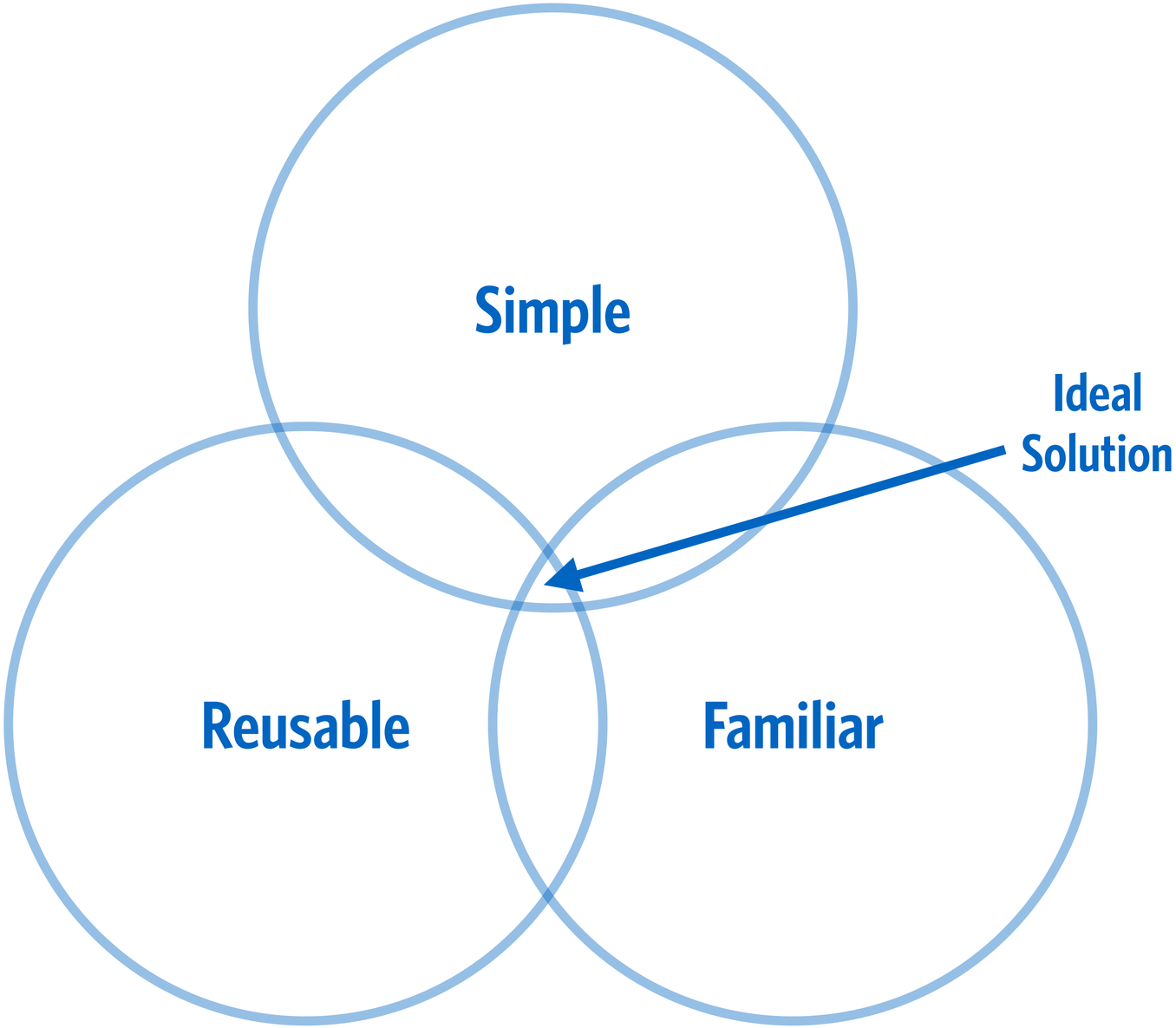
Adopting ETS

Adopting ETS: Cookie Cutter Layout



Adopting ETS: With Ecto




Simple
Ideal
Solution
Reusable
Familiar

Adopting ETS: With Ecto

Solution Benefits

Easier to Understand
maps and structs are pervasive in Elixir

Reuse of Ecto Schemas
facilitating easy migration to Presets

Familiar Query Syntax via Repo callbacks
easy to understand and maintain, reducing conceptual burden

Adopting ETS: With Ecto

Solution Requirements

Converting between Ecto Schemas and ETS Tuples

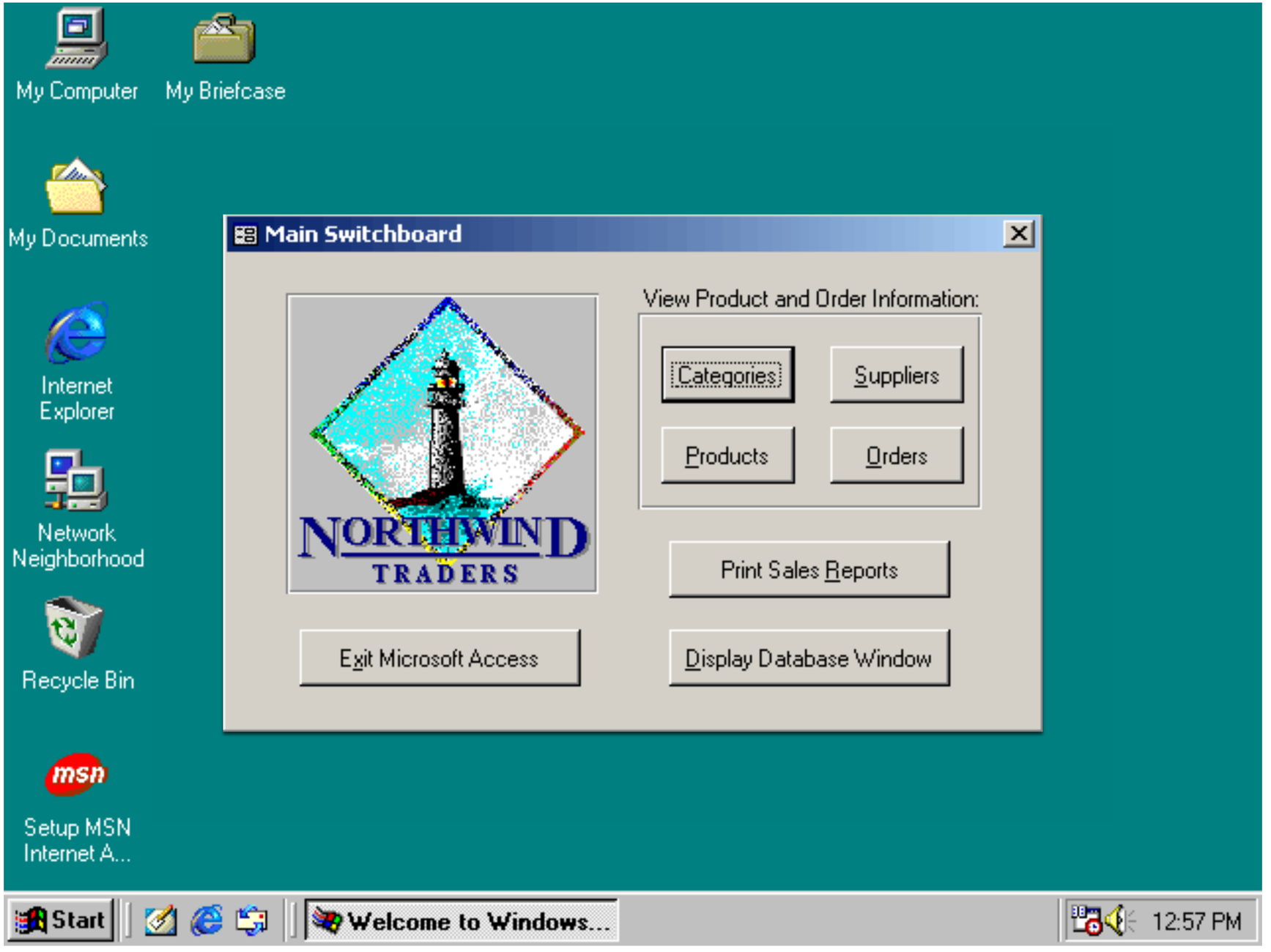
seamless translation + optimisation opportunity for Ordered Sets

Transforming Ecto Fields / Queries for ETS execution

consistent formation of Query Plans

Exposing Streams via Fixation (Safe Enumeration)

ability to enumerate data in the Elixir Way


My Computer
My Briefcase
My Documents
Internet Explorer
Network Neighborhood
Recycle Bin
Setup MSN Internet A...
Main Switchboard
NORTHWIND TRADERS
View Product and Order Information:
Categories
Suppliers
Products
Orders
Print Sales Reports
Exit Microsoft Access
Display Database Window
Start
Welcome to Windows...
12:57 PM

Reference Materials

Reference Materials ⅓

Erlang/OTP: Atomics
http://erlang.org/doc/man/atomics.html

Erlang/OTP: ETS
http://erlang.org/doc/man/ets.html

Purely Functional Data Structures
https://www.cs.cmu.edu/~rwh/theses/okasaki.pdf

Designing Data Intensive Applications
http://dataintensive.net

Reference Materials ⅔

Erlang Patterns of Concurrency
https://github.com/duomark/epocxy

Avoiding Single Process Bottlenecks By Using Ets Concurrency
http://www.erlang-factory.com/static/upload/media/1394716488140115jaynelson.pdf

When ETS Is Too Slow
https://speakerdeck.com/mrallen1/when-ets-is-too-slow

Dispcount: Erlang task dispatcher based on ETS counters
https://github.com/ferd/dispcount

Reference Materials ⅔

A History of Erlang
http://www.cse.chalmers.se/edu/year/2009/course/TDA381_Concurrent_Programming/ARCHIVE/VT2009/general/languages/armstrong-erlang_history.pdf

A Study of Erlang ETS Table Implementations and Performance
http://erlang.org/workshop/2003/paper/p43-fritchie.pdf

On the Scalability of the Erlang Term Storage
http://winsh.me/papers/erlang_workshop_2013.pdf

Thank You