TSL Language

Aos Engine Family

SqLite TSL Integration

Year-2

David Simmons, Smallscript Corp

Tcl 2017 Conference

SqLite: SqLite APIs for Cmd Extensions

- "af" procedures
  - af: normal
  - afd: deterministic
  - afEach: normal tables
  - afEach: deterministic tabl
- Deterministic means
  - cacheable idempotent
- SqLite Api Integration
  - build flags
  - a few other APIs used as hooks

```
sqlite3_create_function_v2(db, "af", -1, SQLITE_UTF8,
  this, af, nullptr, nullptr, SqliteClosed);
sqlite3_create_function_v2(db, "afd", -1, SQLITE_UTF8 | SQLITE_DETERMINISTIC,
  this, af, nullptr, nullptr, SqliteClosed);
//
sqlite3_create_function_v2(db, "afEach", -1, SQLITE_UTF8,
  this, nullptr, afEachStep, afEachFinal, SqliteClosed);
sqlite3_create_function_v2(db, "afdEach", -1, SQLITE_UTF8 | SQLITE_DETERMINISTIC,
  this, nullptr, afEachStep, afEachFinal, SqliteClosed);
```

SqLite: UUID - Universally unique identifier

- UUID `123e4567-e89b-12d3-a456-426655440000`
  `xxxxxxxx-xxxx-Mxxx-Nxxx-xxxxxxxxxxxx`

```
uuid-create-sequential()
```

UUID Record Layout

Name	Length (Bytes)	Length (Hex Digits)	Contents
time_low	4	8	integer giving the low 32 bits of the time
time_mid	2	4	integer giving the middle 16 bits of the time
time_hi_and_version	2	4	4-bit "version" in the most significant bits, followed by the high 12 bits of the time
clock_seq_hi_and_res clock_seq_low	2	4	1-3 bit "variant" in the most significant bits, followed by the 13-15 bit clock sequence
node	6	12	the 48-bit node id

SqLite: Time conversions

- Time Conversion
  - %J: Julian days as float
  - The julianday() function returns the Julian day - the number of days since noon in Greenwich on November 24, 4714 B.C. (Proleptic Gregorian calendar).

Function	Equivalent strftime()
date(...)	strftime('%Y-%m-%d', ...)
time(...)	strftime('%H:%M:%S', ...)
datetime(...)	strftime('%Y-%m-%d %H:%M:%S', ...)
julianday(...)	strftime('%J', ...)

- **time-fmt**: tsl-command
  - custom time parsing and formatting library
  - parses most text forms
  - parses numbers as either epoch seconds in epoch, uuid or FileTime, time 10^{-7} 64-bit time

```
• time-fmt({now +1 day -2 hours
   -7 min, {%J});
• time-fmt(connection.time-created,
   {%Y/%m/%d %H:%M:%S.%f UTC});
• time-fmt(uuid-create-sequential(),
   {%Y/%m/%d %H:%M:%S.%f UTC});
```

SqLite: JSON Interchange & Ad-Hoc Records

- JSON1 Library
  - Json data in records
    - jFieldName
  - Json as tables in sql expressions
- CRUD triggers on columns and indexes
- JSON1 (flaws)
  - Empty and Nullable
  - Caching

1. json(*json*)
2. json_array(*value1*,*value2*,...)
3. json_array_length(*json*)
   json_array_length(*json*,*path*)
4. json_extract(*json*,*path*,...)
5. json_insert(*json*,*path*,*value*,...)
6. json_object(*label1*,*value1*,...)
7. json_patch(*json1*,json2)
8. json_remove(*json*,*path*,...)
9. json_replace(*json*,*path*,*value*,...)
10. json_set(*json*,*path*,*value*,...)
11. json_type(*json*)
    json_type(*json*,*path*)
12. json_valid(*json*)
13. json_quote(*value*)

There are two aggregate SQL functions:

14. json_group_array(*value*)
15. json_group_object(name,*value*)

The two table-valued functions are:

16. json_each(*json*)
    json_each(*json*,*path*)
17. json_tree(*json*)
    json_tree(*json*,*path*)

SqLite: sql triggers

```
CREATE TRIGGER aAccount_DELETE
         AFTER DELETE
            ON aAccount
          WHEN old.aAccountId NOT NULL
BEGIN
    DELETE FROM aPhoneNumber
          WHERE aAccountId = old.aAccountId;
    UPDATE aLxcNumber
       SET aAccountId = NULL
     WHERE aAccountId = old.aAccountId;
END;
```

```
CREATE TRIGGER aAccount_INSERT
         AFTER INSERT
            ON aAccount
BEGIN
    UPDATE aAccount
       SET lxcNumber = json_extract(new.jAccount, '$.lxcNumber'),
           ppdNumber = json_extract(new.jAccount, '$.ppdNumber'),
           lpdNumber = json_extract(new.jAccount, '$.lpdNumber'),
           email = json_extract(new.jAccount, '$.email'),
           imeiNumber = json_extract(new.jAccount, '$.imeiNumber'),
           iccidNumber = json_extract(new.jAccount, '$.iccidNumber'),
           aAccountId = json_extract(new.jAccount, '$.aAccountId'),
           jAccount = json_set(new.jAccount, '$.aAccountRid', new.aAccountRid)
     WHERE rowid = new.rowid/* AND new.jAccount != old.jAccount */ AND
           json_valid(jAccount);
END;
```

```
CREATE TRIGGER aAccount_UPDATE
        AFTER UPDATE
           ON aAccount
BEGIN
    UPDATE aAccount
       SET lxcNumber = json_extract(new.jAccount, '$.lxcNumber'),
           ppdNumber = json_extract(new.jAccount, '$.ppdNumber'),
           lpdNumber = json_extract(new.jAccount, '$.lpdNumber'),
           email = json_extract(new.jAccount, '$.email'),
           imeiNumber = json_extract(new.jAccount, '$.imeiNumber'),
           iccidNumber = json_extract(new.jAccount, '$.iccidNumber')
     WHERE rowid = new.rowid AND
           ifnull(new.jAccount, "") != ifnull(old.jAccount, "") AND
           json_valid(jAccount);
    INSERT INTO aPhoneNumber (
                                phoneNumber,
                                aAccountId,
                                phoneKind
                            )
                            SELECT new.lpdNumber,
                                   new.aAccountId,
                                   'lpd'
                             WHERE ifnull(new.lpdNumber, "") != ifnull(old.lpdNumber, "") AND
                                   new.aAccountId NOT NULL AND
                                   new.lpdNumber NOT NULL AND
                                   (old.lpdNumber IS NULL/* OR NOT EXISTS (SELECT...) */);
    INSERT INTO aPhoneNumber (
                                phoneNumber,
                                aAccountId,
                                phoneKind
                            )
                            SELECT new.ppdNumber,
                                   new.aAccountId,
                                   'pd'
                             WHERE ifnull(new.ppdNumber, "") != ifnull(old.ppdNumber, "") AND
                                   new.aAccountId NOT NULL AND
                                   new.ppdNumber NOT NULL AND
                                   (old.ppdNumber IS NULL/* OR NOT EXISTS (SELECT...) */);
    UPDATE aPhoneNumber
       SET phoneNumber = new.lpdNumber,
           aAccountId = new.aAccountId
     WHERE ifnull(new.lpdNumber, "") != ifnull(old.lpdNumber, "") AND
           new.lpdNumber NOT NULL AND
           old.lpdNumber NOT NULL AND
           aAccountId = old.aAccountId AND
           phoneNumber = old.lpdNumber;
    UPDATE aPhoneNumber
       SET phoneNumber = new.ppdNumber,
           aAccountId = new.aAccountId
     WHERE ifnull(new.ppdNumber, "") != ifnull(old.ppdNumber, "") AND
           new.ppdNumber NOT NULL AND
           old.ppdNumber NOT NULL AND
           aAccountId = old.aAccountId AND
           phoneNumber = old.ppdNumber;
    DELETE FROM aPhoneNumber
          WHERE ifnull(new.lpdNumber, "") != ifnull(old.lpdNumber, "") AND
                new.lpdNumber IS NULL AND
```

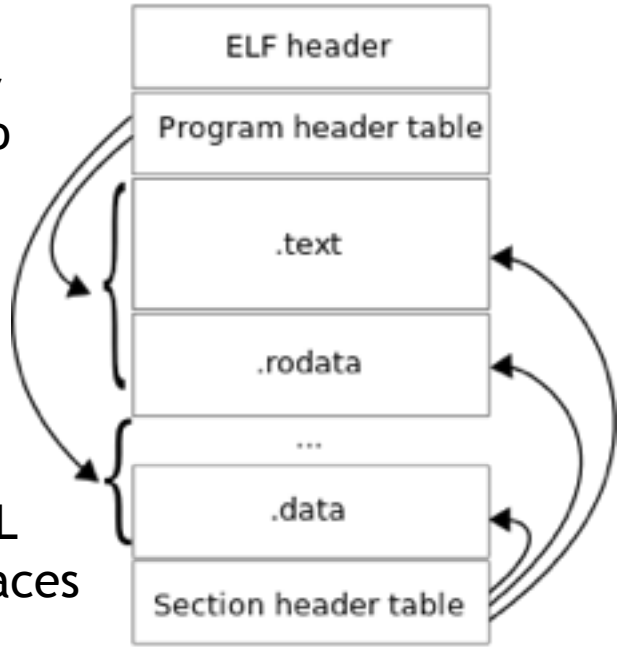
SqLite: sql query using .af command

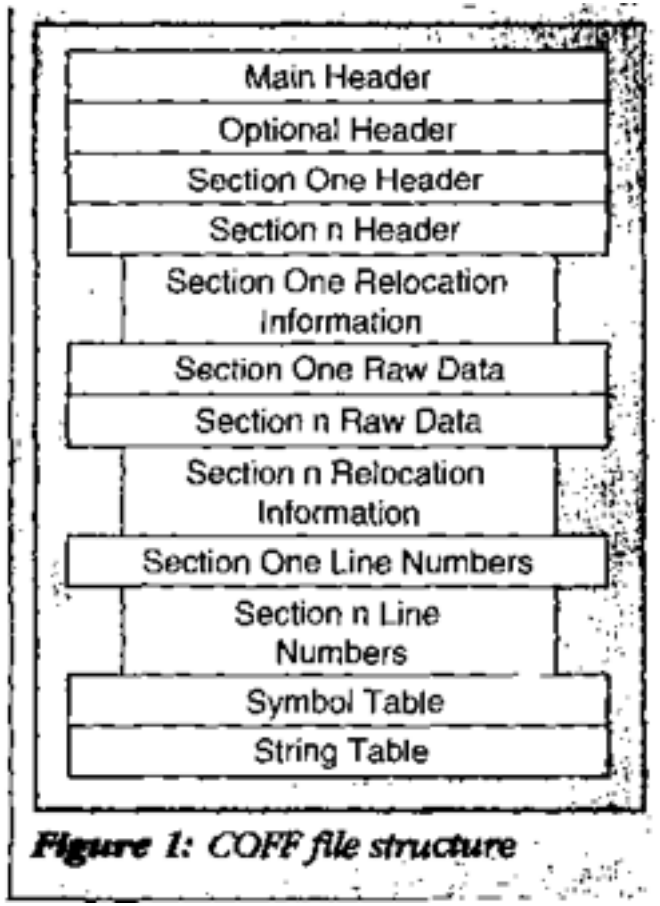
```
this.query() {
  SELECT rowid AS [~]
  FROM acl.user AS a
  WHERE afd("acl-afxDbAccountMatch", a.uid, a.login, a.info, a.cap, a.pw)
} { break; }
```

DEMO

SqLite: Integrating TSL

- sqlite “lib”
  - built w/JSON1
    - convinced a number of the poplar open source tools to add JSON1 to their nightlies
- afm built as a single “lib”
  - built from a single “cpp” file
  - Linked as “afm.exe” console subsystem coffbox
    - Renaming “afm.exe” a .SO or .DLL
    - Exporting standard DLL API interfaces
- AOS/S# COFF binary ffi-thunk bBuilder





SqLite: Packages “DBs as vfs Pkg”

- SqLite Header
  - SiteId, AppId
- Common DB extension forms
  - .afx, .afr, .dpk, .zdb [FsPath VFS model]

SqLite: Schema concepts - sqlite_master

- sqlite_master
  - VersionSchema table concepts
    - VersionSchemaTrigger table
- db-resident scripts
  - StoredProcedures
    - Where to place stored procedure code depends whether a vfs-repo-fs table model is available
- Undefined sql API hook (NFH)
  - Allows lazy NFH lookup into existing environment, or lazy load from db stored procedure tables

End of Slides

TSL Language

Aos Engine Family

Devops and the CoffBox Model

Year-2

David Simmons, Smallscript Corp

Tcl 2017 Conference

CoffBox: DevOps Conce

- Executable Components
  - afm.exe
  - *afm-symlinked-name*.exe
  - *executable-db-vfs-pkg*.afx
    - usually a ".afr" that's been symlinked
    - shebang-#!path or pathext-ext registered.
  - *script-name*.afts
- Registering script types
  - Windows
    - PATHEXT
    - ftype, assoc
    - registry twiddling for overloading
  - nix*
    - shebang - #!path header
    - chmod

CoffBox: Private Cloud

- My home office environment
  (by way of example)
  - Networks
    - 10GB Fiber
    - Mesh WiFi
    - ZWave, Ethernet-over-power
  - over
    - 70TB NAS storage
    - 20 computers
    - 200 devices
  - OS
    - Linux
    - OSX
    - Windows
    - Others

Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016

Consumer Applications to Represent 63 Percent of Total IoT Applications in 2017

Gartner, Inc. forecasts that 8.4 billion connected things will be in use worldwide in 2017, up 31 percent from 2016, and will reach 20.4 billion by 2020. Total spending on endpoints and services will reach almost $2 trillion in 2017.

CoffBox: Explore Desktc

- Show Files and Locations
- Demo App

CoffBox: DevOps Conce

- .af directory patterns
  - .conf customization model
    - search paths
- SymLinks
  - client system relative
    - symlinks, reparse-points
  - host system relative
    - junctions
  - absolute and relative paths
  - repo-treatment
    - ignored, as-file
    - xattr archive-flag
- Virtual Drives
  - .dmg, .vhdx, linux sparse file images

- Packaging Model
  - .af directory pattern
  - .afws
  - .afr, .afx
  - .afts scripts

CoffBox: Build a script

- script w/command line arg processing

Build an afx w/.conf checked into it. Showcase demo webserver

CoffBox: DevOps Conce

Customizing a System

- ChildProcess
  - pipes
    - http(s), message-queues
  - sockets
    - http(s)
- EventLog mechanisms
  - telemetry reporting

Reflecting on a System

- ::system::info
  - elevated, admin, etc
- reg- commands

CoffBox: DevOps Conce

- Binary build model
  - cpp, hxx, hpp, h
- Single Binary for Exec & DLL (.so)
  - Coff Format
  - Subsystems
  - Sections
    - Resources
    - Read-only Memory Repos
- TSL scripts exported as native COFF library binaries
  (.so, .dll, .exe)
  - your-code-thunk.dll - thunk-bridges

End of Slides

TSL Language

Aos Engine Family

afm

Year-2

David Simmons, Smallscript Corp

Tcl 2017 Conference

iOT: Purpose Built Constrained Environments

Constraints

- Maybe single app
- Security
- Storage types
- Hw permission and firmware controls
- Memory
- Power Management Lifecycle
  - CPU Cycle Cost
- RTOS Constraints and Models
  - Embedded toolset contraints

Typical metrics

- Memory
- Cpu
- Storage
- Power Management
- Os Services

iOT: engine requirements for iOT deployment

- Os Platforms
  - Windows, OSX, iOS, Android, Linux, Linux Embedded, FreeRTOS, nuttx, nucleus rtos
- ***see*** https://www.osrtos.com/

- Cpu Targets
  - Arm (32, 64), Intel x86/x64, MIPs

iOT: Device Use Case

- Networking
  - Wifi, Bluetooth, Wired-Ethernet, ZWave, Zigbee
- Web Server/Service
  - Telemetry
  - Updates
  - Device Interop (peer, slave, master)
- UX
  - custom displays
  - custom input/sensors
- VFS
  - Packaging, Versioning
  - Resources, executable components
- Power Management Lifecycle
  - Device On/Off
- Device Firmware Update Models
  - Rollback, Upgrade
  - Build, Package, Sign requirements
  - Flashing Lifecycle Constraints
    - OTA, Wired

iOT: Afm's iOT ready architecture

- Coff binary build model
  - cpp, hxx, hpp, h
  - embedded r/o ".afr"
    - compression
- Codecs
  - compression
- Engine Design
  - Booting Model
  - Thread Model
    - engine-affinity, fiber-co-operative, rpc message streams
      - pipes, tcp, ssl, http
      - json, html, …

- Pal: Host Abstraction
  - Memory – cpu/kernel direct
    - TLS – CPU direct (mirror OS model)
  - Interrupts, Signals, Exception
  - Debugger Aware Channels
- Pal: Cpu Abstraction
  - FPU, Vector
  - MP-Sync Instructions
  - Bit Operations
  - Special Forms
- Pal: FsPath Pluggable Abstraction
  - Native File System, *(tags, versioning)* Fs built on SqLite blobs, Registry, Fuse/Dokan, HTTP(S), BuiltIns
- Communications
  - Transports
    - Ethernet, Bluetooth, Serial
  - Sockets
  - DNS, DHCP, HTTP (1.0, 1.1, 2.0 ALPN)
  - SSL, Certs *(pal and host integrated)*

End of Slides

TSL Language

Aos Engine Family

Language Concepts and Usage

Year-2

David Simmons, Smallscript Corp

Tcl 2017 Conference

Statements: JS & TCL 12 Rules Disambiguation

Engine: Model

Procs and Threads

- Process Model
  - multiple engines allowed per process
- Engines
  - engines have thread affinity
  - co-operative threading within an engine
- Namespaces
  - ::super
  - ::system
    - ::afm
  - ::shell
  - ::app
    - ::script

Code Locations

- FsPath Concepts
  - SymLinks
    - Archive XAttr-Flag
- Process Loading Model
- Script Binding Model
- Web Site and Page Model

Statements: Statement Tokenization and Substitution

- Phase 0: Text Command Statement
  - Statement Delimiting
    - **else** rules *[command must defer-level1]*
    - ; ***eos***
    - {} ***eos***
    - # ***eos***
    - 'ws' ***eos*** *[tcl-mode only]*
- Phase 1: Command
  - Phase 1a: Expr Args
  - Phase 1b: semicolon args
- Phase 2: Binary Operators
- Phase 3: TCL Parameters

- Comment Forms
  - // EOL
  - /* nestable */
  - # - special command, NFH form
    - Allows #command if NO whitespace
- Word Grouping Rules
  - () first following command
    - allows whitespace
  - () {} … no other tokens
    - partitions into discrete words whether whitespace or not
- Deferral Levels 1,2,3
  - =, ? … :
  - for() …
  - do while {}
  - return

Commands: Forms

- Types
  - proc
    - ::super namespace
    - protected from re-definition
  - func (fn)
    - context bound
  - method
    - this, prototype bound
- Mint-Paths
  - All commands are namespace types and get minted
- Modes
  - default
  - ^ uplevel & native proc^
  - @ modules
- Prototypes
- Tson Declaration/Merge &:{}
- Declaration Features
  - parameter binding
    - binding (*a, &b, ?z, {x {}})
- Invoke named parameters
  - (key: value,)
- Observation and NFH
  - Loader Hooks, NFH

Eval: Expressions

- Operators
  - new …
  - func …
  - = (op-assign)
  - numeric ops
    - .qualifiers for typing
  - ==
    - ===, ==~, !=, !==, !=~
  - Short Circuit Boolean Ops
    - ||, &&

- Contexts where it is implicit eval
  - pathx invokes ()
  - pathx indices []
  - tcs-line
    - text command statement

Path Expressions: Operators

- PathX
  - $$,$?,$!,$:
  - &:, &?
  - delete &
  - && upvar
  - (expr), {}, [expr]
- “”, {}, [] and tcp
  - token/macro rules

- PathX Operators *(incl reserved)*
  - . operator
    - .*() invoke
    - .. cascade
  - :: and ! binding operators

Path-Expressions: VarRefs

- Variable References
- & Operator
  - Stack-Context References
    - &(#)
  - &pathx – deferred binding
  - [expr] => {scalar-key}
  - && Upvar model
- Name Partitioning
  - …
  - this
  - super
  - @key, key@qualifier
  - {literal-key-closing}
    - escape rules
- Valid Name Patterns

Namespaces:

Foundation

- Namespaces
  - ::super
  - ::system
    - ::afm
  - ::shell
  - ::app
    - ::script
  - ::global
    - modules

Concepts

- Minting
  - namespace command
  - mint-name concepts
  - minting rules
- Where Procs Live
- Where System information is
- User owned
- Names
  - CamelCase
  - Hyphenated

Commands: Categories - patterns

- Naming Patterns
  - Case Usage
    - Uppercase – factory *(new)*
    - Lowercase - variable
  - CamelCase - methods
  - Hyphenated - functions
  - Special commentary
    - @names
    - this-… names
    - super
- Locations
  - Modules
    - ::super proc space
    - ::app, ::lib
    - reserved
      - ::globals – for local composers
      - ::afm, ::system, ::shell
      - ::app, ::script
- Module Naming pattern
  - domain – like package model
  - uuids – anon modularization

Commands: Categories - groupings

- file-, dir-, fs-
- string-
- mint-
- sql-
- time-
- ?put*
- @*

Commands: include, require, fs-find

Key variables

- ::system::path
- ::app::path
- ::shell::path
- ::shell::ext
- ::shell::jails

Extensions

- -ncf
- -aei

End of Slides

What we are going to see this morning

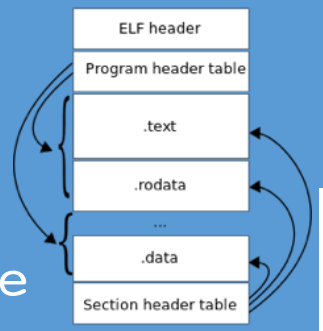
Aos Engine Family
25th Anniversary

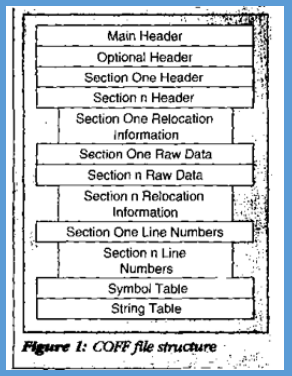
TSL

Text Scripting Language

AFM

Design and demos





Rich Web Servers	DevOps Toolset	Repo VFS CMS	Integrated SqLite	Single Binary
Open Ssl, Certs, Json, Markdown, HTML, TCP, Sockets	CLI, Ansi, EventLog, Services, Pipes, Exec, Codecs	acl-security, file-system, registry, env, time-codec, uuids	1st class Sql support JSON, Sql Procs	Config Free SymLinks

TSL Language

Aos Engine Family

afm

Year-2

David Simmons, Smallscript Corp

Tcl 2017 Conference

Refresher, demos, discussion

For those I did not meet last year, my name is **David Simmons**

- A year has gone by…
  - This morning we are going to see how TSL/AFM has evolved.
- The beginning of this talk will be a refresher
- Then we will go into a series of demos and discussions

Thoughts to share…

Over the last year, the **afm** system has been actively serving 10,000+/engin PBX cloud telephony users

- Single-binary, single db pkg-file
  - coff-box symlink model, single db-file package system (vfs)
  - drop install on any machine.
- Setting up new servers is only a few minute operation
  - fail-over, scaling, and versioned upgrades are mostly automated by simple remote check-in/out.
- Full server and devops set of stabilized features
  - More than is possible to cover in a talk.

On the subject of Names

- What is the acronym “st”?
  - Site, Smalltalk, Smallscript
- What is the acronym meaning of “AOS®”?
  - Agents Object System, Agile Object System, Actor Object System
- What is the acronym root “af” of “afm” stand for?
  - collates top of the alphabet ☺
  - **A**os-**F**amily, **A**os **F**ile Manager, **A**os **F**ossil Manager
- But in all seriousness:
  - Carefully thought through names and naming conventions are critical to a language design and best practices for code libraries.

My language implementation timelir

1978-1997

- NBS Net/ArpaNet HW w/**Basic** (6512)
  - Forth, APL, Mix of Shell Languages, many others of the era
- **Fortran**, Lisp, PostScript, …
- Message-C, SIAL, Informix SQL's Full Text engine for International Publishing on BRS/Dialog with TeX
- QKS Smalltalk, Prolog, Scheme,…
  - MacOS AOS 1991/1992-1998
    - Embedded in QuickTime
  - Windows 1994-1998
- Newton, AppleScript, Kaleida/SK8, Taligent

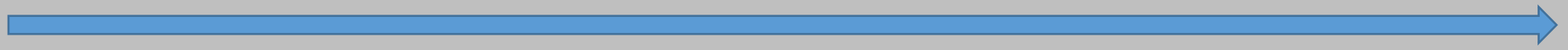
1997-2017 (Owner/MSFT Architect)

- Smallscript, S# Language
  - Native engine, .NET Runtime DLR 2000 PDC
- VBA, Visual Basic.NET
- **JavaScript**
- .NET Mobile Runtime and XNA
- PowerShell
- AFM/TSL

20MHz Mac Smalltalk

4GHz 12-core PC

1.5+GHz 4-core Arm



Public Cloud VMs

$8 Cellular Wristwatch

iOT, Edge, Private Clouds

Mesh Networks, 10GB fiber NAS

DEMO

1997

Flashback to the good old days

History: Smalltalk's failure to be relevant

1986-1996 *(gave life to the era of Java)*

- Awesomely productive and rich toolset capable of talking to almost anything
- Immense integrated frameworks
- Pioneered many aspects of modern software development from patterns, agile programming, unit testing
- Disaster for working with text and file-based code management and practices
- Monolithic image system unable to integrate within other systems, execution engine scaling issues
- Decompositional Model challenge to schema version, package, and be small and bootstrap from nothing

2007-2017 *(Rise of the mobile iOT Device Net)*

- Everything connected, people message, and watch/create content on their schedule
- Compute is cheap $8/full watch cpu competitive with 2004 PC
- Embedded, real-time, devices proliferate and with it immense scaling challenges
- HTTP, JSON, HTML-UI dominate compute systems
- Text processing and file-based assets dominate compute design with massive stack-based libraries
- devops challenges and complexity single largest impediment to evolution

Back to the present

2017

Why Text matters more than ever

FsPath system: virtual versioned file system

- Let's begin by talking about the FsPath system
  - pluggable and defaults to supporting:
    - Disk
      - fully symlink, xattr and stream aware
    - Versioned repo (scm)
      - System core files baked into the afm coff binary as a r/o repo
    - Web based files
  - has a specific path syntax for disambiguating content
    - server-type
      - file-system (default, current directory)
        - streams/xattr support
      - repo vfs
        - branches, tags, versions support
      - af web-server generalized

Fuse
Dokan
Registry

include command: .tsl; .afts; .md; …

include
require
fs-find

/~:alias/path/file.ext path aliases are located in ::app::path ^ ::system::path ma

①

File folders
on disk

/path/file.ext
N:/path/file.ext
N:/path/dir:xattr-
stream.ext

②

Branch folders
in local repo
(versioned)

/:branch:repo/file.ext

.afr;
.afx;
.afws

Branch folders
in r/o coff repo

~:/path/file.ext

afm.exe
afx.exe
afts.exe

Branch folders
in https repo
(versioned)

https://af.st/:branch:repo/

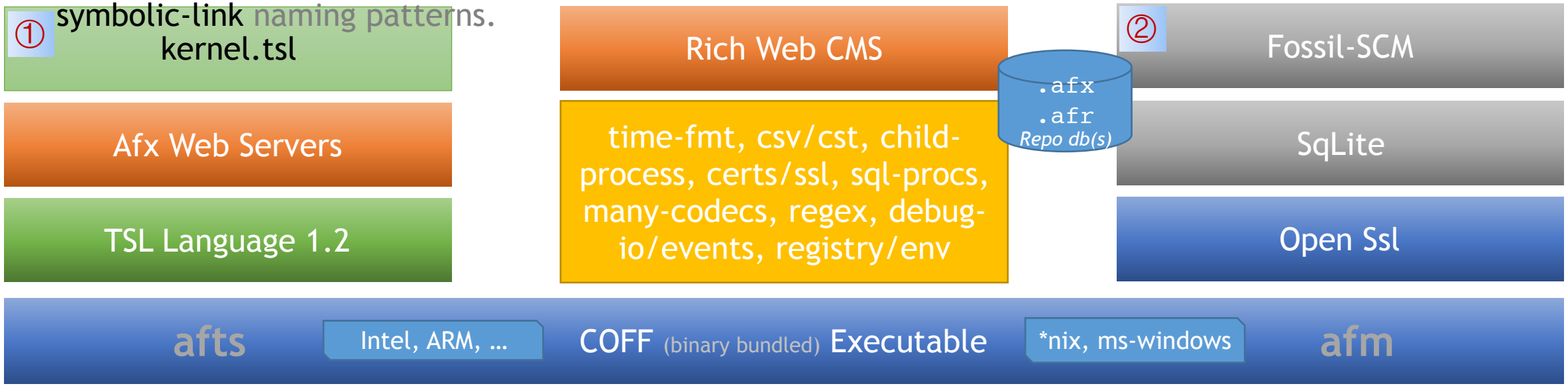
./.af/site/
./.af/sites/domain

End of Slides

AFM executable

*Self-installing single host executable that uses **symbolic-link** file-naming to determine its execution behavior*

The executable's **File Name** is used by kernel**.tsl** code to determine the execution path as it examines the **exec** *command line*; this allows efficient creation of **single shared binary** tools using symbolic-link naming patterns.



afts.exe 4.05 MB Application (SymLink) Yesterday 9:44 PM -a---- I386, Unsigned, Product Version: 1.35.8.1016, File Version: 1.35.8.1016 (Target: N:\local\bin\afr.exe [afr.exe])

afm.exe 4.05 MB Application Yesterday 11:01 PM -a---- I386, Unsigned, Product Version: 1.35.8.1016, File Version: 1.35.8.1016

Variable Path Expressions

JSON Expressions and TCL quirks mode support

```
$    &      ::   identifier .    (…)
     &&     ^      {…}    ::
            .      […]
```

Examples:

```
$![putl "Hello"][1]          $!{string "Hello"}†    $!(17 + 5 % 3)[3]
```

$::*identifier-expr* $^*identifier-expr* $*identifier-expr*

$*identifier*.{*literal-key*}.[*expr-key*] $a.b[3] $a::b[3] $a.b::[3] $a{b}[3]["z"]

$*receiver-path-expr*::*namespace-expr*(*?invoke-expr-param?, …*)

Cases of Note:

`&:{` *… JSON pojo …* `}` — **JSON** pojo supporting TSL **`eval`** *expr* extensions and auto-conversion.

`&:[` *… JSON poja …* `]` — **JSON** poja supporting TSL **`eval`** *expr* extensions.

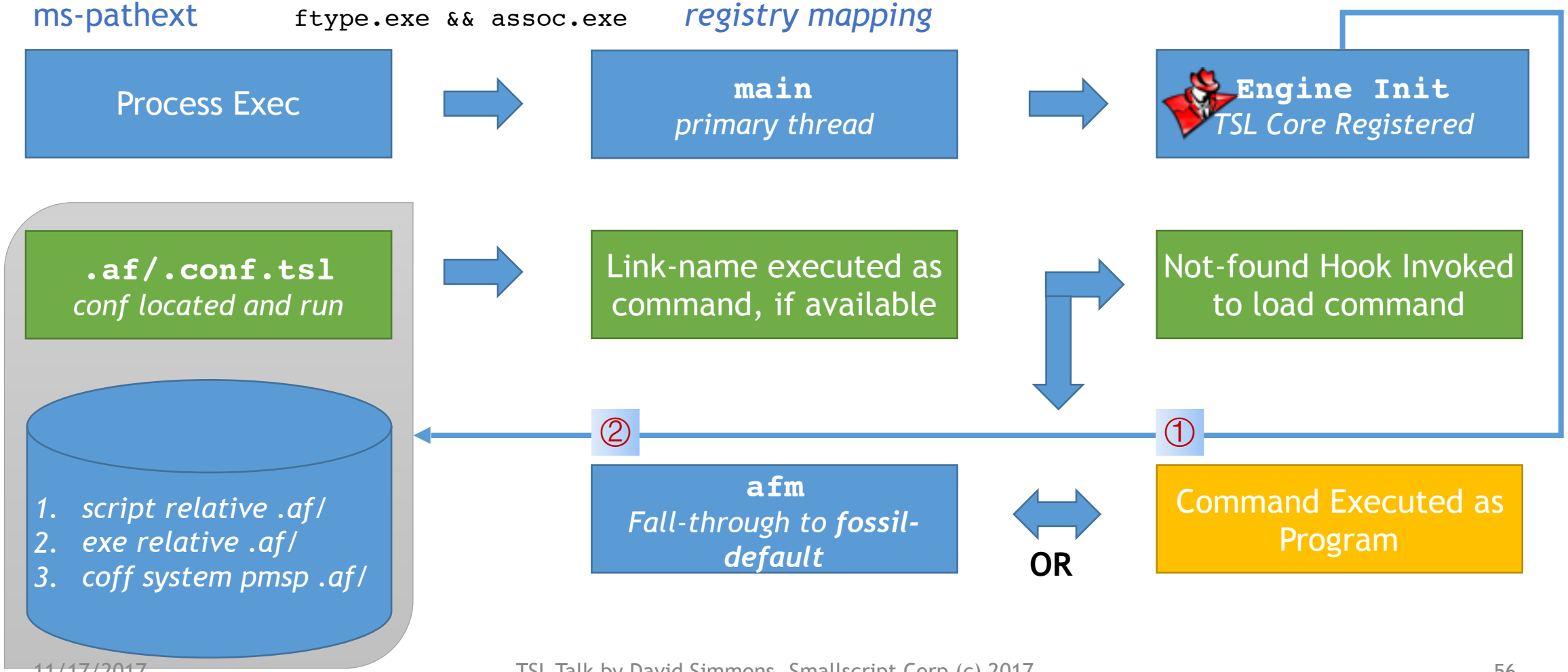
`var` *identifier-path* `=` `&&`*var-ref-path*`;` **Binds two variables**, as opposed to assignment.

Startup sequence: she`bang-script binding

shebang
ms-pathext

```
#! /local/bin/afts
ftype.exe && assoc.exe
```

registry mapping



Command *re-examined as* Message


```
cputl $msg;                    # A TCL command with macro-param

cputl(msg)                     # A TSL function with expr-param

this.cputl(msg);               # A TSL method with an expr-param
```

TSL Language

Aos Engine Family

afm

David Simmons, Smallscript Corp

Tcl/Tk 2016 Conference

What we are going to briefly tour this afternoon..

Aos Engine Family

TSL

Text Scripting Language

AFM

Its COFF binary executable host

DEMO

`afts`

Command Line Script

TSL

DSL:
Domain Specific
Language

*As a language TSL is founded on the **macro** and **command** patterns used in TCL*

As C++ can be described in relation to C,
TSL can be described in relation to TCL

It would be incorrect to describe TSL as a dialect of TCL

TCL
Macro & Command
Patterns

JavaScript *blended*
syntax & *intrinsic*
JSON

Smalltalk, Lisp, Self
messaging behavior
and

Intrinsic C++
object integration

Executable Libraries: *Source, Op-Codes, and precompiled Machine Code*

Frameworks, IDE,
Image-Snapshots,
Intrinsic-Repos

Object Model:
prototype, mixins

Path Binding
Engine:
predicate binder

Execution
Invocation: *Interp-*
JIT, FFI

Host Integration
& Lifecycle Model

Perspective Types:
Versioning, Modularization,
Security

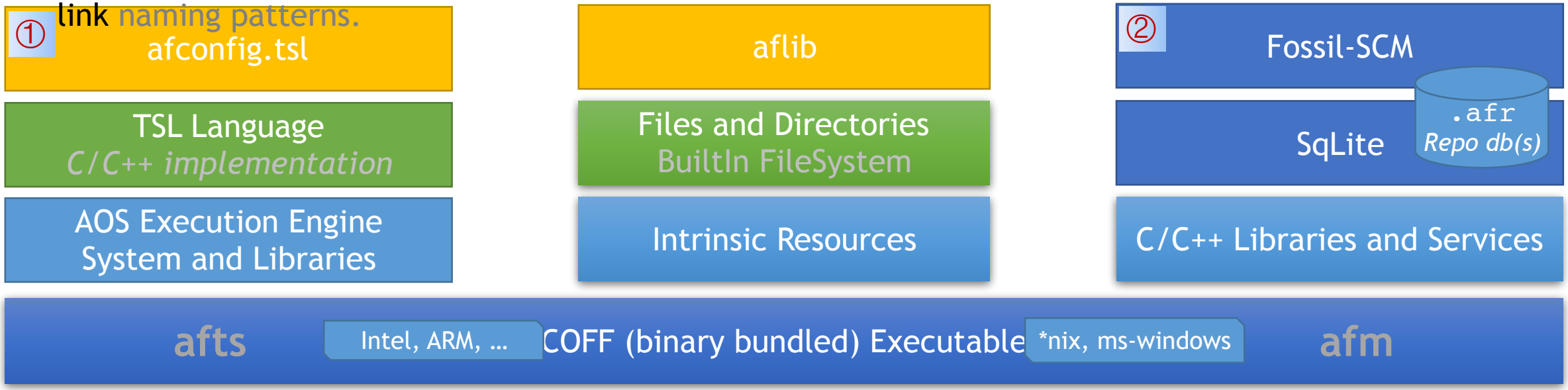
AOS Execution Engine and Container System

Dynamic Composition
Aspect-weaving and introspection

AFM executable

*Self-installing single host executable that uses **symbolic-link** file-naming to determine its execution behavior*

The executable's **File Name** is used by **afconfig** code to determine the execution path as it examines the **exec** *command line*; this allows efficient creation of **single shared binary** tools using symbolic-link naming patterns.



Brief Intro, then on to demos and tour

Hello, my name is **David Simmons**

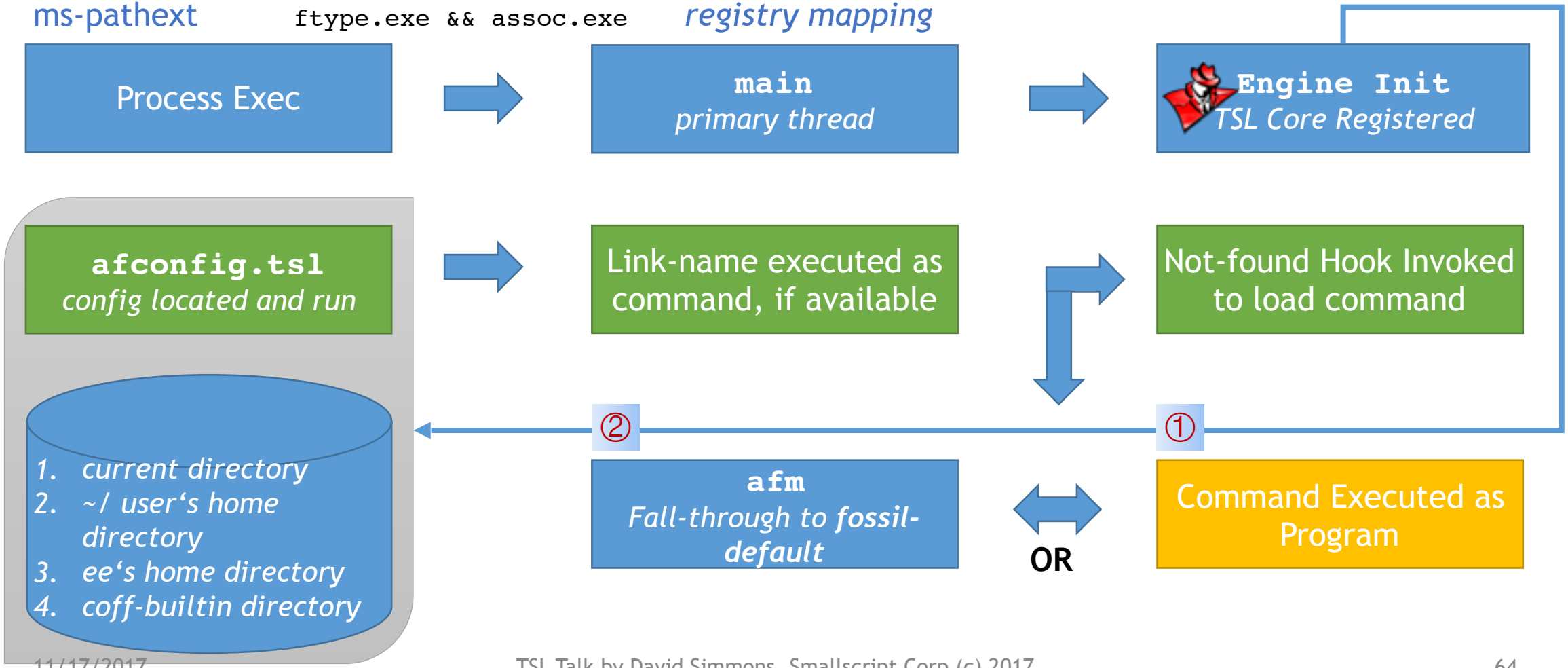
- For 25 years I've had a particular specialization in high-performance hybrid dynamic-static language runtime systems.
- By education, I am an Electrical Engineer and Astrophysist specialized in VLSI design. However, most of my career has been as a Software Engineer and Architect
  - Working on Operating Systems, Hardware Devices, Real Time Systems and Programming Languages and their Execution Machinery and Infrastructure
    - I began my career some 40 years ago, in the summer of 1976 at The National Bureau of Standards which is known today as NIST (The National Institute of Science and Technology) working on Fortran Runtime Real Time Libraries, Perkin Elmer Operating System Development, and ArpaNet NBS Net Packet Switch Hardware and Analyzers.
- Employed working with *or* for:
  - NBS/NIST, U.S. Congress, Air Force, Faculty at UofMd NSF Systems Research Center, Tokyo Gas, Apple, Suse Linux, Microsoft and worked for or owned a number of small to mid-size companies of up to 250 employees.
- Worked on, collaborated on or was lead architect on the following language systems:
  - Apple's Kaleida, Apple's AppleScript, Apple's Quicktime-Track-Scripting, Script on Newtons
  - Basic, Forth, Message-C, SIAL (1990 ODBC equivalent with full-text search engine), Smalltalk and Prolog, S#
  - Microsoft Visual Basic, Javascript/JScript, .NET Runtime (desktop, mobile), XNA, Powershell

Startup sequence

Script binding:

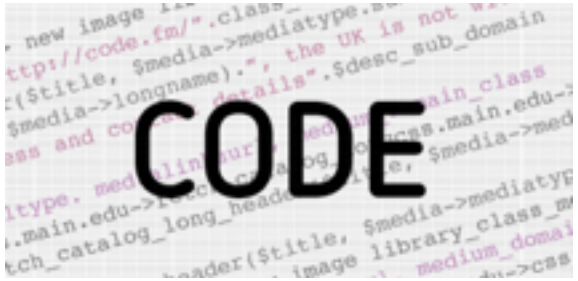
shebang `#! /local/bin/afts`

ms-pathext `ftype.exe && assoc.exe` *registry mapping*



Language

Statements, Expressions and other Concepts



TCL

- Statement
  - Composed of Words
    - First Word is **Command**
    - Rest is *Command* Macro **Params**
- Expressions
- Things
  - Scalars
  - Key-Value Containers

TSL

- Statement
  - Composed of First Word and Rest
  - First Word is **Message**
    - Command, Function or Method
  - Rest is Message DSL **Params**
    - Literal, Macro, Eval
- Things
  - Scalars
  - Mixin Prototype Dynamic Objects

Operators and Keywords

Syntax, Semantics and DSL patterns

Operator	Language	Type	Patterns
{…}	tcl	*Literal*	text patterns.
"…" '…' `…`	tcl+	*Macro* for "[…]".	prefix-names. $(…), $[…], ${…} forms. (…) invocation. Quirks
[…]	tcl†	*Command* patterns.	**proc**, **func**, **method** patterns. Unary and binary **symbol method**
(…)	tsl	Expr *(eval)*	implicit **eval** pattern.
$	tcl+	MX *path* Operator tokens.	macro-expand operator for variables and other
&	tsl	Ref *path* Operator expressions.	reference operator for variables and path
\	tcl+	Escape Operator	enables UTF8, ascii and inline-operator escapes.
;	tcl †	End-of-statement	TCL quirks mode for newline and curly-brace EOS.
,	tsl	End-of-expression	Expr **eval** param delimiter.
# /* */ //	tcl+ †	Comments TCL #.	JavaScript/C++ compatible comments. Quirks for

Keywords		Description
super *this* †	tsl	inheritance operator and contextual variable for accessing inheritance

Command *re-examined as* Message


```
putl $msg;              # A TCL command with macro-param

putl(msg)                  # A TSL function with expr-

param

stream putl(msg);       # A TSL method with an expr-paran

stream.putl() $msg;     # A TSL method with a macro-paran
```

Variables *re-examined*

Declaration

```
set varName valueParamWord;          # TCL assignment form
set varName;                         # TCL get command

var varRef = eval-expr;              # TSL assignment form
var varRef;                          # TSL decl command form
```

Variable Reference is a **path**

Path roots are:

Global, or in Stack Frame Context down-level of global-root-frame.

Variable Path Expressions

JSON Expressions and TCL quirks mode support

```
$   &     ::  identifier.   (…)
    &&    ^     {…}   ::
          .     […]
```

Examples:

```
$[putl "Hello"][1]          ${putl "Hello"} †          $(17 + 5 % 3)[3]
$::identifier-expr          $^identifier-expr          $identifier-expr
$identifier.{literal-key}.[expr-key]  $a.b[3]  $a::b[3]  $a.b::[3]   $a{b}[3]["z"]
$receiver-path-expr::namespace-expr(?invoke-expr-param?, …)
```

Cases of Note:

`&{ … JSON pojo … }` — **JSON** pojo supporting TSL **`eval`** *expr* extensions and auto-conversion.

`&[ … JSON poja … ]` — **JSON** poja supporting TSL **`eval`** *expr* extensions.

`var identifier-path = &&var-ref-path;` — **Binds two variables**, as opposed to assignment.

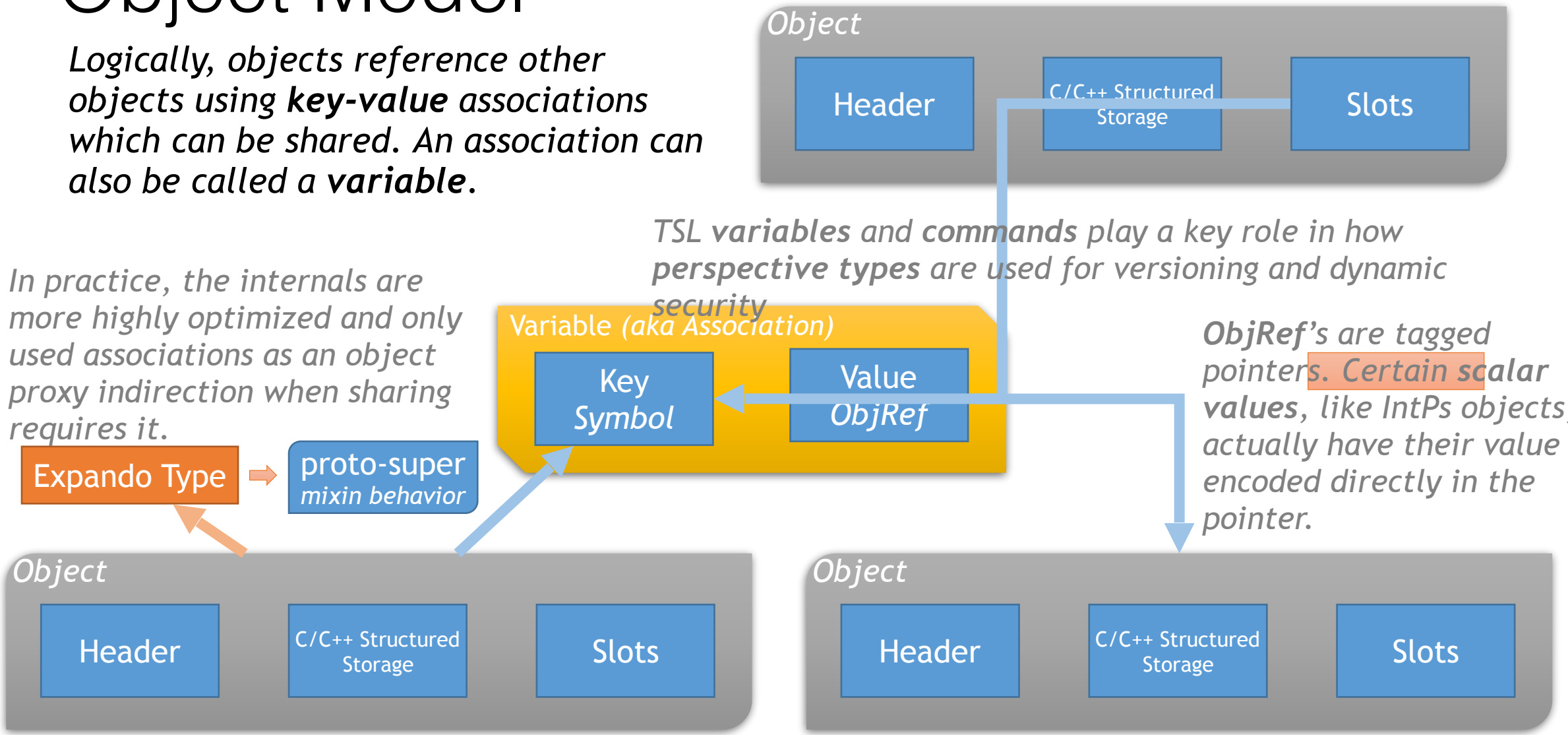
DEMO

`var sharing`

Script Example

Object Model

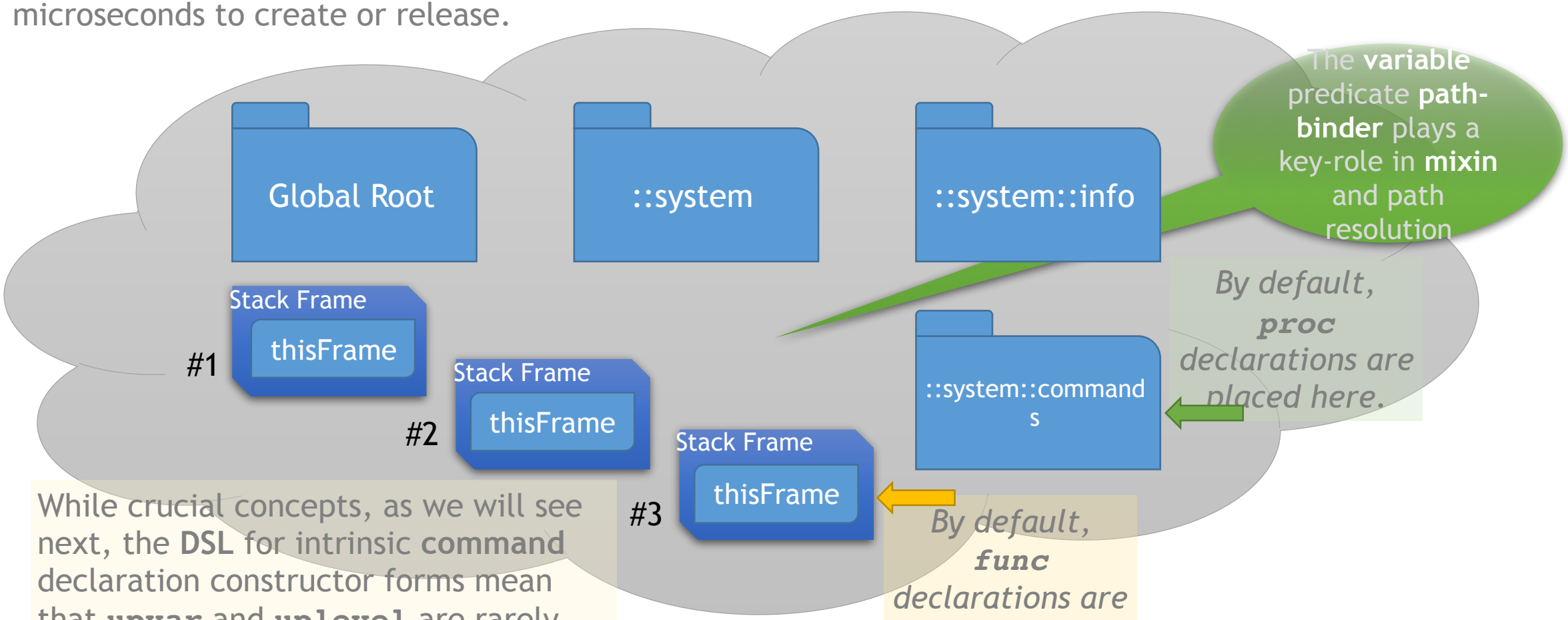
*Logically, objects reference other objects using **key-value** associations which can be shared. An association can also be called a **variable**.*



Object Space

One per engine created within a process.

There can be an unlimited number of engines within a process. Each engine takes microseconds to create or release.



While crucial concepts, as we will see next, the **DSL** for intrinsic **command** declaration constructor forms mean that **`upvar`** and **`uplevel`** are rarely needed.

JSON

Given a uniform internal object model, all objects can be converted to or from JSON via serialization.

*The original technology name for this in AOS 1992 was PIPOs. **Platform Independent Portable Objects**, which contained TOCs, DBs, versioning, schema-migration features and supported cross-machine migration of threads and UX components with automatic re-wiring.*

*When TSL networking protocols for HTTP/HTTPS with JSON are combined with **SQL TRIGGERS** in **SqLite** using **JSON1**, powerful robust systems can be easily built.*

Object

```
var j = &{
    "id": 1,
    "name": "A green door",
    "price": 12.50,
    "tags": ["home", "green"]
}

json-out(j);
```

```
{
  "id": 1,
  "name": "A green door",
  "price": 12.50,
  "tags": [
    "home",
    "green"
  ]
}
```

Object

```
var zJson = {
    "id": 1,
    "name": "A green door",
    "price": 12.50,
    "tags": ["home", "green"]
}

json-out(json-in(zJson));
```

Commands are objects

```
1  // Discrete thisFrame context
2  proc myCmd1(param, ...) { }
3  // ^ Implicit uplevel thisFrame context
4  proc^ myCmd2(param, ...) { }
5  // lambda
6  var lambda = func (...) { }
7  // Declaring a subtype
8  func localCommand(...) &{
9    // super-type [commands are objects - they have supers too],
10   super: lambda,
11   // inherited instance methods
12   prototype: {
13     foo: func () { },
14   },
15   // @factory initializer
16   @body: {
17     putl "Hello";
18 }}
19 // Creating an instance
20 var inst = new localCommand();
```

Scripts are *lambda* **commands**

- **Modules** are *commands* with additional metadata.
- **Source** has ***provenance***, which plays a role in perspective-type binding of ***versions*** and access ***security***.
- Since *objects* are *pathable* and *commands* are global or contextual *objects*, **commands** are commonly used as **namespaces**.

QUESTIONS

More demos

offline and BOF