Modern APL in real life

LambdaConf 2020 (1 Anno Coronaviri)

Aaron W. Hsu arcfide@sacrideo.us

Why?

Theory and Practice

The coding gap

Beautiful code

Novice code

Production/Legacy

Production/Legacy

Modern APL moving forwards

Many styles/approaches

Missing best practices guidelines

Discipline benefits Liberty

Too many options,
not enough role models

The 3 religious structures

Desert Englightenment

Monastery

Public Churches

APL involves Change

What can stay the same?

Recommendations

	Concrete	**Abstract**
Technical	Tooling	Architecture
Social	People	Methods

How do you begin?

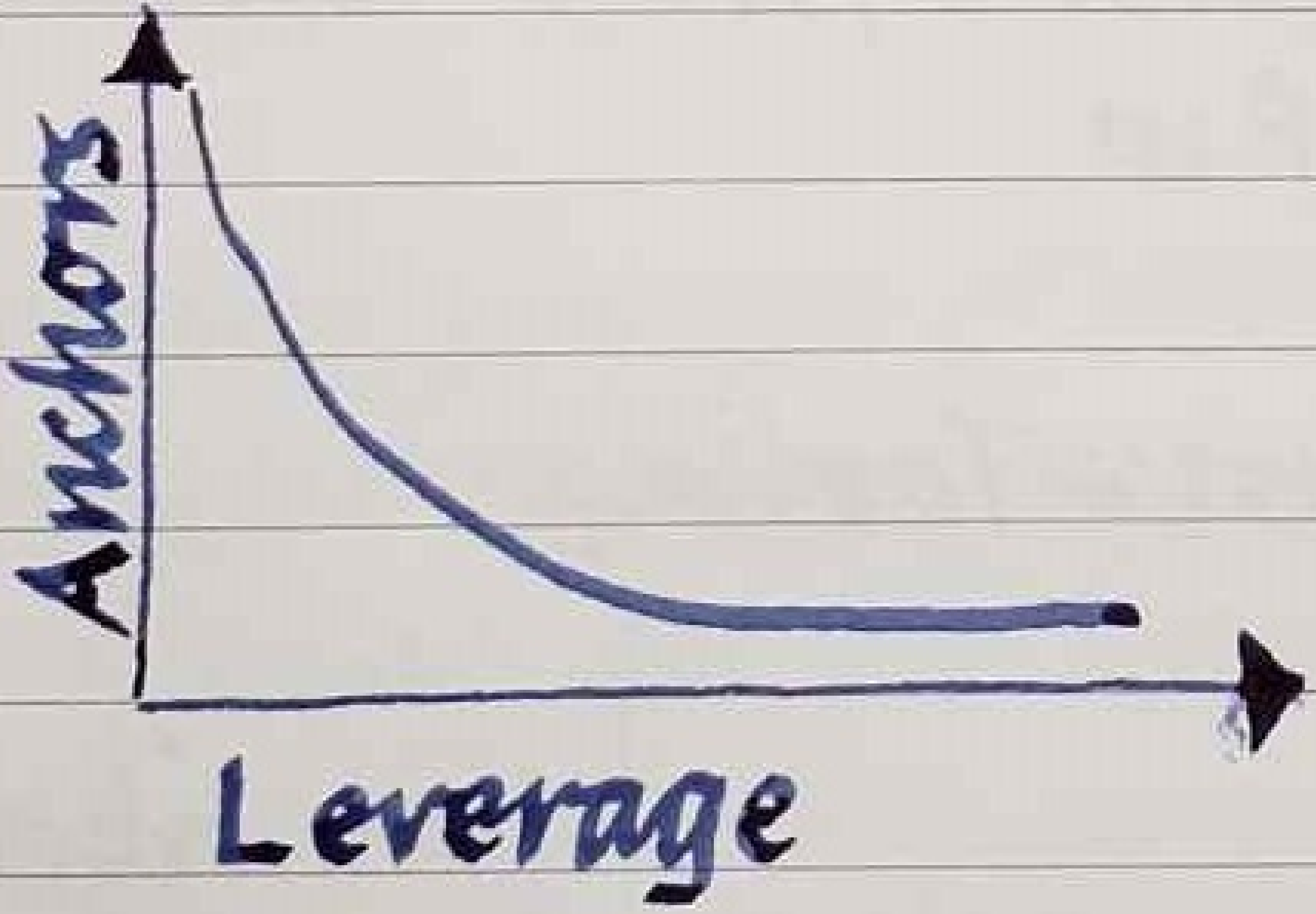
Social×Concrete→People

Psychology of APL

Expectation is Everything

APL Leverage Spectrum

Expected Scaffolding


Anchors
Leverage

High cost of removing boilerplate

Shift your expectations

Designing vs. Coding

Recognize different affordances

Type thinking

Mathematical Domains

Write literature, not tax code

Affordances:
C. S. Lewis vs. IRS

Linear vs. Non-linear

Emergent vs. Referent

Complex Systems Analysis

Intentional, Incremental

Time and Patience

Time and Patience

Targeted Learning

Primitives, Domains

Booleans,
Number Systems

Information as Arrays

Primitives!

Increase your Working Set of Idioms

Read many different programs

Write, a lot

Control flow as Data

This is similar to the “code is data” Lisp insight

Read Idiom Libraries

Read the Classics

ACM Quote-Quad

Vector Journal

Example

I want all of the blue disks from a set but only if there are any blue disks, and if there are no blue disks in the set, then I just want all the disks anyways.

```
m∨~∨/m←p⍵
```

“Some or everything” mask idiom

Technical×Concrete →Tooling

Awareness

Built-in

- Namespaces
- ⎕JSON,⎕CSV,⎕XML
- Object System
- Debugger
- RIDE
- Charts and Chart Wizard
- Office Integration
- RDBMS/SQAPL
- Jarvis, SockPuppet, MiServer
- Python, R, .NET Core Bridges
- FFI/DWA
- Dfns library
- Co-dfns, Isolates
- Conga
- Component File Server
- Date/Time Systems

Built-in

- PCRE Regular Expressions
- Serialization
- Compression
- Win32, XAML, Windows GUI
- SAWS (SOAP, WSDL)
- Reports, PDF, SharpLeaf
- APLMON, Profile
- Math library
- Vector DB
- Util library
- Code libraries (examples)
- Exception Handling
- Triggers
- Versioning, Distribution
- Syntax Coloring API
- Link

Built-in/Available

- Comp, Native, MMAP files
- Green threading
- Lexical and Dynamic Scope
- Structured Statements
- `⎕DMX`
- Randomness, Search, Hash
- TamStat statistical suite
- Code sharing

- Primitive Parallelism
- Haven + APLCart
- Dyalog GitHub
- APLTree
- Jupyter Notebooks
- APLUnit
- Mystika
- *Idiom Libraries*

Maximize the debugger

Options, but stay clean

Tooling for analysis

Tooling at the boundaries

Mixing and nesting are not your friends

Stuff you think is missing

But it’s probably not

Data Persistence

ML Libraries

Data Analytics

Key-Value and other NoSQL systems

Computational, Traditional Algorithm Libraries

Technical×Abstract →Architecture

Architecture Spectrum

Minimal Architecture

Requires clear, shared expectations

Disciplined habits vs. Explicit Architecture

Social methods, spirit over inflexible architecture

Explicit architecture is a form of boilerplate

Implicit architecture is the bedrock of good design

Integrating APL

On machine

Full stack in APL

- Interactive APL Session
- Win32 GUI "Stand Alone Executable"
- Cross-Platform GUI (Desktop AND Web)

Embedded APL

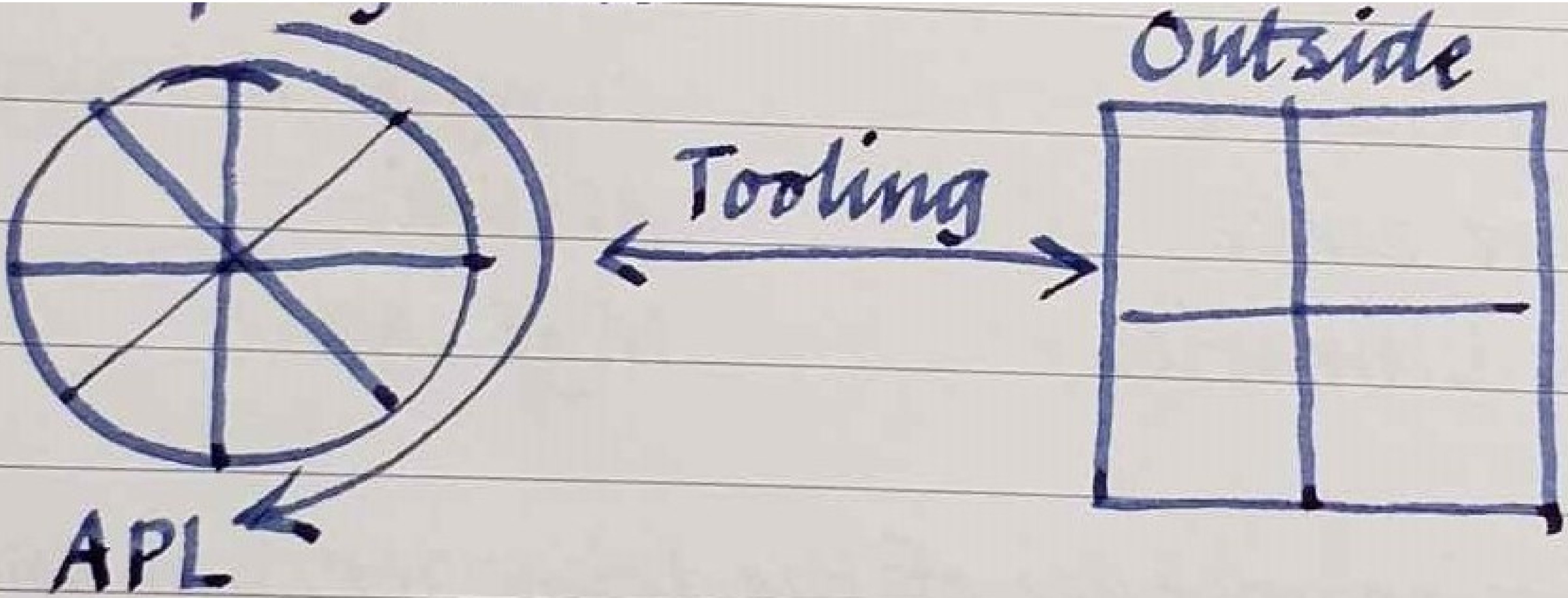
- Shared Library (called from Python)
- Microsoft.NET Assembly (called from PowerShell)

Services

- Web Service in Docker Container
- Serverless Lambda (Amazon Web Services)

- See also: Jarvis, SockPuppet, SAWS, SQAPL

How I organize code


Outside
Tooling
APL

Slice @ domain, interface

Architecture is Hard Abstraction

Avoid internal architecture

Domain, Ground-level

Missing Link: Front-end

Social×Abstract→Method

Spectrum, again.

Method Gap

Agile

APL supports more extreme approaches

Some practices mitigate language limitations

Maximize APL as shared notation

Lean into shared knowledge practices

Lean into intra-system transfer

Mandate learning, refactoring as explicit objectives

Competency is your limiting factor

Refactor and revision: essential to learning

Factor in the time

Do not underestimate time to fluency

Traditional Teams

Specification Resolution

More frequent refactors

Discovery process

APL as a Specification Language

Delineate the APL space

Leverage your advantage:

Upfront Design

What do I recommend?

Just start with Extreme Programming

Well suited to Modern APL

APL mitigates XP’s drawbacks

Extreme Personal Habits

Biohacking clean code

Personality

High Orderliness, low Industriousness, high trait Openness

Maximize strengths

Psychologically incentivize clean code

Notepad

Visual Limits

No unit testing

Minimal control flow

Linear, data flow

Minimal nesting

Almost no comments

Global name space

Minimal filesystem structure

Naming conventions

Short globals

Few, longer locals

Minimal filenames

Or, well, none?

Version control for notes

Not comments!

Version control for notes

Not comments!

Open, transparent structures

I often use leaky, ephemeral abstractions

Make complexity painful

Style:
Odious anomalies

Do not hide bad code

Force refactoring

Not orderly?

Leverage Industriousness

Low Openness?

Idioms, style, rules

Clarity, Macro Thinking

Don’t embed fear in your code

Tools should motivate the mind, not subjugate it.

Maximize simplifying affordances