μ-service Architectures

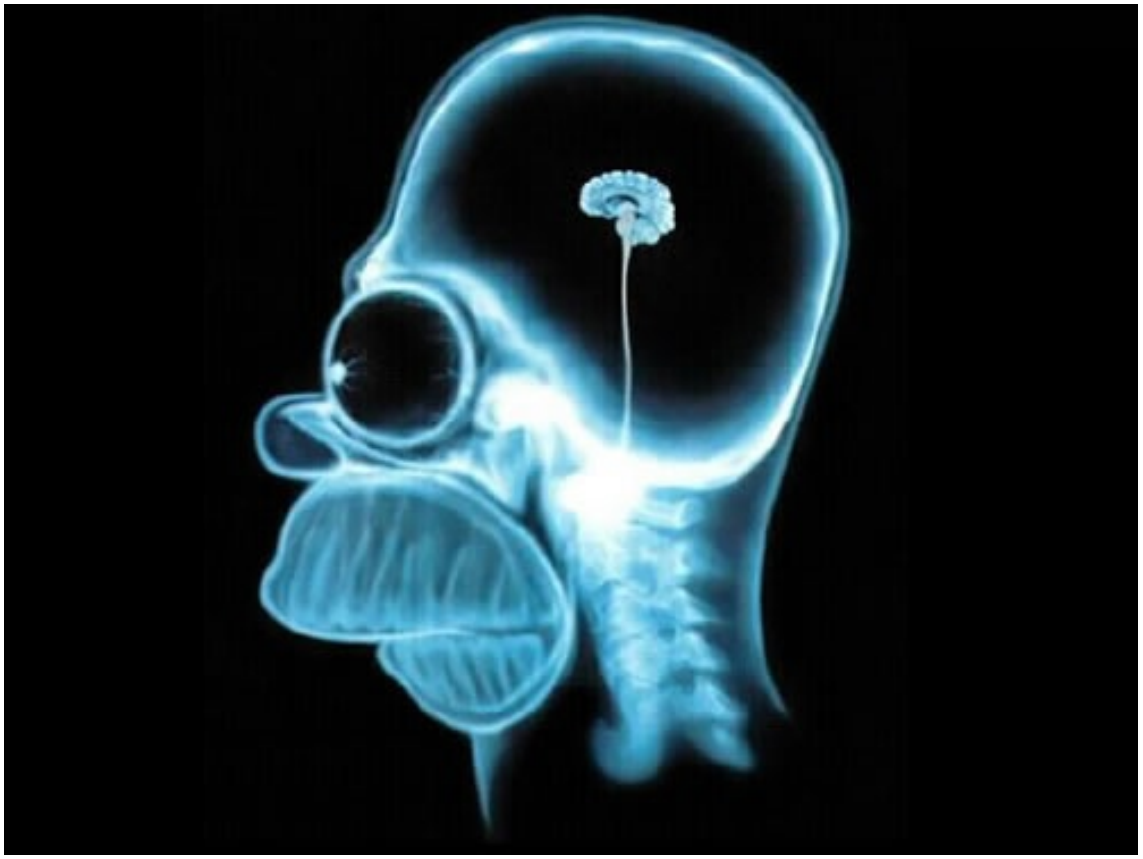
Peter Elger

@pelger

So wtf is a µ-service ?

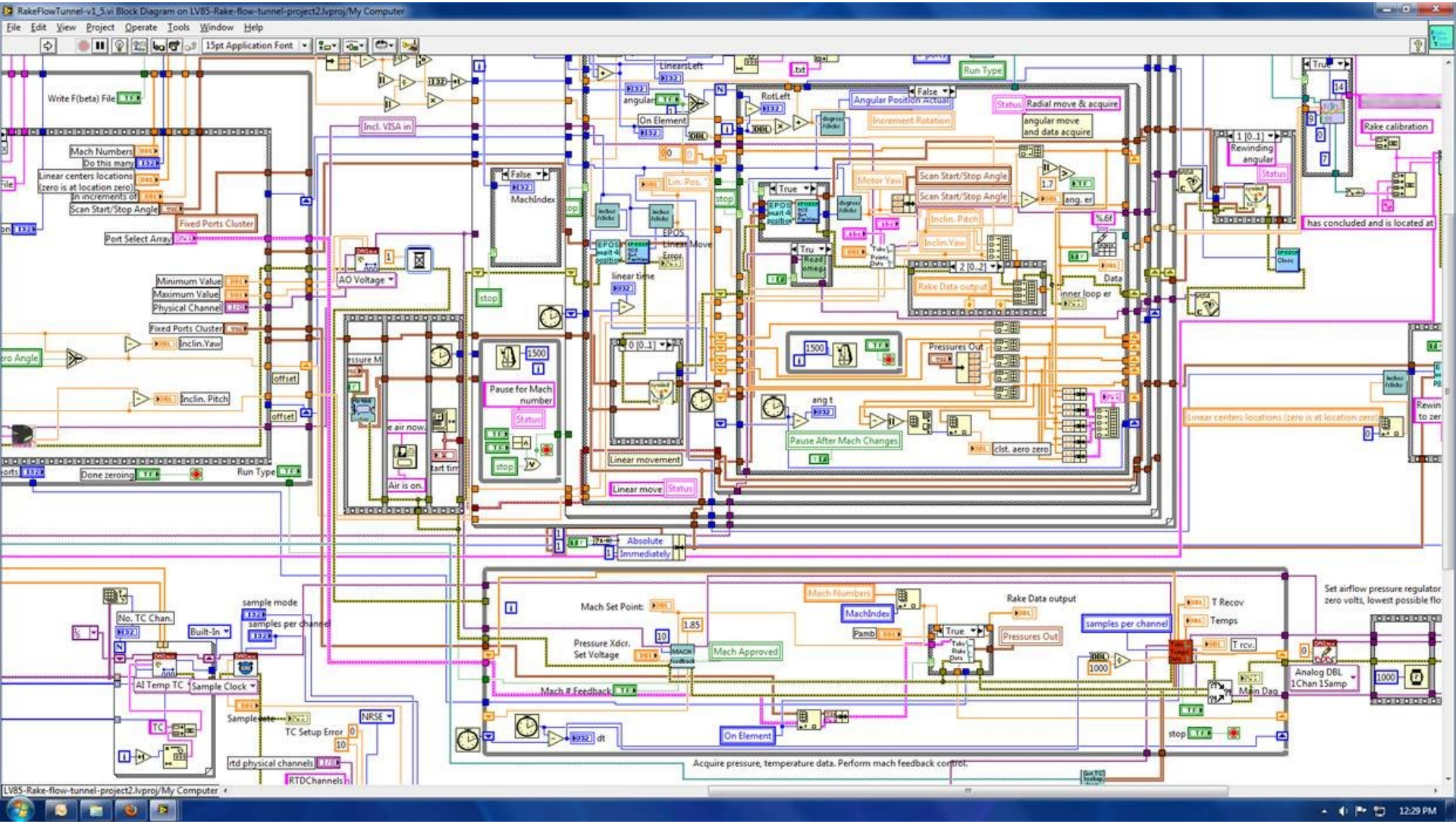
- ‘Sharp’ unit of functionality
- Typically around 100 lines of code
- Does one thing and one thing only
  - And does it well
- A µ-services architecture is a system built from the aggregation of **LOTS** of µ-services…
- A system compon
- Why is that interesting…?

Things tend to get complicated quickly

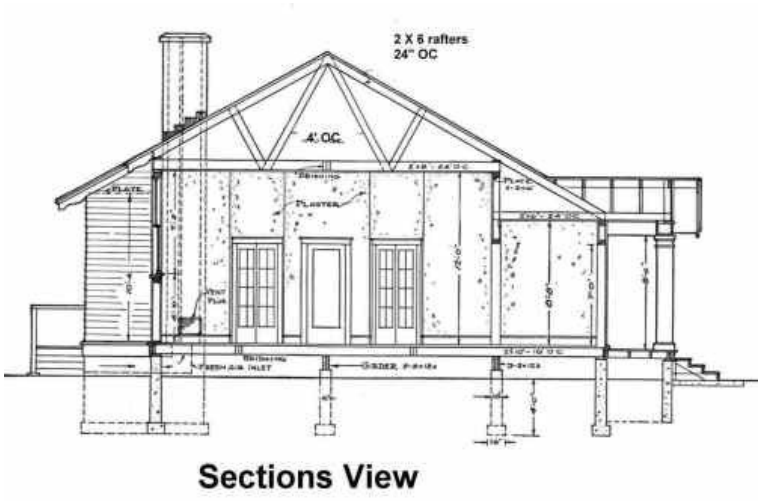


- I have a limited cranial capacity
- So I’m ‘big on simple’

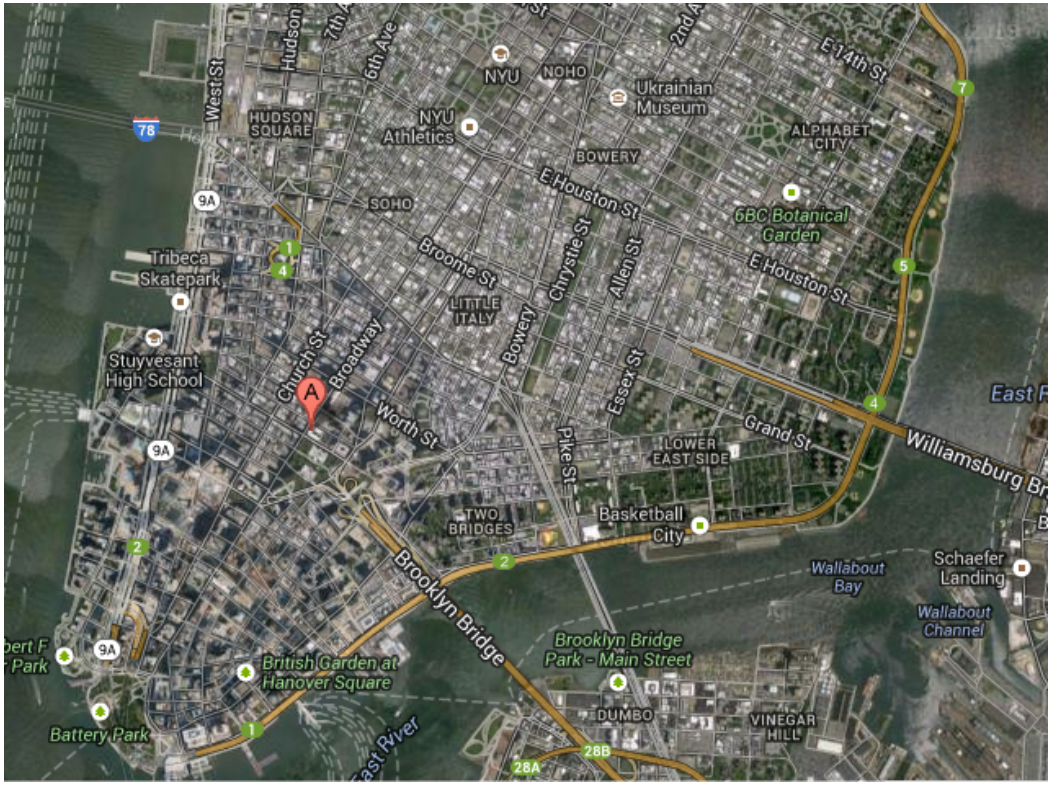
Software becomes complex quickly



Building ‘large’ software systems becomes exponentially complex



Sections View



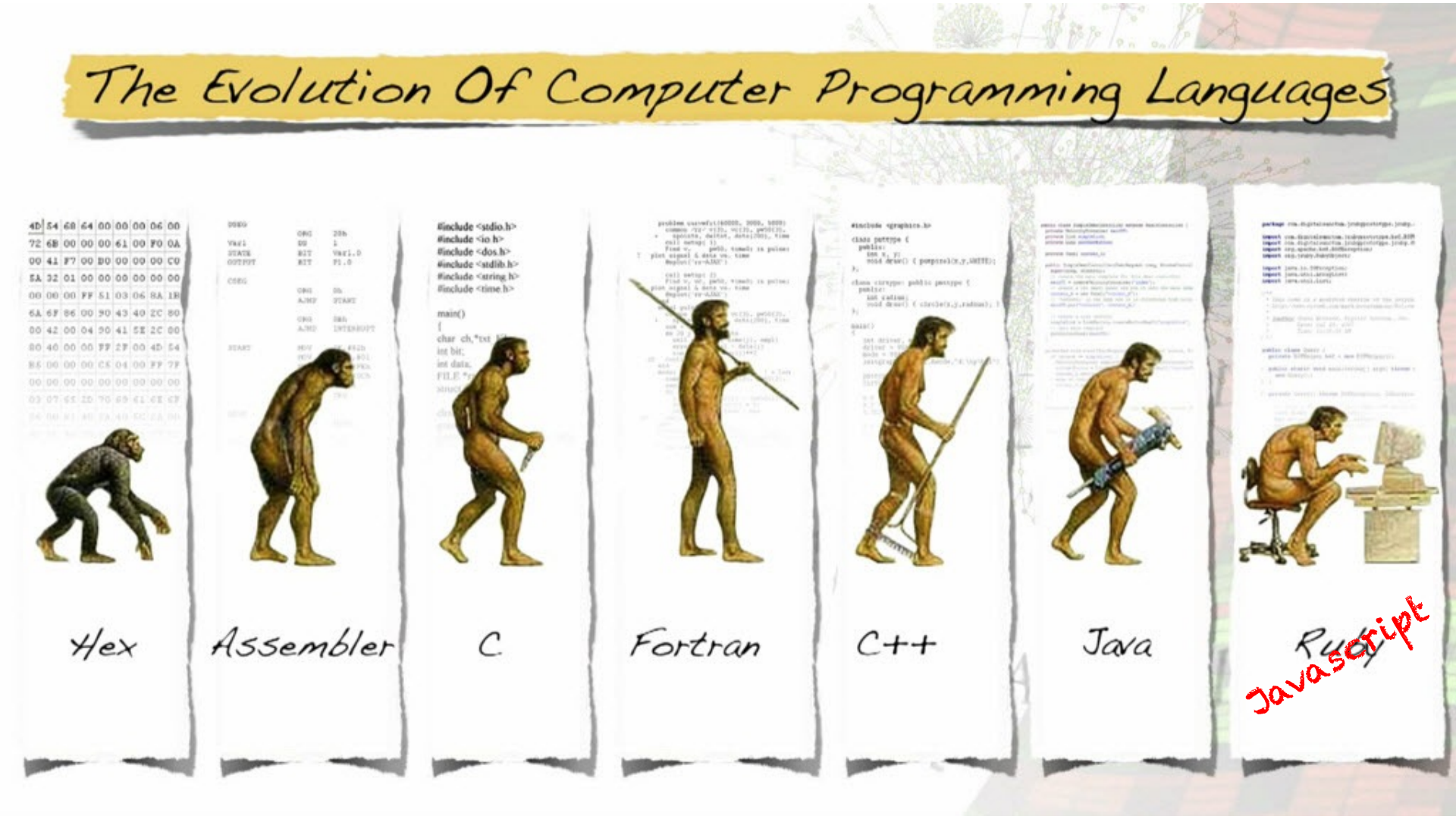
Know your enemy

- Complexity
- Entropy
- Coupling
- Repetition

Technical Debt

- No one ever wrote perfect code, ever.
- All software has some level of technical debt
- The most successful software systems are those that effectively manage technical debt
- As creators of software we should adopt architectures and techniques that allow us to effectively manage technical debt over the entire lifetime of a system.

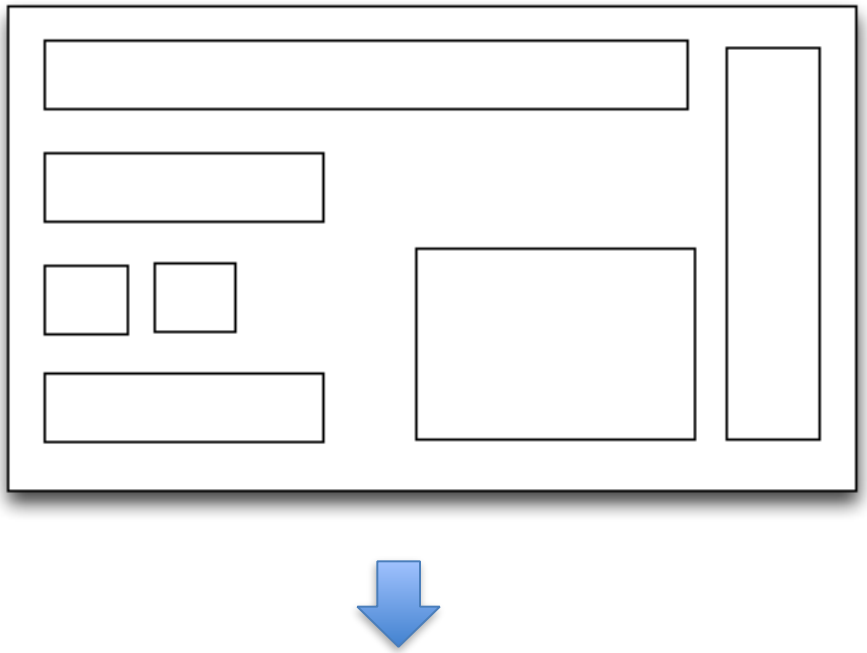
Evolution


The Evolution Of Computer Programming Languages
Hex
Assembler
C
Fortran
C++
Java
Ruby
Javascript

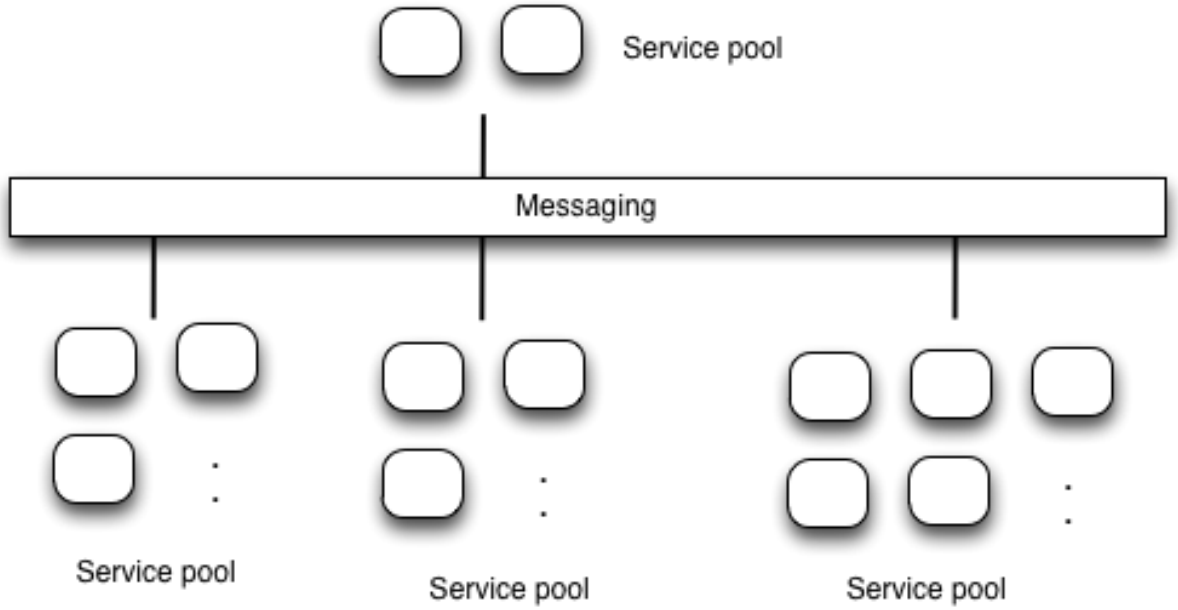
Architectural Evolution

- Mush
- Mainframe/terminal
- Client Server
- N-Tiered
- SOA
- ESB
- µ-services ??

Layered / OO



μ-services



Characteristics

- Messaging channel
  - Direct Http, Queue, Message bus…
- Lots of little services
  - ~100 loc
- Each does one thing only
- Multiple instances
- Multiple versions

Messages

- Pattern match on messages
- Messages are asynchronous as are responses

- For Example:

```
{
command:"tweet"
Author:”@pelger"
status:”hello world"
}
```

Tweet service

- If I see a message with this property:
  - command:"tweet”
- It's Mine!
- I don't care how you get it to me…
- …and I'll ignore anything else

Simple rules

- Report status
  - Each process self reports status
- Die and restart on error
  - Don’t attempt error handling just reboot!
- Continuously test the business outputs of the system

Lifecycle

- Services have short lifetimes
  - Add a service, remove/update a service without affecting the overall system
- Multiple version of the same service can coexist
- Services are language agnostic

Living software system

- Long-lived system; short-lived services

- Extremely dynamic with continuous deployments
- 5-10 minutes between deployments typical
- Accept it is complex (especially for testing)
- Acceptance test on business outcomes instead

Key difference

- Allows system to ‘Grow’ from the bottom up
  - Following simple rules
  - Rather than top down planned control
- Rapid deployment of individual services
  - Isolated change
- Written in any language
  - Pick the right tool for the job

- Think of command line tools v’s Bloated GUI

Systems using µ-services

- Sunday business post
- Social analytics platform
- Nearform Hardware Cloud
  - Just demonstrated
- Fred George
  - Forward Labs
  - Check out his presentations on youtube!

Microbial

- Toolkit that embodies µ-service principals
  - Based on in the field experiences
- Built on node.js
- Javascript is particularly well suited to implementing asynchronous µ-services
- npm install microbial

Summary

- Very, very small
- Loosely coupled
- Highly Cohesive
- Self-execution monitoring of each service
- Pattern match on messages
- Die and restart on error
- Measure the live system
- Rapid and continuous deployments

- We are discovering through live projects how this approach allows us to develop and operate systems more effectively.

Questions ?