Stories Every Developer Should Know

ThoughtWorks

NEAL FORD

Director / Software Architect / Meme Wrangler

@neal4d

nealford.com

Forensic engineering

Forensic engineering is the investigation of materials, products, structures or components that fail or do not operate or function as intended, causing personal injury or damage to property. The consequences of failure are dealt with by the law of product liability. The field also deals with determining processes and procedures leading to accidents in operation of vehicles or machinery. The subject is applied most commonly in civil law cases, although it may be of use in criminal law cases. Generally, the purpose of a forensic engineering investigation is to locate cause or causes of failure with a view to improve performance of the item or component, or to assist a court in determining the facts of an accident. It can also involve investigation of intellectual property claims, especially patents.

Contents [hide]

1 History
2 Investigation
3 Analysis
4 Examples
5 Applications
6 Historic examples
7 Publications
8 See also
9 References

Part of a series on
Forensic science

Physiological
Social
Criminalistics
Digital forensics
Related disciplines

Related articles

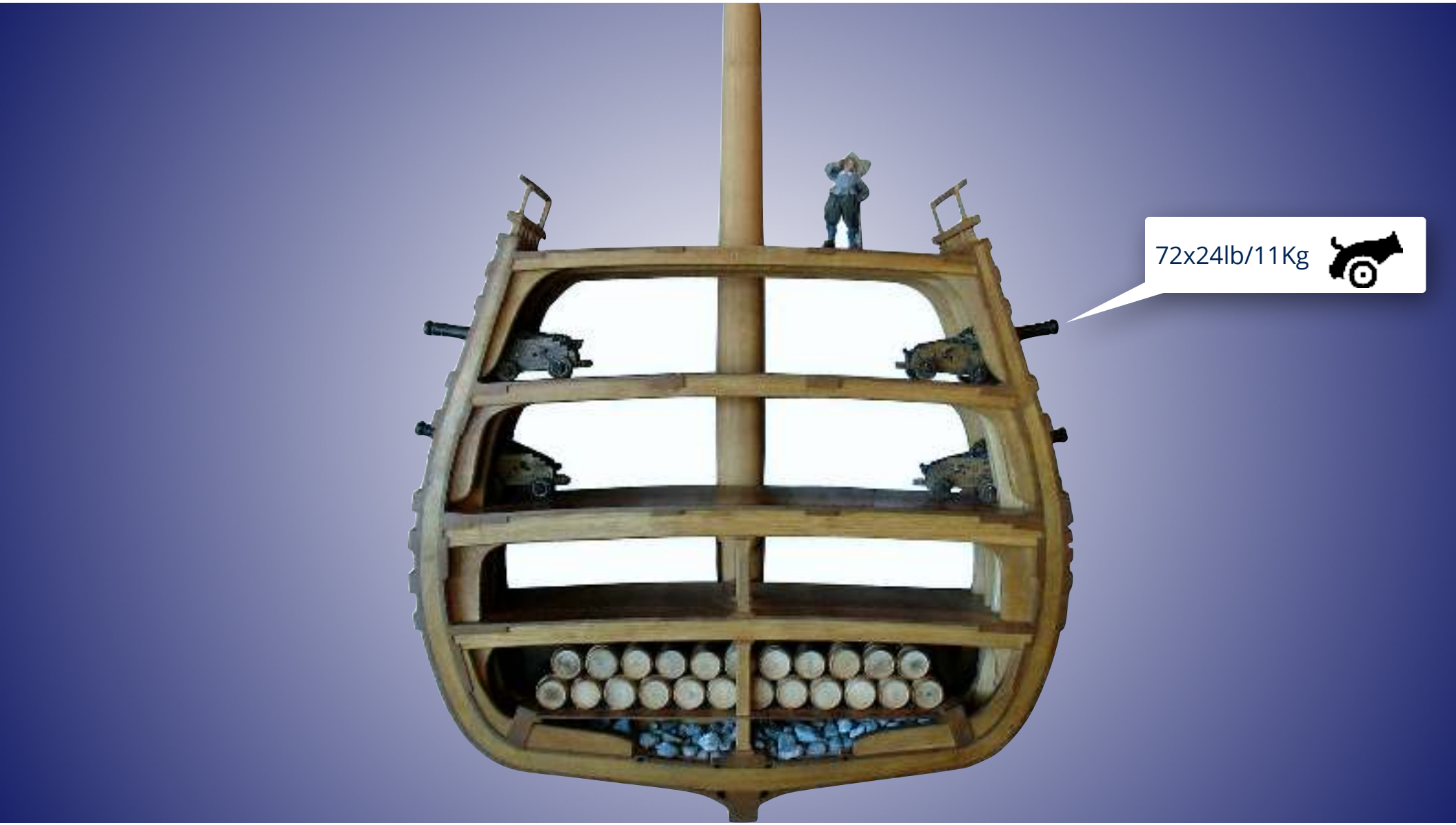
History

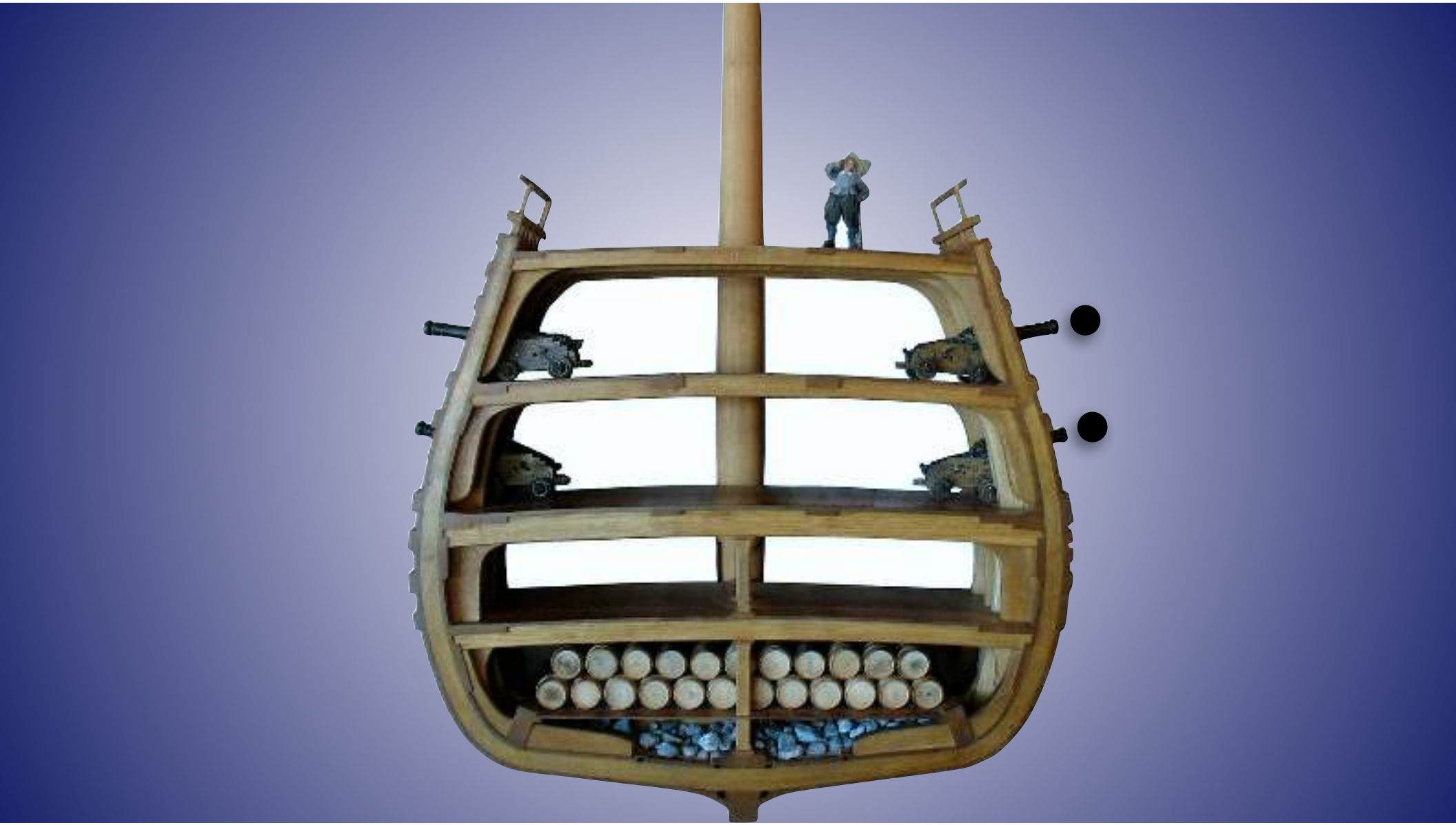
As the field of engineering has evolved over time, so has the field of forensic engineering. Early examples include investigation of bridge failures such as the Tay rail bridge disaster of 1879 and the Dee bridge disaster of 1847. Many early railroad accidents promoted the invention of tensile testing of samples and fractography of failed components.[1]

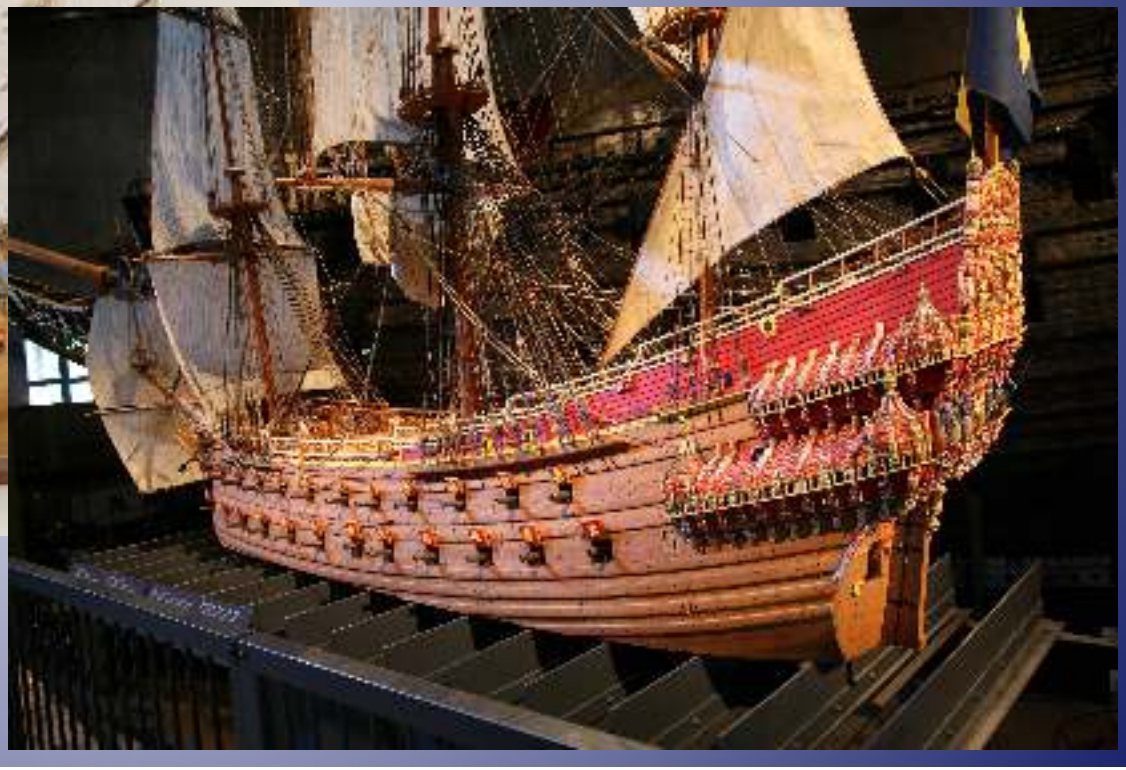
Investigation

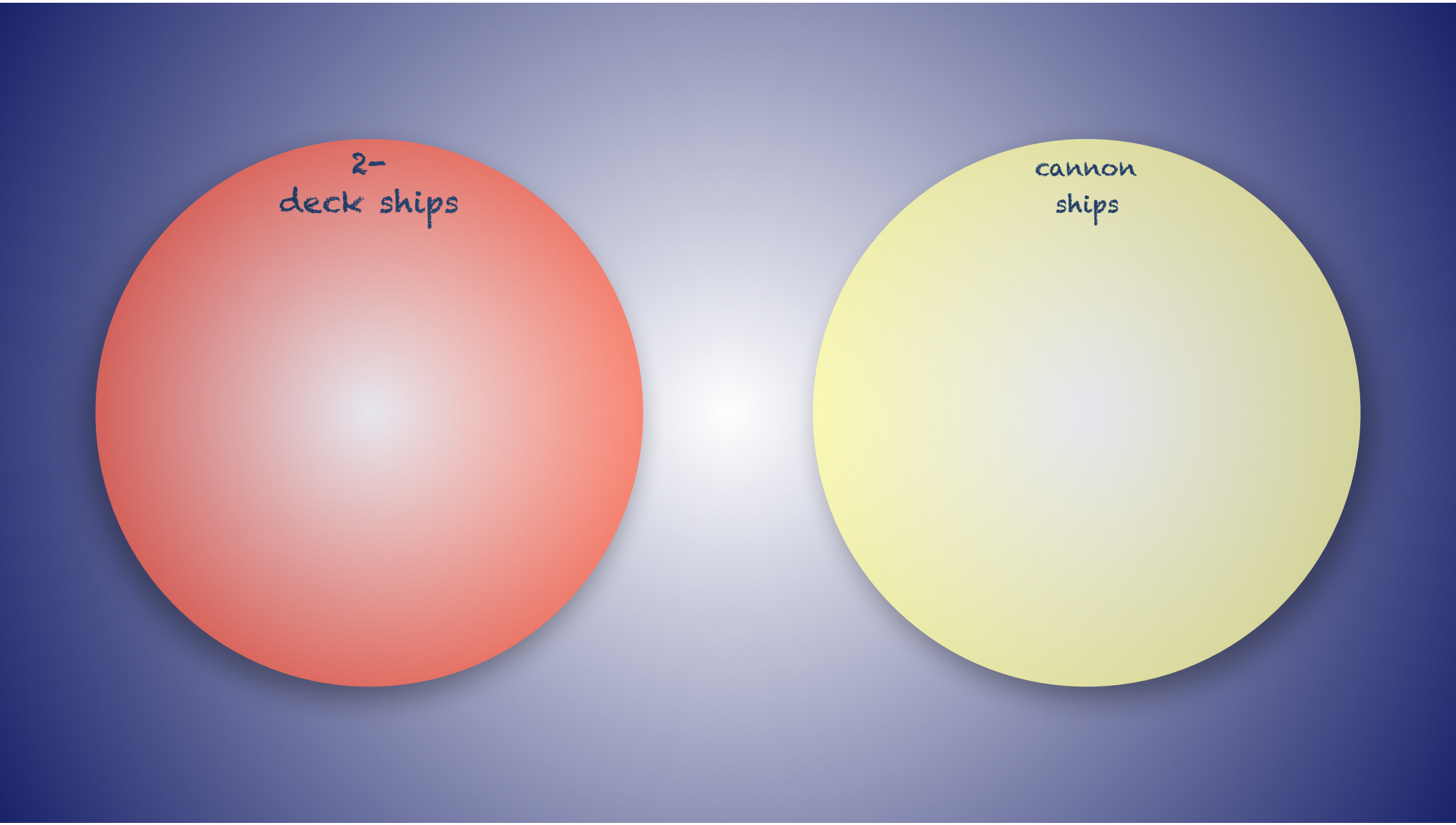
Vasa

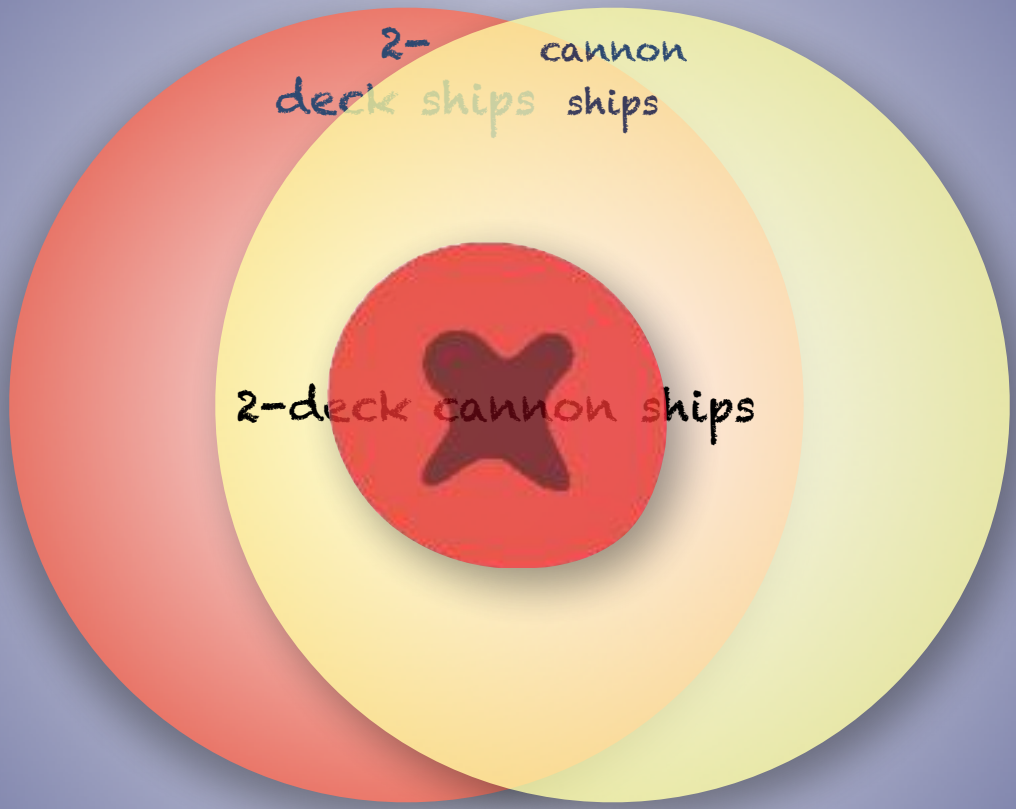

Trondelag
1658 (1660)
Jämtland
1645
Dalarna
Åland
Gulf of Bothnia
Finland
1581 (1809)
Karelia
1617 (1721)
Ladoga Lake
Kexholm
Viborg
(1721)
Narva
Ingria
1583-95, 1617 (1721)
Estland
1561 (1721)
Stockholm
Dagö
Gotland
1648
Ösel
1645 (1721)
Livonia
1629 (1721)
Riga
1621
Bohuslän
1658
Halland
1658
Småland
Öland
Scania
1658
Kaliningrad
Klaipéda/Memel
1629-35
Copenhagen
Stralsund
(1815)
Barnholm
1658 (1660)
Pillau
1629-35
Elbing
1629-35
Wismar
1648 (1803)
Cammin
(1679)
Western Pomerania
1648 (1720)
Bremen-Verden
1648 (1719)
NORWAY
RUSSIA
NORTH SEA
DENMARK
BALTIC SEA
LITHUANIA
PRUSSIA
POLAND
HOLY ROMAN EMPIRE
Sweden under Gustav Vasa until 1560
Acquisitions under Gustav Vasa's sons until 1611
Acquisitions under Gustav II Adolf and Christina until 1654
Acquisitions under Karl X until 1660
Numbers describe the year of acquisition, numbers in parentheses describe the year of loss


72x24lb/11Kg

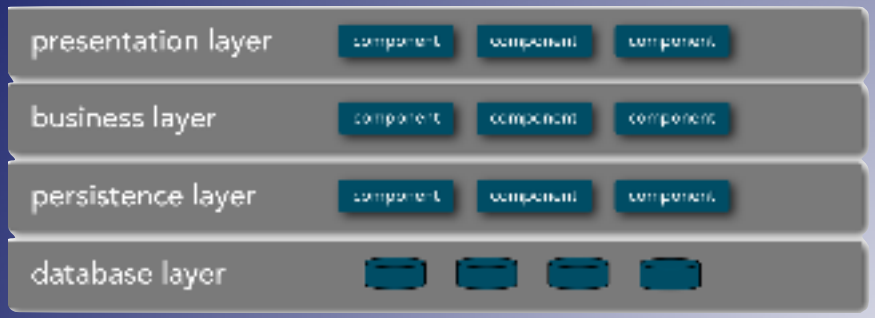




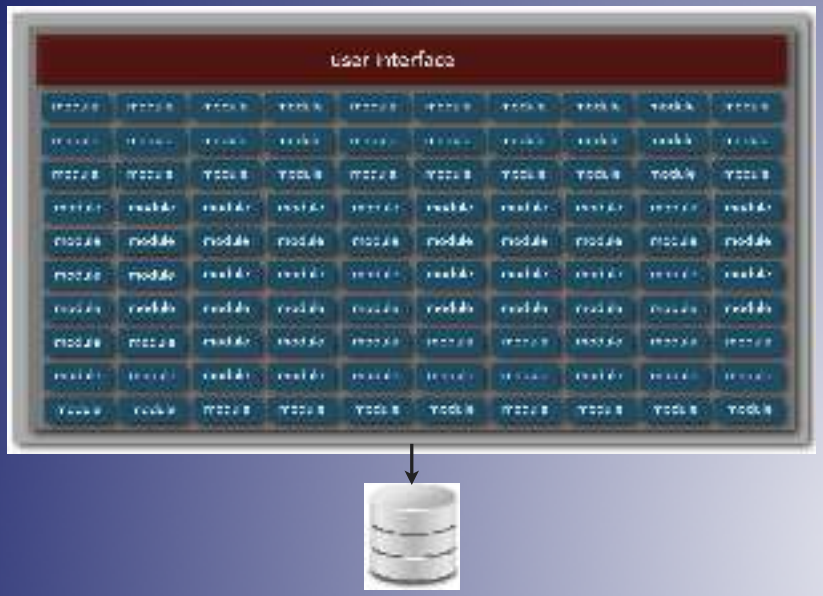

2-
deck ships
cannon
ships


2-deck ships
cannon ships
2-deck cannon ships

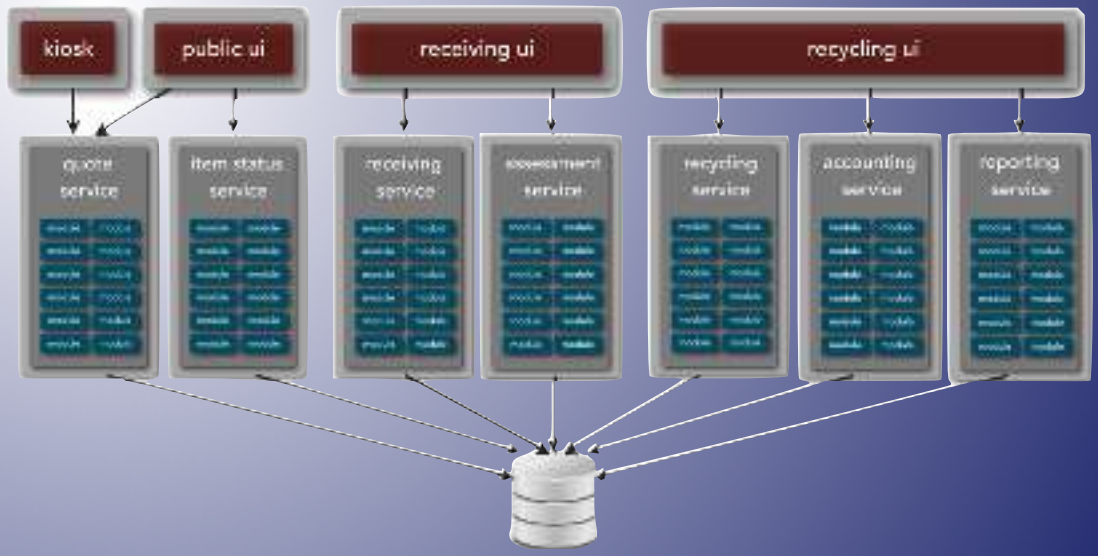
#scaling_architecture



some things in architecture don't scale linearly



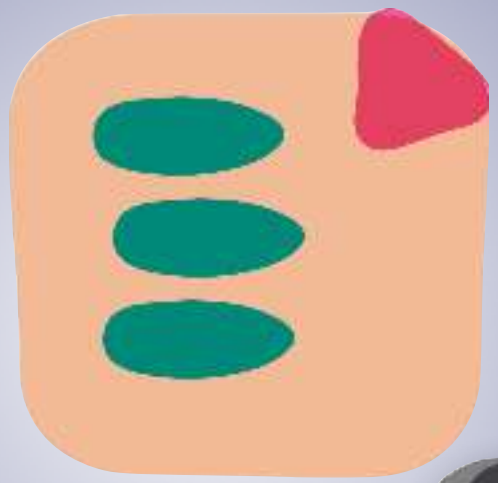
when restructuring architecture...

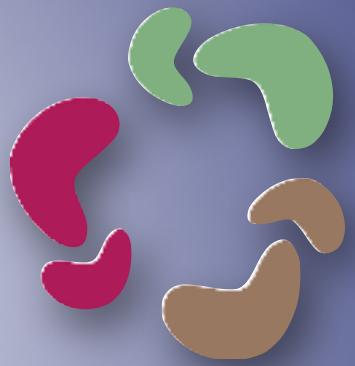


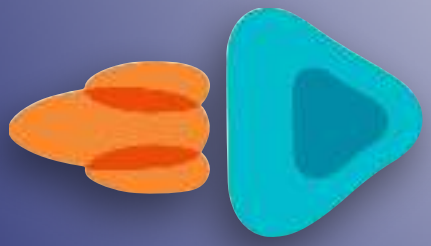
...reevaluate "-ilities".

requirements

requirements + “—ilities”

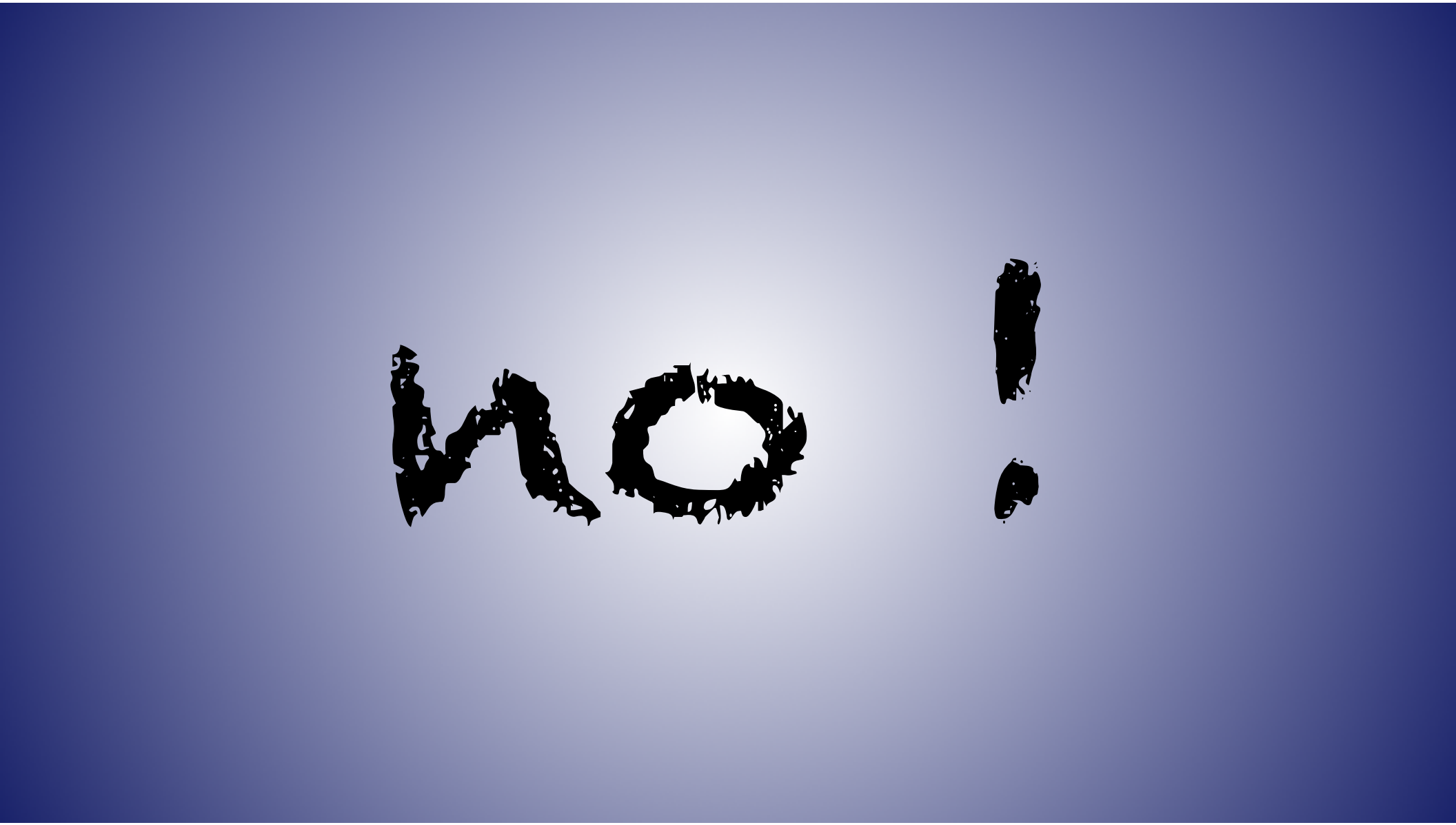






defining architecture


WENGER


no !

#tradeoffs

Tacoma Narrows Bridge


Tacoma
Narrows
Bridge


Disaster

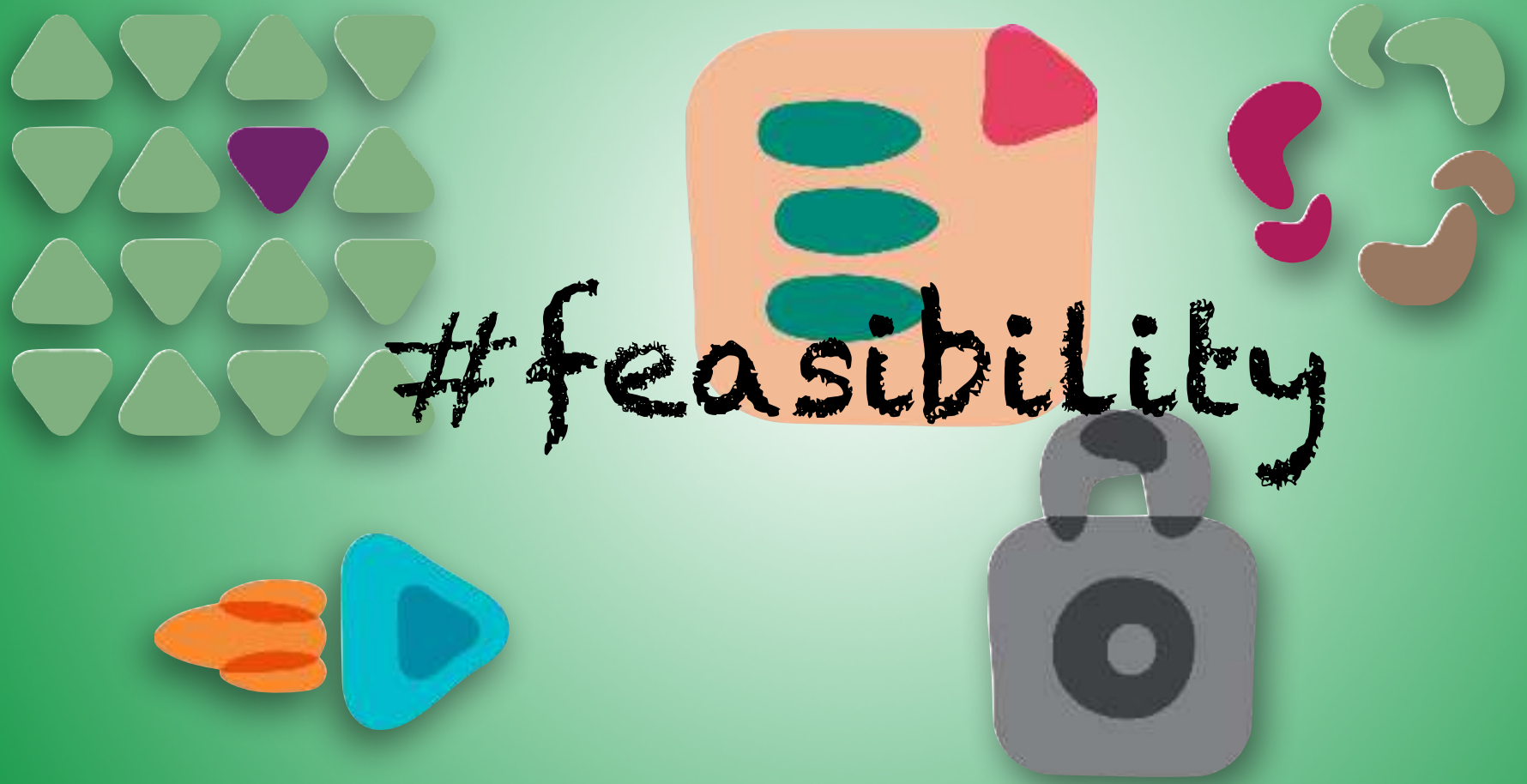

Disaster

#scaling_architecture

F16

1974

requirements + “—ilities”



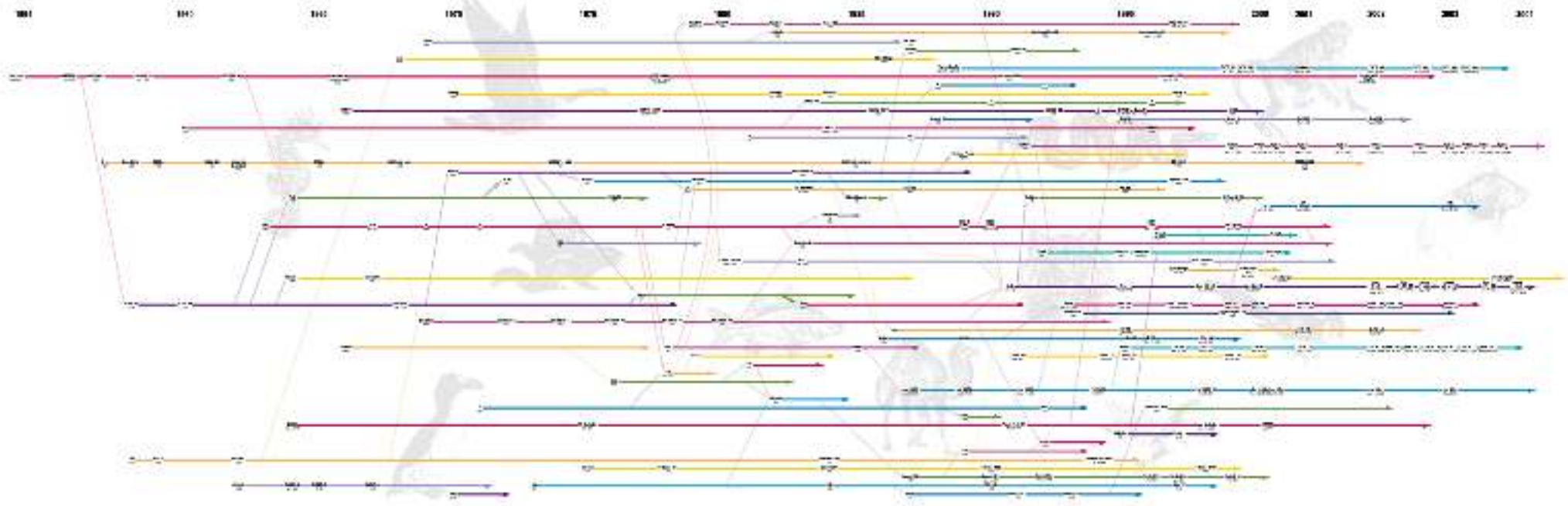

Why?

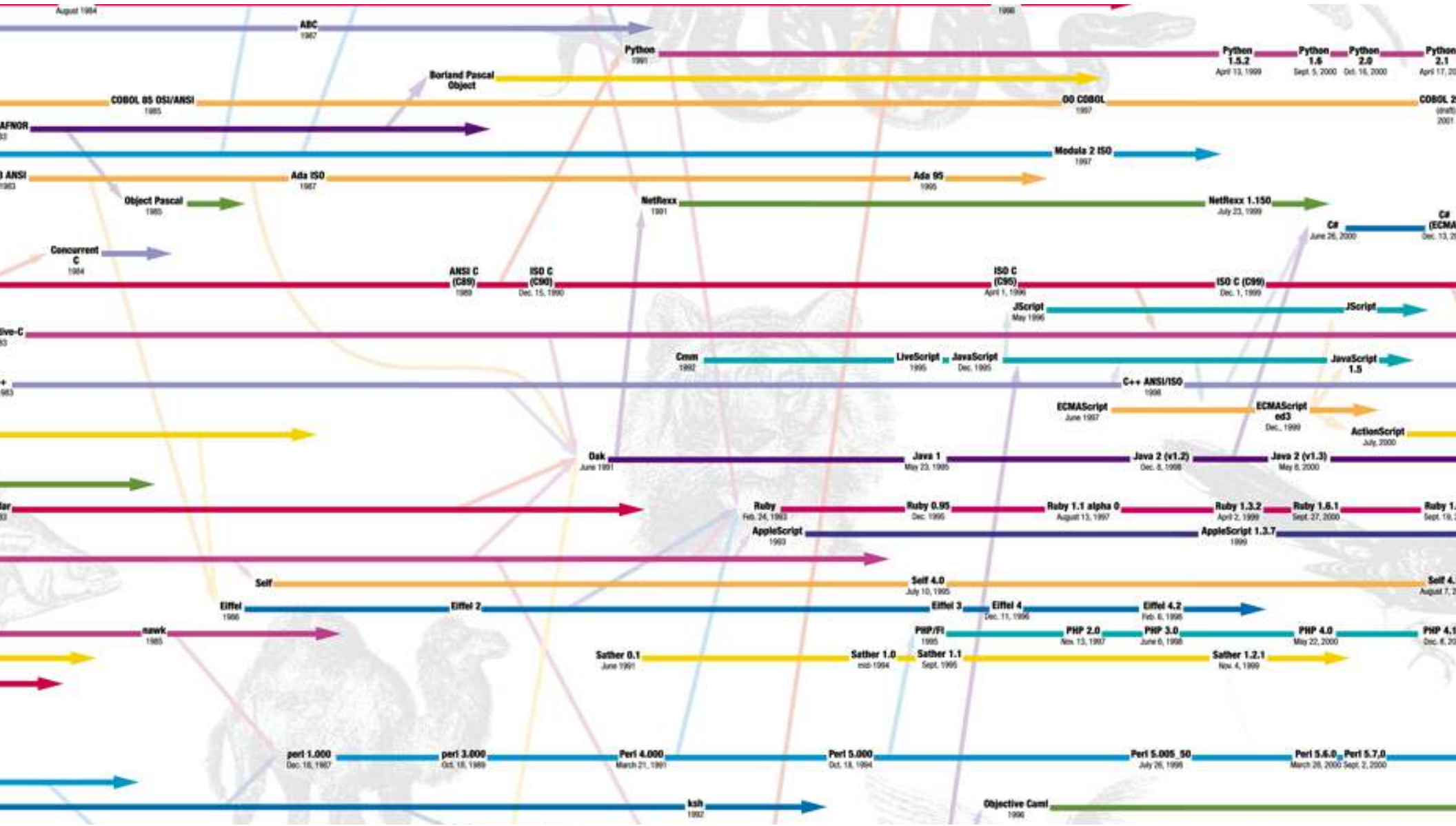
null

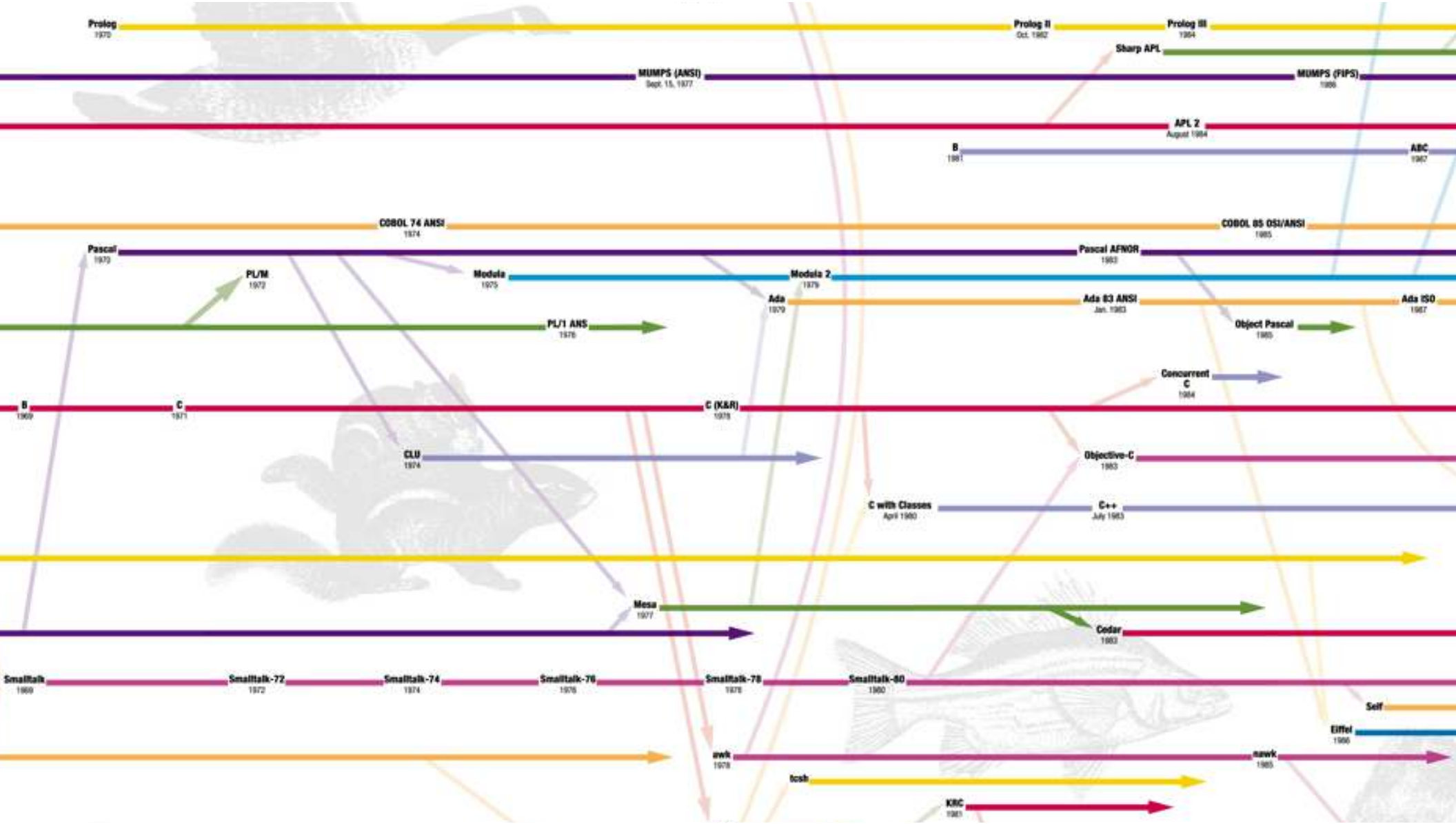
https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare

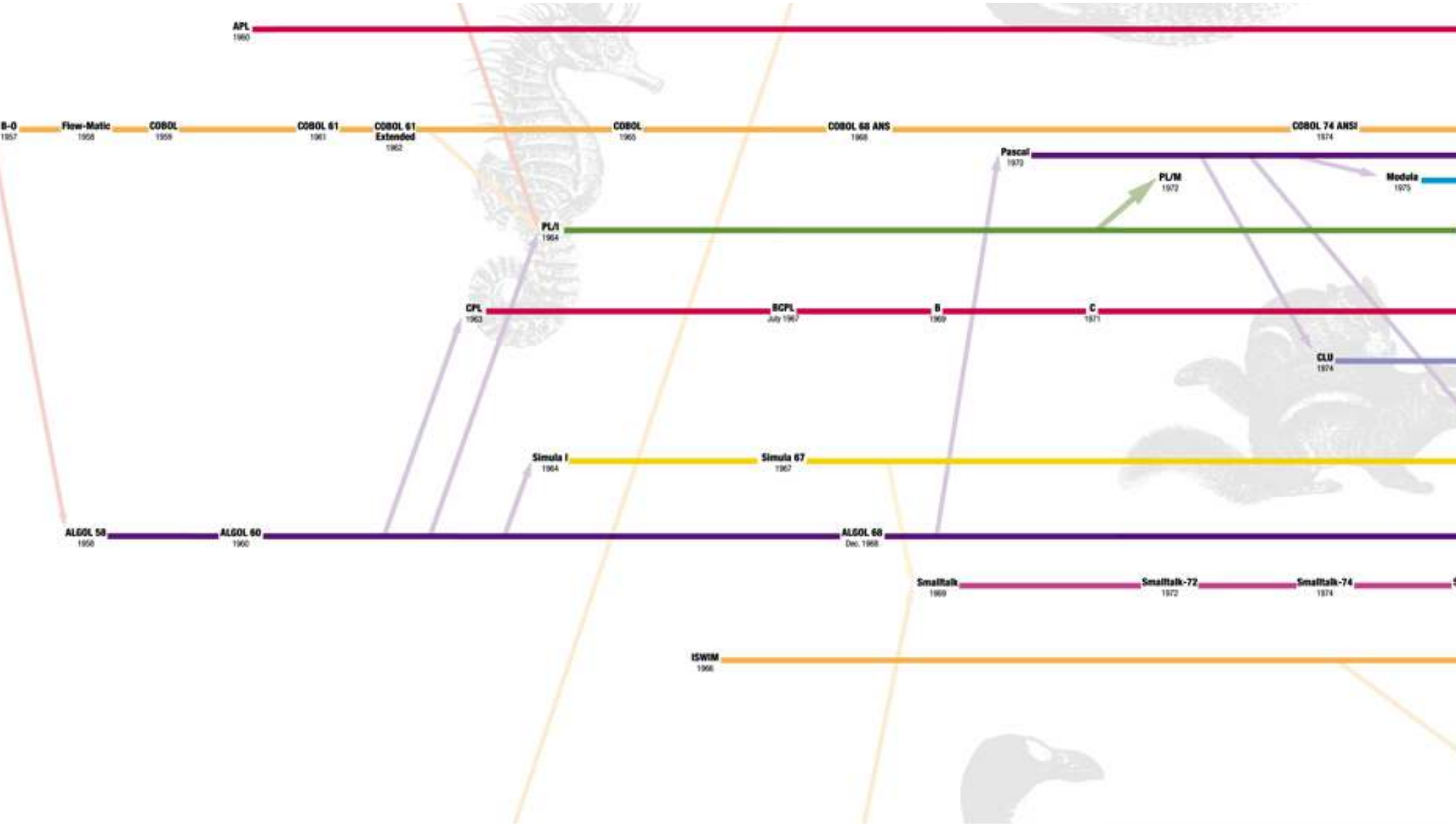
History of Programming Languages

O'REILLY



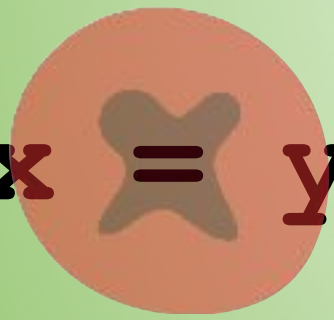

August 1984
ABC
1987
Python
1991
Python 1.5.2
April 13, 1999
Python 1.6
Sept. 5, 2000
Python 2.0
Oct. 16, 2000
Python 2.1
April 17, 20
Borland Pascal
Object
COBOL 85 OSI/ANSI
1985
OO COBOL
1997
COBOL 2
(draft)
2001
AFNOR
Modula 2 ISO
1997
Ada ISO
1987
Ada 95
1995
Object Pascal
1985
NetRexx
1991
NetRexx 1.150
July 23, 1999
C#
June 26, 2000
C#
(ECMA
Dec. 13, 2
Concurrent
C
1984
ANSI C
(C89)
1989
ISO C
(C90)
Dec. 15, 1990
ISO C
(C95)
April 1, 1996
ISO C (C99)
Dec. 1, 1999
JScript
May 1996
JScript
Cmm
1992
LiveScript
1995
JavaScript
Dec. 1995
JavaScript
1.5
C++ ANSI/ISO
1998
ECMAScript
June 1997
ECMAScript
ed3
Dec., 1999
ActionScript
July, 2000
Oak
June 1991
Java 1
May 23, 1995
Java 2 (v1.2)
Dec. 8, 1998
Java 2 (v1.3)
May 8, 2000
Ruby
Feb. 24, 1993
Ruby 0.95
Dec. 1995
Ruby 1.1 alpha 0
August 13, 1997
Ruby 1.3.2
April 2, 1999
Ruby 1.6.1
Sept. 27, 2000
Ruby 1.
Sept. 19,
AppleScript
1993
AppleScript 1.3.7
1999
Self
Self 4.0
July 10, 1995
Self 4.
August 7, 2
Eiffel
1986
Eiffel 2
Eiffel 3
Eiffel 4
Dec. 11, 1996
Eiffel 4.2
Feb. 6, 1998
nawk
1985
PHP/FI
1995
PHP 2.0
Nov. 13, 1997
PHP 3.0
June 6, 1998
PHP 4.0
May 22, 2000
PHP 4.1
Dec. 6, 20
Sather 0.1
June 1991
Sather 1.0
mid-1994
Sather 1.1
Sept. 1995
Sather 1.2.1
Nov. 4, 1999
perl 1.000
Dec. 18, 1987
perl 3.000
Oct. 18, 1989
Perl 4.000
March 21, 1991
Perl 5.000
Oct. 18, 1994
Perl 5.005_50
July 26, 1998
Perl 5.6.0
March 28, 2000
Perl 5.7.0
Sept. 2, 2000
ksh
1992
Objective Caml
1996


Prolog
1970
Prolog II
Oct. 1982
Prolog III
1984
Sharp APL
MUMPS (ANSI)
Sept. 15, 1977
MUMPS (FIPS)
1986
APL 2
August 1984
B
1981
ABC
1987
COBOL 74 ANSI
1974
COBOL 85 OSI/ANSI
1985
Pascal
1970
Pascal AFNOR
1983
PL/M
1972
Modula
1975
Modula 2
1979
Ada
1979
Ada 83 ANSI
Jan. 1983
Ada ISO
1987
PL/1 ANS
1976
Object Pascal
1985
Concurrent
C
1984
B
1969
C
1971
C (K&R)
1978
CLU
1974
Objective-C
1983
C with Classes
April 1980
C++
July 1983
Mesa
1977
Cedar
1983
Smalltalk
1969
Smalltalk-72
1972
Smalltalk-74
1974
Smalltalk-76
1976
Smalltalk-78
1978
Smalltalk-80
1980
Self
Eiffel
1986
awk
1978
nawk
1985
tcsh
KRC
1981

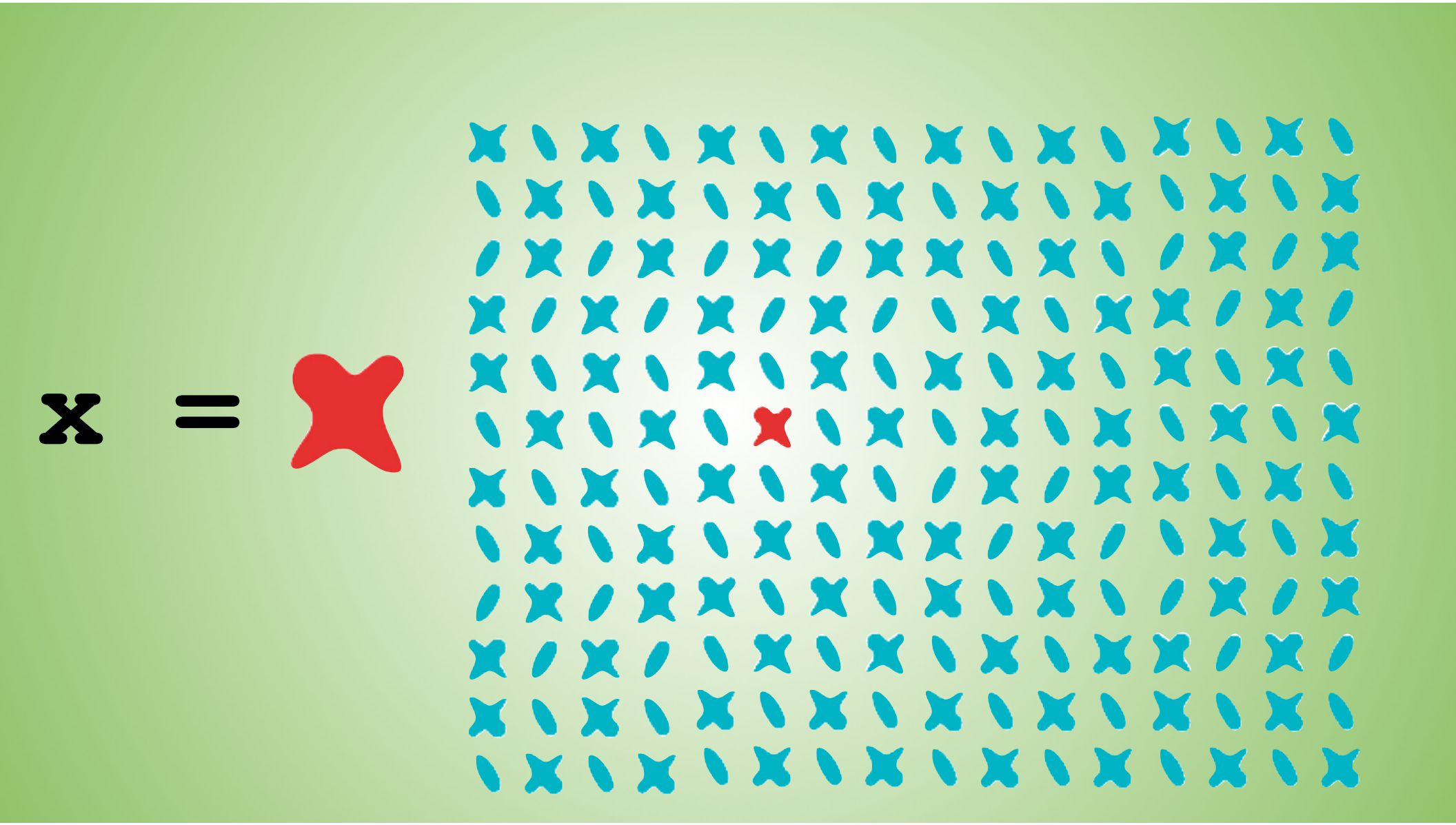

APL
1960
B-0
1957
Flow-Matic
1958
COBOL
1959
COBOL 61
1961
COBOL 61
Extended
1962
COBOL
1965
COBOL 68 ANS
1968
COBOL 74 ANSI
1974
Pascal
1970
PL/M
1972
Modula
1975
PL/I
1964
CPL
1963
BCPL
July 1967
B
1969
C
1971
CLU
1974
Simula I
1964
Simula 67
1967
ALGOL 58
1958
ALGOL 60
1960
ALGOL 68
Dec. 1968
Smalltalk
1969
Smalltalk-72
1972
Smalltalk-74
1974
ISWIM
1966

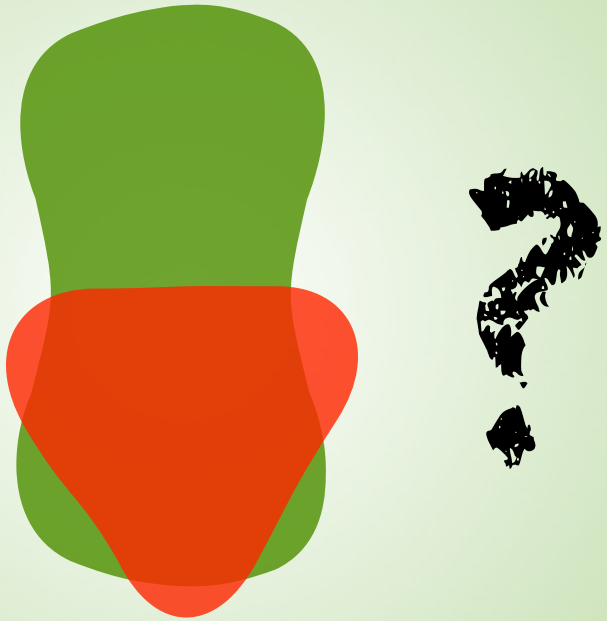
https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare

```
Person x = new Person();
Cat y = new Cat();

x = y; //not permitted
```

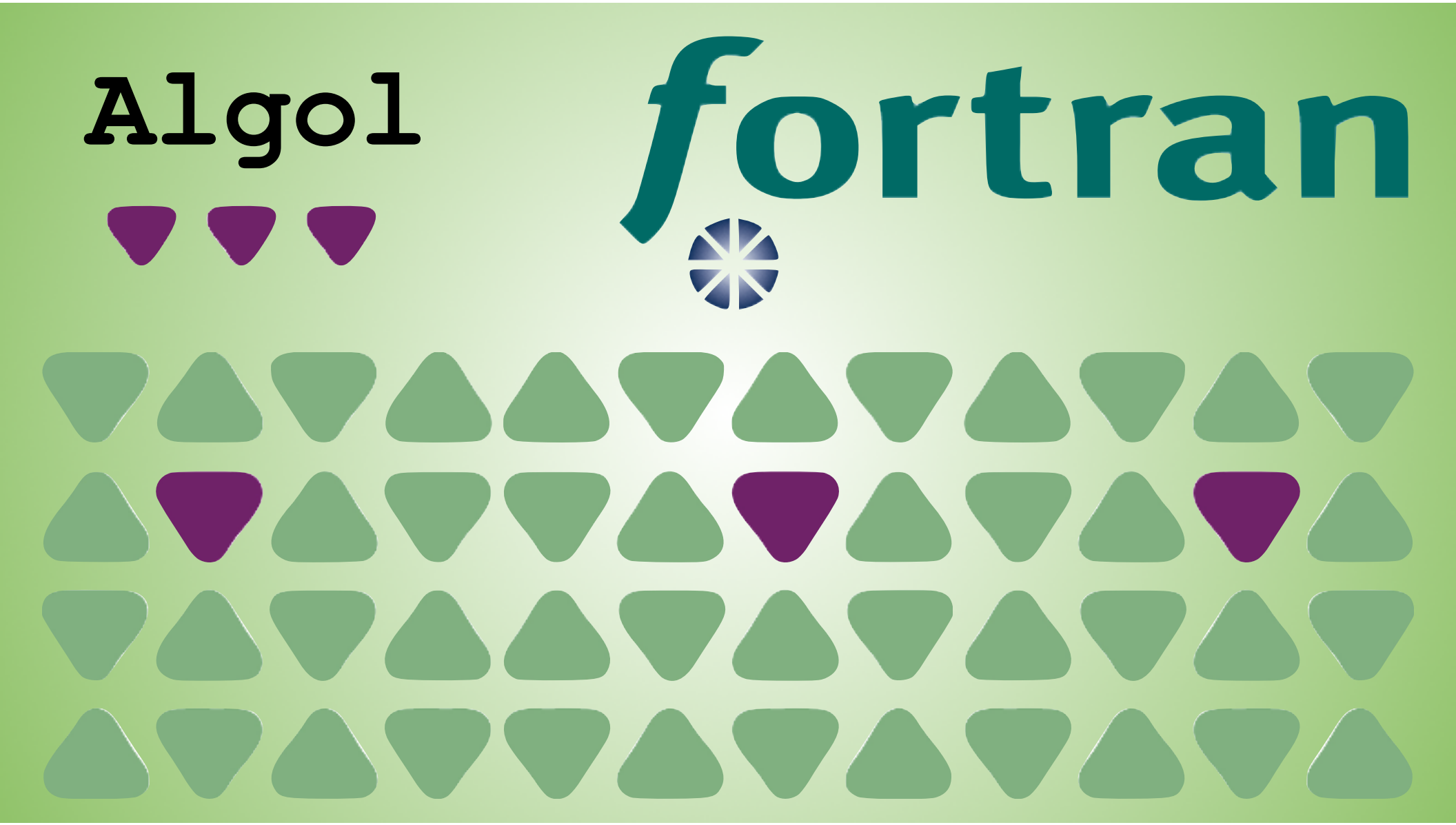


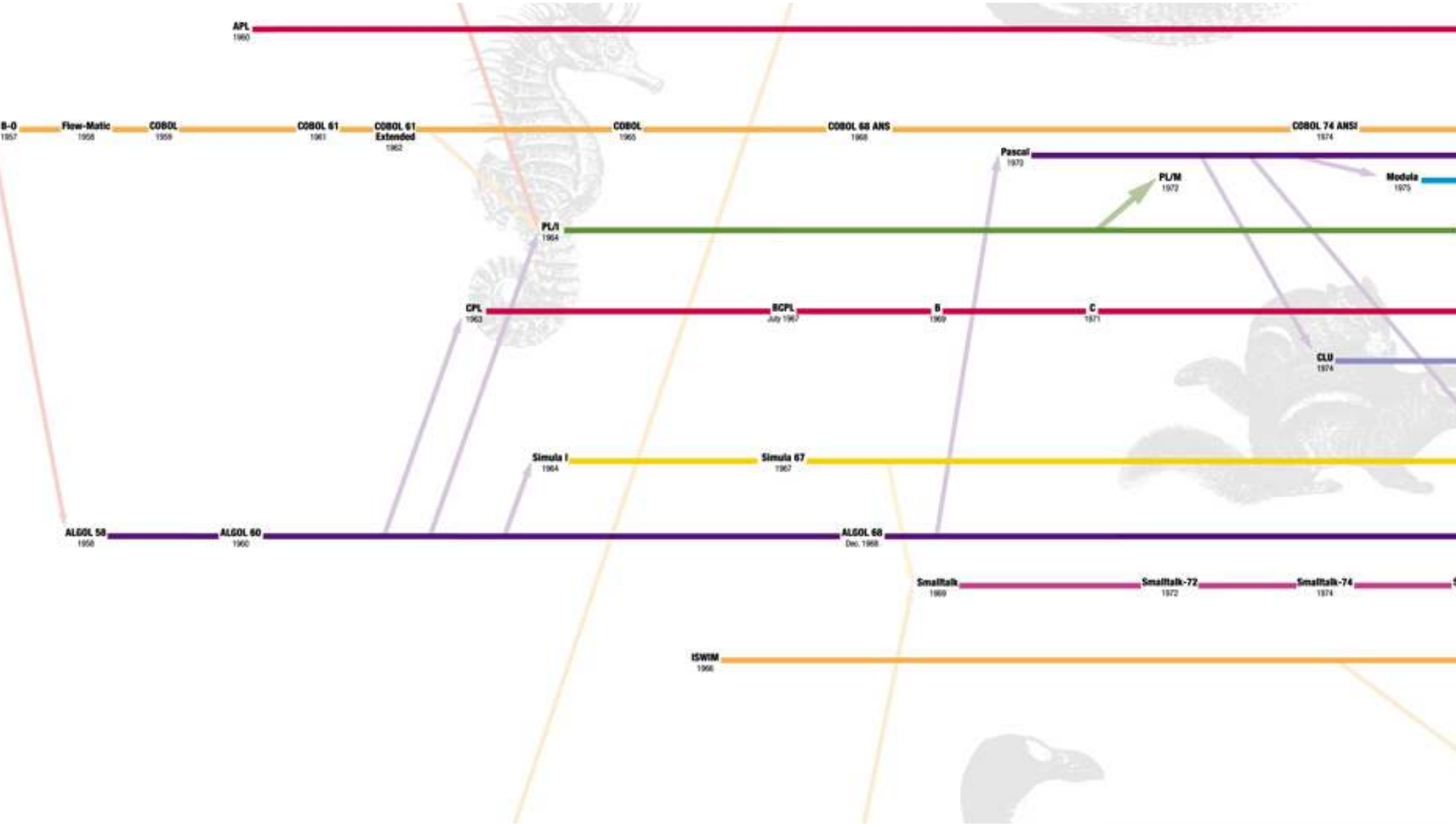
$$x = ?$$

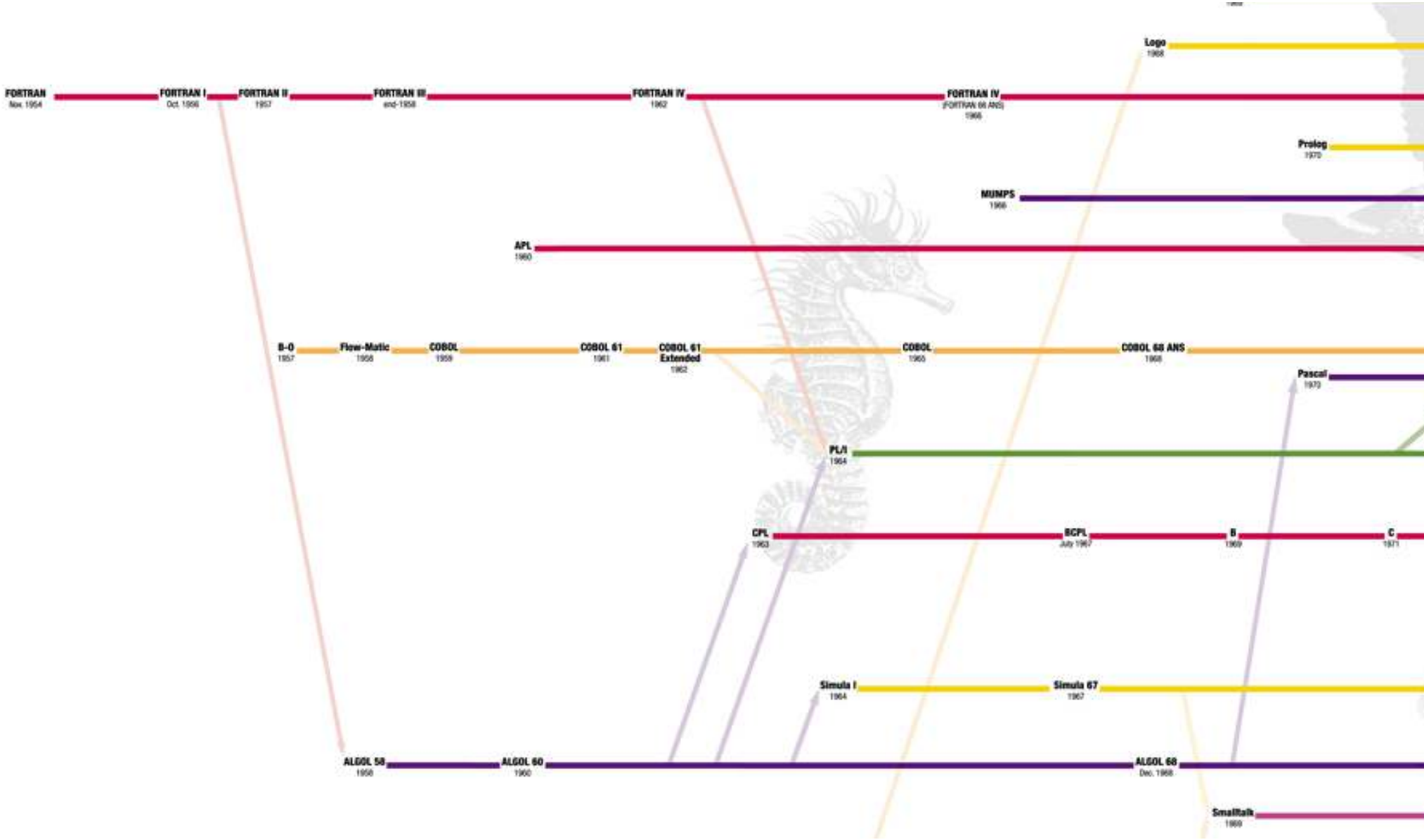




Algol


Algol
fortran


APL
1960
B-0
1957
Flow-Matic
1958
COBOL
1959
COBOL 61
1961
COBOL 61
Extended
1962
COBOL
1965
COBOL 68 ANS
1968
COBOL 74 ANSI
1974
Pascal
1970
PL/M
1972
Modula
1975
PL/I
1964
CPL
1963
BCPL
July 1967
B
1969
C
1971
CLU
1974
Simula I
1964
Simula 67
1967
ALGOL 58
1958
ALGOL 60
1960
ALGOL 68
Dec. 1968
Smalltalk
1969
Smalltalk-72
1972
Smalltalk-74
1974
ISWIM
1966


FORTRAN
Nov. 1954
FORTRAN I
Oct. 1956
FORTRAN II
1957
FORTRAN III
end-1958
FORTRAN IV
1962
FORTRAN IV
(FORTRAN 66 ANS)
1966
Logo
1968
Prolog
1970
MUMPS
1966
APL
1960
B-0
1957
Flow-Matic
1958
COBOL
1959
COBOL 61
1961
COBOL 61
Extended
1962
COBOL
1965
COBOL 68 ANS
1968
Pascal
1970
PL/I
1964
CPL
1963
BCPL
July 1967
B
1969
C
1971
Simula I
1964
Simula 67
1967
ALGOL 58
1958
ALGOL 60
1960
ALGOL 68
Dec. 1968
Smalltalk
1969

#legacy

why we can't have nice things

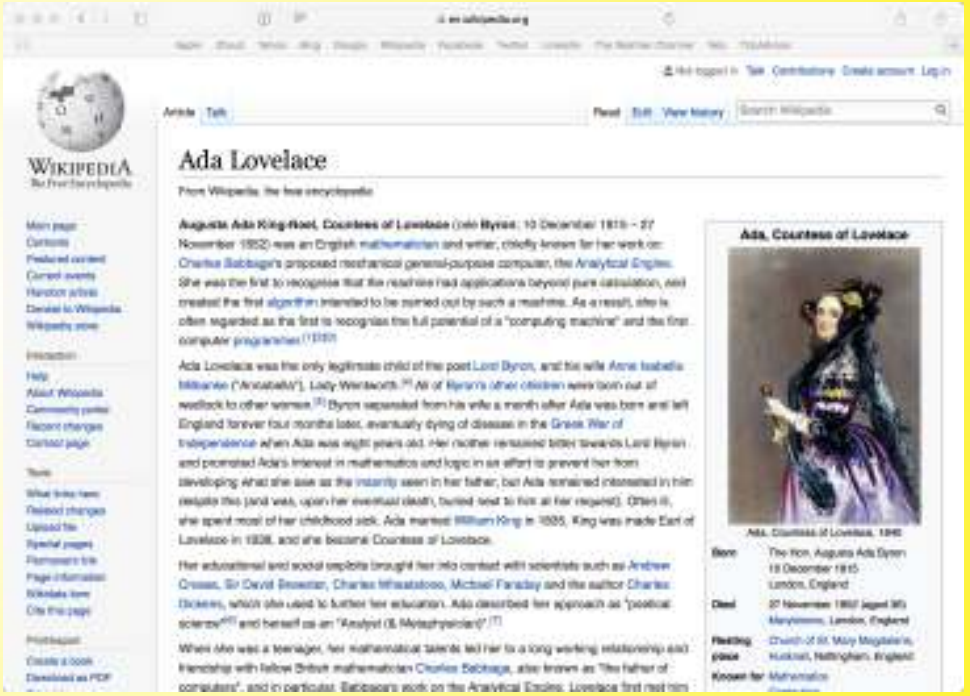
#reuse

serialization

#implications

Ada

Lady Ada Lovelace



Ada Lovelace

From Wikipedia, the free encyclopedia

Augusta Ada King-Noel, Countess of Lovelace (née **Byron**; 10 December 1815 – 27 November 1852) was an English mathematician and writer, chiefly known for her work on Charles Babbage's proposed mechanical general-purpose computer, the Analytical Engine. She was the first to recognise that the machine had applications beyond pure calculation, and created the first algorithm intended to be carried out by such a machine. As a result, she is often regarded as the first to recognise the full potential of a "computing machine" and the first computer programmer.[1][2][3]

Ada Lovelace was the only legitimate child of the poet Lord Byron, and his wife Anne Isabella Milbanke ("Annabella"), Lady Wentworth.[4] All of Byron's other children were born out of wedlock to other women.[5] Byron separated from his wife a month after Ada was born and left England forever four months later, eventually dying of disease in the Greek War of Independence when Ada was eight years old. Her mother remained bitter towards Lord Byron and promoted Ada's interest in mathematics and logic in an effort to prevent her from developing what she saw as the insanity seen in her father, but Ada remained interested in him despite this (and was, upon her eventual death, buried next to him at her request). Often ill, she spent most of her childhood sick. Ada married William King in 1835. King was made Earl of Lovelace in 1838, and she became Countess of Lovelace.

Her educational and social exploits brought her into contact with scientists such as Andrew Crosse, Sir David Brewster, Charles Wheatstone, Michael Faraday and the author Charles Dickens, which she used to further her education. Ada described her approach as "poetical science"[6][7] and herself as an "Analyst (& Metaphysician)".[7]

When she was a teenager, her mathematical talents led her to a long working relationship and friendship with fellow British mathematician Charles Babbage, also known as "the father of computers", and in particular, Babbage's work on the Analytical Engine. Lovelace first met him

Ada, Countess of Lovelace

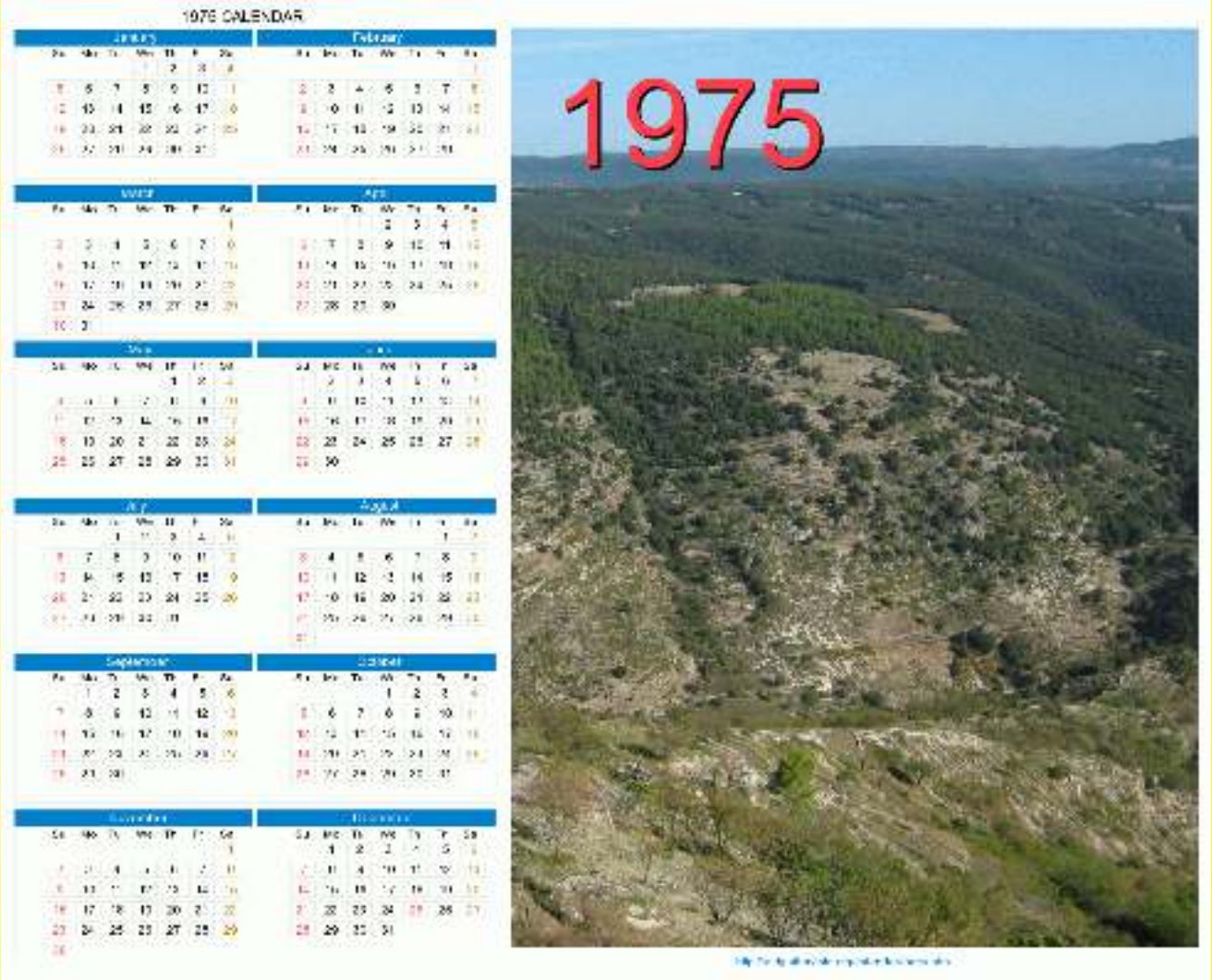
Ada, Countess of Lovelace, 1840

Born	The Hon. Augusta Ada Byron 10 December 1815 London, England
Died	27 November 1852 (aged 36) Marylebone, London, England
Resting place	Church of St. Mary Magdalene, Hucknall, Nottingham, England
Known for	Mathematics

Ada

Ada

HOLWG



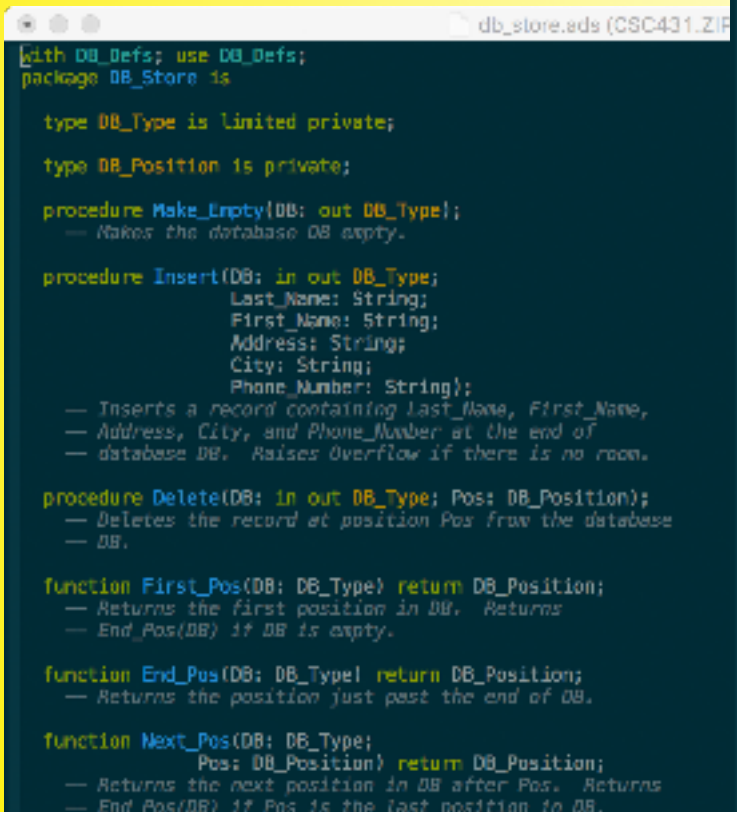
Ada

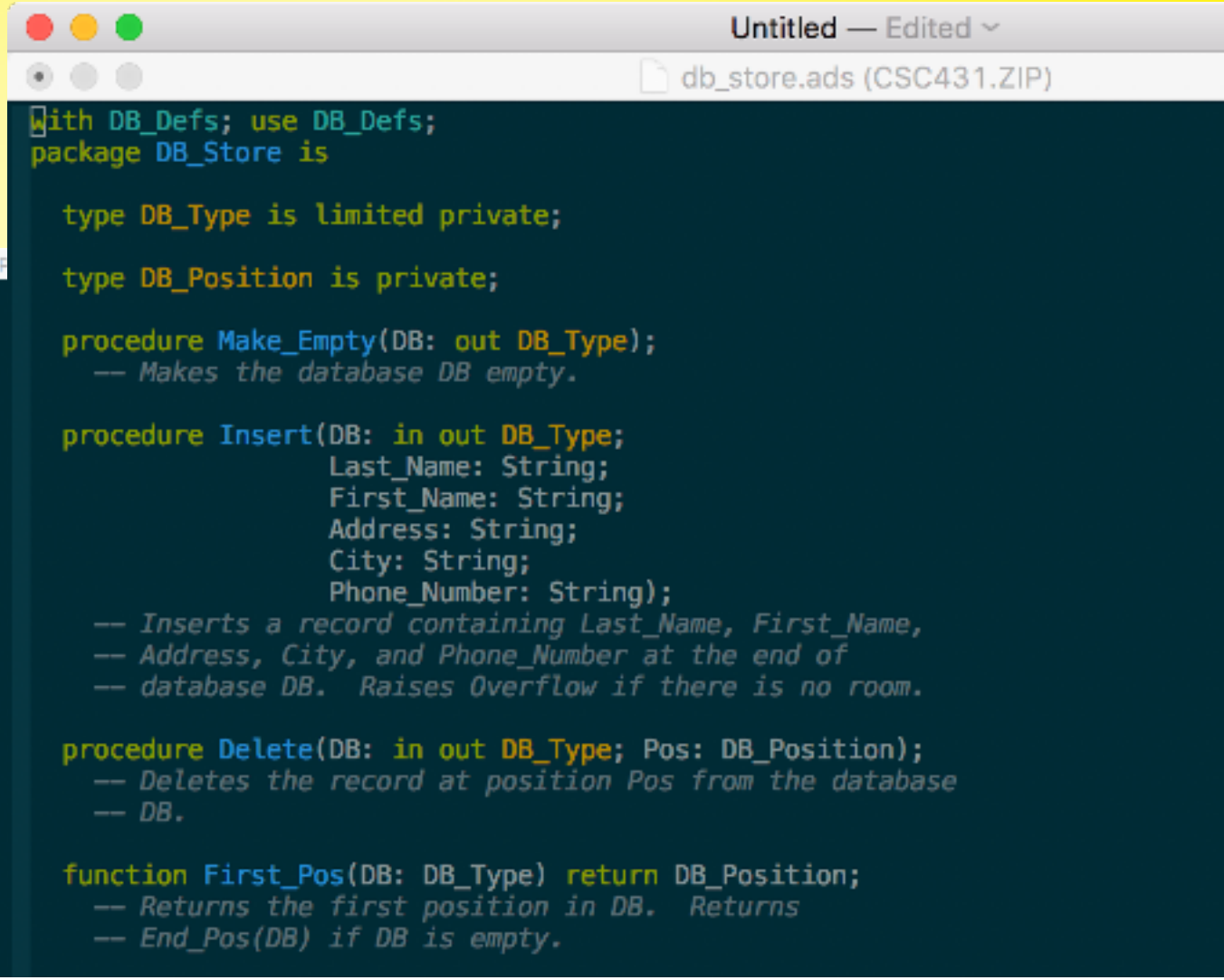
Ada

db_store.ads (CSC431.ZIP)

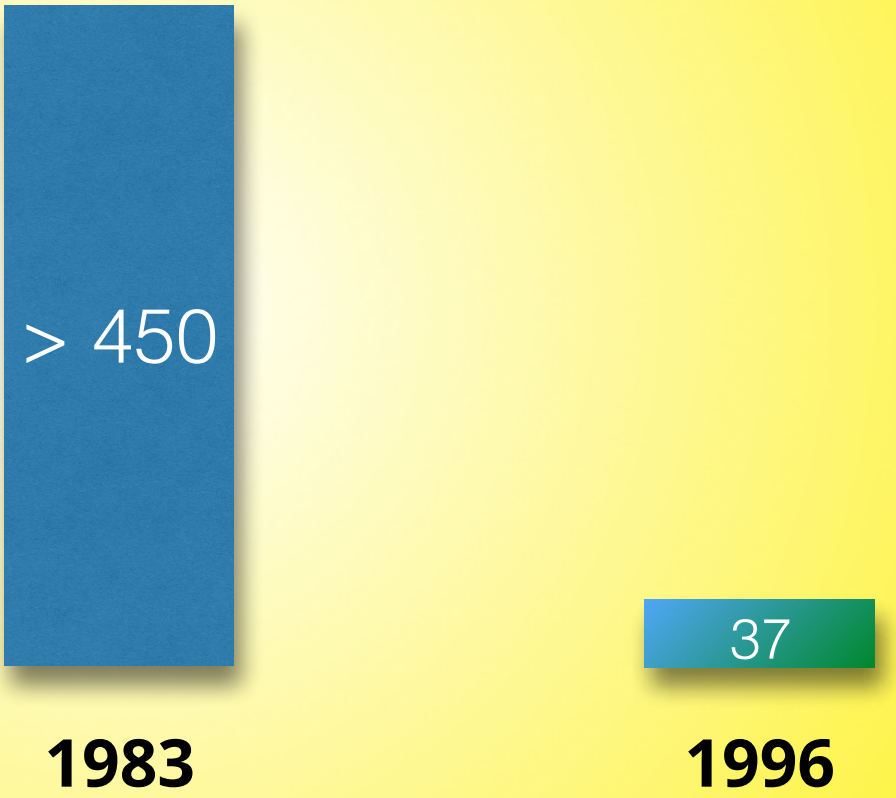
```
with DB_Defs; use DB_Defs;
package DB_Store is

  type DB_Type is limited private;

  type DB_Position is private;

  procedure Make_Empty(DB: out DB_Type);
    -- Makes the database DB empty.

  procedure Insert(DB: in out DB_Type;
                   Last_Name: String;
                   First_Name: String;
                   Address: String;
                   City: String;
                   Phone_Number: String);
    -- Inserts a record containing Last_Name, First_Name,
    -- Address, City, and Phone_Number at the end of
    -- database DB.  Raises Overflow if there is no room.

  procedure Delete(DB: in out DB_Type; Pos: DB_Position);
    -- Deletes the record at position Pos from the database
    -- DB.

  function First_Pos(DB: DB_Type) return DB_Position;
    -- Returns the first position in DB.  Returns
    -- End_Pos(DB) if DB is empty.

  function End_Pos(DB: DB_Type) return DB_Position;
    -- Returns the position just past the end of DB.

  function Next_Pos(DB: DB_Type;
                    Pos: DB_Position) return DB_Position;
```

Ada


db_store.ads (CSC431.ZIP)
with DB_Defs; use DB_Defs;
package DB_Store is
type DB_Type is limited private;
type DB_Position is private;
procedure Make_Empty(DB: out DB_Type);
-- Makes the database DB empty.
procedure Insert(DB: in out DB_Type;
Last_Name: String;
First_Name: String;
Address: String;
City: String;
Phone_Number: String);
-- Inserts a record containing Last_Name, First_Name,
-- Address, City, and Phone_Number at the end of
-- database DB. Raises Overflow if there is no room.
procedure Delete(DB: in out DB_Type; Pos: DB_Position);
-- Deletes the record at position Pos from the database
-- DB.
function First_Pos(DB: DB_Type) return DB_Position;
-- Returns the first position in DB. Returns
-- End_Pos(DB) if DB is empty.
function End_Pos(DB: DB_Type) return DB_Position;
-- Returns the position just past the end of DB.
function Next_Pos(DB: DB_Type;
Pos: DB_Position) return DB_Position;
-- Returns the next position in DB after Pos. Returns


Untitled — Edited
db_store.ads (CSC431.ZIP)
with DB_Defs; use DB_Defs;
package DB_Store is
type DB_Type is limited private;
type DB_Position is private;
procedure Make_Empty(DB: out DB_Type);
-- Makes the database DB empty.
procedure Insert(DB: in out DB_Type;
Last_Name: String;
First_Name: String;
Address: String;
City: String;
Phone_Number: String);
-- Inserts a record containing Last_Name, First_Name,
-- Address, City, and Phone_Number at the end of
-- database DB. Raises Overflow if there is no room.
procedure Delete(DB: in out DB_Type; Pos: DB_Position);
-- Deletes the record at position Pos from the database
-- DB.
function First_Pos(DB: DB_Type) return DB_Position;
-- Returns the first position in DB. Returns
-- End_Pos(DB) if DB is empty.

DOD Languages Used




1997 CALENDAR
JANUARY
FEBRUARY
MARCH
APRIL
MAY
JUNE
JULY
AUGUST
SEPTEMBER
OCTOBER
NOVEMBER
DECEMBER

COTS

#ubuiltit

#uownit

#standardization

(#forced)

#overengineering

Ariane 5

flight data (64 bit):

0110101101011010011010110101101001101011010110100110101101011010

guidance system (16 bit):

0110101101011010

flight data (64 bit):

Some of those bits might have been important!

guidance system (16 bit):

0110101101011010011010110101101001101011010110100110101101011010

guidance system (16 bit):

v5 faster than v4

#reuse

guidance system (16 bit):

only used on the ground...

...kept on for first 40s of flight...

#debuggingNproduction

#debuggingNproduction

#legacy

#abstractiondistraction

2005

increased tech stack complexity

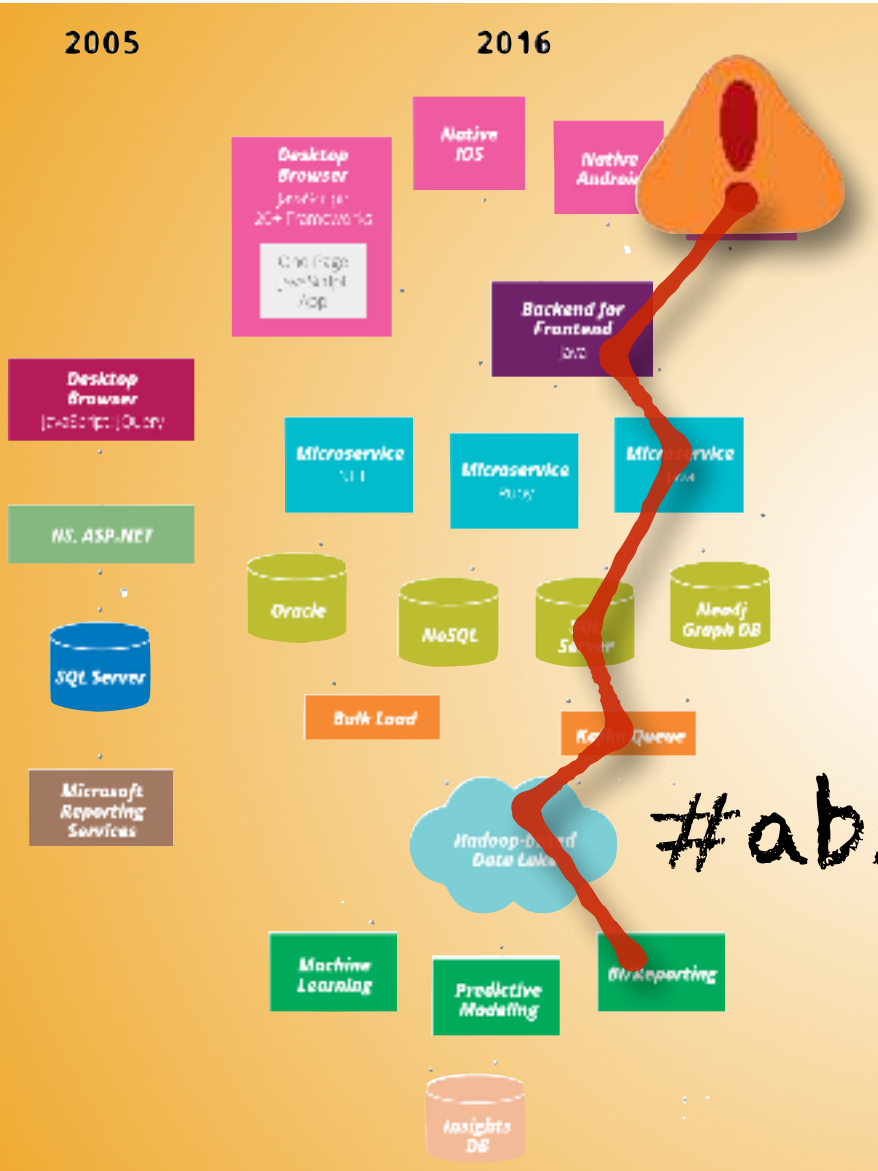
#abstractiondistraction

https://www.thoughtworks.com/insights/blog/implications-tech-stack-complexity-executives

increased tech stack complexity

#abstractiondistraction

https://www.thoughtworks.com/insights/blog/implications-tech-stack-complexity-executives



#abstractiondistraction

https://www.thoughtworks.com/insights/blog/implications-tech-stack-complexity-executives

#debuggingNproduction

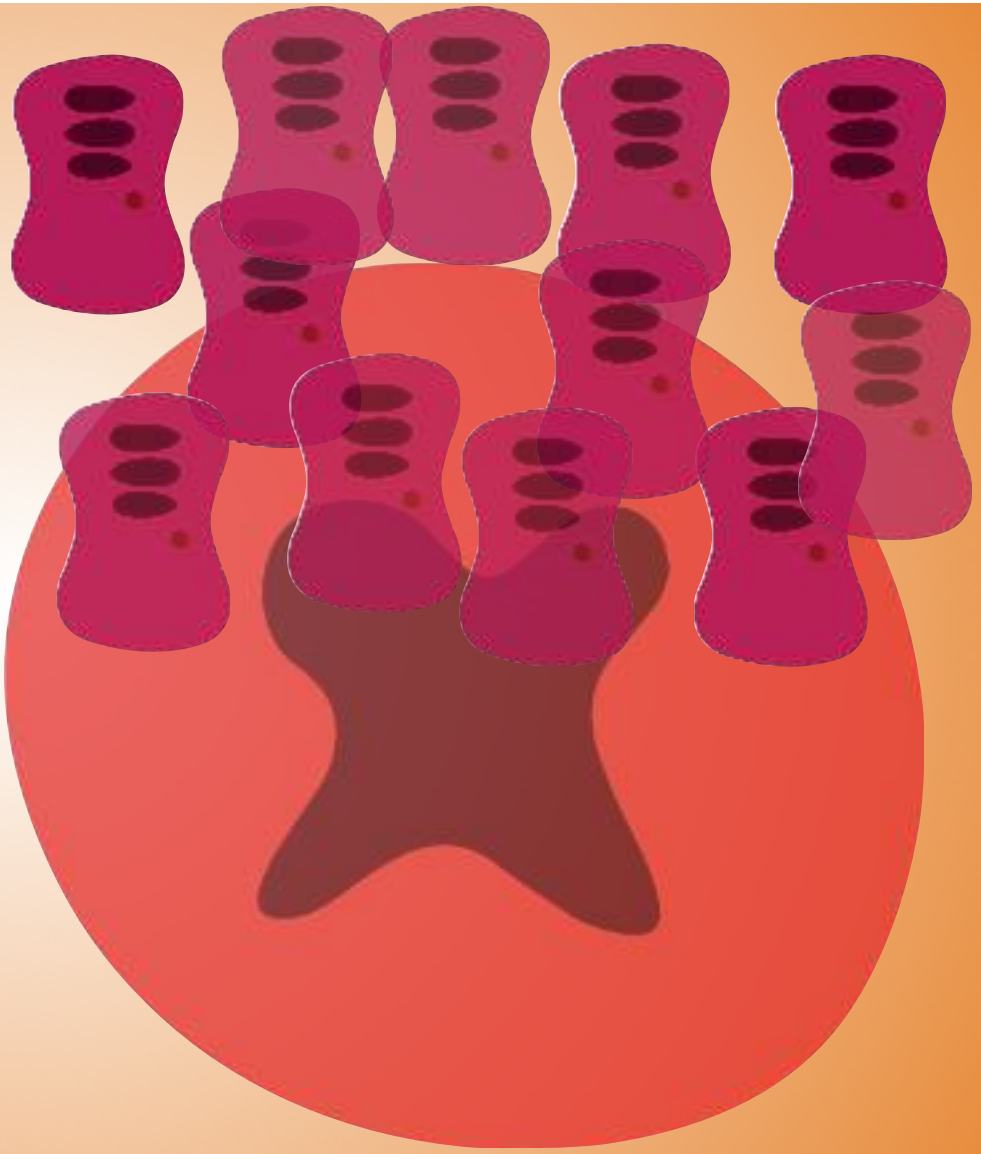
#legacy

#abstractiondistraction

an `int` is an `int`...


pets.com


pets.com



#toolittleinfrastructure


webvan


webvan.com
webvan.com

why webvan failed

why webvan failed

why webvan failed

#toomuchinfrastructure

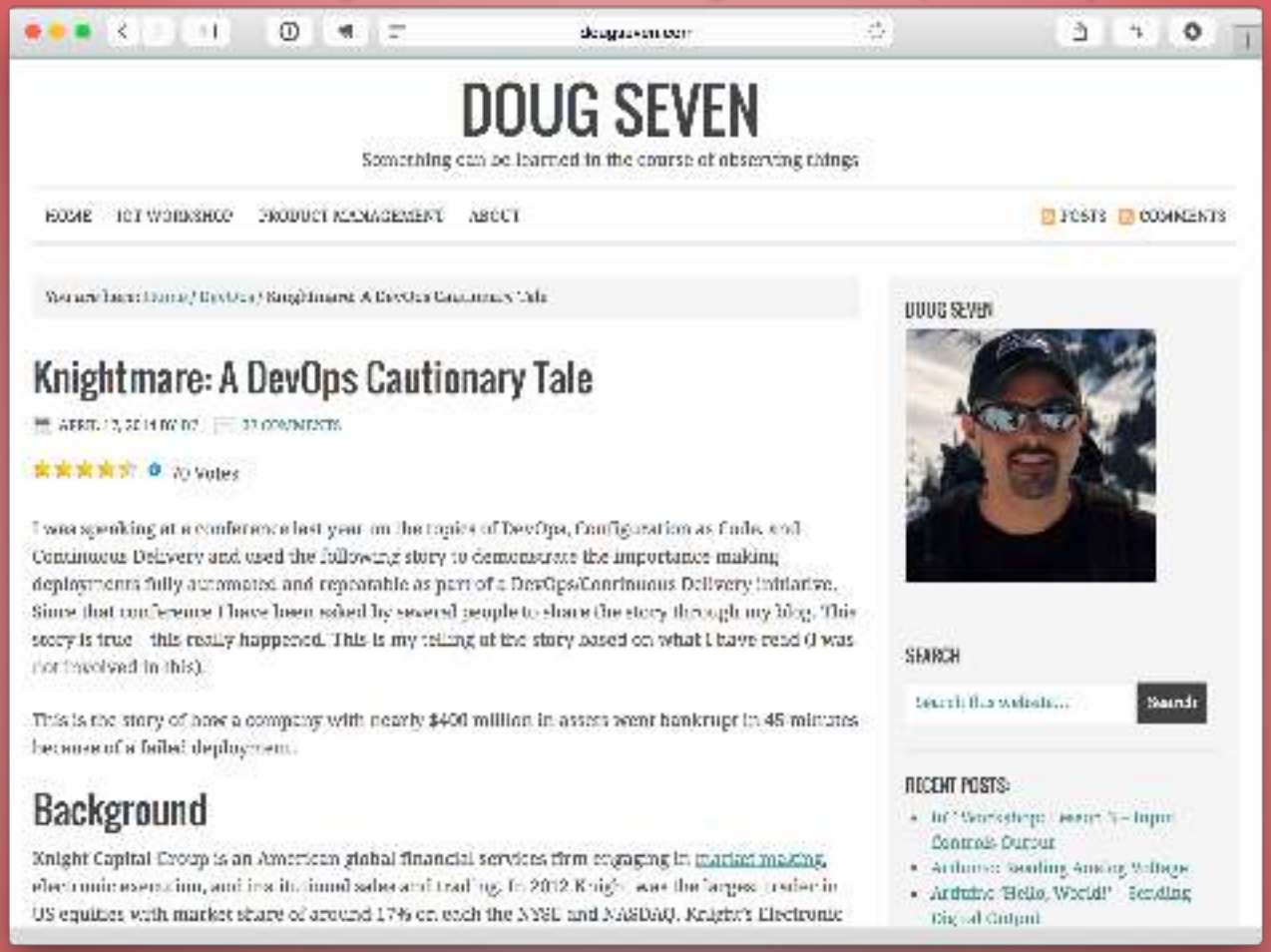
#toomuchinfrastructure

Let’s build all the
frameworks we’ll need first!

#misunderstanding"-ilities"

Knight Capital

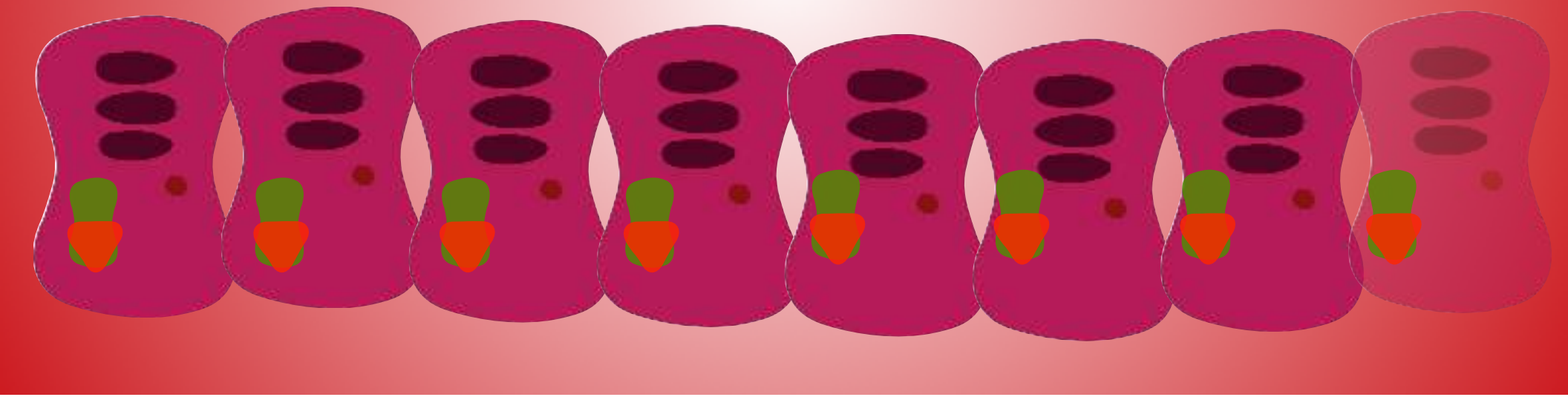
dougseven.com/2014/04/17/knightmare-a-devops-cautionary-tale/



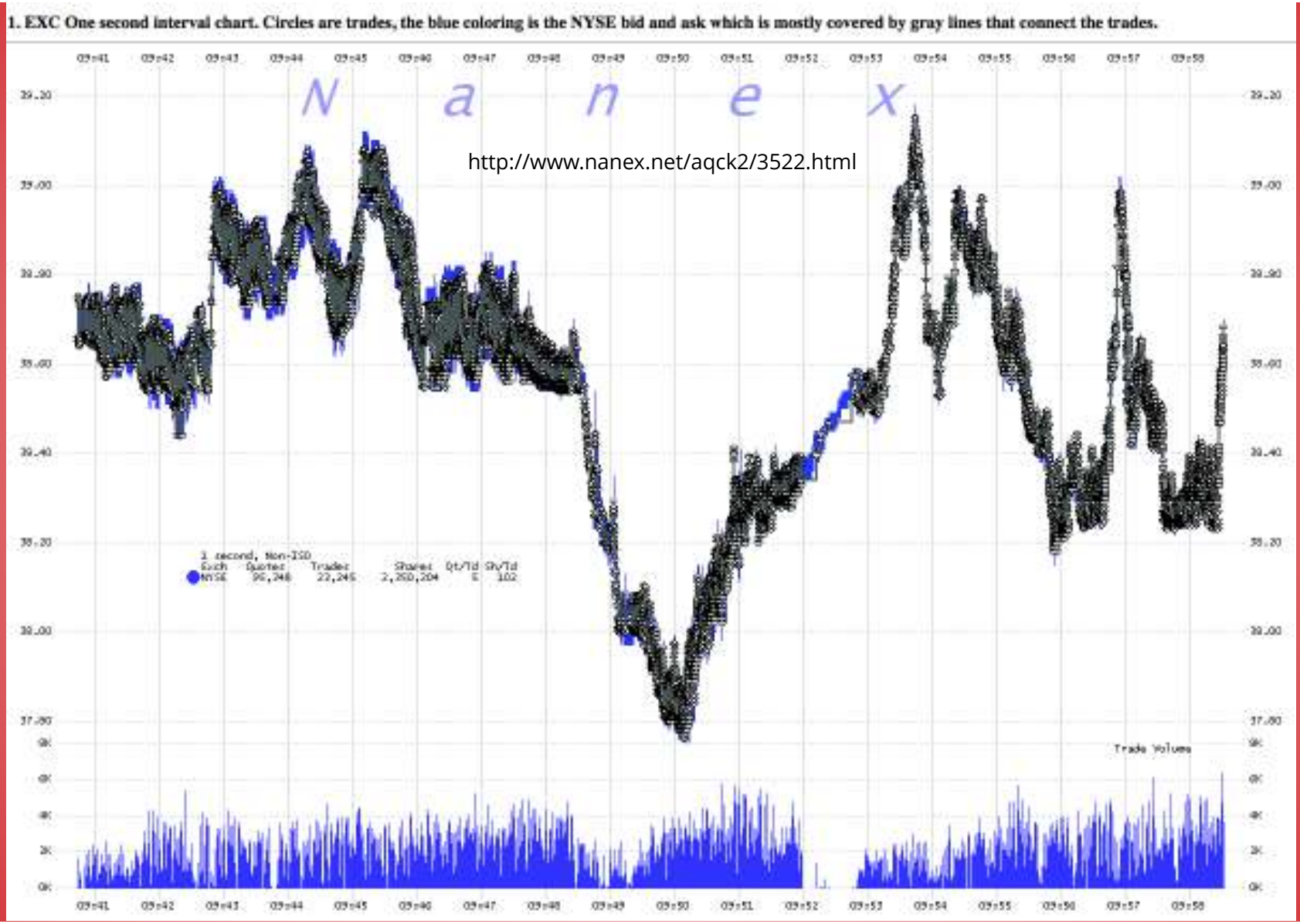
"bankrupt in 45 minutes"

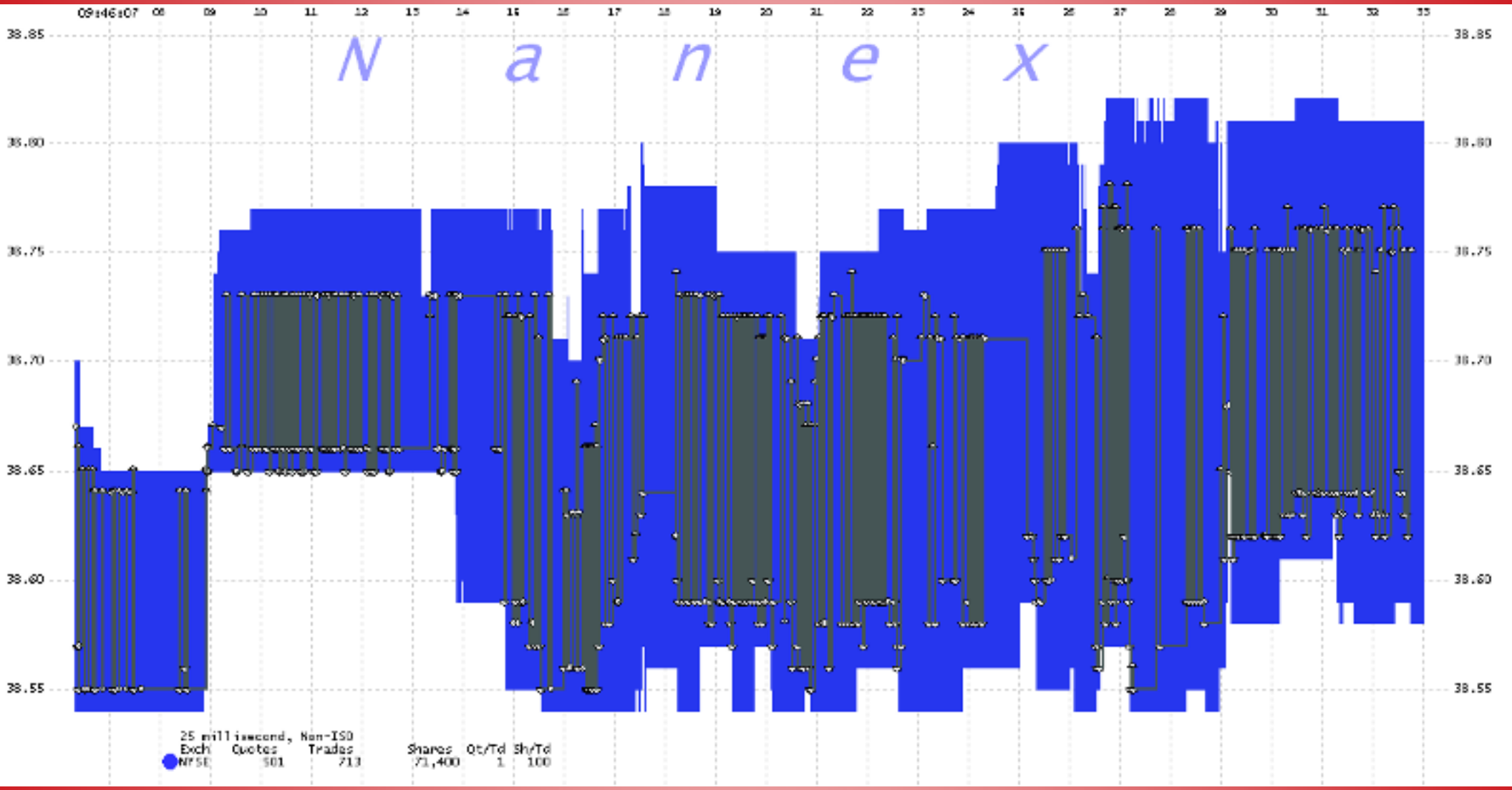
SMARS

PowerPeg



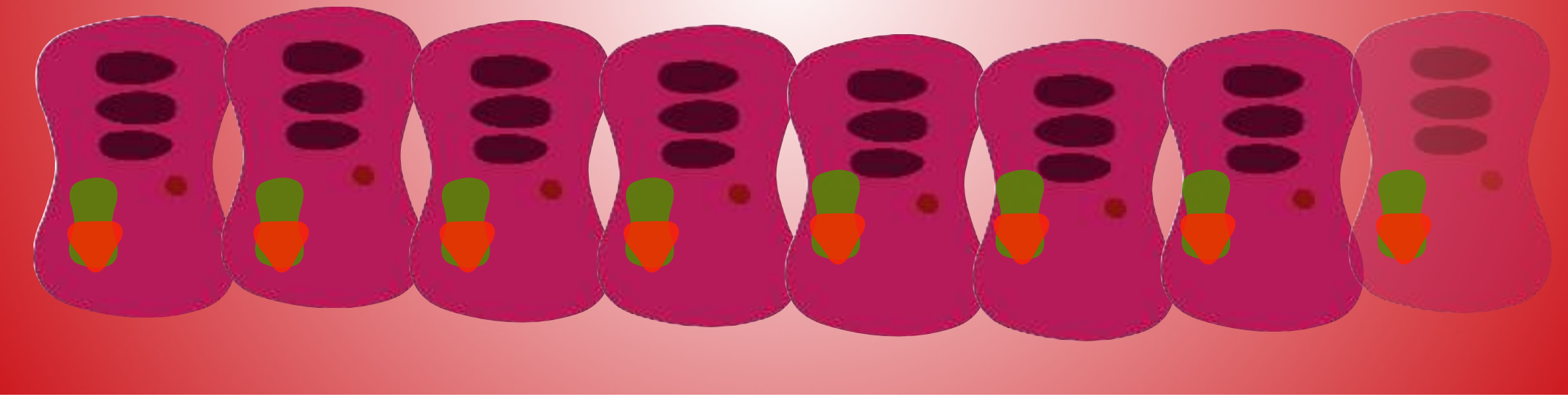
1. EXC One second interval chart. Circles are trades, the blue coloring is the NYSE bid and ask which is mostly covered by gray lines that connect the trades.




09:46:07
38.85
38.80
38.75
38.70
38.65
38.60
38.55
Nanex
25 millisecond, Non-ISO
Exch Quotes Trades Shares Qt/Td Sh/Td
NYSE 501 713 71,400 1 100

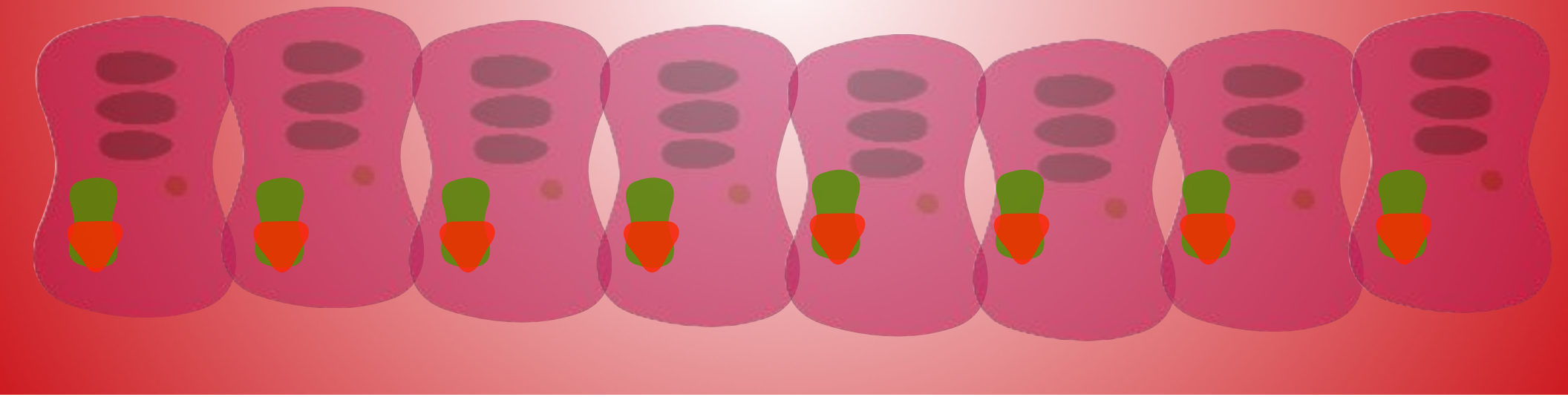
SMARS

PowerPeg



SMARS

PowerPeg



#doyouevenDevOps?

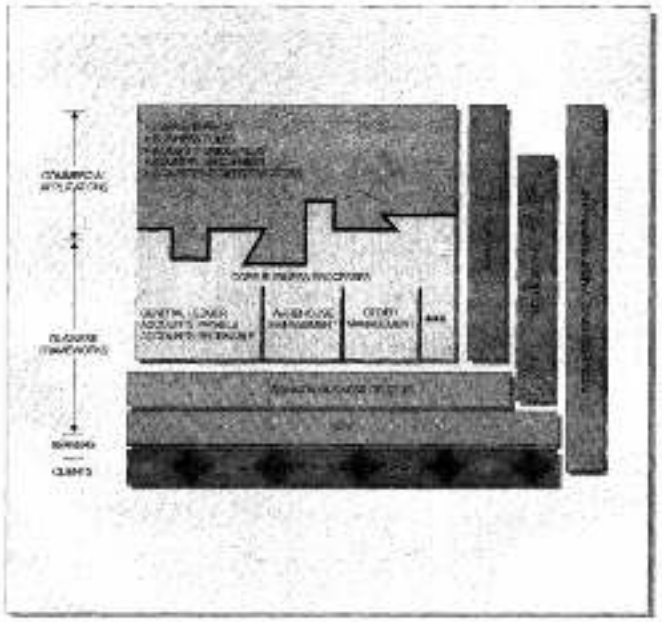
clean up technical debt

#reusegonebad

San Francisco Project

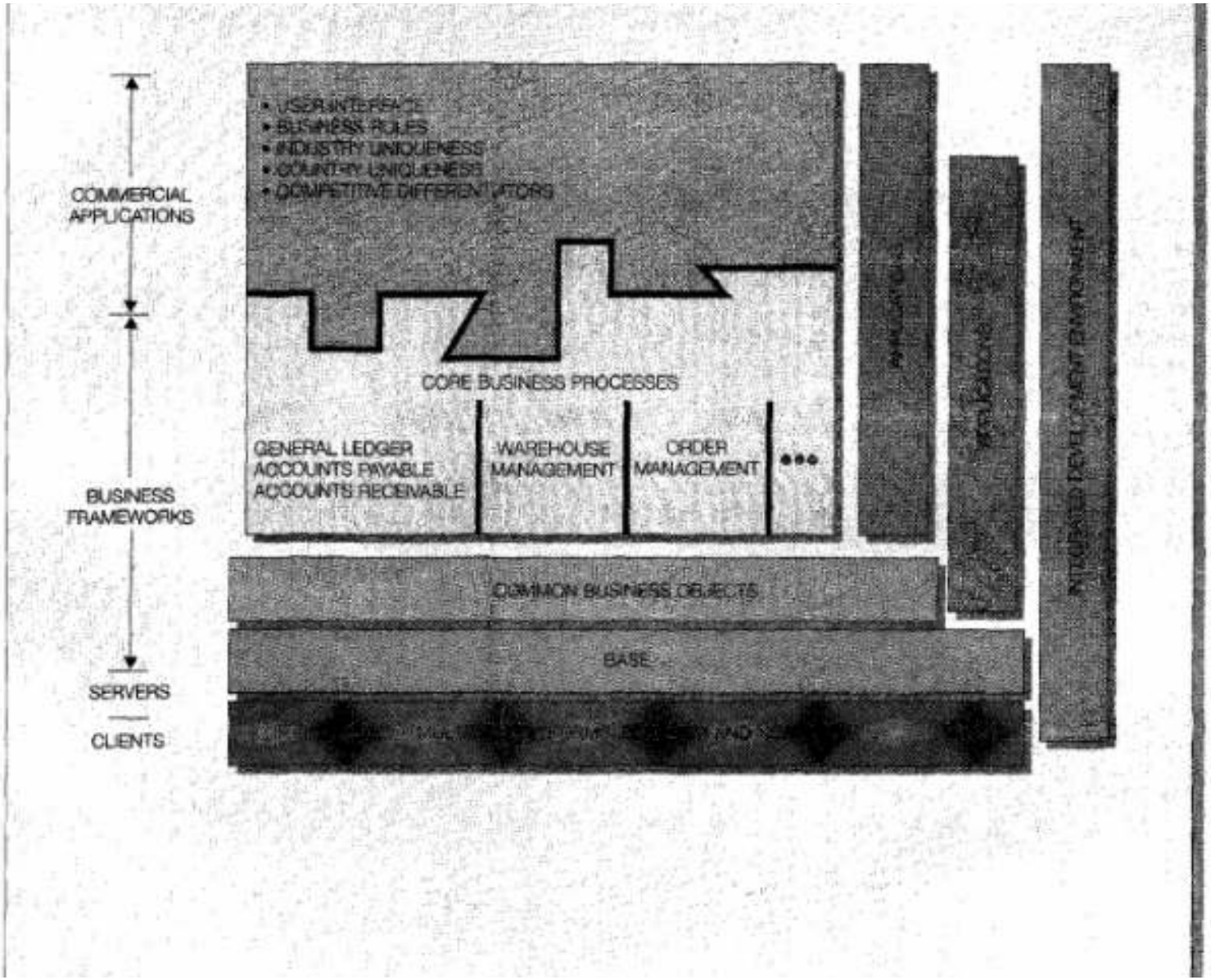
spoken by someone who has never implemented a real-world business solution

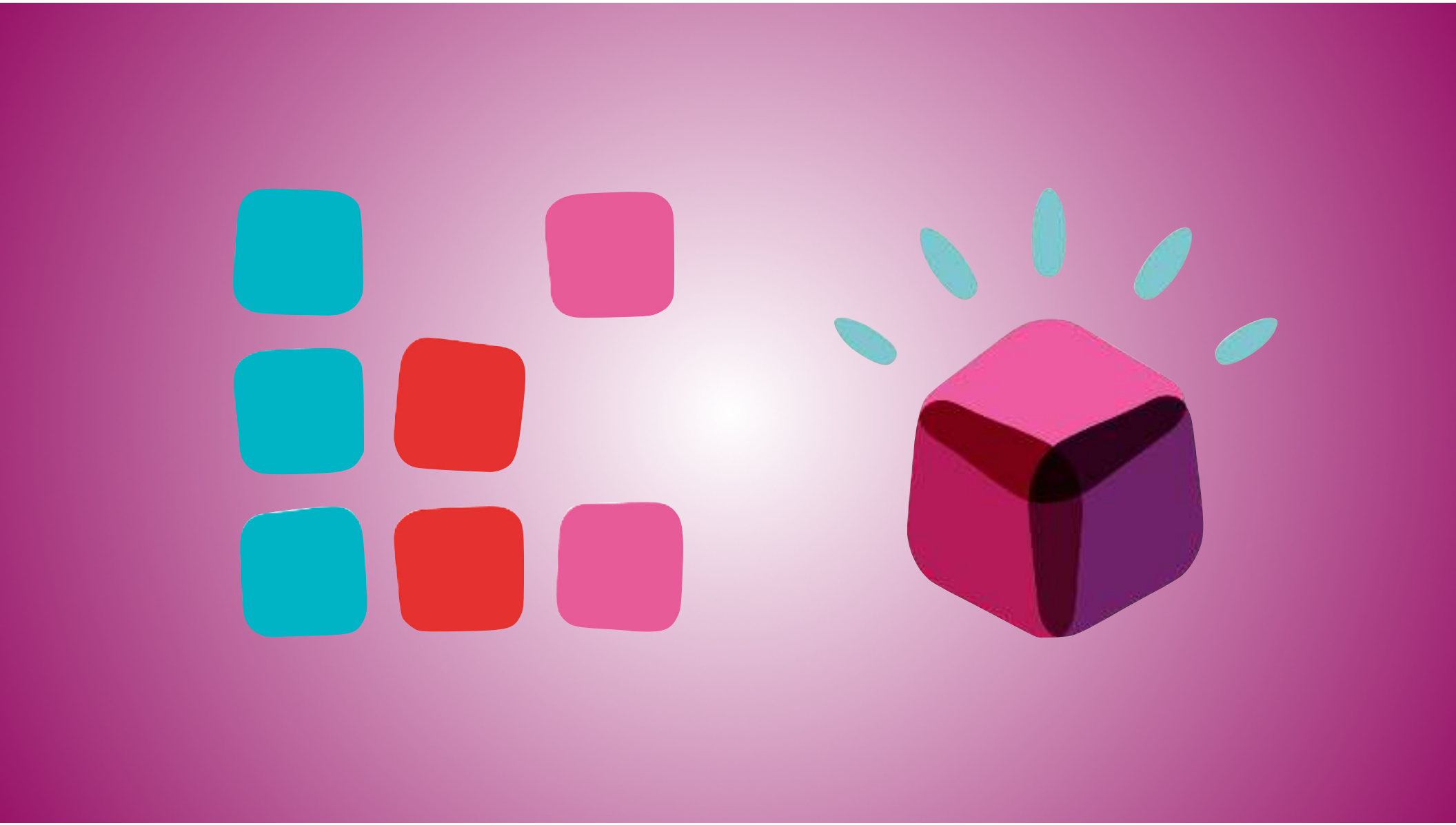
Figure 1 IBM Business Frameworks

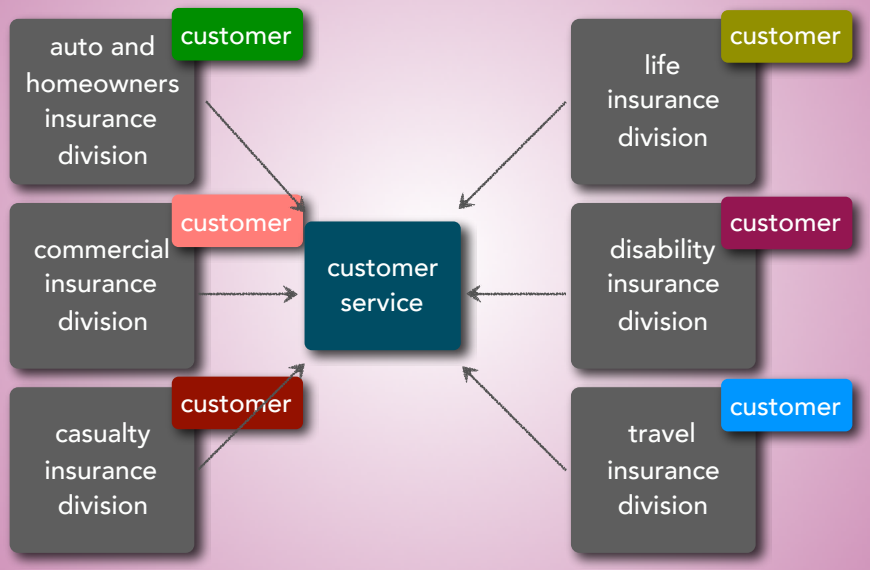


Two categories of functions in the base layer are directly visible to developers: base object model interfaces and utilities. To support distributed, transactional applications, the base layer also provides a set of kernel services. In most cases, the kernel services are not directly visible to developers. Instead they are invoked indirectly by the base object model interfaces. This approach helps to simplify the application programming model. It also will allow application developers to make use of new technology that IBM might incorporate into the infrastructure without modifying their application code. The idea is that application developers one would retain control, only the underlying implementation of the infrastructure would change.

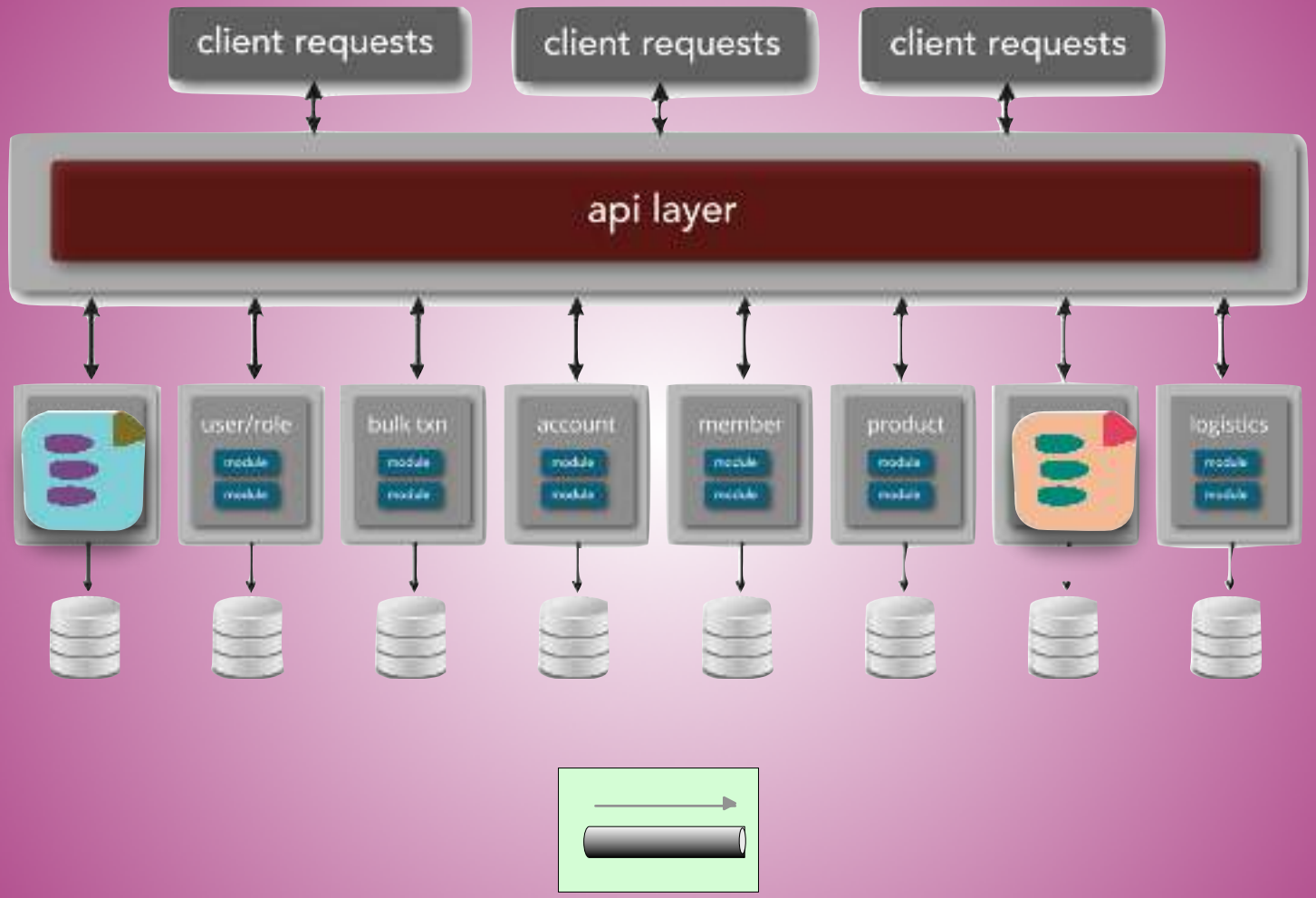
Many of the services in this layer are based on object services definitions from the Object Management Group (OMG). For example, the license service provides an object transaction service, collection inter-


COMMERCIAL APPLICATIONS
BUSINESS FRAMEWORKS
SERVERS
CLIENTS
• USER INTERFACE
• BUSINESS RULES
• INDUSTRY UNIQUENESS
• COUNTRY UNIQUENESS
• COMPETITIVE DIFFERENTIATORS
CORE BUSINESS PROCESSES
GENERAL LEDGER
ACCOUNTS PAYABLE
ACCOUNTS RECEIVABLE
WAREHOUSE MANAGEMENT
ORDER MANAGEMENT
COMMON BUSINESS OBJECTS
BASE




auto and homeowners insurance division
customer
commercial insurance division
customer
casualty insurance division
customer
customer service
life insurance division
customer
disability insurance division
customer
travel insurance division
customer

#canonicality


client requests
client requests
client requests
api layer
user/role
module
module
bulk txn
module
module
account
module
module
member
module
module
product
module
module
logistics
module
module

bounded context ≠ entity

The more *reusable* something is, the less *usable* it is.

— *Evolutionary Architectures*

http://evolutionaryarchitecture.com

Microservice is a *label*, not a *description*.

— *Martin Fowler*

https://martinfowler.com/articles/microservices.html

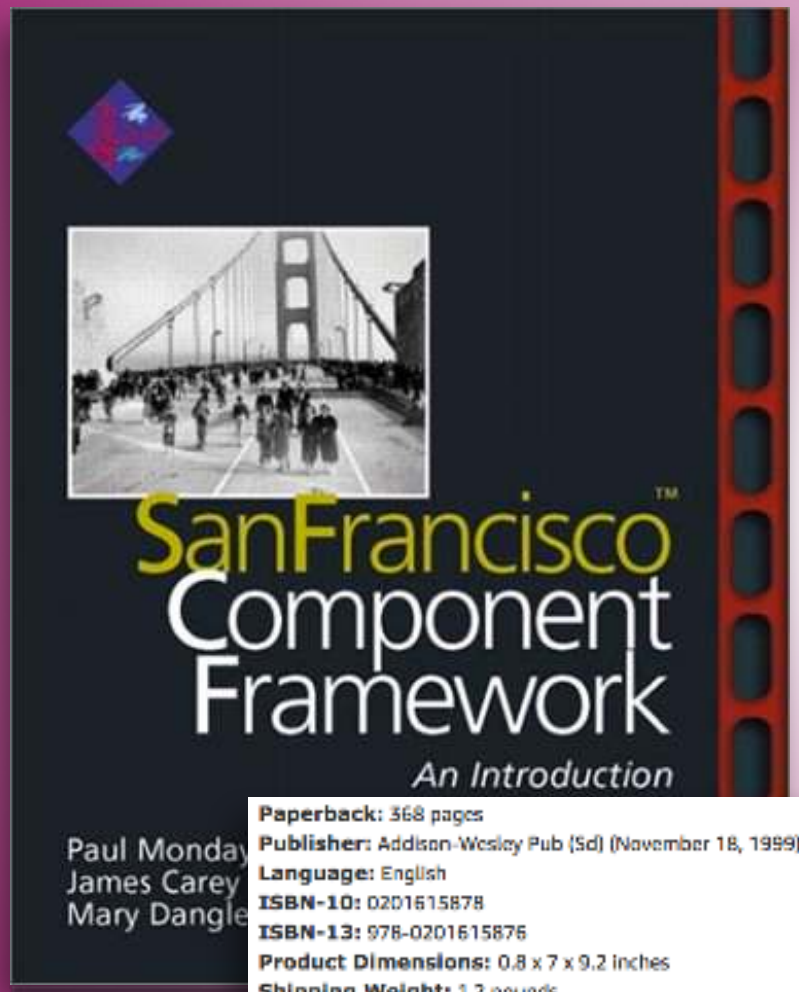
San Francisco Foundation and Utilities

San Francisco Common Business Objects

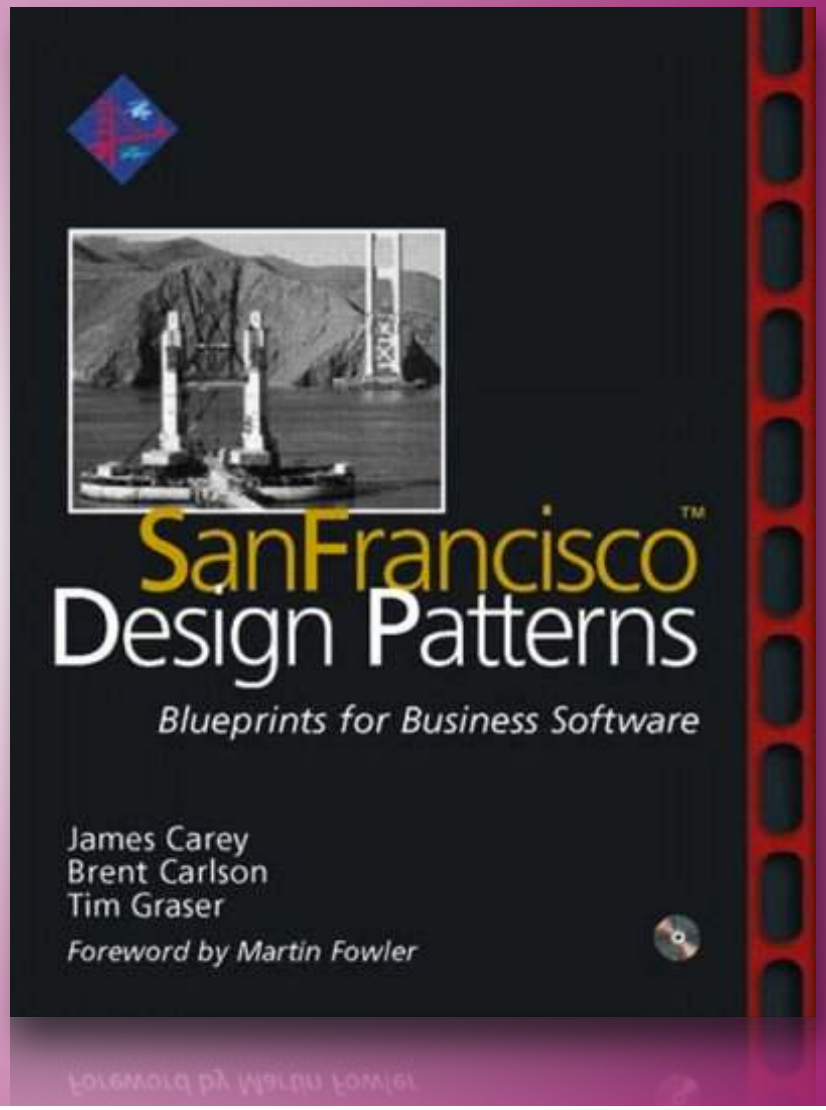
San Francisco Foundation and Utilities

San Francisco Common Business Objects

EJB



Paperback: 368 pages
Publisher: Addison-Wesley Pub (Sd) (November 18, 1999)
Language: English
ISBN-10: 0201615878
ISBN-13: 978-0201615876
Product Dimensions: 0.8 x 7 x 9.2 inches
Shipping Weight: 1.2 pounds
Average Customer Review: ★★★★★★ (1 customer review)
Amazon Best Sellers Rank: #3,425,499 in Books (See Top 100 in Books)



#last10%rule

#last10%rule

"Users always want 100% of what they want (& are never satisfied with less)."

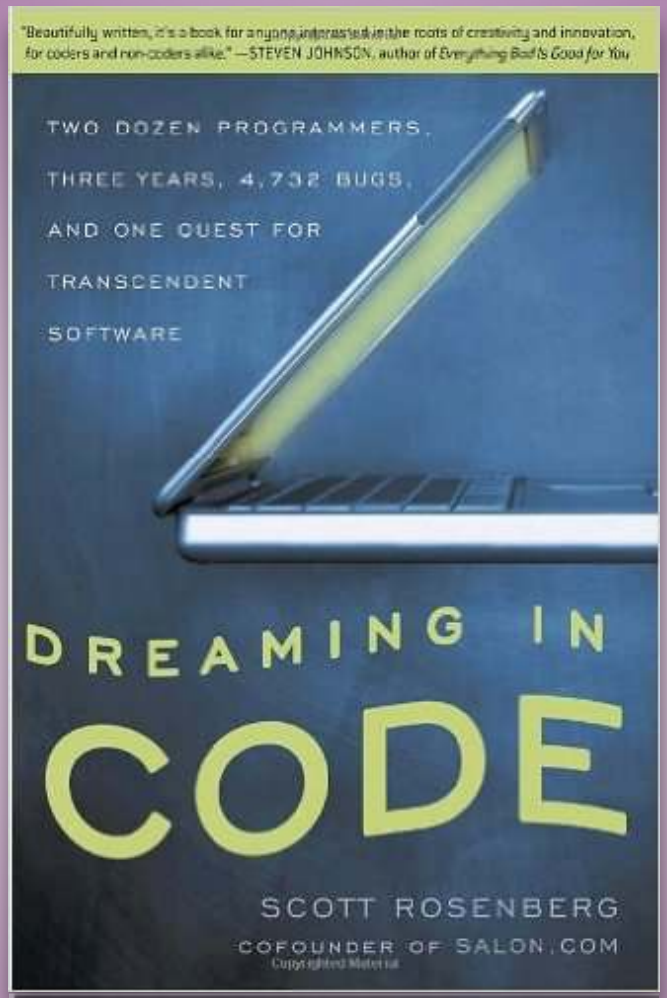
what the user wants

#last10%rule

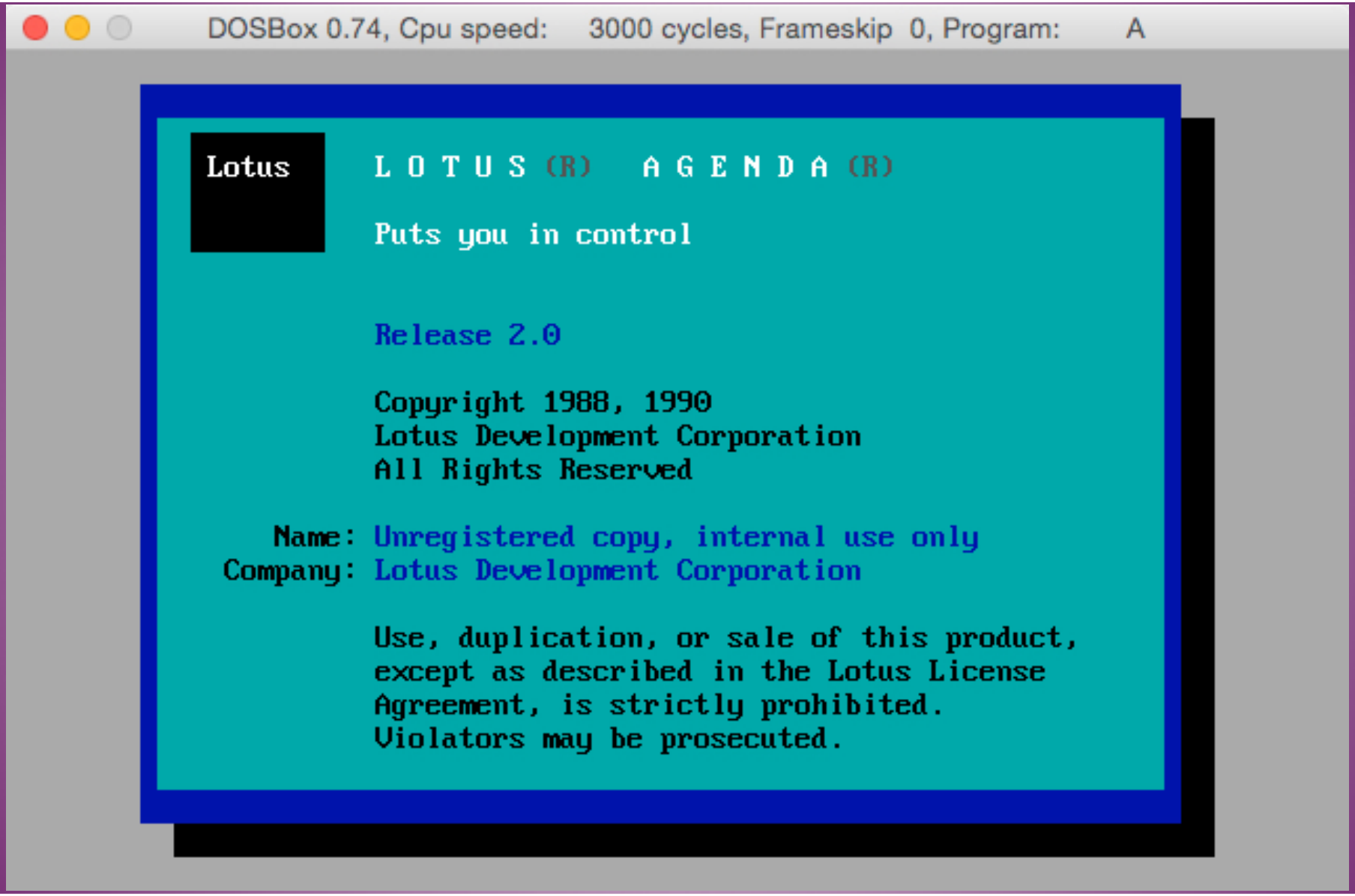
#reuse

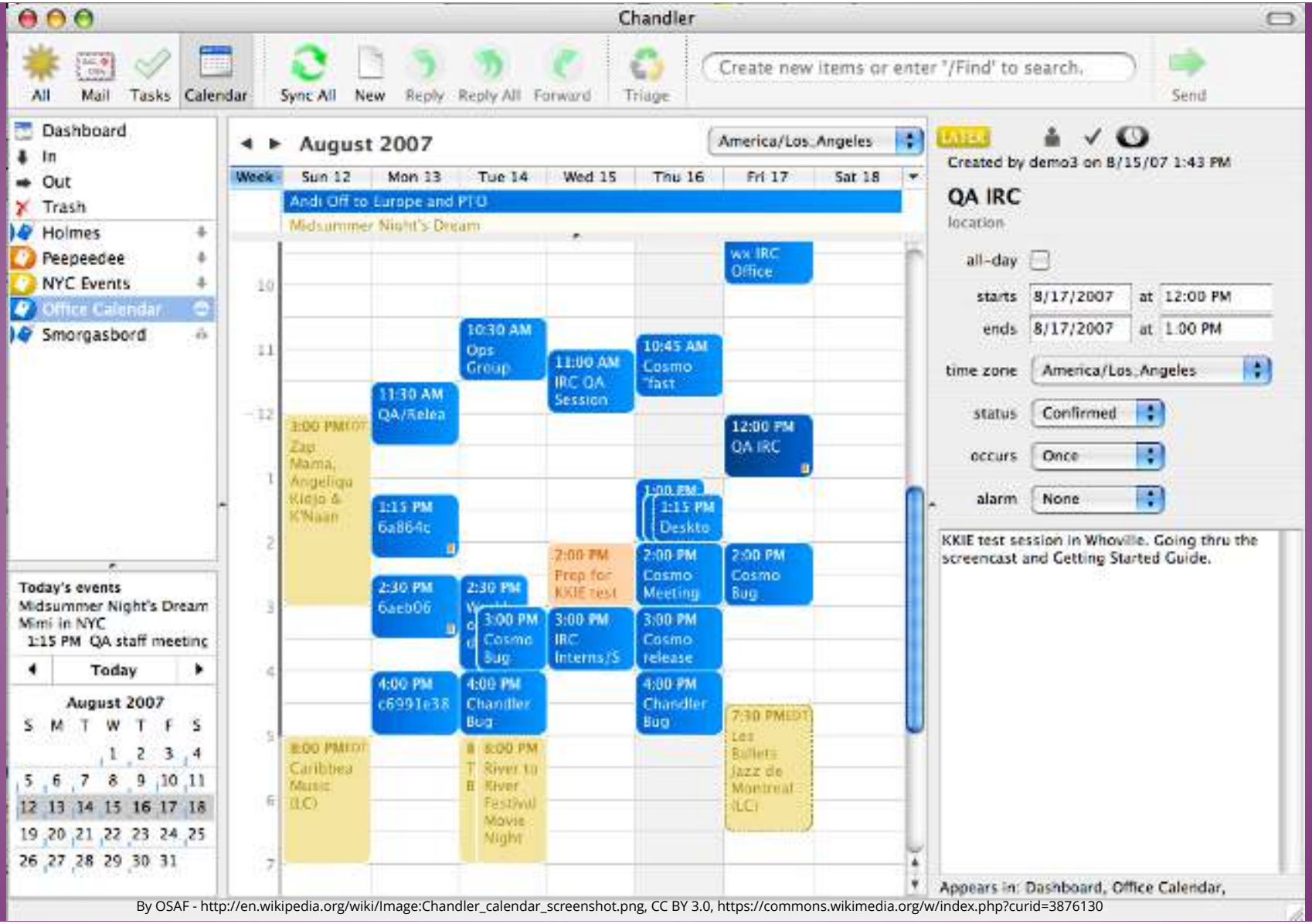
Chandler Project


infinite time
infinite resources
no legacy



https://www.amazon.com/Dreaming-Code-Programmers-Transcendent-Software/dp/1400082471


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: A
Lotus
L O T U S (R) A G E N D A (R)
Puts you in control
Release 2.0
Copyright 1988, 1990
Lotus Development Corporation
All Rights Reserved
Name: Unregistered copy, internal use only
Company: Lotus Development Corporation
Use, duplication, or sale of this product,
except as described in the Lotus License
Agreement, is strictly prohibited.
Violators may be prosecuted.



“We’ve consistently overinvested in infrastructure and design, the fruits of which won’t be realized in the next development cycle or even two—that is, not in the next six or twelve months.”

I’m more and more feeling like the art here is to do agile development without losing the long-term vision—and, frankly, I didn’t even define the problem as that to start with.

#metawork

is more interesting than *work*.

#metawork

is more interesting than *work*.

Sagrada Familia

#experimental

NETFLIX

LMAX

Vasa

Sagrada Familia

Tacoma Narrows

Chandler

F 16

SF Project

null

Knight Capital

Ada

webvan

Ariane 5

pets.com

#tradeoffs

#experimental

#scaling_architecture

#metawork

#feasibility

#canonicality

#legacy

Those who do not learn
history are doomed to repeat it.

—George Santayana

#doyouDevOps

#standardization

#toomuch

#reuse

#toolittle

Vasa

null

Chandler

#tradeoffs

SF Project

F 16

Ada

#tradeoffs

#experimental

#scaling_architecture

#metawork

#feasibility

#canonicality

#legacy

Those who do not learn
history are doomed to repeat it.

—George Santayana

#doyouDevOps

#standardization

#toomuch

#reuse

#toolittle

Chandler

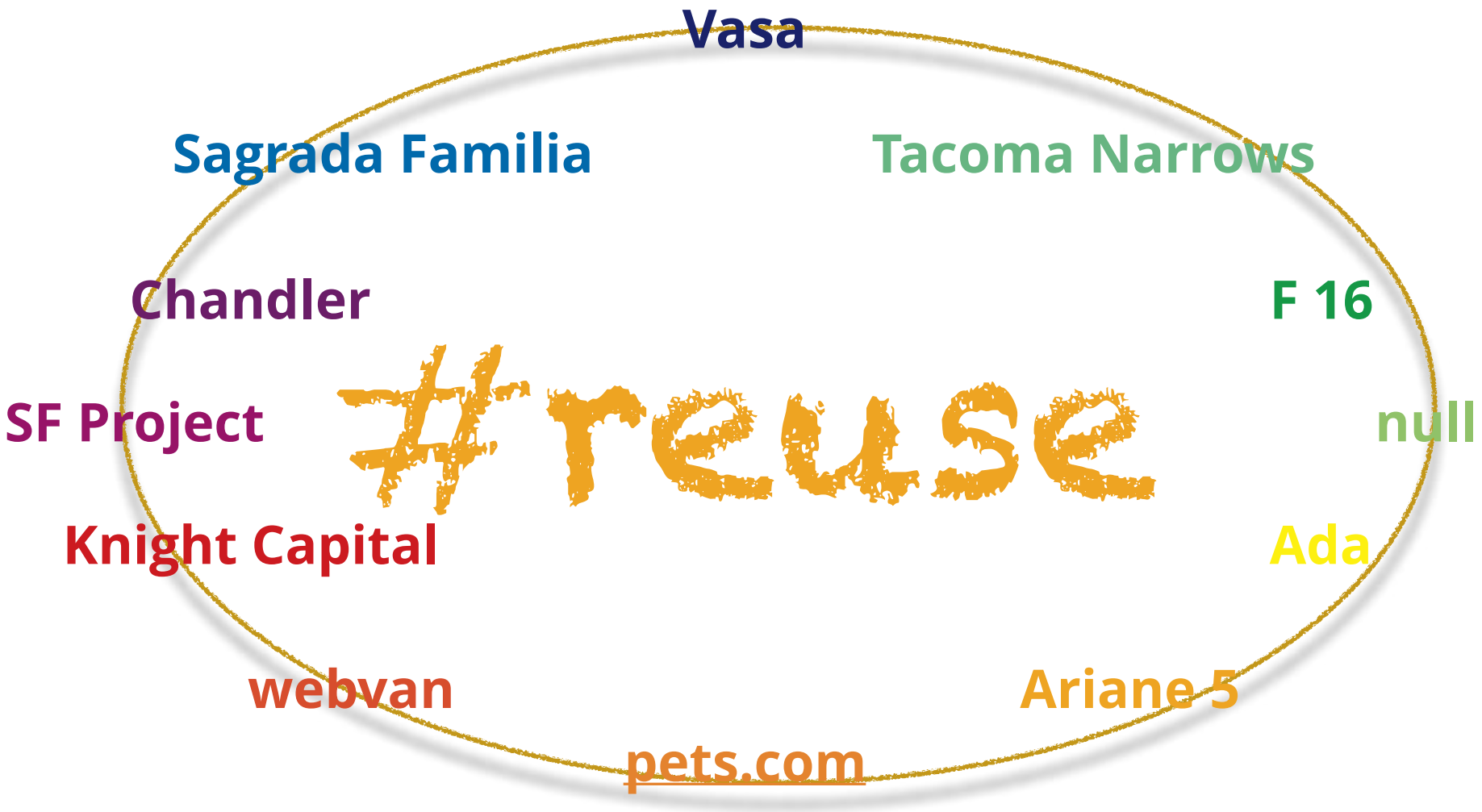
Envelopes

#metawork

SF Project

Ada

webvan


Vasa
Sagrada Familia
Tacoma Narrows
Chandler
F 16
SF Project
#reuse
null
Knight Capital
Ada
webvan
Ariane 5
pets.com

#reuse

design

Vasa

#reuse

design

Tacoma Narrows

#reuse

old code

null

#reuse

everything

Ada

#reuse

old code

Ariane 5

#reuse

not enough!

pets.com

#reuse

invest too early

webvan.com

#reuse

old toggles

Knight Capital

#reuse

San Francisco Project

#reuse

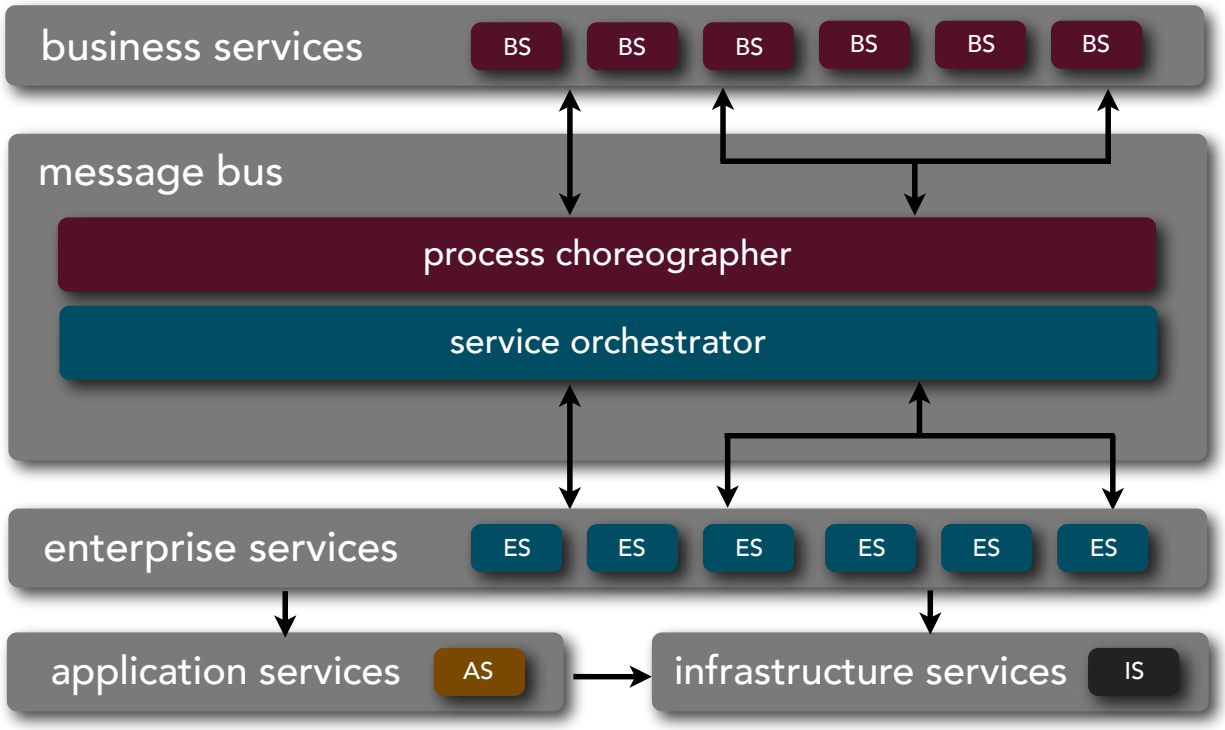
frameworks

design

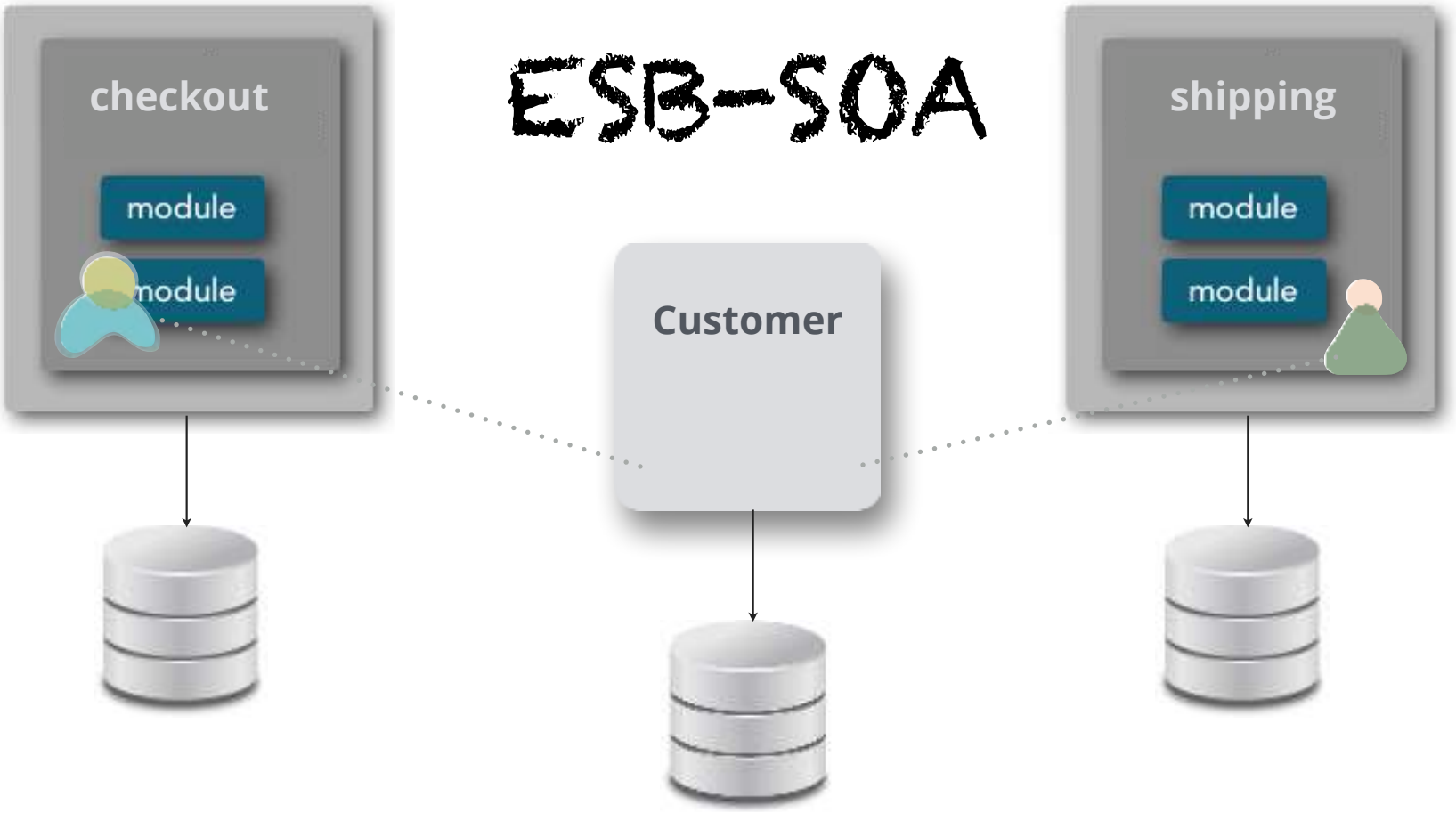
platforms code

Chandler

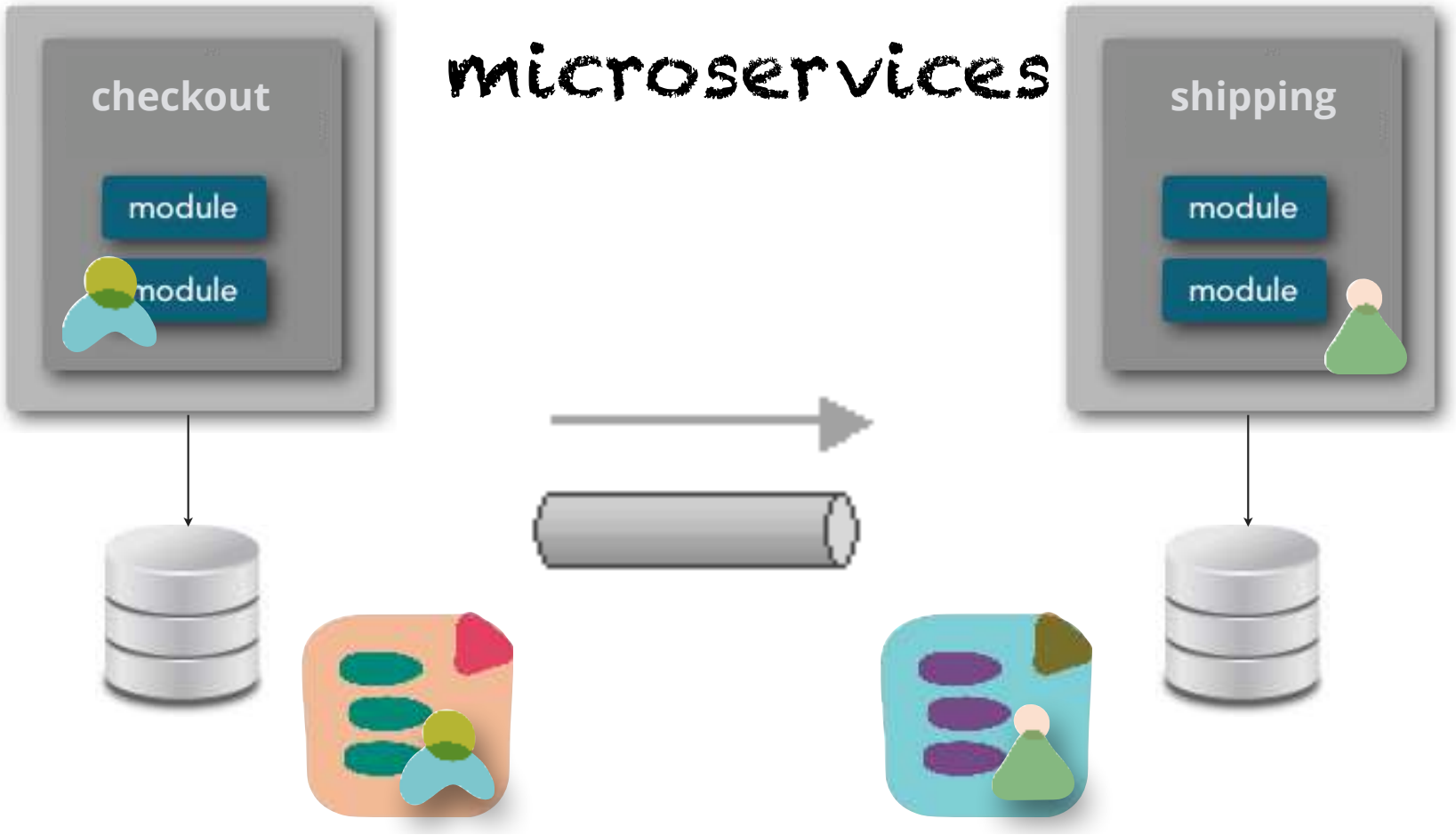
ESB-SOA

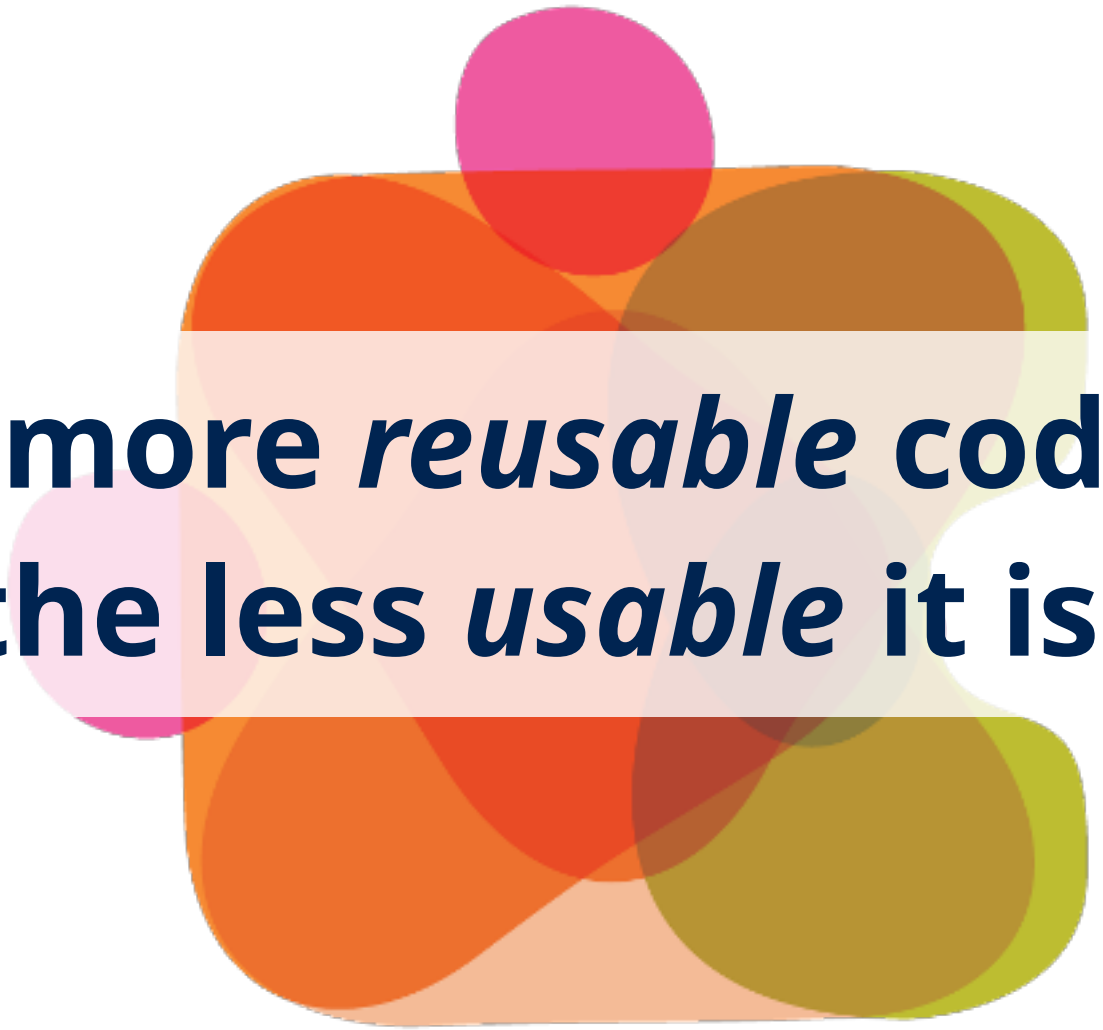

business services
BS
BS
BS
BS
BS
BS
message bus
process choreographer
service orchestrator
enterprise services
ES
ES
ES
ES
ES
ES
application services
AS
infrastructure services
IS

Code Reuse (Over Time)



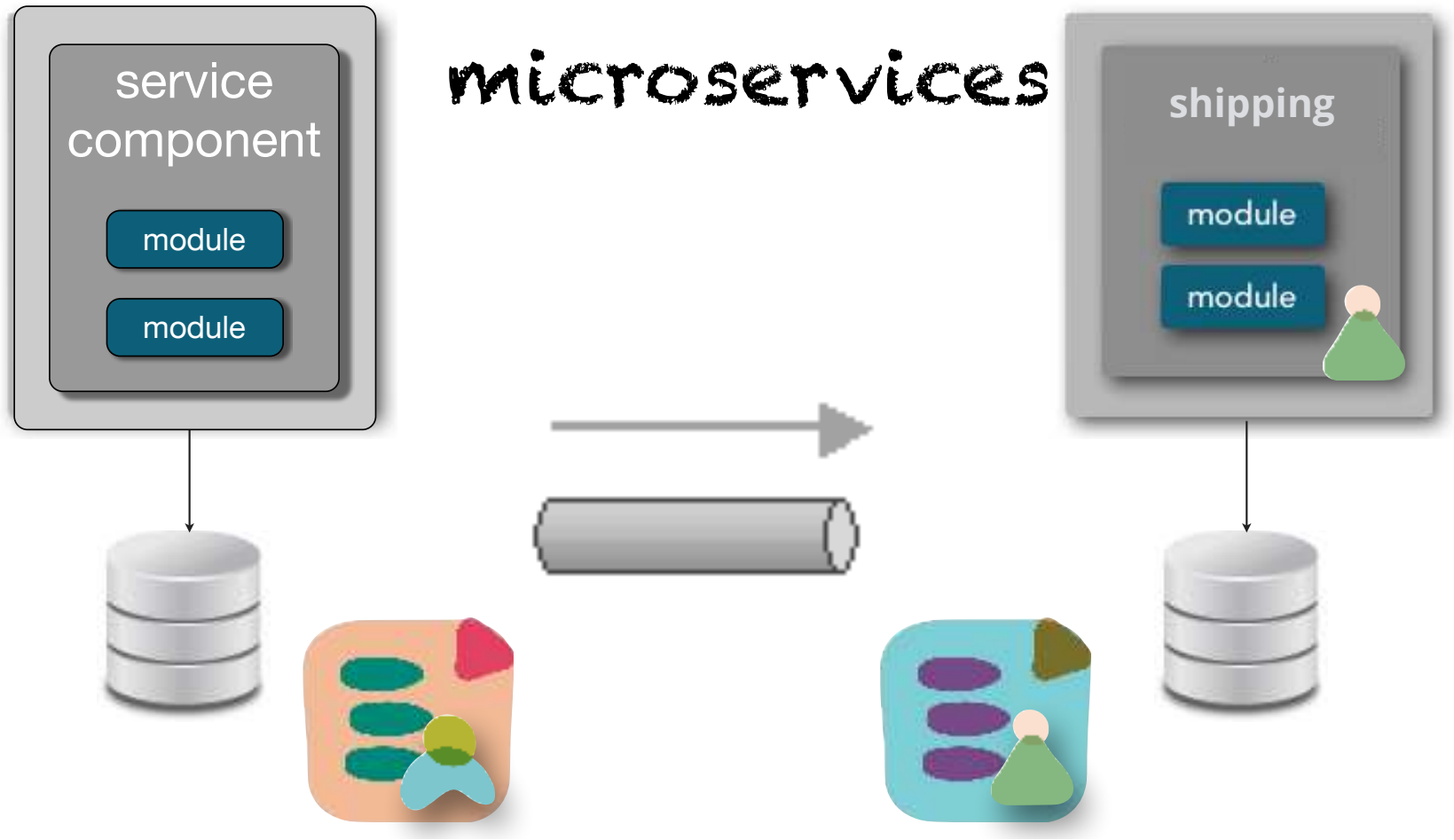
Code Reuse (Over Time)

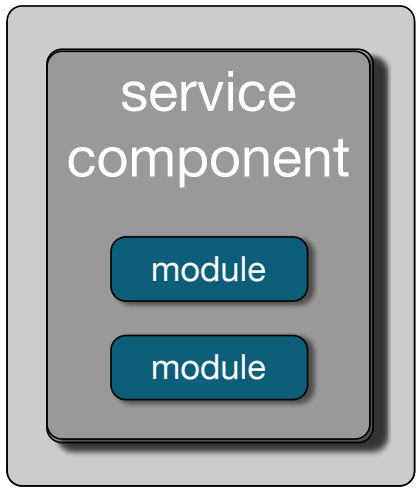




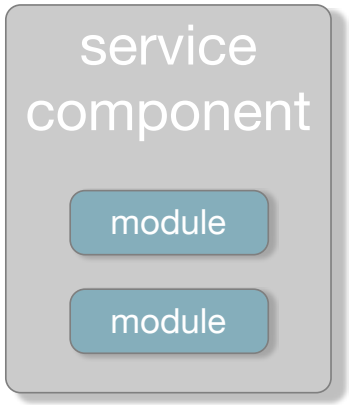
The more *reusable* code is, the less *usable* it is.

Code Reuse (Over Time)



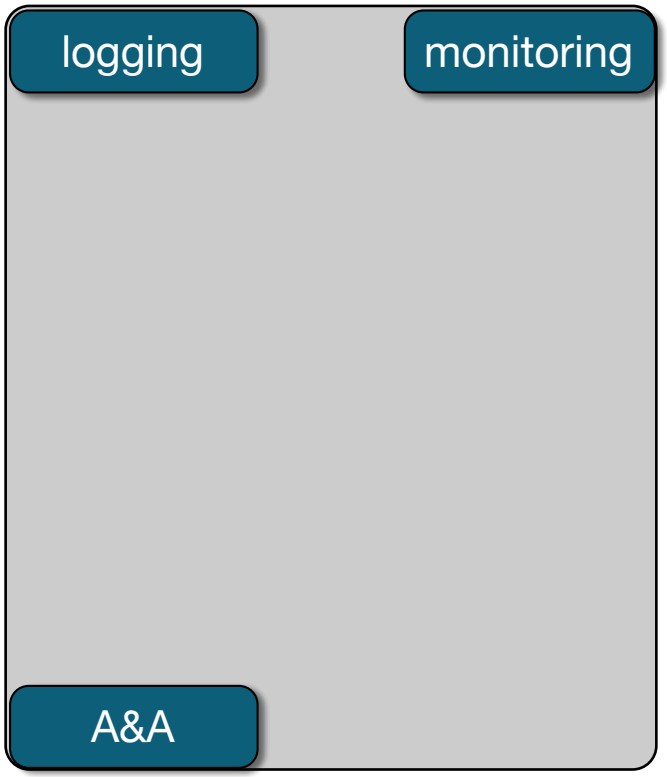

service
component
module
module

service templates



https://projects.spring.io/spring-boot/

http://www.dropwizard.io/



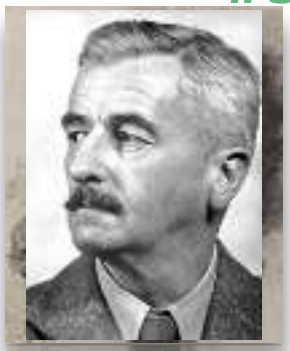
#reuse

carefully

#tradeoffs

#experimental

#scaling_architecture



#metawork

#feasibility

#canonicality

#legacy

#doyouDevOps

#standardization

#toomuch

#reuse

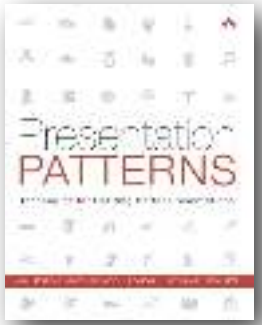
#toolittle

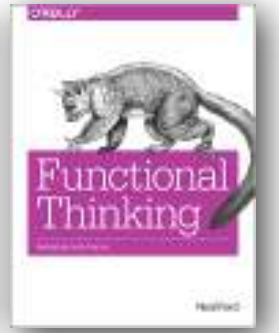
The past is never dead. It's not even past.

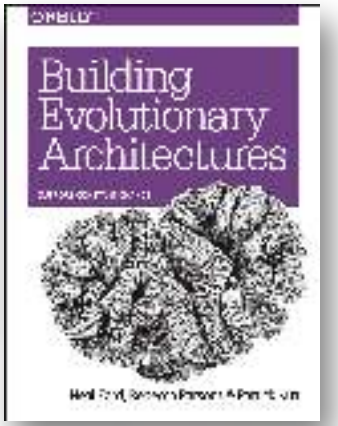
—William Faulkner

Those who cannot remember the past are condemned to repeat it.

—George Santayana

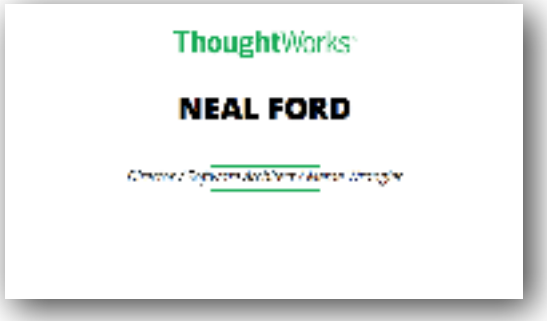


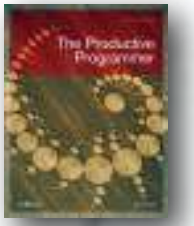


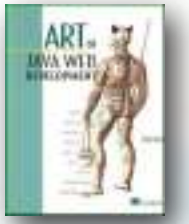


nealford.com

@neal4d







nealford.com/videos

www.oreilly.com/software-architecture-video-training-series.html

nealford.com/books