

this is water

ThoughtWorks®

NEAL FORD

Director / Software Architect / Meme Wrangler



This Is Water Some
Thoughts, Delivered on a Significant
Occasion, about Living a Compassionate
Life **David Foster Wallace**



[http://web.ics.purdue.edu/~drkelly/
DFWKenyonAddress2005.pdf](http://web.ics.purdue.edu/~drkelly/DFWKenyonAddress2005.pdf)

λ

μ

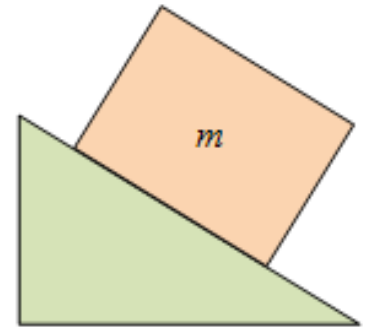
Δ

μ

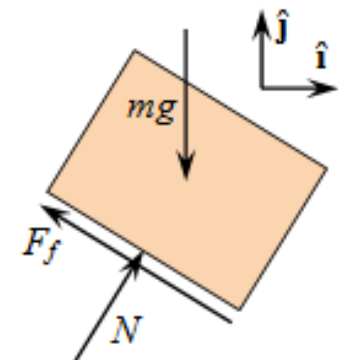
$$F_f \leq \mu F_n$$

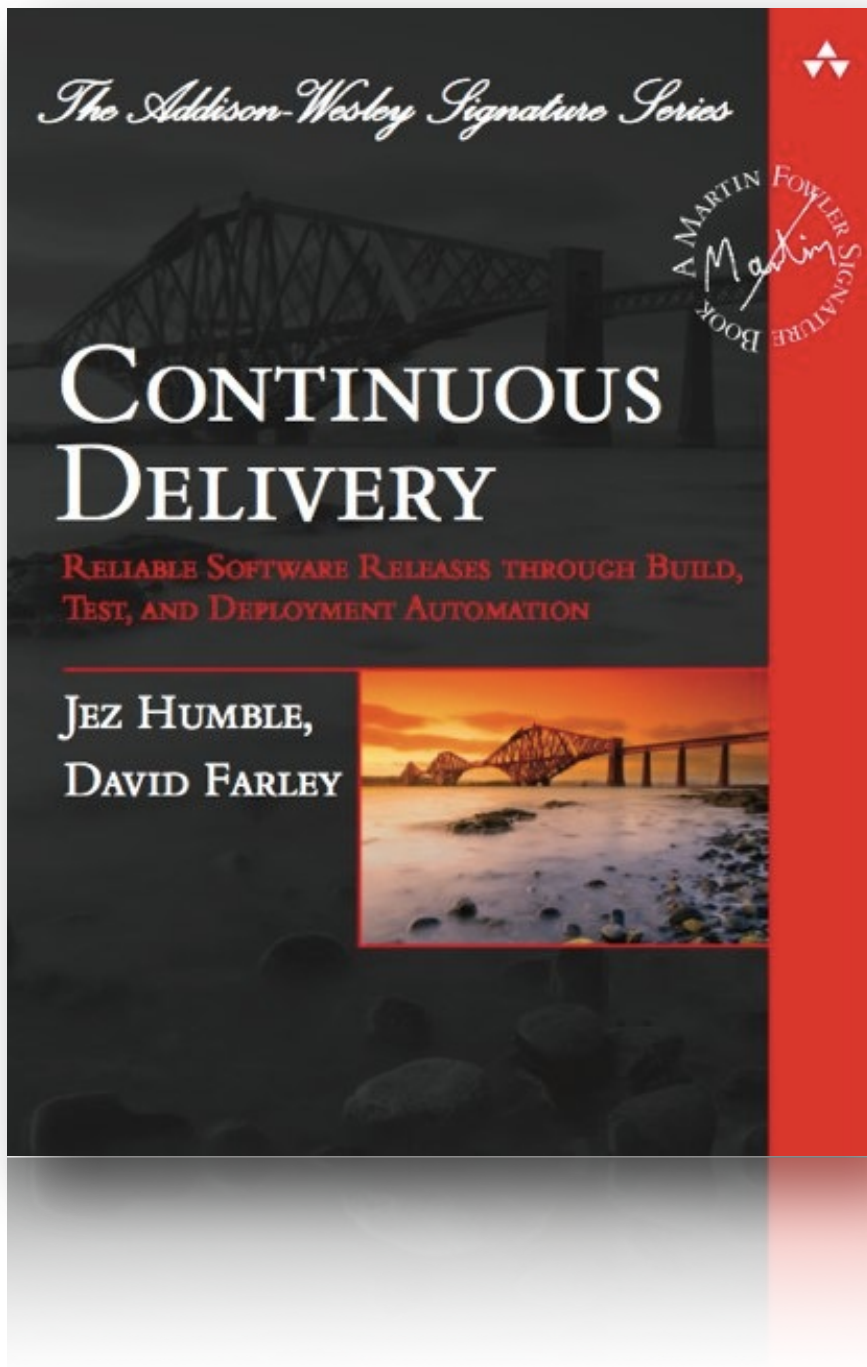
μ is a dimensionless scalar value which describes the ratio of the force of friction between two bodies and the force pressing them together.

A block on a ramp

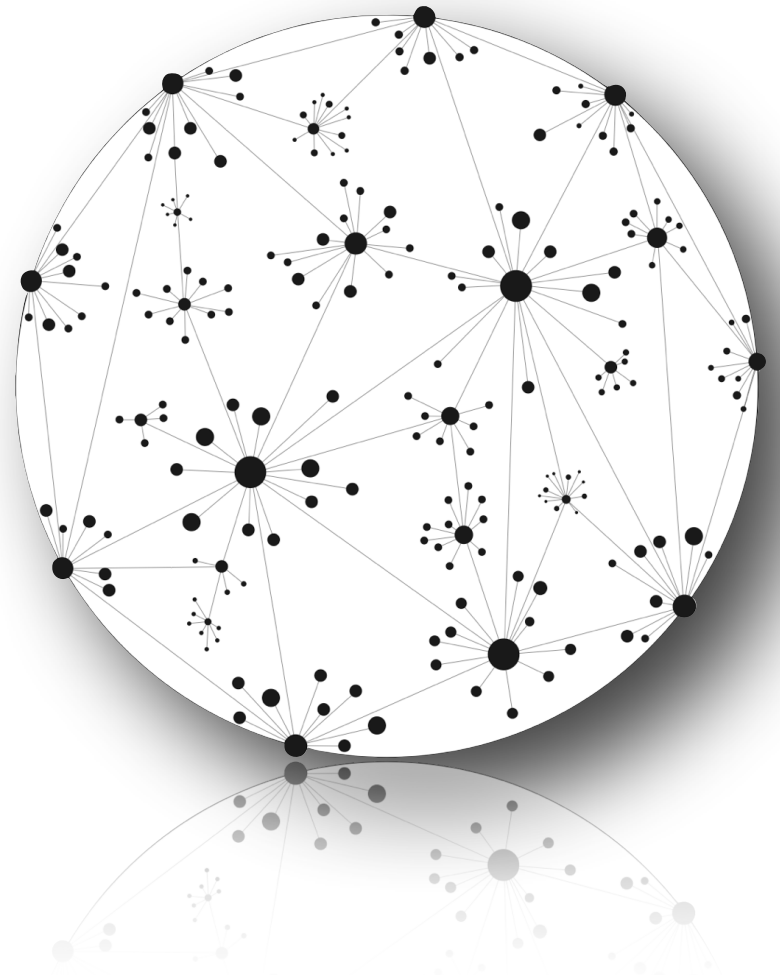


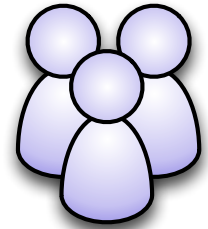
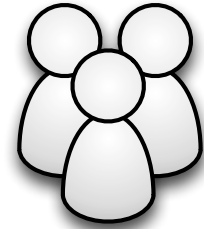
Free body diagram of just the block



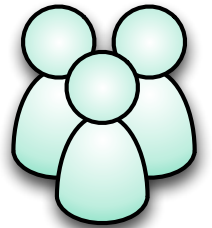
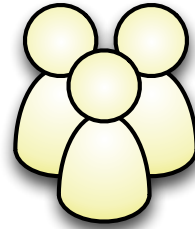


μ





eXtreme Programming



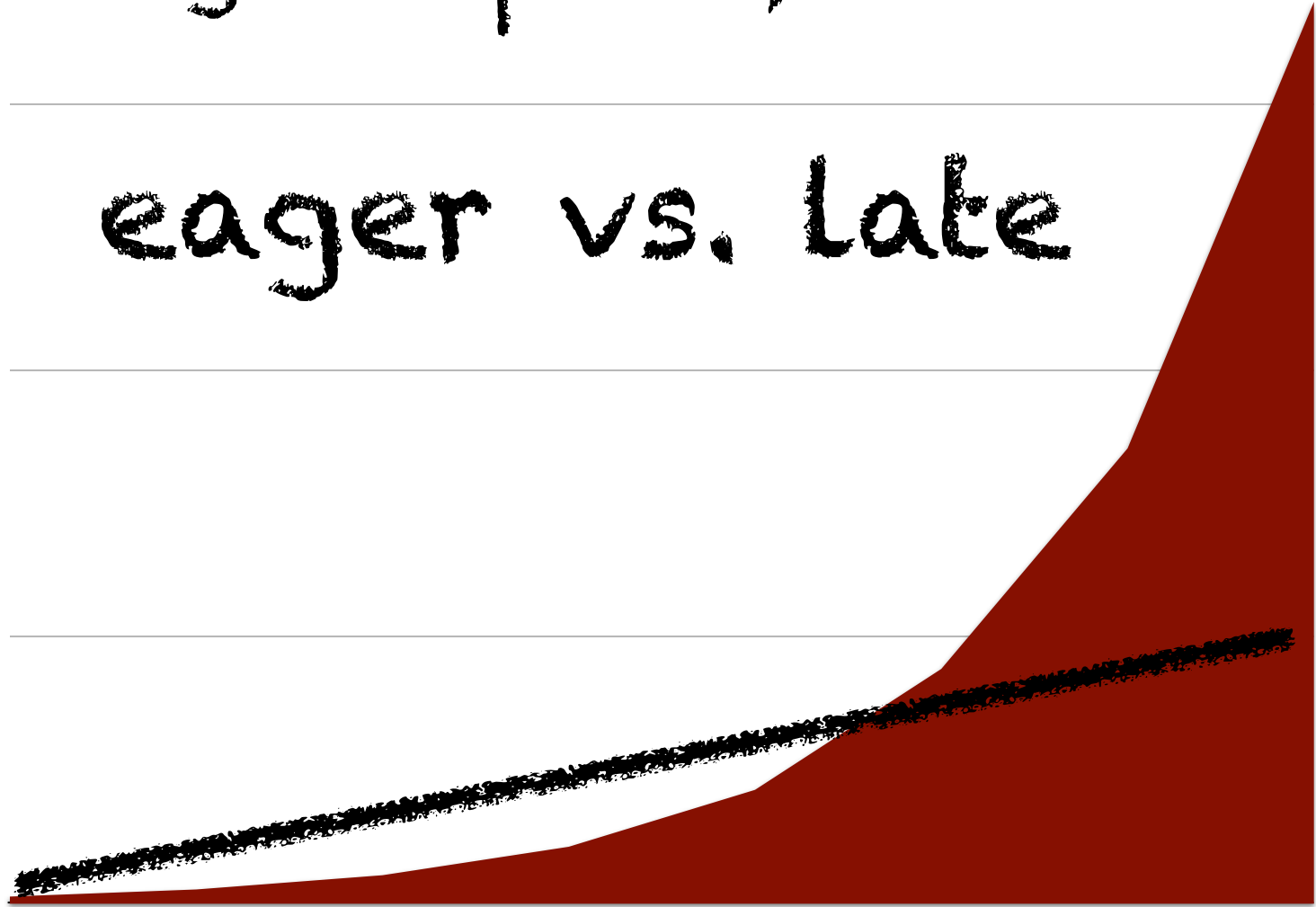
continuous integration

Everyone commits to the trunk at least once a day.

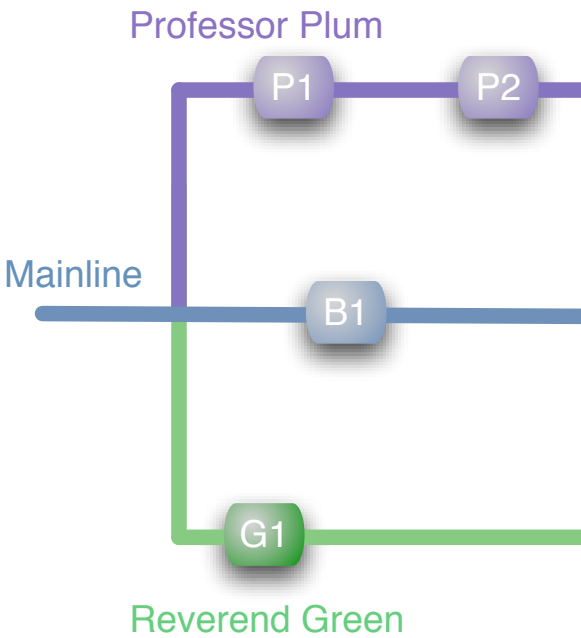
Bring the pain forward.

eager vs. late

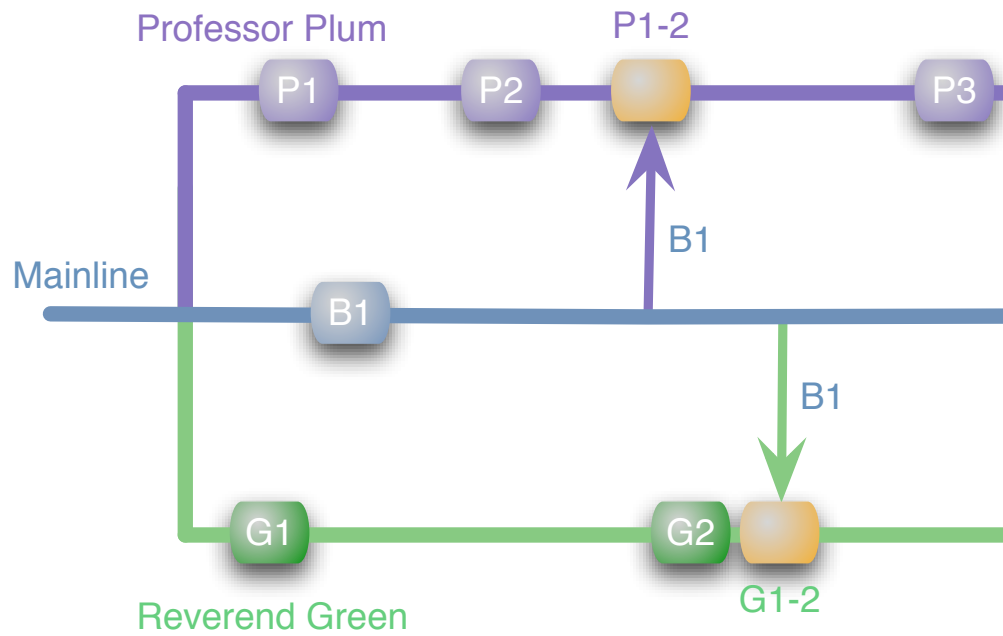
pain



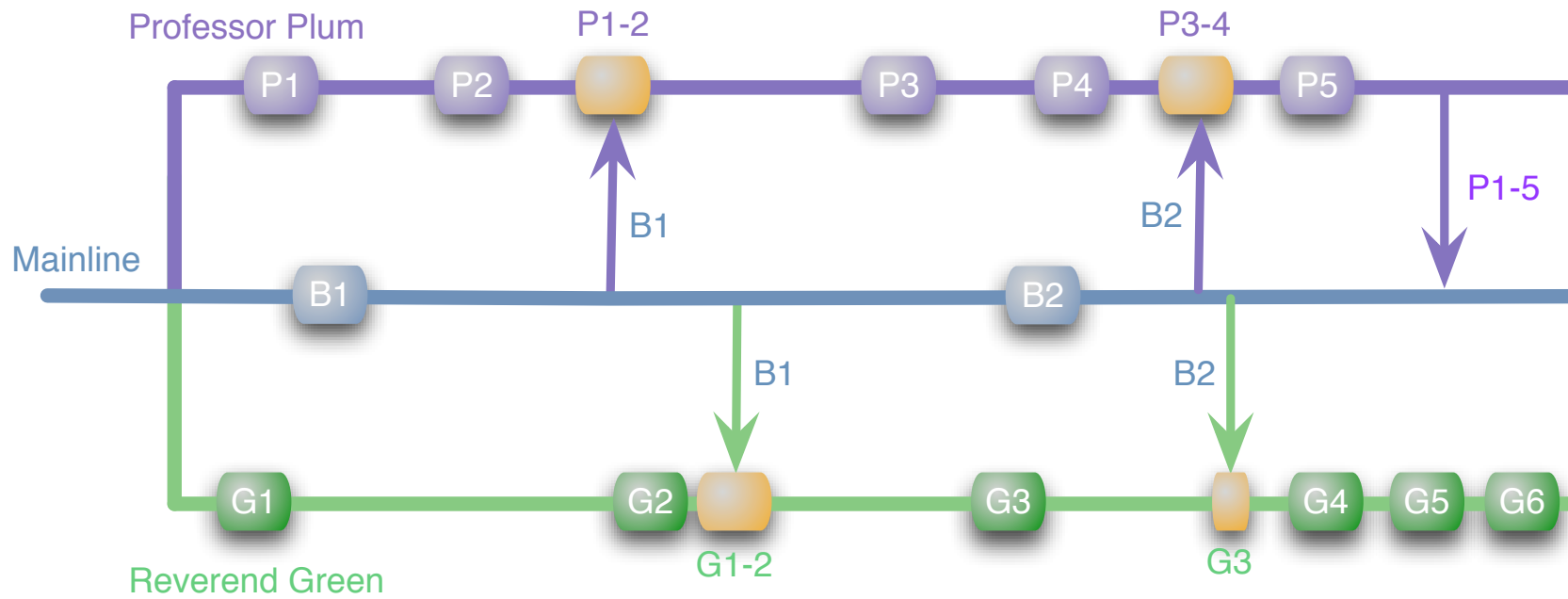
time



Feature Branching

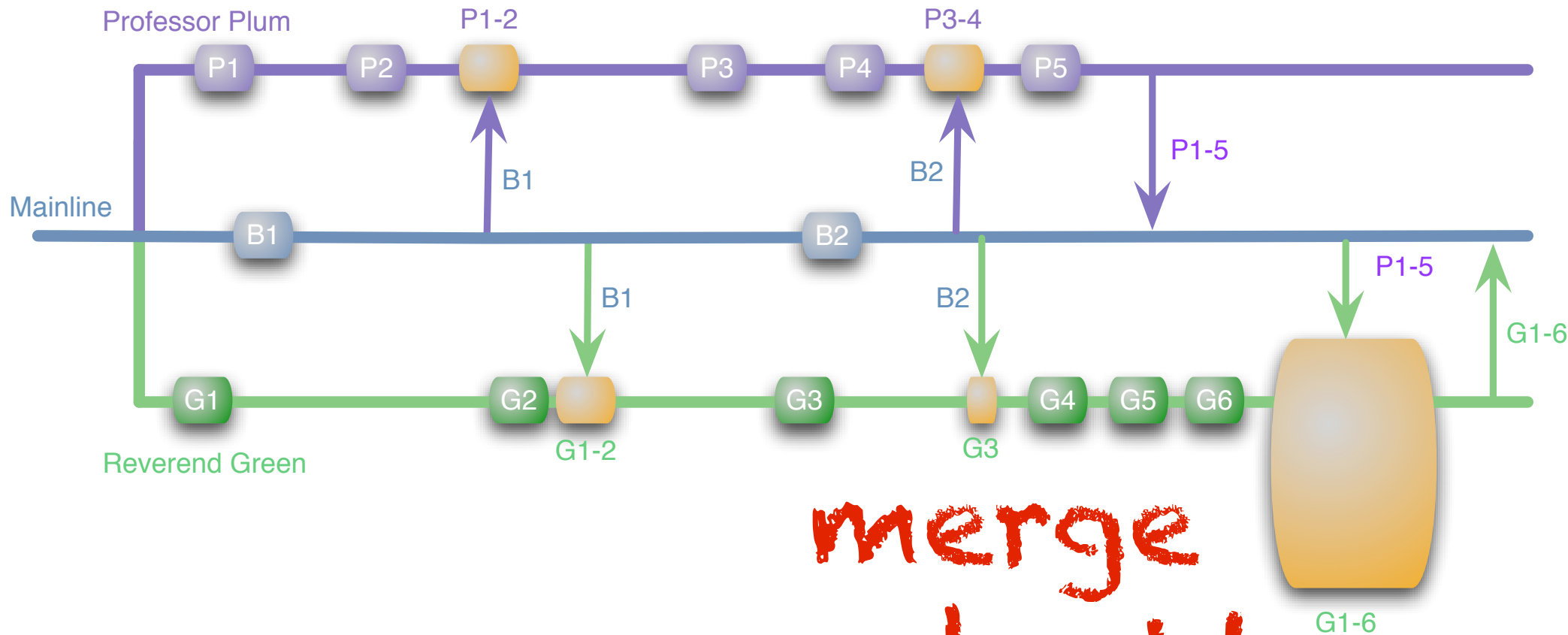


Feature Branching



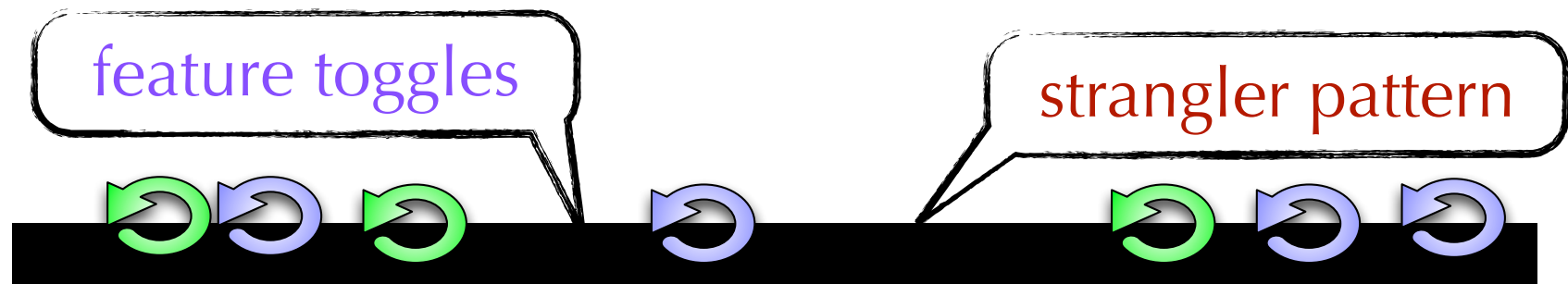
Feature Branching

copy/paste
reuse !!

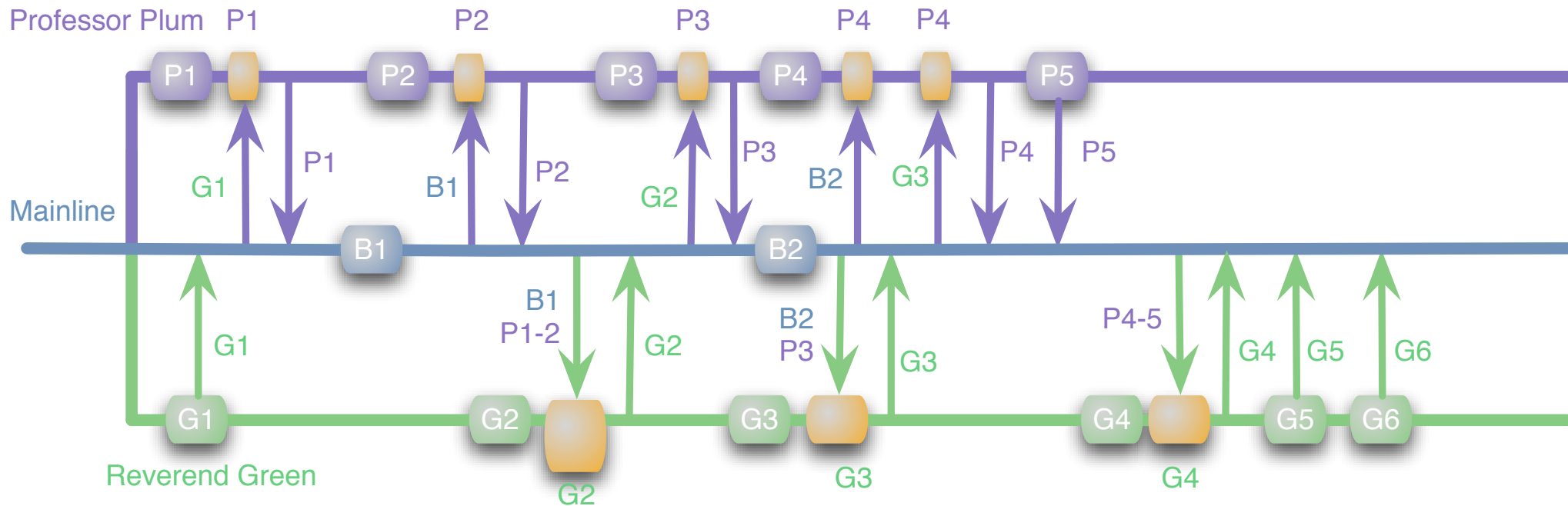


merge
ambush!

Feature Branching



trunk-based development



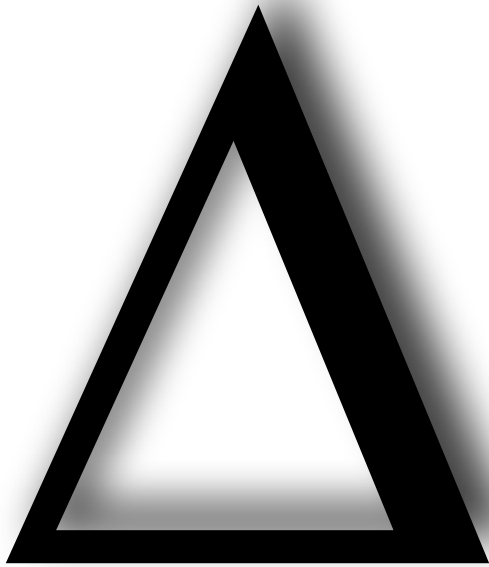
Continuous Integration

META-WORK IS
MORE INTERESTING
THAN WORK



λ

μ



$$\frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x}$$

Change of any changeable quantity, in mathematics and the sciences (more specifically, the difference operator).





1

2

3

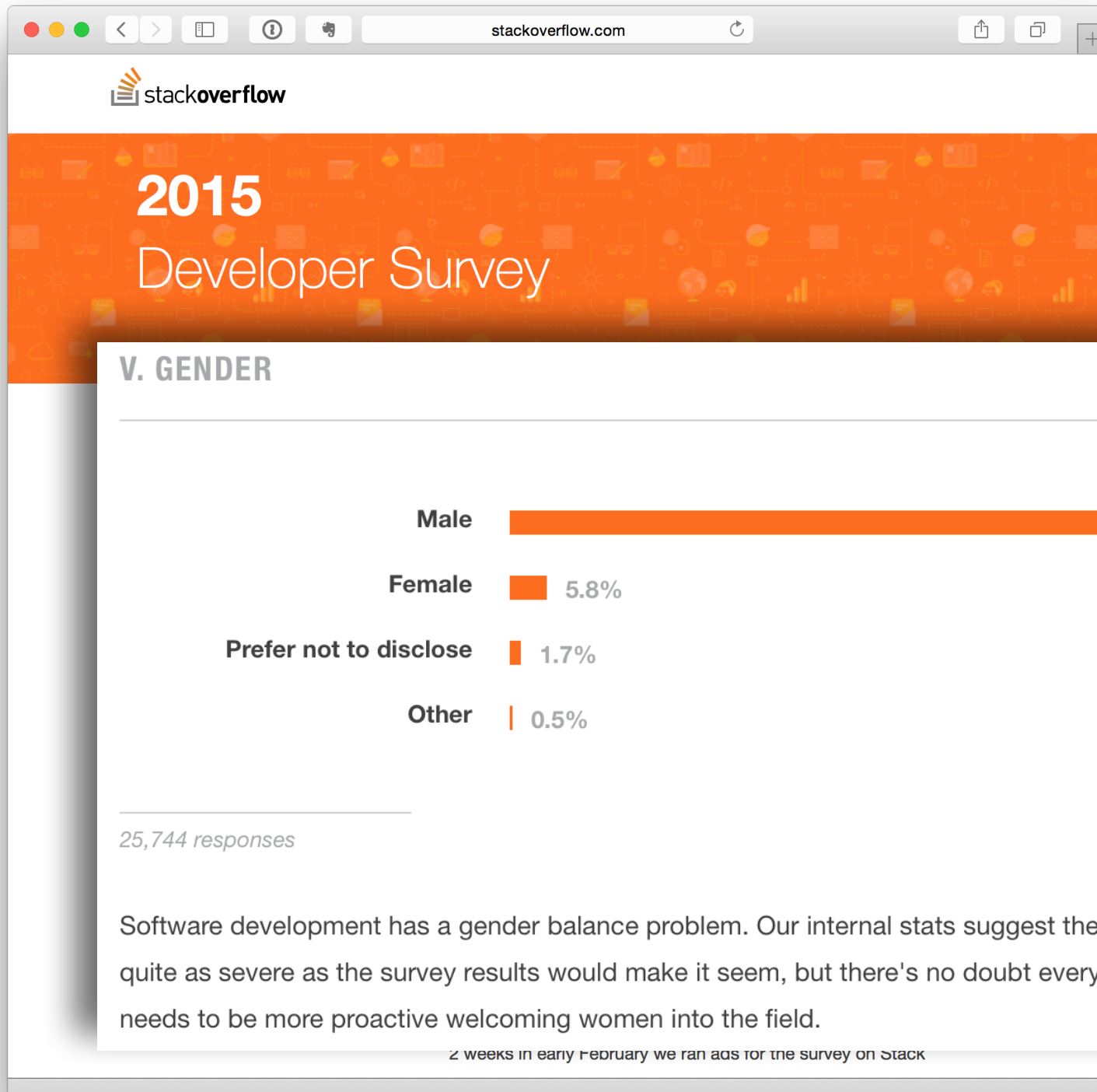
1 1/2

Out of every 100 software developers/engineers in Los Angeles, approximately 10-12 are female.

In 2011, only 17% of job placements (all for positions in technology in L.A.) were women.

Only 7% of developers hired through Q were women.

www.forbes.com/sites/lorikozlowski/2012/03/22/women-in-tech-female-developers-by-the-numbers/



<http://stackoverflow.com/research/developer-survey-2015>

Evidence for a Collective Intelligence Factor in the Performance of Human Groups

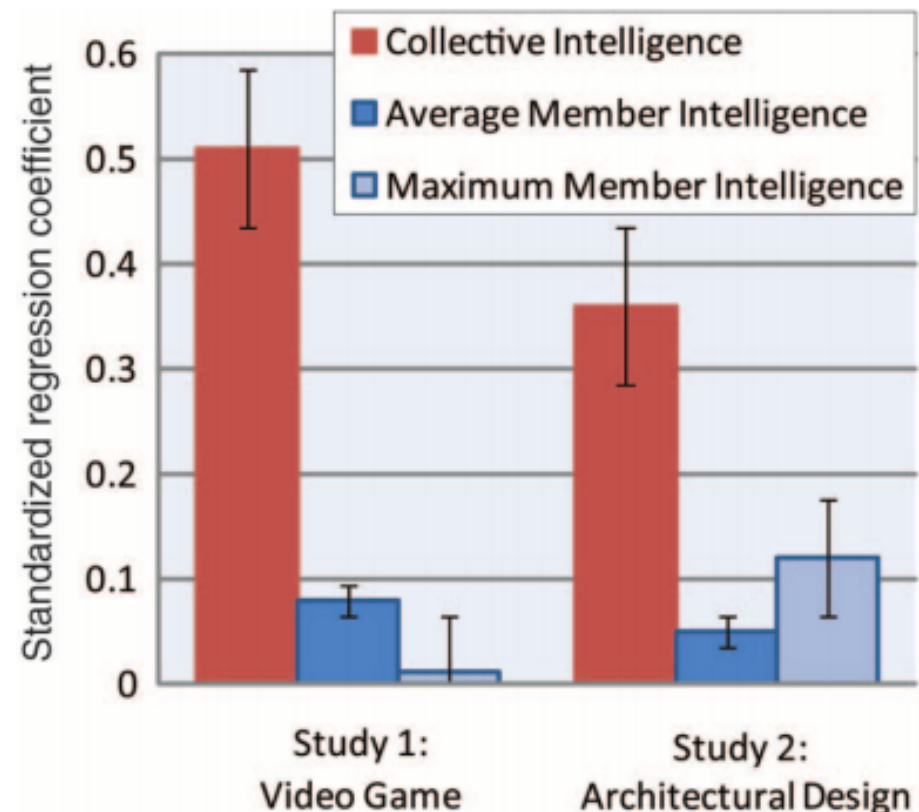
Anita Williams Woolley,^{1*} Christopher F. Chabris,^{2,3} Alex Pentland,⁴ Nada Hashmi,^{3,5} Thomas W. Malone^{3,5}

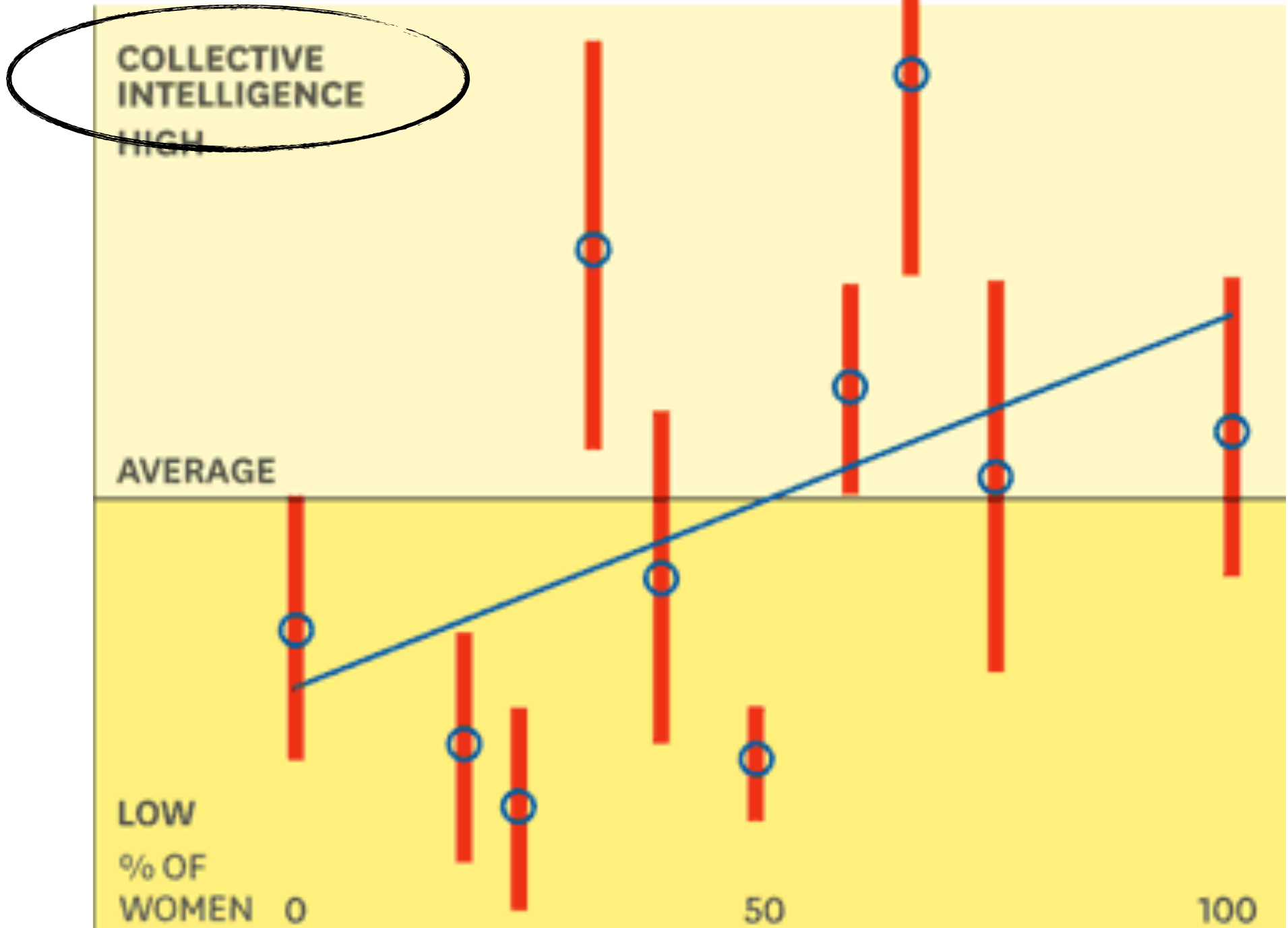
Psychologists have repeatedly shown that a single statistical factor—"intelligence"—emerges from the correlations among people's performance on tasks. But no one has systematically examined whether a similar kind of "c" factor exists in groups of people. In two studies with 699 people, working in groups of four, we found evidence of a general collective intelligence factor that explains a group's performance on a variety of tasks. This "c factor" is not strongly correlated with the average or maximum intelligence of group members but is correlated with the average social sensitivity of group members, the distribution of conversational turn-taking, and the proportion of females in the group.

As research, management, and many other kinds of tasks are increasingly accomplished by groups—working both face-to-face and virtually (1–3)—it is becoming ever more important to understand the determinants of group performance. Over the past century,

psychologists have made progress in defining and systematically measuring intelligence in individuals. A statistical approach to the study of intelligence to systems of groups.

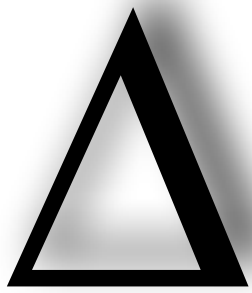
gists and others have studied for decades how well groups perform specific tasks (5, 6), they have not attempted to measure group intelligence in the same way individual intelligence is measured—by assessing how well a single group can perform



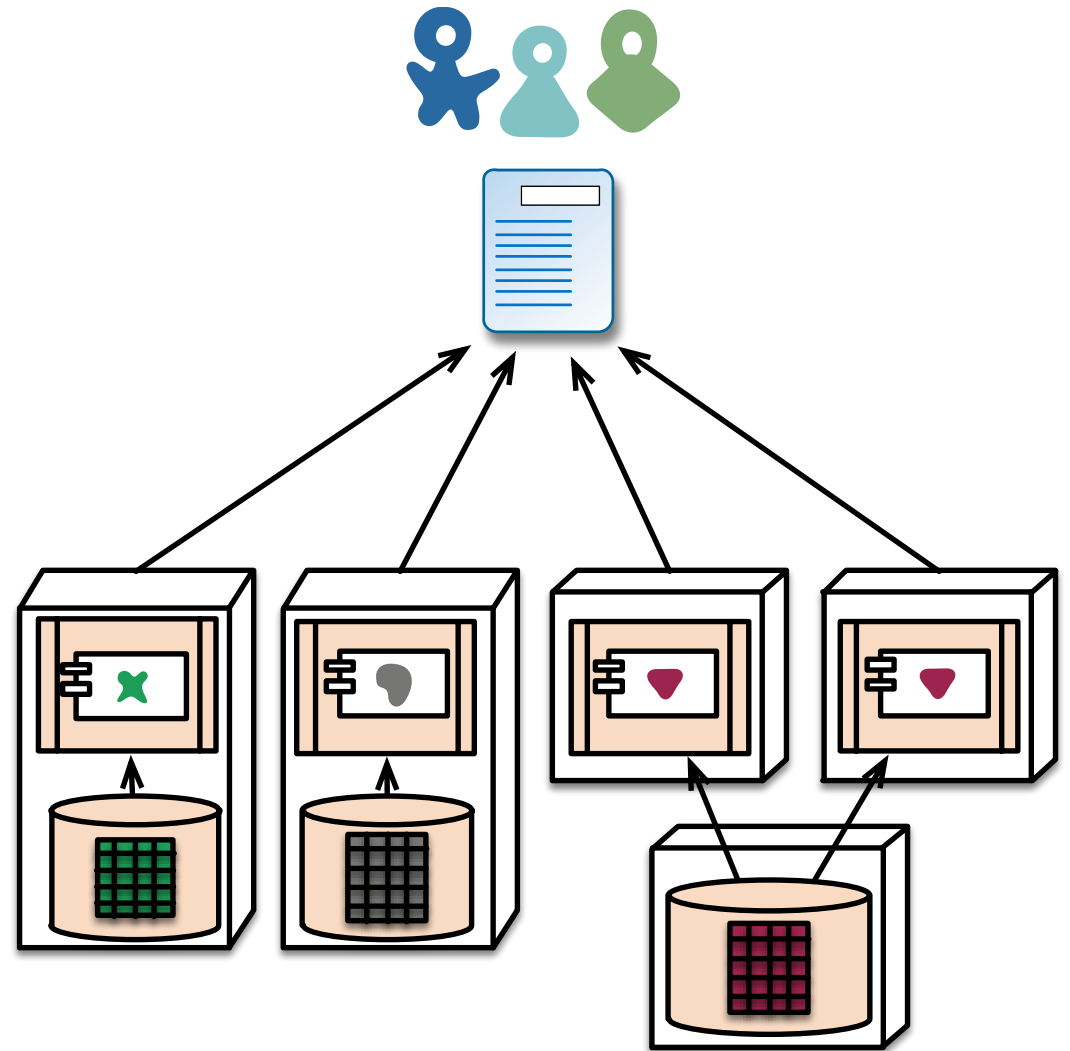
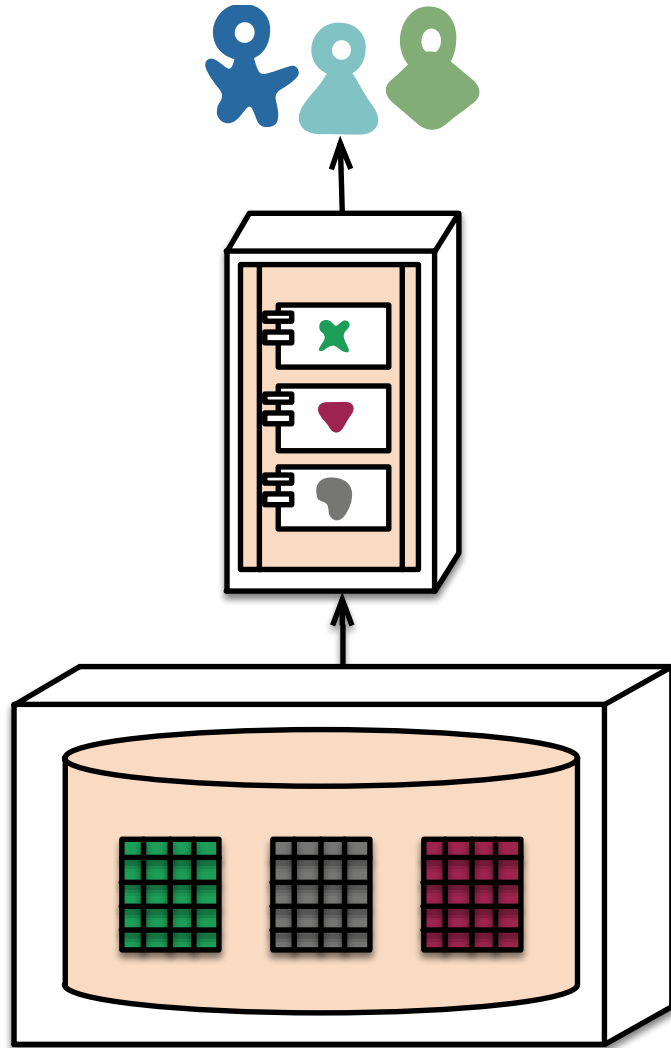


NEED BETTER
WAYS TO FOSTER
DIVERSITY





transactions

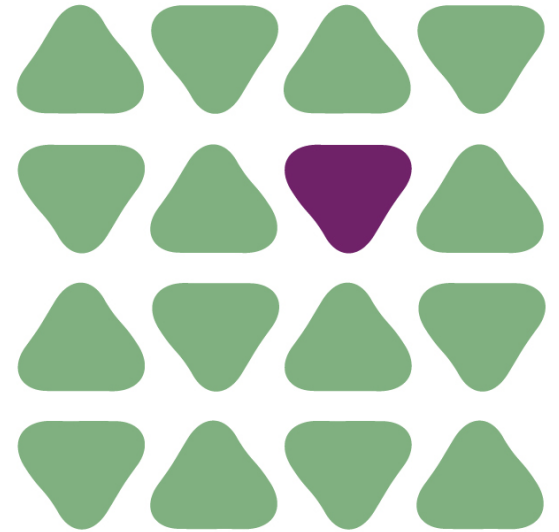




<http://highscalability.com/blog/2013/5/1/myth-eric-brewer-on-why-banks-are-base-not-acid-availability.html>

http://www.eaipatterns.com/docs/IEEE_Software_Design_2PC.pdf

ACID versus BASE



ACID

- **A**tomic
- **C**onsistent
- **I**solated
- **D**urable

BASE

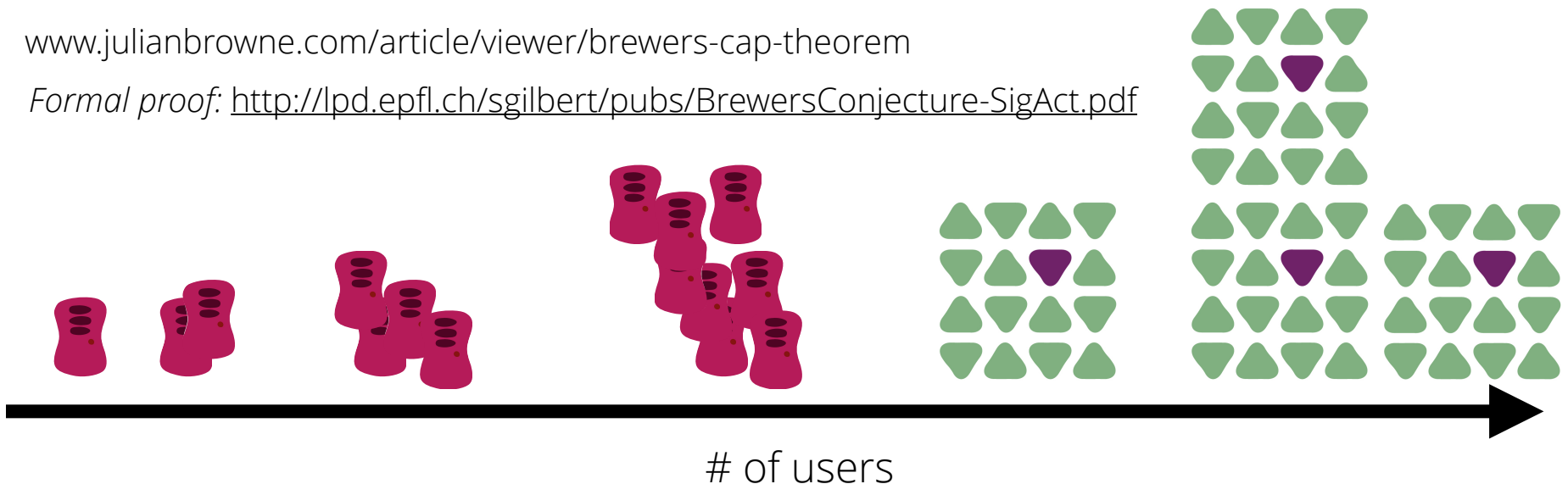
- **B**asic **A**vailability
- **S**oft-state
- **E**ventual Consistency

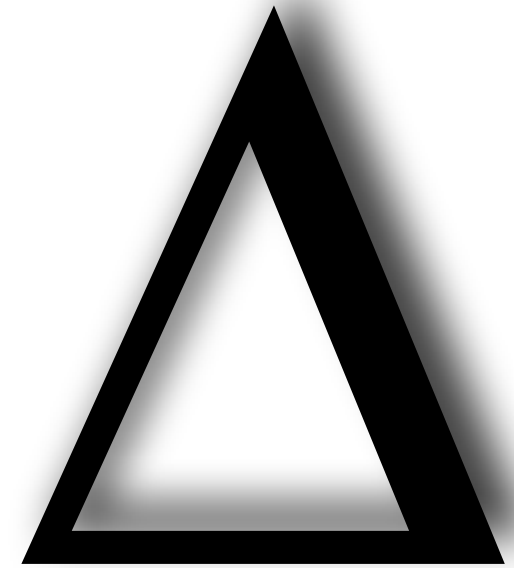
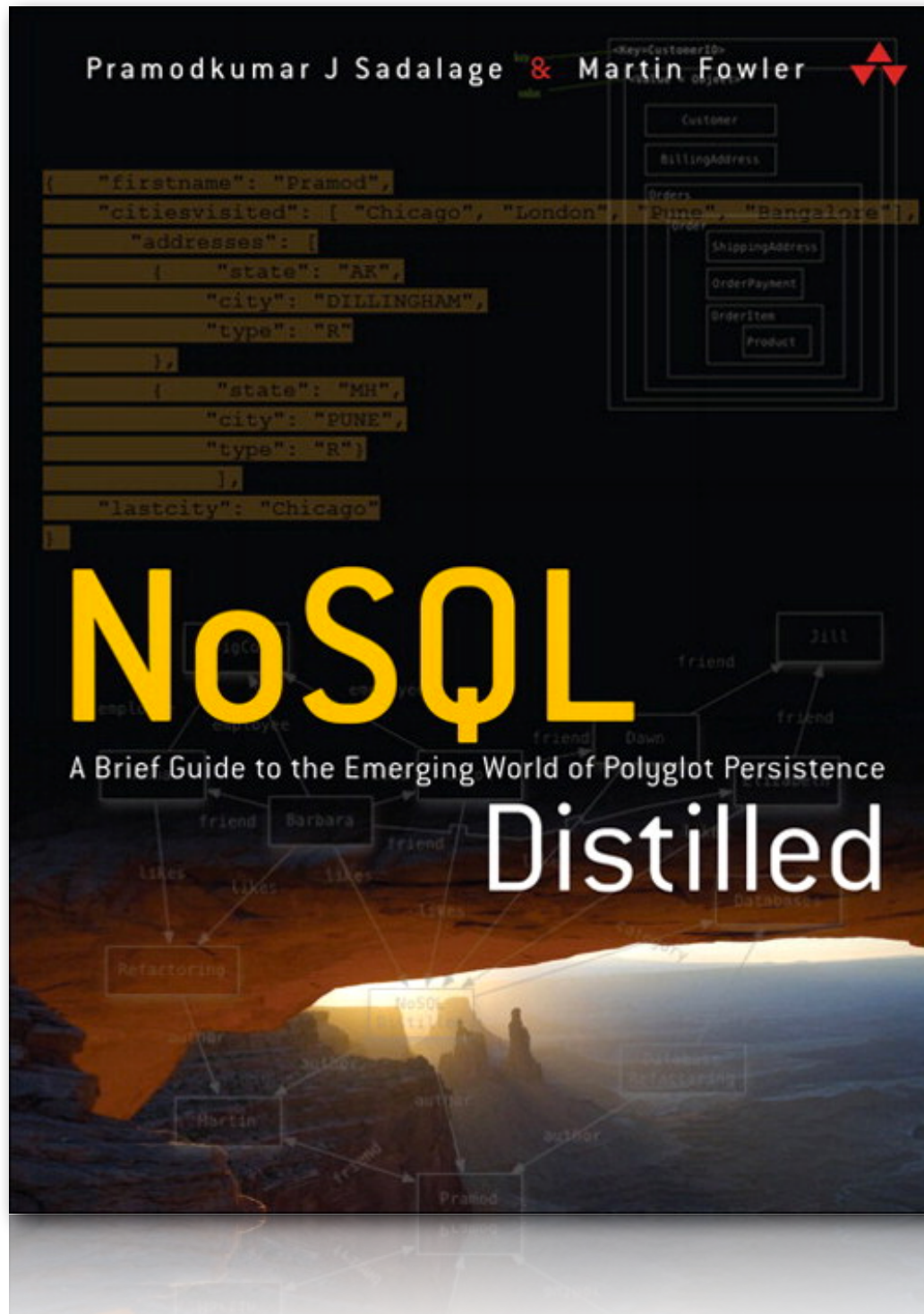
**As your system becomes more distributed,
prefer BASE to ACID...**

...because CAP Theorem

www.julianbrowne.com/article/viewer/brewers-cap-theorem

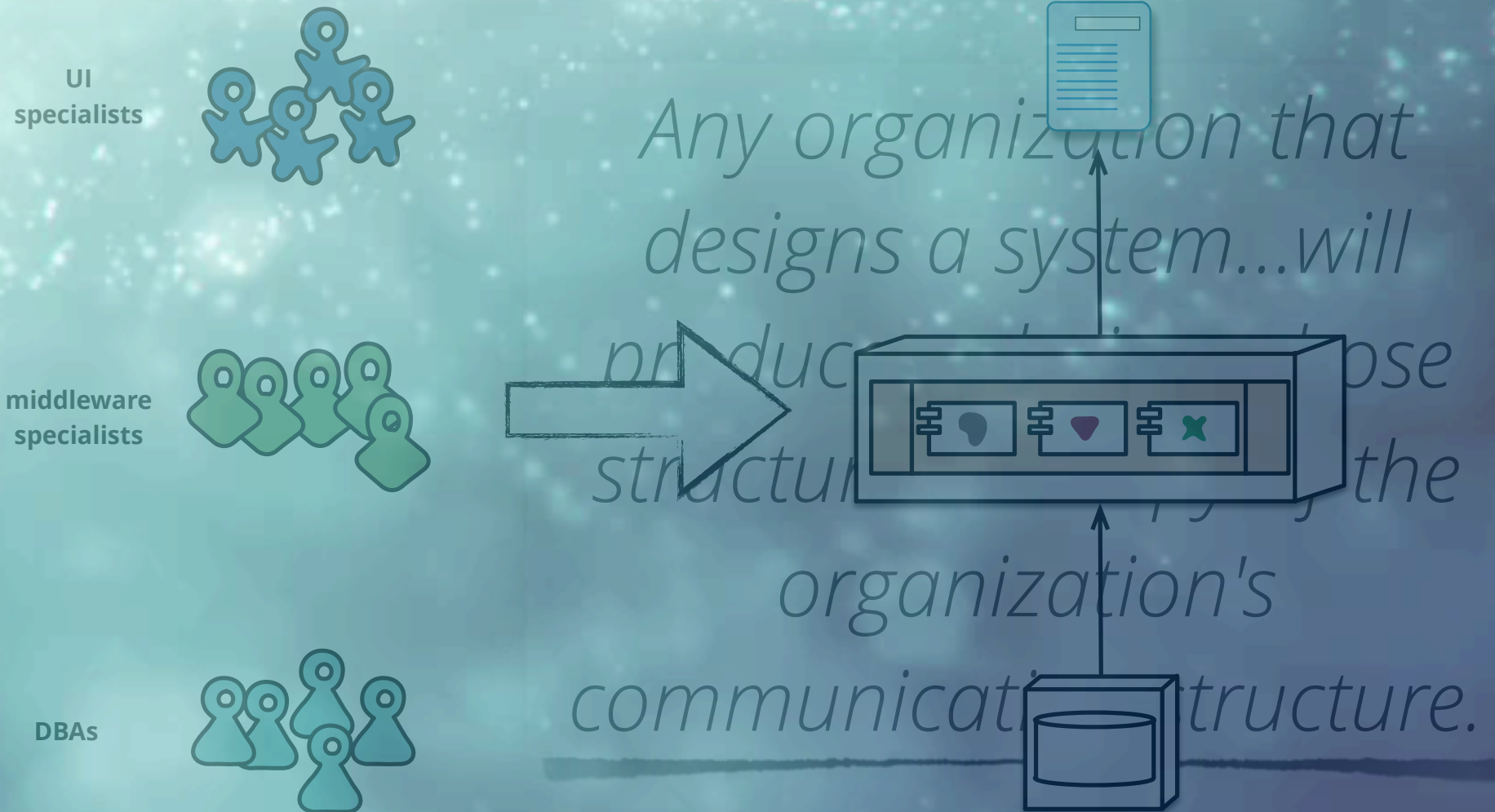
Formal proof: <http://lpd.epfl.ch/sgilbert/pubs/BrewersConjecture-SigAct.pdf>



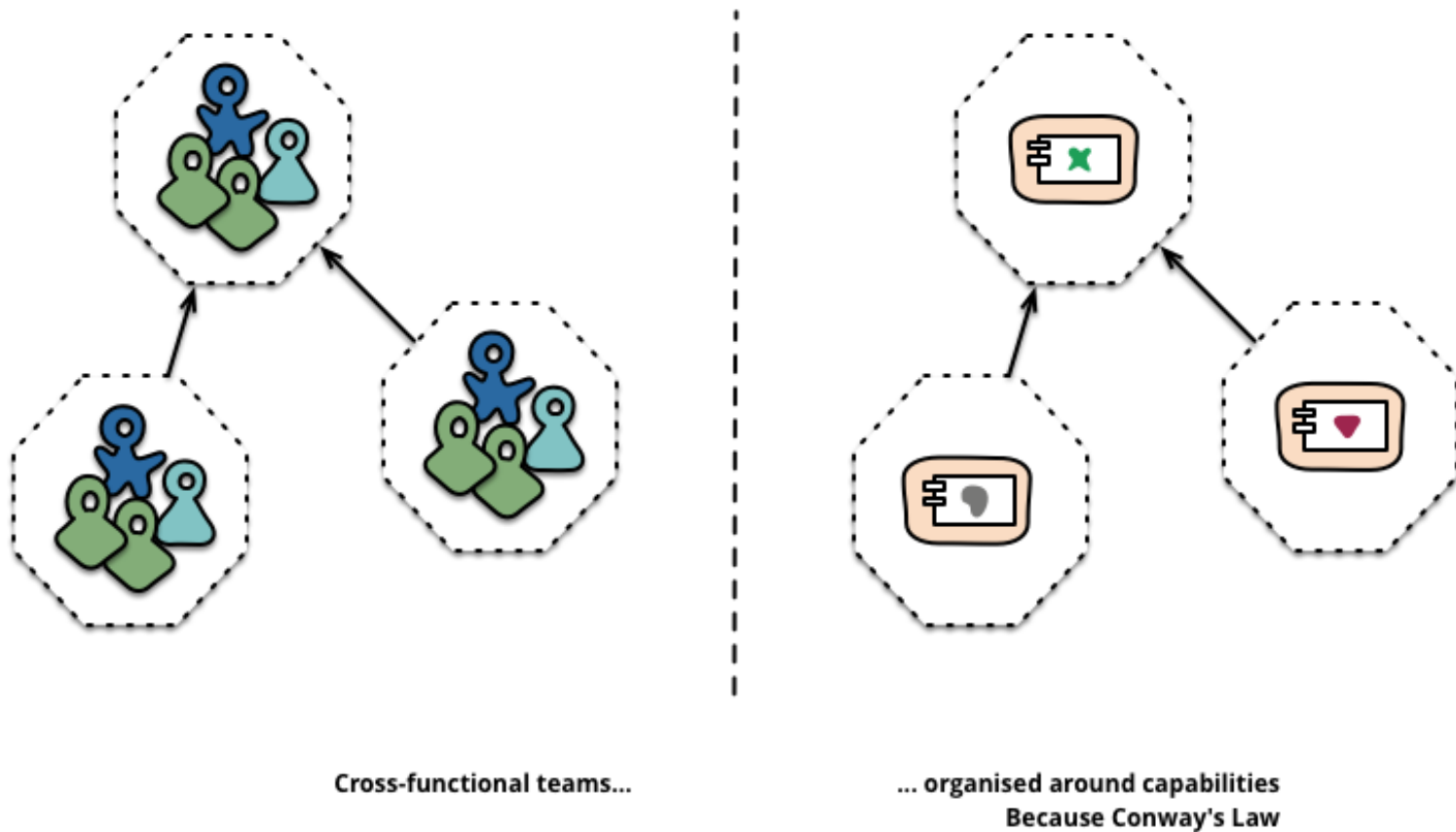


end of the
relational database
dark ages

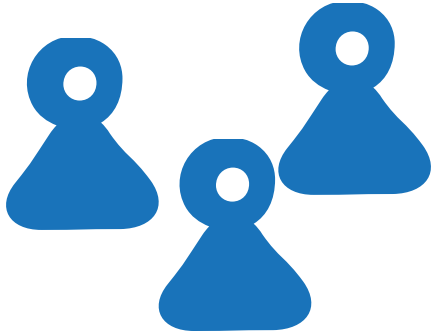
Conway's Law



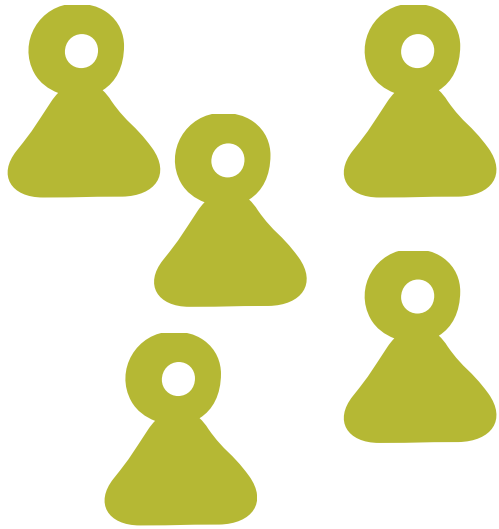
Inverse Conway Maneuver



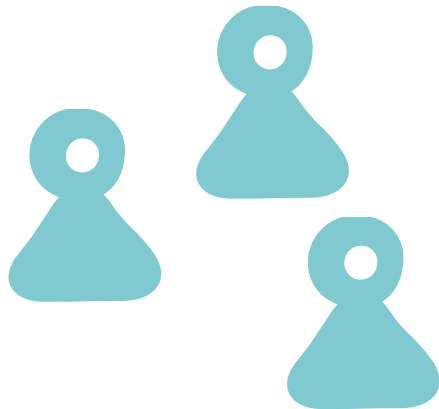
Build teams that look like
the architecture you want
(and it will follow).



user interface



server-side



DBA

Orders



Shipping



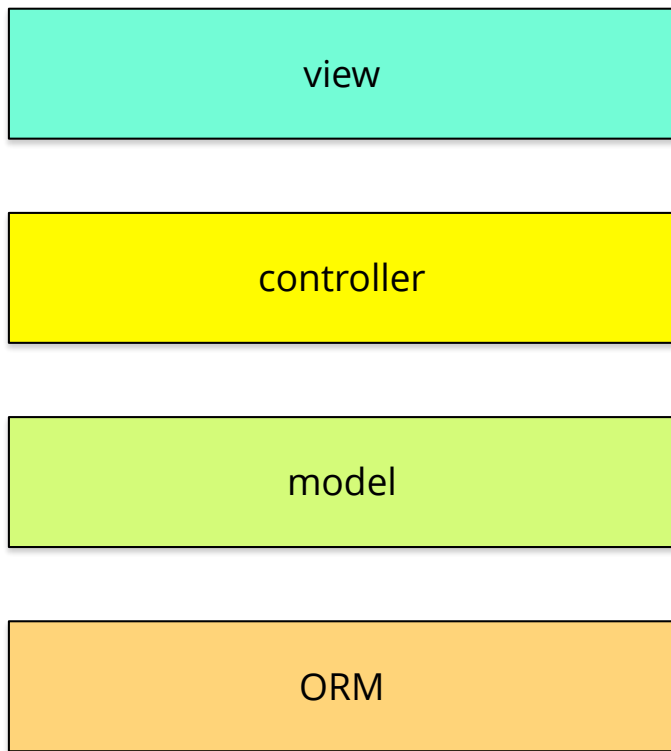
Catalog



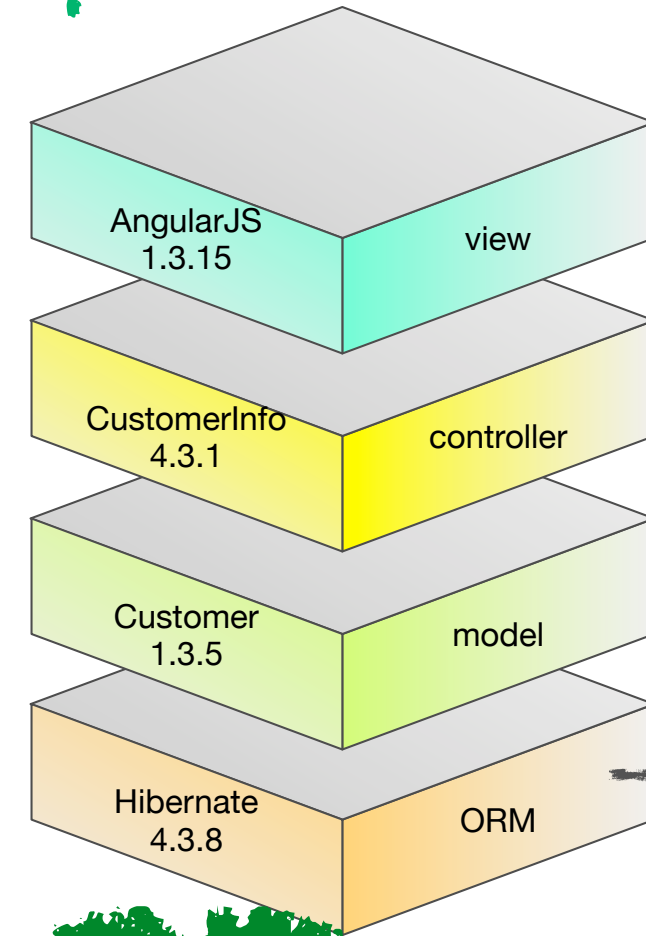


Architecture is abstract
until operationalized.

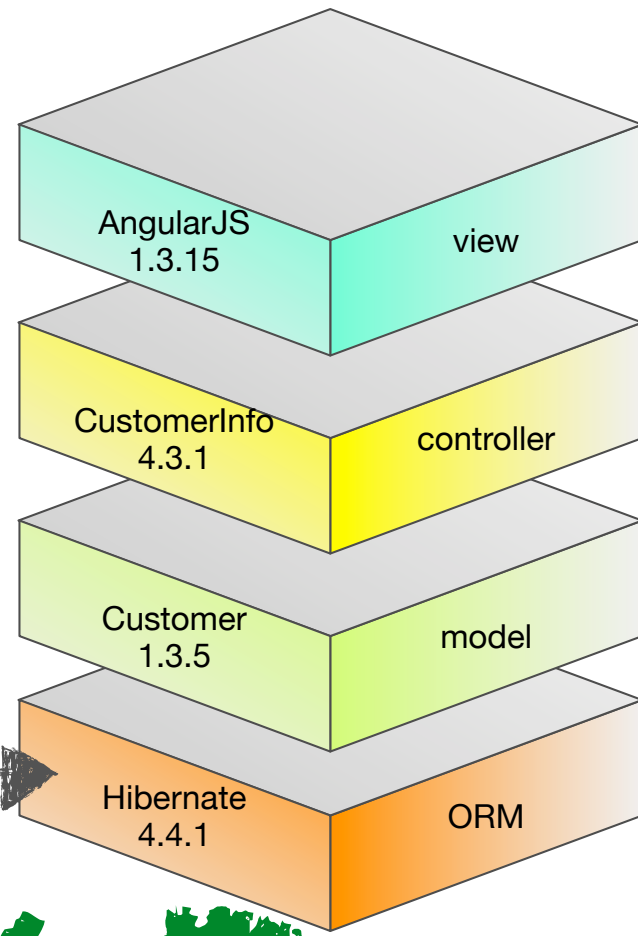
Architecture is abstract until operationalized.



2D

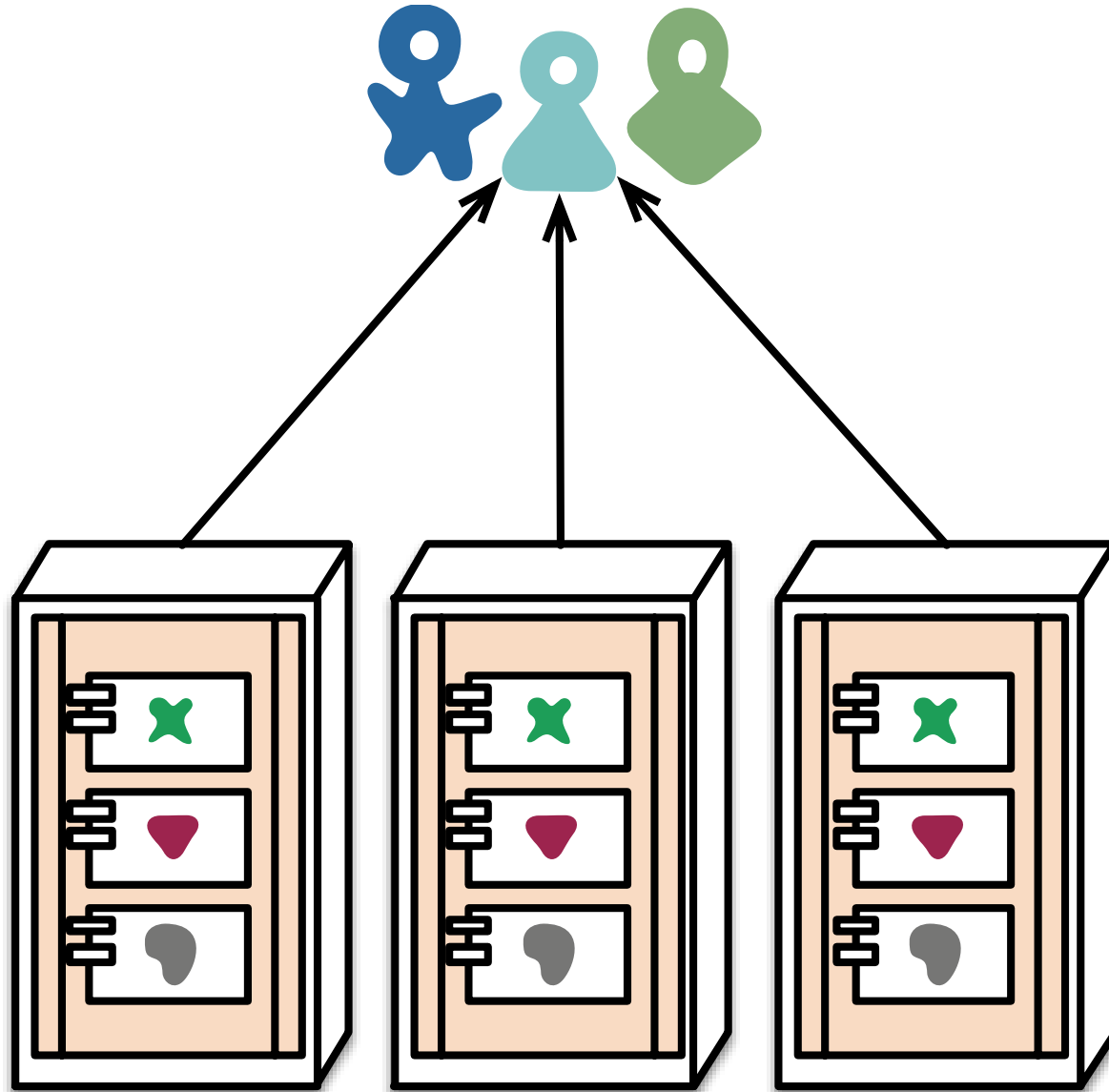


3D



4D

Shared Resources



Coupling!

~~Shared Resources~~

prevents isolated changes

encourages big-bang deployments

makes upgrades difficult

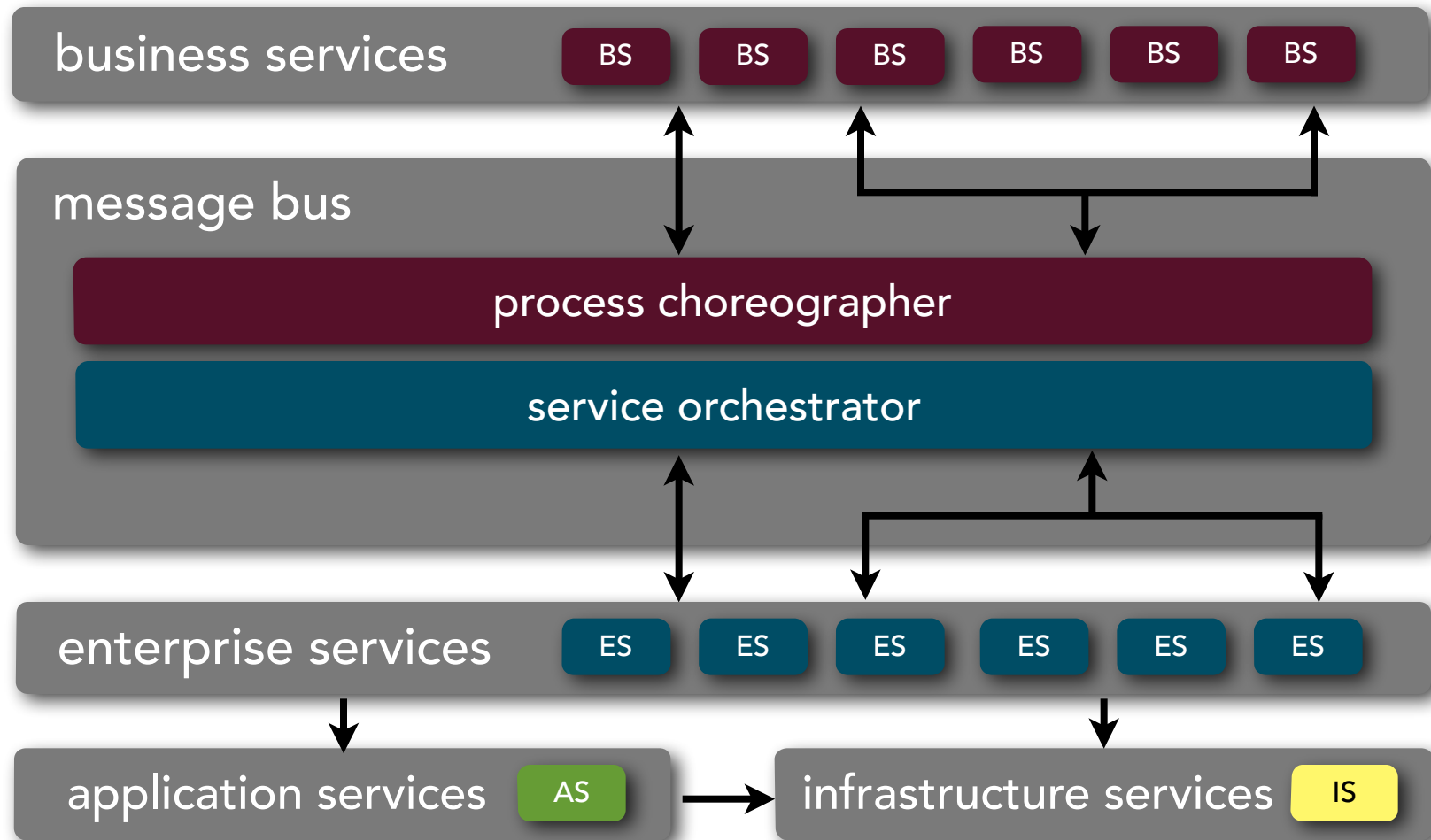
prevents change



Yesterday's best
practice is tomorrow's
anti-pattern.

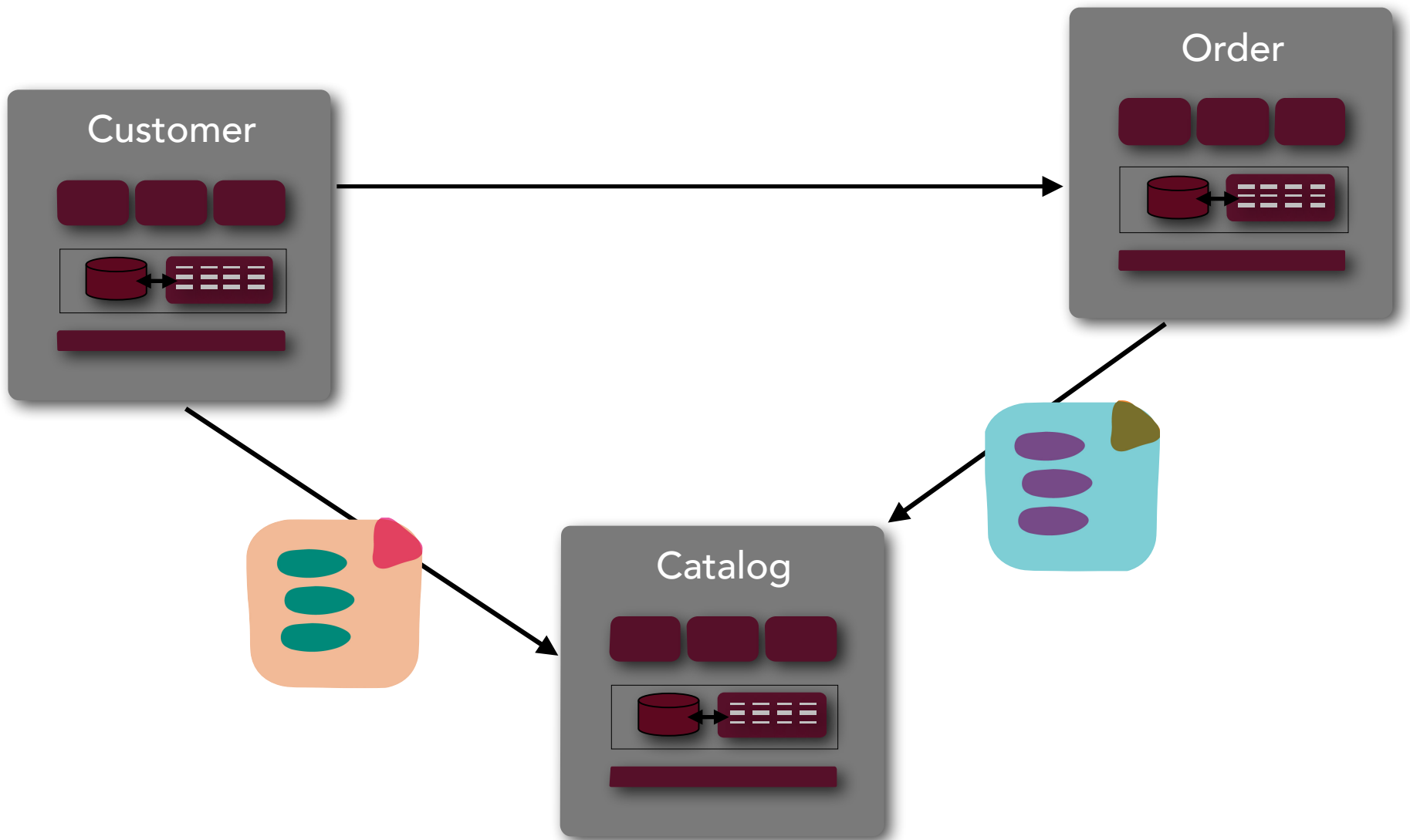
We inadvertently build
architectures to solve
outdated problems.

Traditional SOA

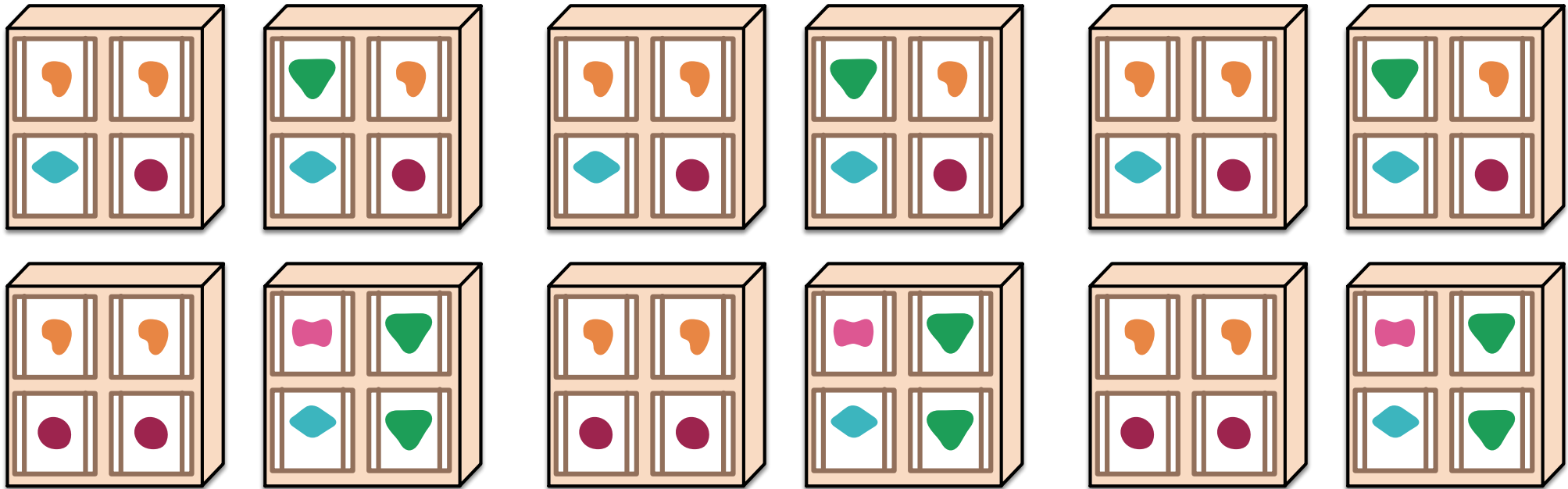


maximize shared resources

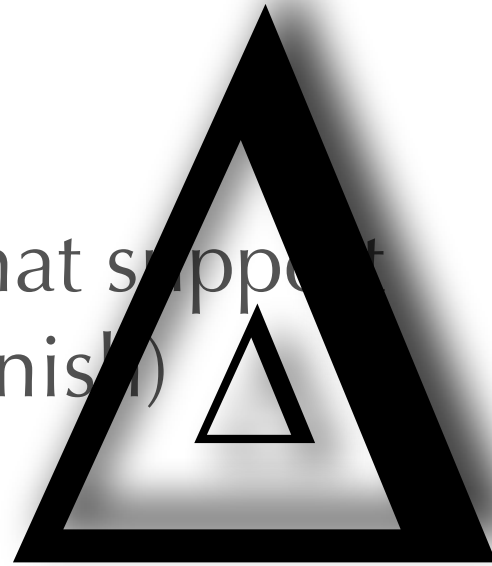
Microservice



maximize easy evolution



build architectures that support
(rather than punish)



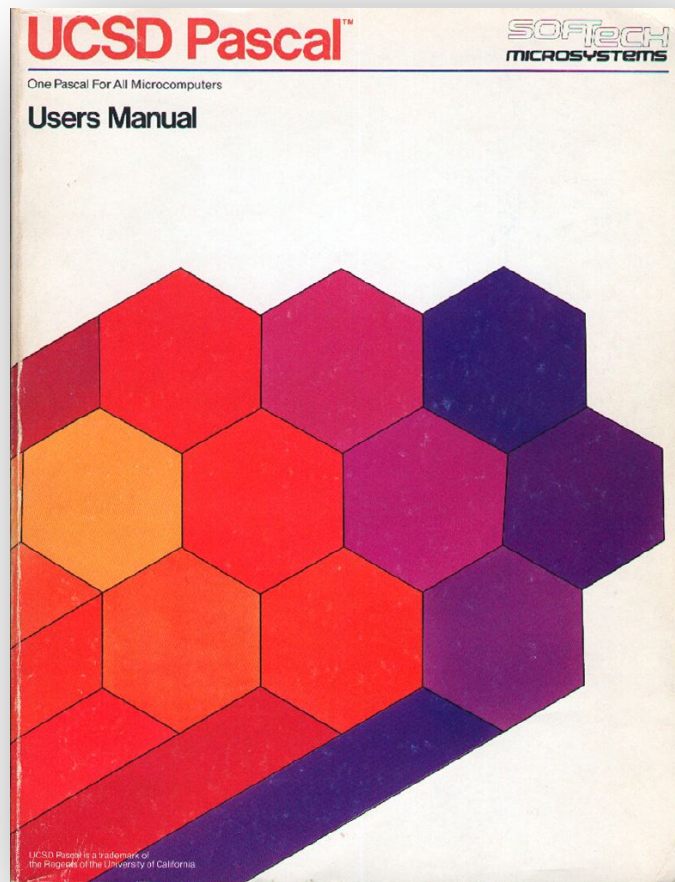
λ

μ

Δ



λ



www.threedee.com/jcm/psystem/

The background features a series of silhouettes showing the evolution of a human-like figure from an ape-like ancestor on the left to a modern person sitting at a desk with a computer on the right. The silhouettes are light gray and have a reflection effect below them. A thick, black, hand-drawn arrow points from the left towards the right, positioned at the top of the image.

memory allocation

garbage collection

state via closures

caching

concurrency

A series of light gray silhouettes on a light blue background, illustrating the evolution of man from an ape-like ancestor to a modern human. The sequence includes a crouching ape, several upright hominids, and a modern human sitting at a desk with a computer. The text is overlaid on the middle of this sequence.

Life's too short for
malloc!

OO makes code
understandable by
encapsulating moving
parts.

FP makes code
understandable by
minimizing moving
parts.

Michael Feathers, author of "Working with Legacy Code"



```
class Classifier {  
  def static isFactor(number, potential) {  
    number % potential == 0;  
  }  
  
  def static factorsOf(number) {  
    (1..number).findAll { isFactor(number, it) }  
  }  
  
  def static sumOfFactors(number) {  
    factorsOf(number).inject(0, {i, j -> i + j})  
  }  
  
  def static isPerfect(number) {  
    sumOfFactors(number) == 2 * number  
  }  
  
  def static isAbundant(number) {  
    sumOfFactors(number) > 2 * number  
  }  
  
  def static isDeficient(number) {  
    sumOfFactors(number) < 2 * number  
  }  
}
```

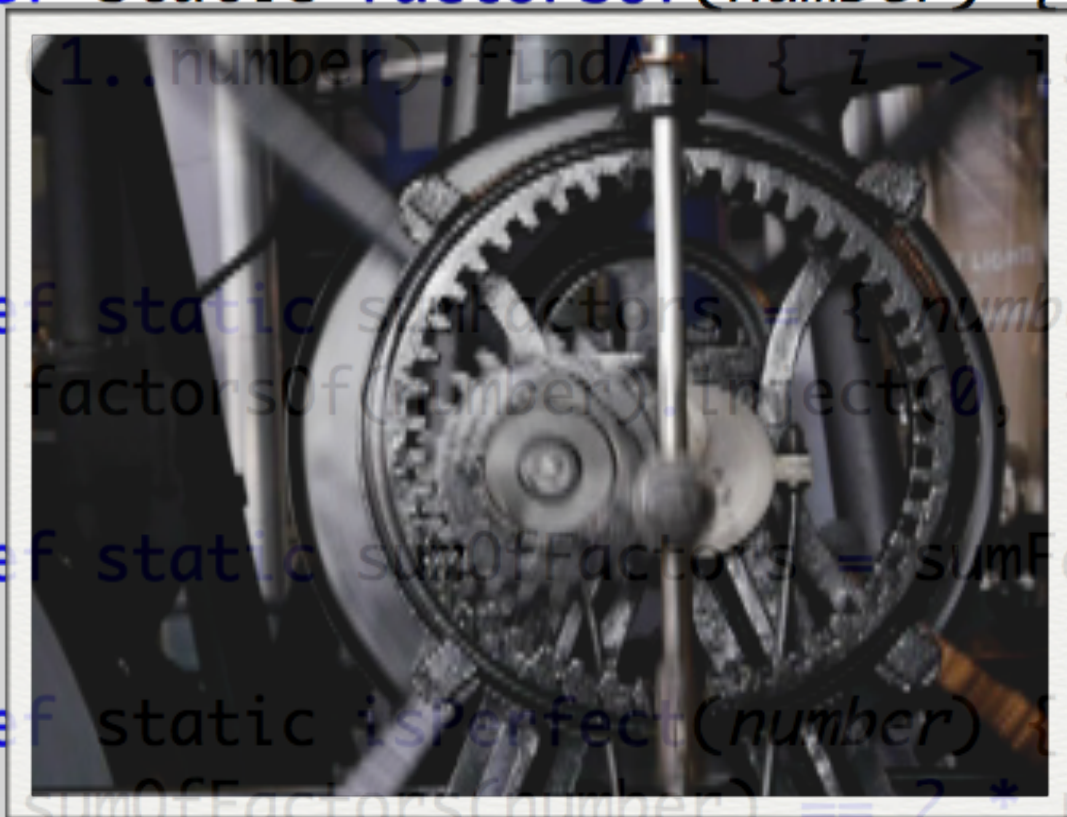


hand-written cache

```
class ClassifierCachedSum {  
    private sumCache = [:]  
  
    def sumOfFactors(number) {  
        if (! sumCache.containsKey(number)) {  
            sumCache[number] = factorsOf(number).sum()  
        }  
        return sumCache[number]  
    }  
}  
// remainder of code unchanged...
```

memoization

```
def static factorsOf(number) {  
  (1..number).findAll { i -> isFactor(number, i) }  
}  
  
def static sumFactors = { number ->  
  factorsOf(number).inject(0, {i, j -> i + j})  
}  
  
def static sumOfFactors = sumFactors.memoize()  
  
def static isPerfect(number) {  
  sumOfFactors(number) == 2 * number  
}
```



Paradigm Over Syntax



Functional Thinking

O'REILLY®

Neal Ford

O'REILLY®

Neal Ford

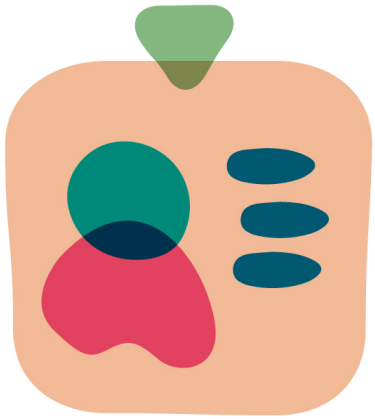
memoization

```
def static factorsOf(number) {  
  (1..number).findAll { i -> isFactor(number, i) }  
}
```

```
def static sumFactors = { number ->  
  factorsOf(number).inject(0, {i, j -> i + j})  
}
```

```
def static sumOfFactors = sumFactors.memoize()
```

```
def static isPerfect(number) {  
  sumOfFactors(number) == 2 * number  
}
```



The Company Process

Given a list of names:

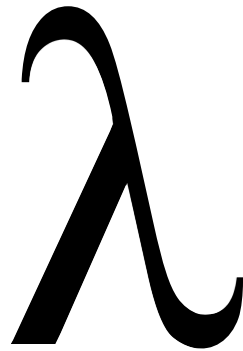
▶ remove single letter entries

▶ capitalize first letter

▶ return a comma-separated list

```
public class TheCompanyProcess {  
    public String cleanNames(List<String> listOfNames) {  
        StringBuilder result = new StringBuilder();  
        for(int i = 0; i < listOfNames.size(); i++) {  
            if (listOfNames.get(i).length() > 1) {  
                result.append(FormatUtil.capitalizeString(listOfNames.get(i))).append(",");  
            }  
        }  
        return result.substring(0, result.length() - 1).toString();  
    }  
  
    public String capitalizeFirstLetter(String s) {  
        return s.substring(0, 1).toUpperCase() + s.substring(1, s.length());  
    }  
}
```

Java 8 company Process



```
public String cleanNames(List<String> names) {  
    if (names == null) return "";  
    return names  
        .stream()  
        .filter(name -> name.length() > 1)  
        .map(name -> capitalize(name))  
        .collect(Collectors.joining(", "));  
}  
public String cleanNamesP(List<String> names) {  
    if (names == null) return "";  
    return names  
        .parallelStream()  
        .filter(n -> n.length() > 1)  
        .map(e -> capitalize(e))  
        .collect(Collectors.joining(", "));  
}
```

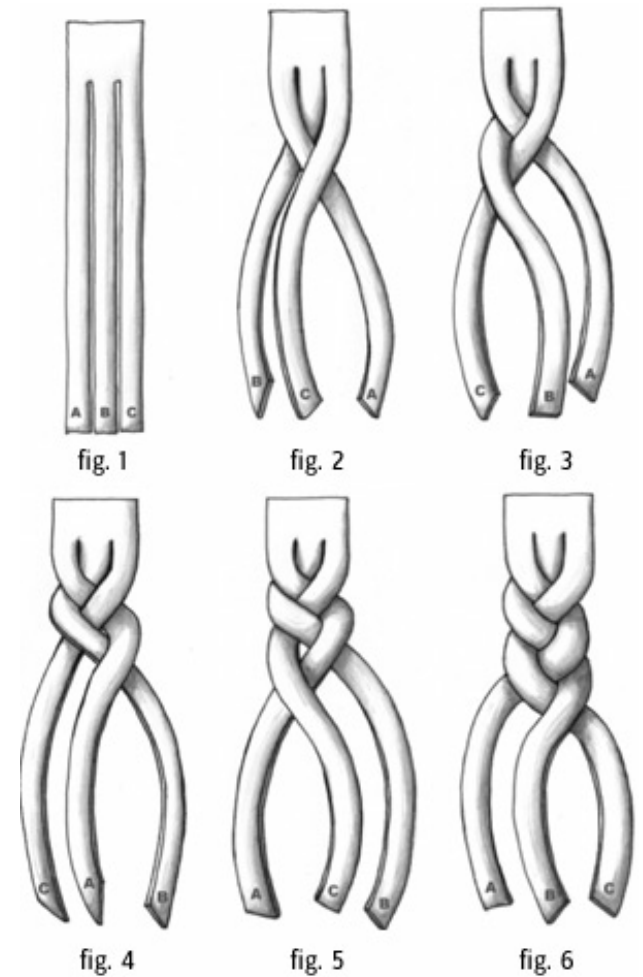
```

public class TheCompanyProcess {
    public String cleanNames(List<String> listOfNames) {
        StringBuilder result = new StringBuilder();
        for(int i = 0; i < listOfNames.size(); i++) {
            if (listOfNames.get(i).length() > 1) {
                result.append(capitalizeString(listOfNames.get(i))).append(",");
            }
        }
        return result.substring(0, result.length() - 1).toString();
    }

    public String capitalizeString(String s) {
        return s.substring(0, 1).toUpperCase() + s.substring(1, s.length());
    }
}

```

complect, *transitive verb*:
 intertwine, embrace, especially
 to plait together



```

public String cleanNames(List<String> names) {
    if (names == null) return "";
    return names
        .stream()
        .filter(name -> name.length() > 1)
        .map(name -> capitalize(name))
        .collect(Collectors.joining(","));
}

```

```

public class TheCompanyProcess {
    public String cleanNames(List<String> listOfNames) {
        StringBuilder result = new StringBuilder();
        for(int i = 0; i < listOfNames.size(); i++) {
            if (listOfNames.get(i).length() > 1) {
                result.append(capitalizeString(listOfNames.get(i))).append(",");
            }
        }
        return result.substring(0, result.length() - 1).toString();
    }

    public String capitalizeString(String s) {
        return s.substring(0, 1).toUpperCase() + s.substring(1, s.length());
    }
}

```

complect, *transitive verb*:
 intertwine, embrace, especially
 to plait together



```

public String cleanNames(List<String> names) {
    if (names == null) return "";
    return names
        .stream()
        .filter(name -> name.length() > 1)
        .map(name -> capitalize(name))
        .collect(Collectors.joining(", "));
}

```



Functional
programming
allows you to
operate at a
higher level of
abstraction.



λ

μ

Δ

look for places you
are drowning...



"Out-
of-town
Consultant
Effect"



Be your own "out of
town consultant"

Thank you for attending!

Neal Ford

Director / Software Architect /
Meme Wrangler

ThoughtWorks®

2002 Summit Blvd, Level 3, Atlanta, GA 30319, USA
T: +1 404 242 9929 Twitter: @neal4d
E: nford@thoughtworks.com W: thoughtworks.com

E: nford@thoughtworks.com W: thoughtworks.com
T: +1 404 242 9929 Twitter: @neal4d
2002 Summit Blvd, Level 3, Atlanta, GA 30319, USA