

please

Ask questions  
through the app



*Rate Session*

Thank you!



# Unconditional Code

Michael Feathers  
R7K Research & Conveyance

$$n + 4$$

```
public class MyClass {  
    private int lineCount;  
    private String sFileName = "myfile";  
    public static void main(String[] args) throws IOException {  
        try {  
            lineCount = LineCounter.countLines(sFileName);  
        } catch (IOException e) {  
            throw new IllegalArgumentException("Unable to load " + sFileName, e);  
        }  
    }  
}
```

```

function [FFTVals] = EEG_FFT(eeg, L, Interval)
    nchn = min(size(eeg));
    max_index = floor((length(eeg) - L) / Interval);
    FFTVals = cell(nchn, 1);

    for n = 1:nchn
        FFTVals{n} = zeros(max_index + 1, L/2 + 1);
    end

    % calculate the fft values
    for index = 0:max_index
        for n = 1:nchn
            temp_dat = eeg(n, index*Interval + (1:L));
            % hamming
            temp_han = temp_dat'.*hamming(L);
            temp_fft = fft(temp_han);
            temp_mag = abs(temp_fft/L);
            temp_mag = temp_mag(1:L/2+1);
            temp_mag(2:end-1) = 2*temp_mag(2:end-1);
            FFTVals{n}(index+1,:) = temp_mag;
        end
    end
end

```

# Logging



*The Addison-Wesley Signature Series*



# GROWING OBJECT-ORIENTED SOFTWARE, GUIDED BY TESTS

STEVE FREEMAN  
NAT PRYCE



```
}catch MyError.AnError {  
    print("AnError")  
}catch MyError.AnotherError {  
    print("AnotherError") //AnotherError will be caught and printed  
}catch{  
    print("Something else happened")  
}  
  
do{  
    do{  
        try throwsError()  
    }catch MyError.AnError {  
        print("AnError")  
    }  
}catch MyError.AnotherError {  
    print("AnotherError") //AnotherError will be caught and printed
```





*Noticeable Error Handling is a Symptom of Bad Design*

## Review Your Order


Look this over. If it all looks right, click the "place your order" button to finish buying your stuff.

### SHIPPING & PAYMENT

 **Address Book**

 **To complete the Amazon Pay checkout you must accept third-party cookies in the options/settings of your internet browser. After you have accepted third-party cookies please refresh this page to continue check out.**

[Amazon Pay](#)  
[Privacy](#)

 **Payment Method**

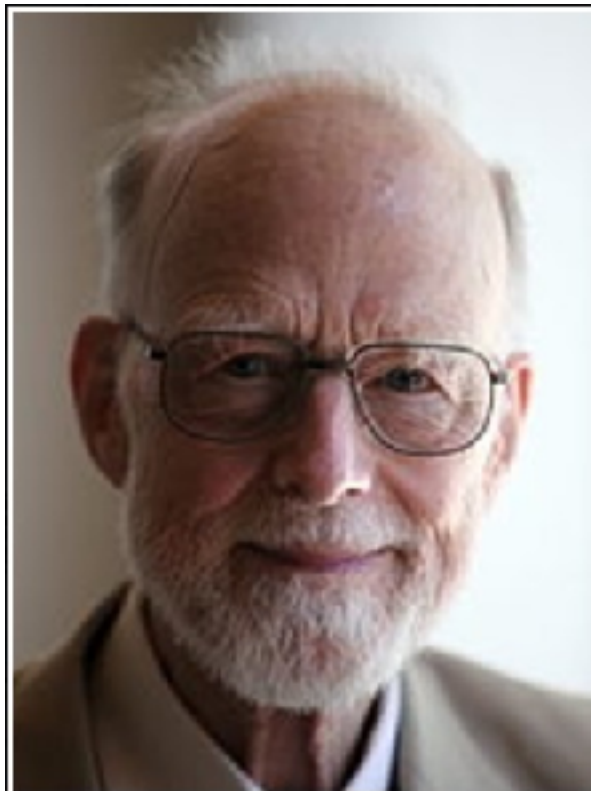
**Your session has expired. Please sign in again by clicking on the Amazon Pay Button.**

[Amazon Pay](#)  
[Privacy](#)

*Code Should Just Run - Unconditionally*

*Deep Dive..*

```
public Item itemForBarcode(String barcode) {  
    Item item = items.get(barcode);  
    if (item != null)  
        return item;  
    return null;  
}
```



I call it my billion-dollar mistake. It  
was the invention of the null  
reference in 1965.

*Tony Hoare*

AZ QUOTES

```
public Item itemForBarcode(String barcode) throws ItemNotFound {  
    Item item = items.get(barcode);  
    if (item == null)  
        throw new ItemNotFound(barcode);  
    return item;  
}
```



*“Use exceptions when you can’t know in advance whether a call will succeed or fail.”*

Bertrand Meyer



```
public void populateSigns(List<SignProvider> providers) {  
    // ...  
}
```

```
public void populateSigns(List<SignProvider> providers) {  
    // ...  
}
```

```
public void run() {  
    // ...  
    populateSigns(null);  
    //...  
}
```

```
public void formReport() {  
    boolean hasMarkedAccounts = findMarkings();  
    boolean inTallyPeriod = CalculationPeriods.hasTallyDate(currentDate);  
  
    // ... 1000 lines here  
  
    if (hasMarkedAccounts && !inTallyPeriod) {  
        // ...  
    }  
}
```

# Tunneling



Is interpretation the problem?

The string is a stark data structure and everywhere it is passed there is much duplication of process. It is a perfect vehicle for hiding information.

*Alan Perlis*

*Can We Eliminate Tunnels?*

```
public Item itemForBarcode(String barcode) {  
    return items.getOrDefault(barcode, new Item("Item not found", 0));  
}
```



```
public Item itemForBarcode(String barcode) {  
    return items.getOrDefault(barcode, new Item("Item not found", 0));  
}
```

NaN

[] null object

special case

```
public interface SaleListener {  
    void itemAdded(Item item);  
    void saleTotalled(int total);  
}
```

```
public interface SaleListener {  
    void itemAdded(Item item);  
    void saleTotalled(int total);  
    void itemNotFound(String barcode);  
}
```

## Tell, Don't Ask

Alec Sharp, in the recent book *Smalltalk by Example* [\[SHARP\]](#), points up a very valuable lesson in few words:

*Procedural code gets information then makes decisions. Object-oriented code tells objects to do things.*  
— Alec Sharp

```
Person person = dataSource.getPersonById(personId);  
if (person != null) {  
    person.setPhoneNumber(phoneNumber);  
    dataSource.updatePerson(person);  
}
```

```
data_source.person(id) do |person|  
  person.phone_number = phone_number  
  data_source.update_person person  
end
```

```
dataSource.person(id, new Action<Person>() {  
    public void act(Person person) {  
        person.setPhoneNumber(phoneNumber);  
        dataSource.updatePerson(person);  
    }  
});
```

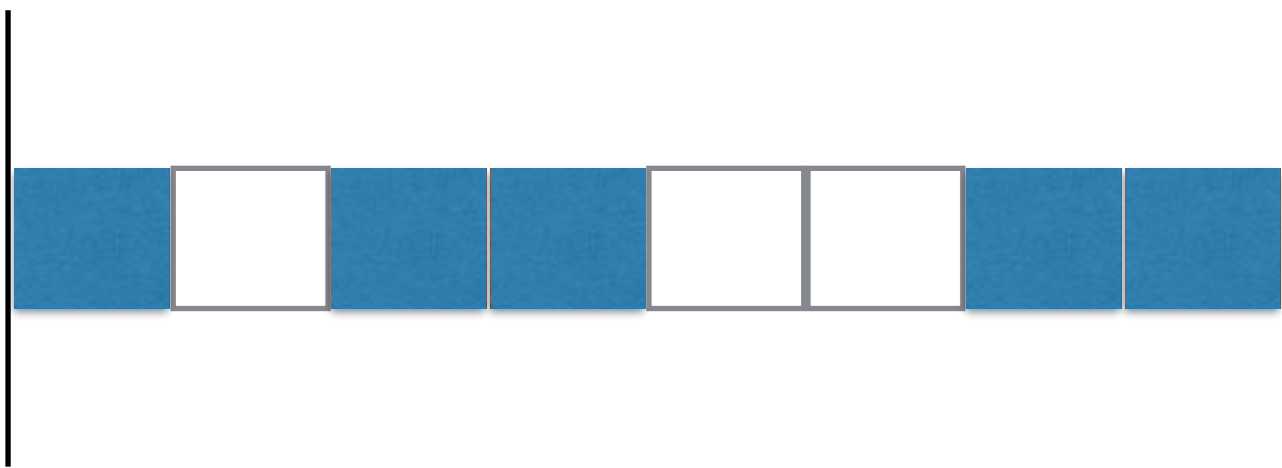
*Extend the Domain*

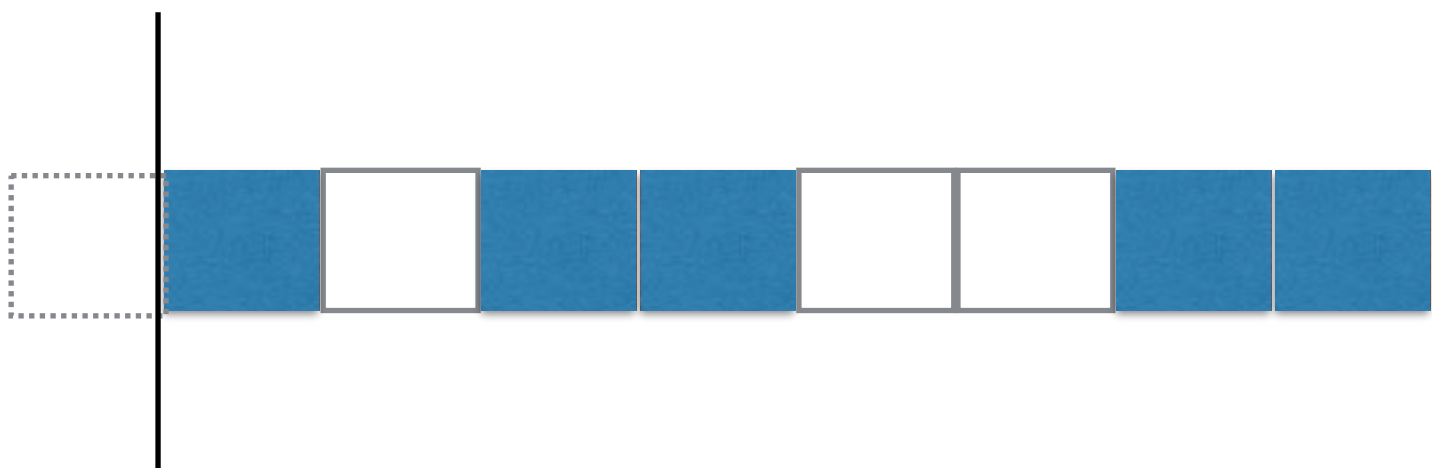


*`Domain' Means Something In Mathematics*



```
def span_count ary
  return 0 if ary.size == 0
  count = 0
  if ary[0] > 0
    count = 1
  end
  i = 0
  while i < ary.size - 1
    if ary[i] == 0 && ary[i+1] != 0
      count = count + 1
    end
    i = i + 1
  end
  return count
end
```





```
# return the number of contiguous spans of non-zeros in an array
#
# :: [Integer] -> Integer
def span_count ary
  ([0] + ary).lazy
    .each_cons(2)
    .count{|c,n| c == 0 && n != 0 }
end
```

0--5--3--2--0-----1--0-----0-----0--1--0-----  
-----3-----1--3-----3--0-----  
-----7--2--2-----2--4--0-----0--2--  
-----2--0-----1--0-----  
-----

0--5--3--2--0-----1--0-----0-----0--1--0-----  
 -----3-----1--3-----3--0-----  
 -----7--2--2-----2--4--0-----0--2--  
 -----2--0-----1--0-----  
 -----

1 0  
 1 5  
 1 3  
 1 2  
 1 0  
 2 3  
 1 1  
 1 0  
 3 7  
 3 2  
 3 2  
 2 1  
 2 3  
 1 0  
 3 2  
 3 4  
 3 0



0 5 3 2 0 1 0 0 0 0 1 0

3 1 3 0

7 2 2 2 4 0 2 0 0 2

2 0 1 0

1 0

1 5

1 3

1 2

1 0

2 3

1 1

1 0

3 7

3 2

3 2

21

1 0  
1 5  
1 3  
1 2  
1 0  
2 3  
1 1  
1 0  
3 7  
3 2  
3 2  
2 1  
2 3  
1 0  
3 2  
3 4  
3 0  
4 2  
4 0  
5 1  
5 0  
1 0  
1 1  
1 0  
2 3  
2 0  
3 0  
3 2

- 0. Command line argument for the filename may be missing
- 1. Unable to open an input file
- 2. File is empty
- 3. File contains empty lines
- 4. Our input file is not a text file
- 5. A line has more than two numbers
- 6. A line has less than two numbers
- 7. A line has fields that can not be parsed as numbers
- 8. The string number is less than one or more than six
- 9. The fret number is less than zero or more than twenty-four

```
STRING_COUNT = 6
```

```
def tab_column string, fret
  ["---"          ] * (string - 1) +
  [fret.ljust(3, '-') ] +
  ["---"          ] * (STRING_COUNT - string)
end
```

```
unless File.exist? ARGV[0]
  abort "Unable to open #{ARGV[0]}"
end
```

```
File.open(ARGV[0], "r") do |f|
  puts f.each_line
    .map(&:split)
    .map {|string, fret| tab_column(string.to_i, fret) }
    .transpose
    .map(&:join)
    .join($/)
end
```

```
STRING_COUNT = 6
```

```
def tab_column string, fret
  ["---"          ] * (string - 1) +
  [fret.ljust(3, '-') ] +
  ["---"          ] * (STRING_COUNT - string)
end
```

```
unless File.exist? ARGV[0]
  abort "Unable to open #{ARGV[0]}"
end
```

← Hmmm..

```
File.open(ARGV[0], "r") do |f|
  puts f.each_line
    .map(&:split)
    .map {|string, fret| tab_column(string.to_i, fret) }
    .transpose
    .map(&:join)
    .join($/)
end
```

```
STRING_COUNT = 6
```

```
def tab_column string, fret
  ["---"          ] * (string - 1) +
  [fret.ljust(3, '-')] +
  ["---"          ] * (STRING_COUNT - string)
end
```

```
puts ARGF.each_line
      .map(&:split)
      .map {|string, fret| tab_column(string.to_i, fret) }
      .transpose
      .map(&:join)
      .join($/)
```

- X 0. Command line argument for the filename may be missing
- X 1. Unable to open an input file
- X 2. File is empty
- 3. File contains empty lines
- 4. Our input file is not a text file
- 5. A line has more than two numbers
- 6. A line has less than two numbers
- 7. A line has fields that can not be parsed as numbers
- 8. The string number is less than one or more than six
- 9. The fret number is less than zero or more than twenty-four

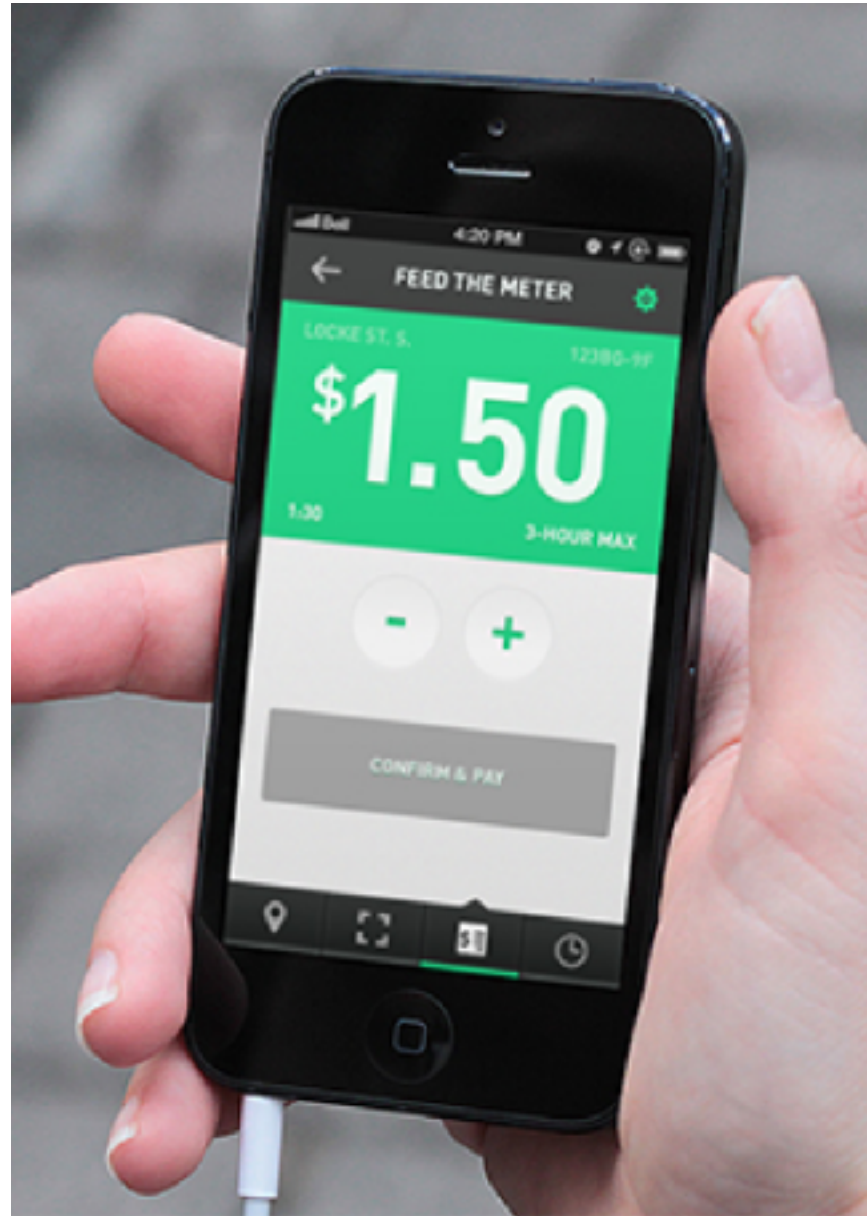
1  
1 5  
1 3  
1 2  
1  
2 3

```
#include<cmath.h>  
double sqrt (double x );
```



Indices may also be negative numbers, to start counting from the right:

```
>>> word[-1]  # last character
'n'
>>> word[-2]  # second-last character
'o'
>>> word[-6]
'p'
```



Extend the domain only when you feel that people can eventually consider it the normal domain

- ✗ 0. Command line argument for the filename may be missing
- ✗ 1. Unable to open an input file
- ✗ 2. File is empty
- 3. File contains empty lines
- ✗ 4. Our input file is not a text file
- 5. A line has more than two numbers
- 6. A line has less than two numbers
- 7. A line has fields that can not be parsed as numbers
- 8. The string number is less than one or more than six
- 9. The fret number is less than zero or more than twenty-four

```

STRING_COUNT = 6

def tab_column string, fret
  ["---"          ] * (string - 1) +
  [fret.ljust(3, '-')] +
  ["---"          ] * (STRING_COUNT - string)
end

lines = ARGF.each_line
      .select {|l| l =~ /\S/ }
      .map(&:split)

check("each line should have two fields") do |line_fields|
  line_fields.count == 2
end

check("all fields should be integers") do |string, fret|
  converts_to_int(string) && converts_to_int(fret)
end

check("strings should be in the range 1..6") do |string, _|
  string >= 1 && string <= 6
end

puts lines.each_line
      .map {|string, fret| tab_column(string.to_i, saturate(fret.to_i, (0..99))) }
      .transpose
      .map(&:join)
      .join($/)

```

```
STRING_COUNT = 6
```

```
def tab_column string, fret
  ["---"          ] * (string - 1) +
  [fret.ljust(3, '-')] +
  ["---"          ] * (STRING_COUNT - string)
end
```

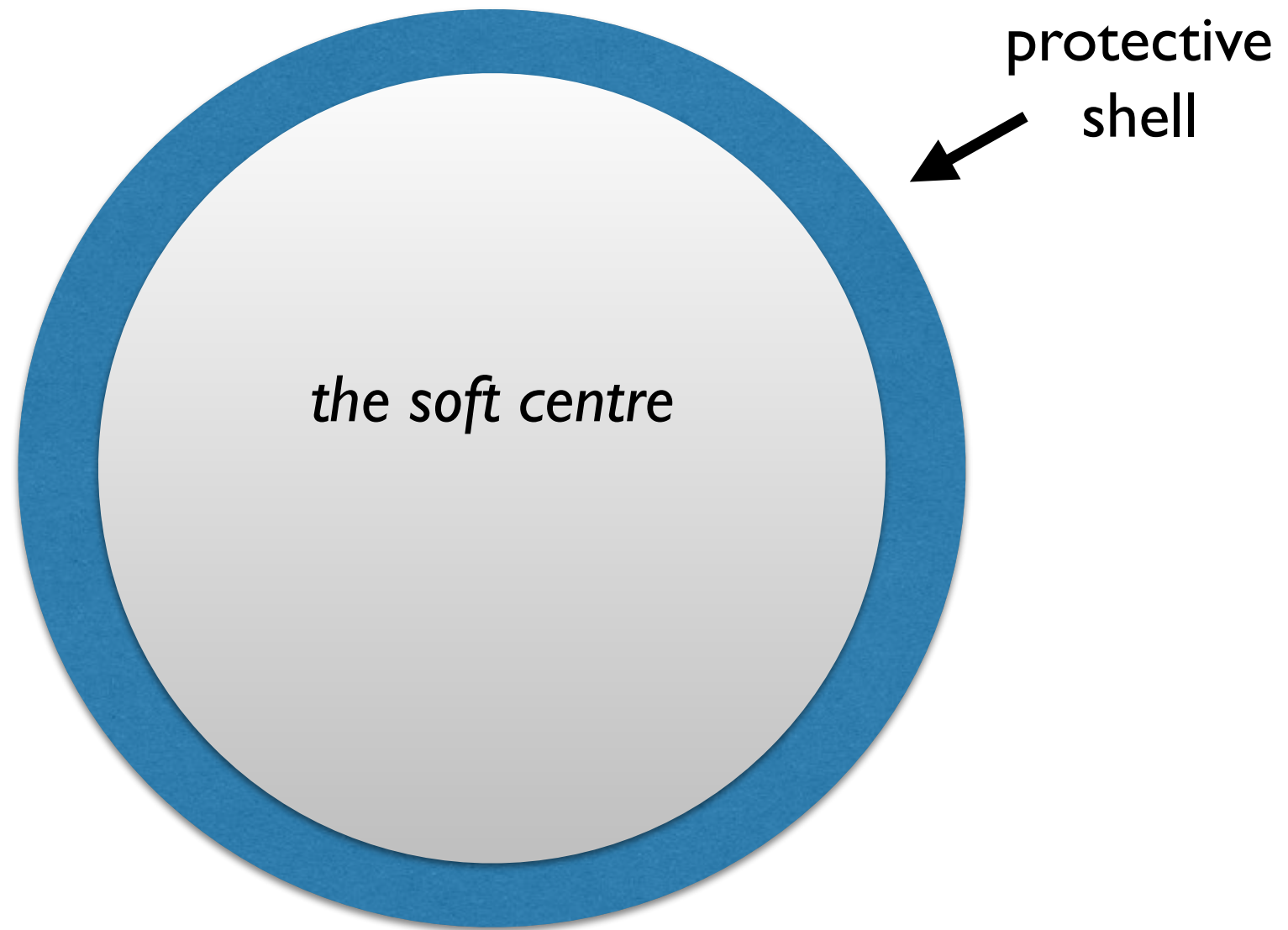
```
lines = ARGF.each_line
      .select {|l| l =~ /\S/ }
      .map(&:split)
```

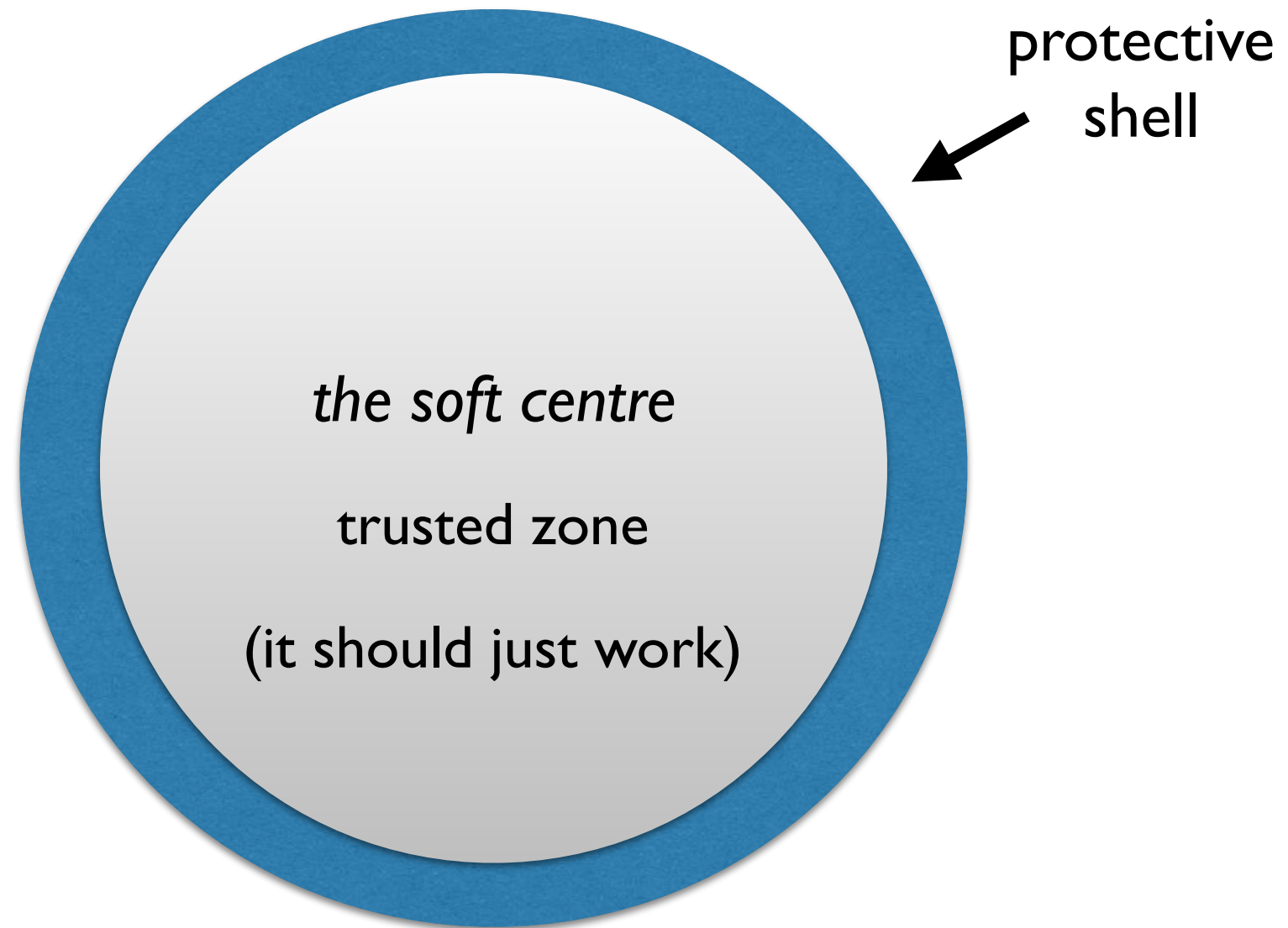
```
check("each line should have two fields") do |line_fields|
  line_fields.count == 2
end
```

```
check("all fields should be integers") do |string, fret|
  converts_to_int(string) && converts_to_int(fret)
end
```

```
check("strings should be in the range 1..6") do |string, _|
  string >= 1 && string <= 6
end
```

```
puts lines.map {|string, fret| tab_column(string.to_i, saturate(fret.to_i, (0..99))) }
      .transpose
      .map(&:join)
      .join($/)
```





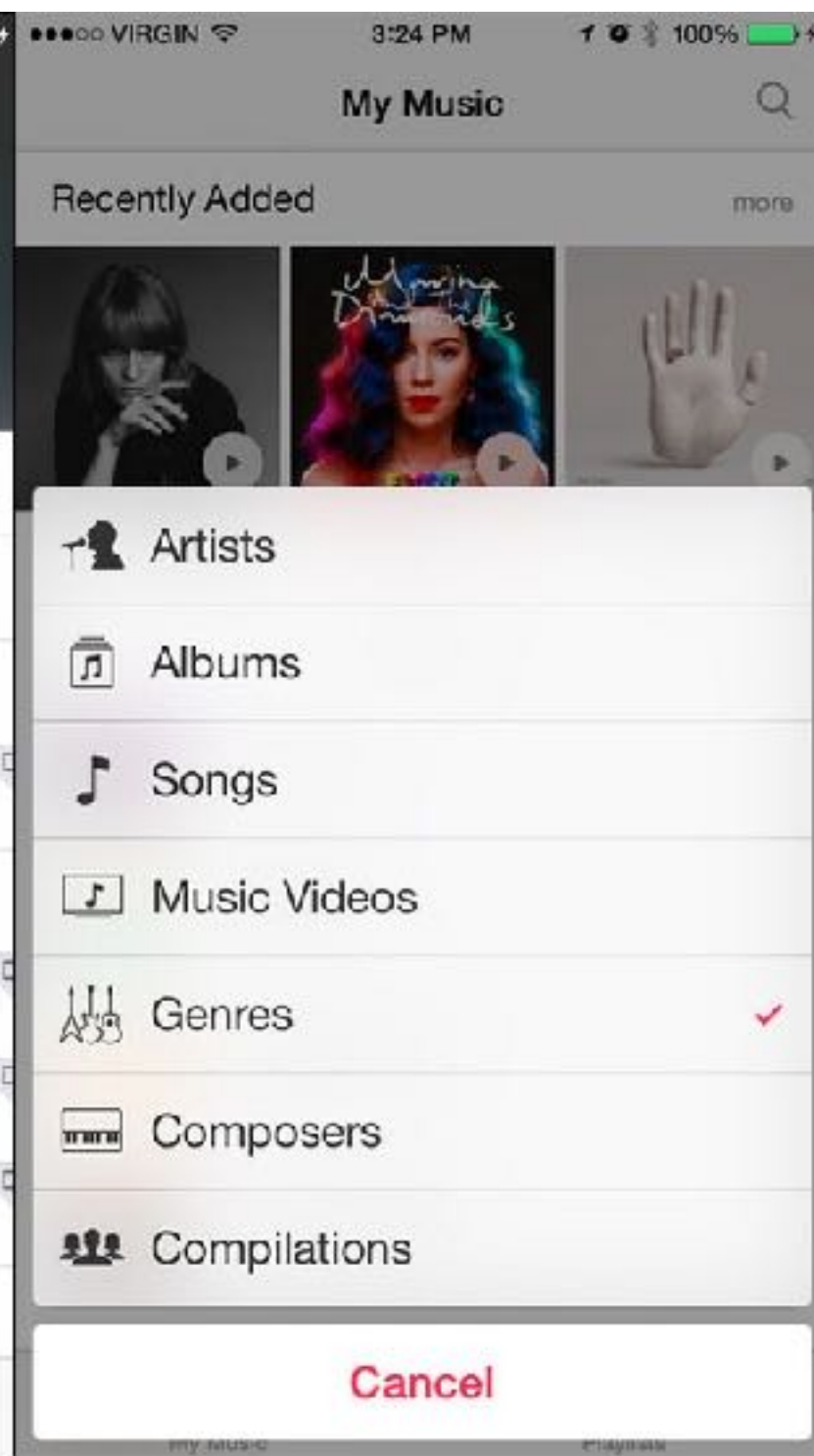


```
STRING_COUNT = 6
```

```
def tab_column string, fret
  ["---" * (string - 1) +
   [fret.ljust(3, '-')] +
   ["---" * (STRING_COUNT - string)
  end
```

```
puts ARGF.each_line
  .either
  .map(&:split)
  .check("two fields per line") { |fs| fs.count == 2 }
  .check("two ints per line")   { |fs| fs.all? { |f| int?(f) } }
  .check("string # in [1..6]")  { |fs| in_range(1,6,fs[0].to_i) }
  .check("fret # in [0..24]")   { |fs| in_range(1,24,fs[1].to_i) }
  .map { |string,fret| tab_column(string.to_i, fret) }
  .transpose
  .map(&:join)
  .join($/)
```

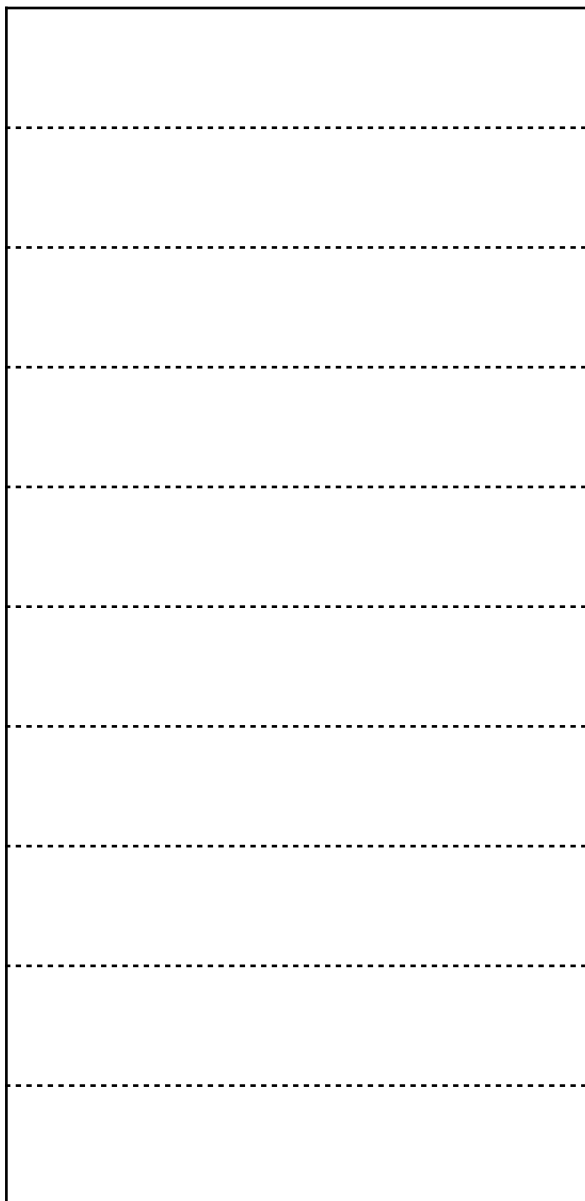
# *Intentions as a Higher Model*



0


n-1

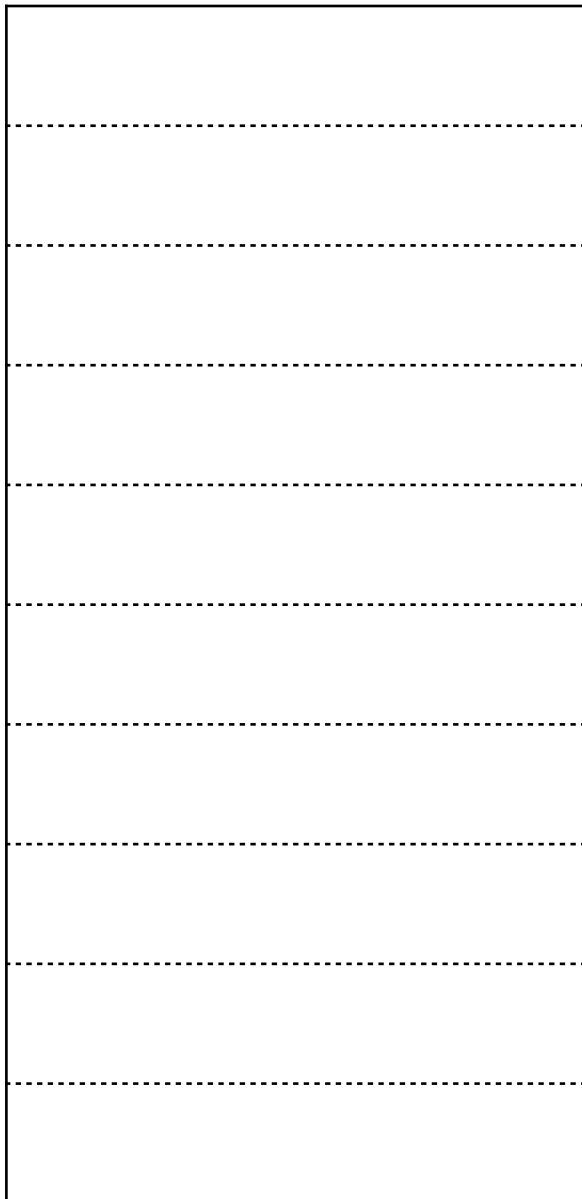
0



n-1

`current_rand_n = rand(N)`

0



n-1

do

current\_rand\_n = rand(N)

while current\_rand\_n == last\_rand\_n

last\_rand\_n = current\_rand\_n

0

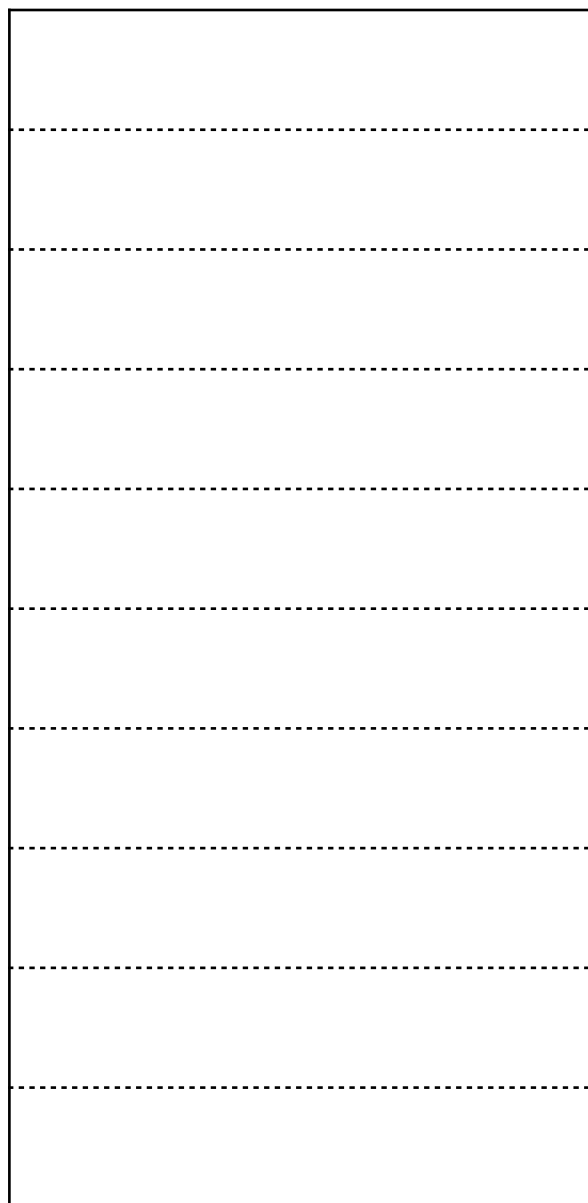

n-1

```
current_rand_n = rand(N)
if current_rand_n == last_rand_n
    current_rand_n = (last_rand_n + 1) % N
last_rand_n = current_rand_n
```

copy

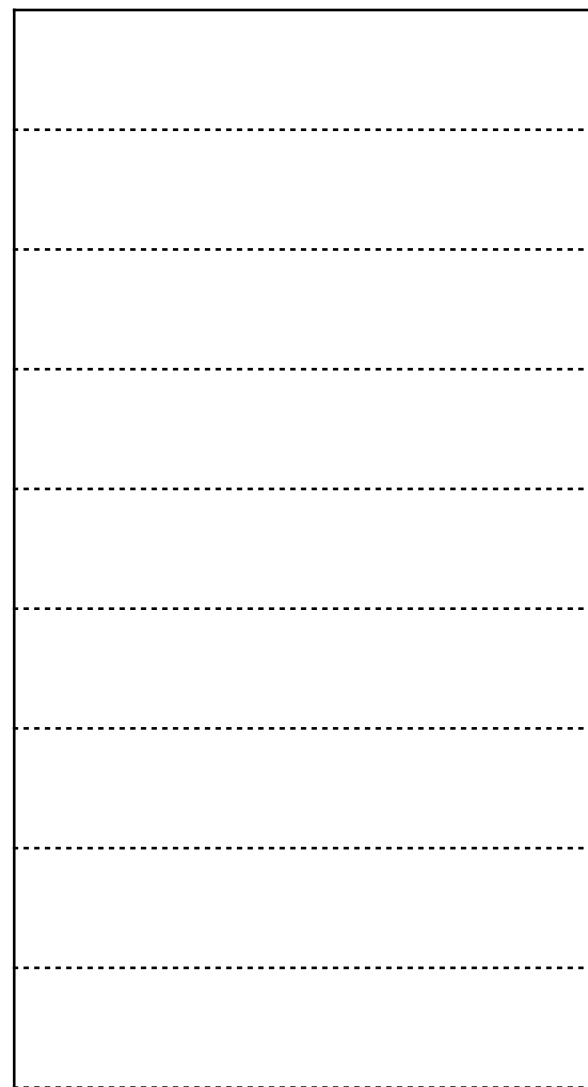


0



n-1

0



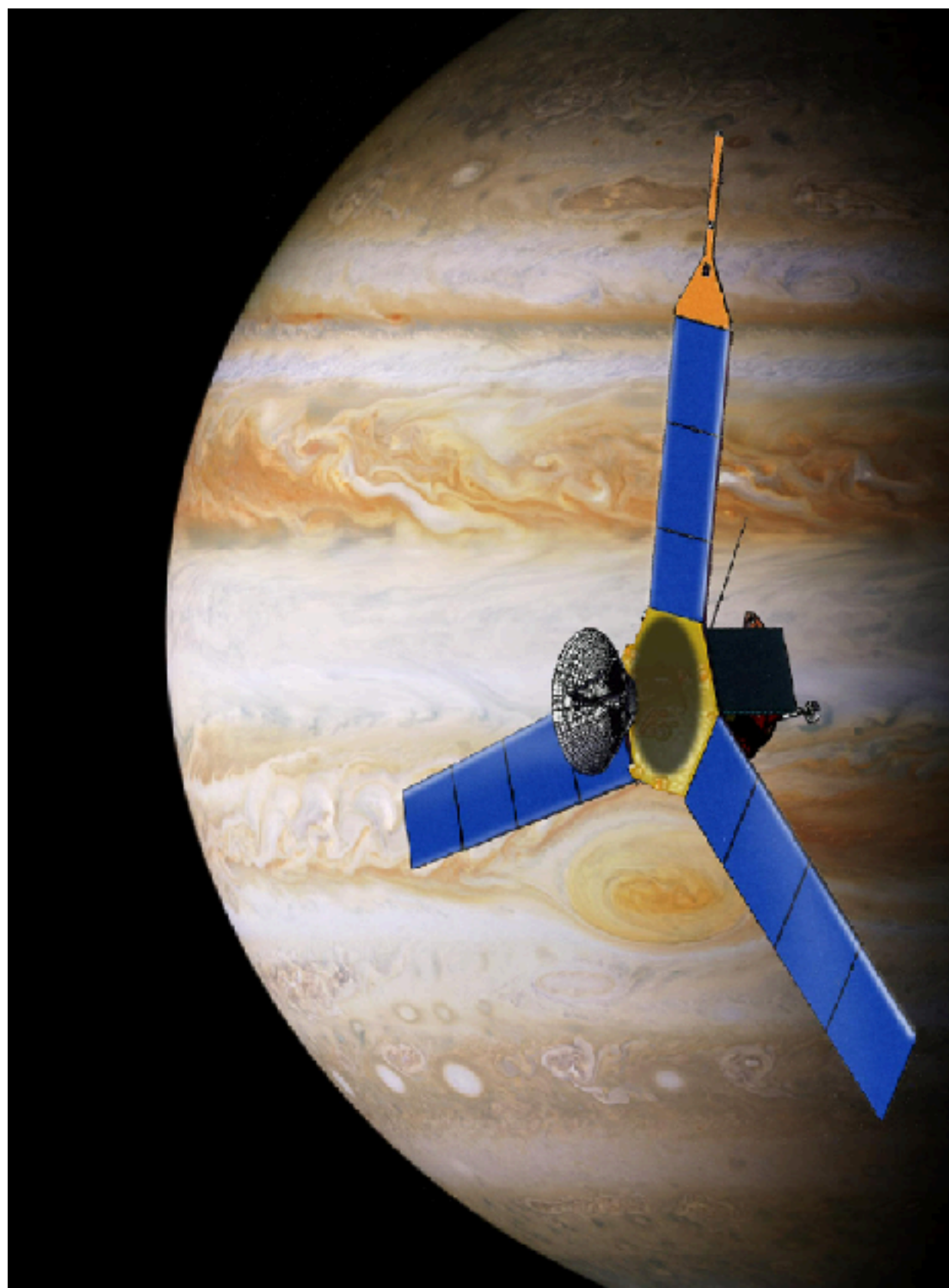
n-1





# *The Edge of the System*









For robustness you need a human  
in the loop



**Antifragile**









# **Why Do Computers Stop and What Can Be Done About It?**

Jim Gray

# Why Do Computers Stop and What Can Be Done About It?

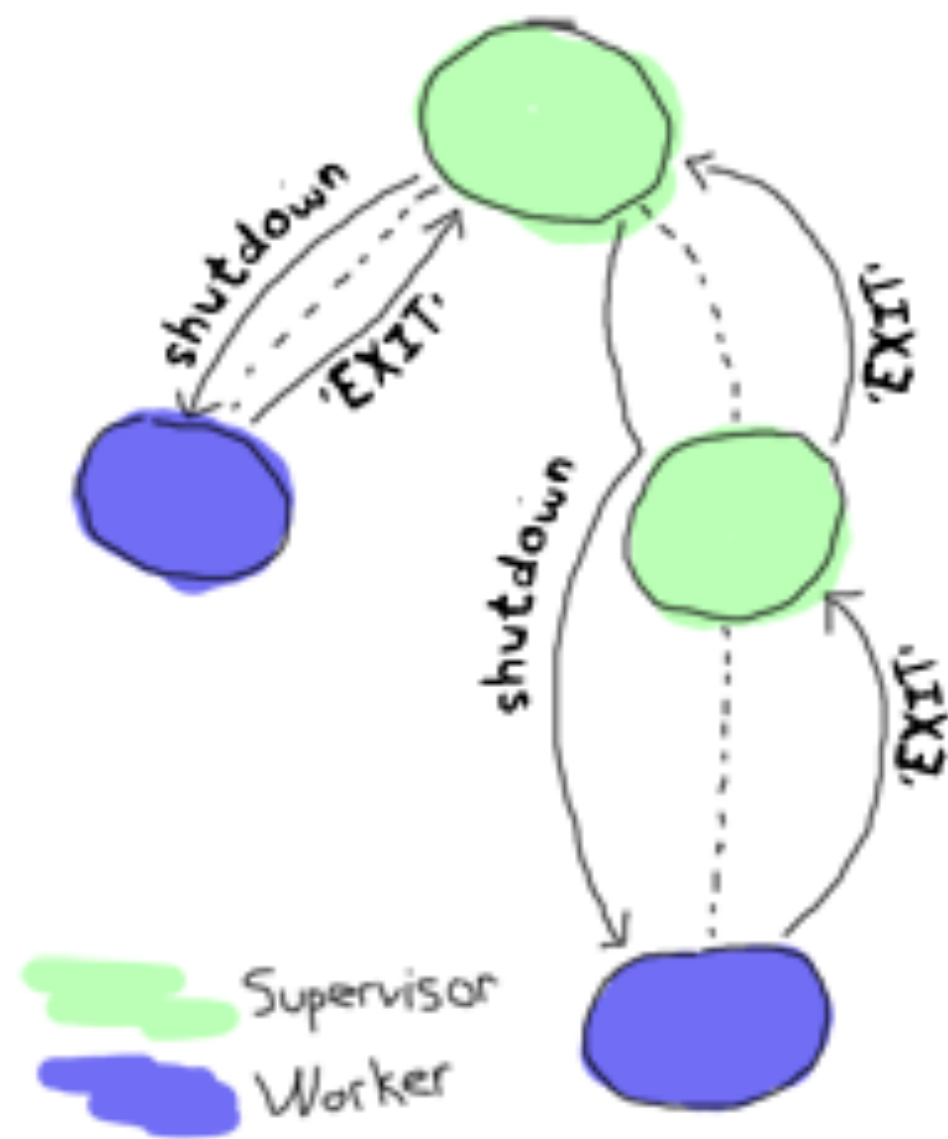
I conjecture that there is a similar phenomenon in software -- most production software faults are soft. If the program state is reinitialized and the failed operation retried, the operation will usually not fail the second time.

Jim Gray



ERLANG

**KEEP  
CALM  
AND  
LET IT  
CRASH**



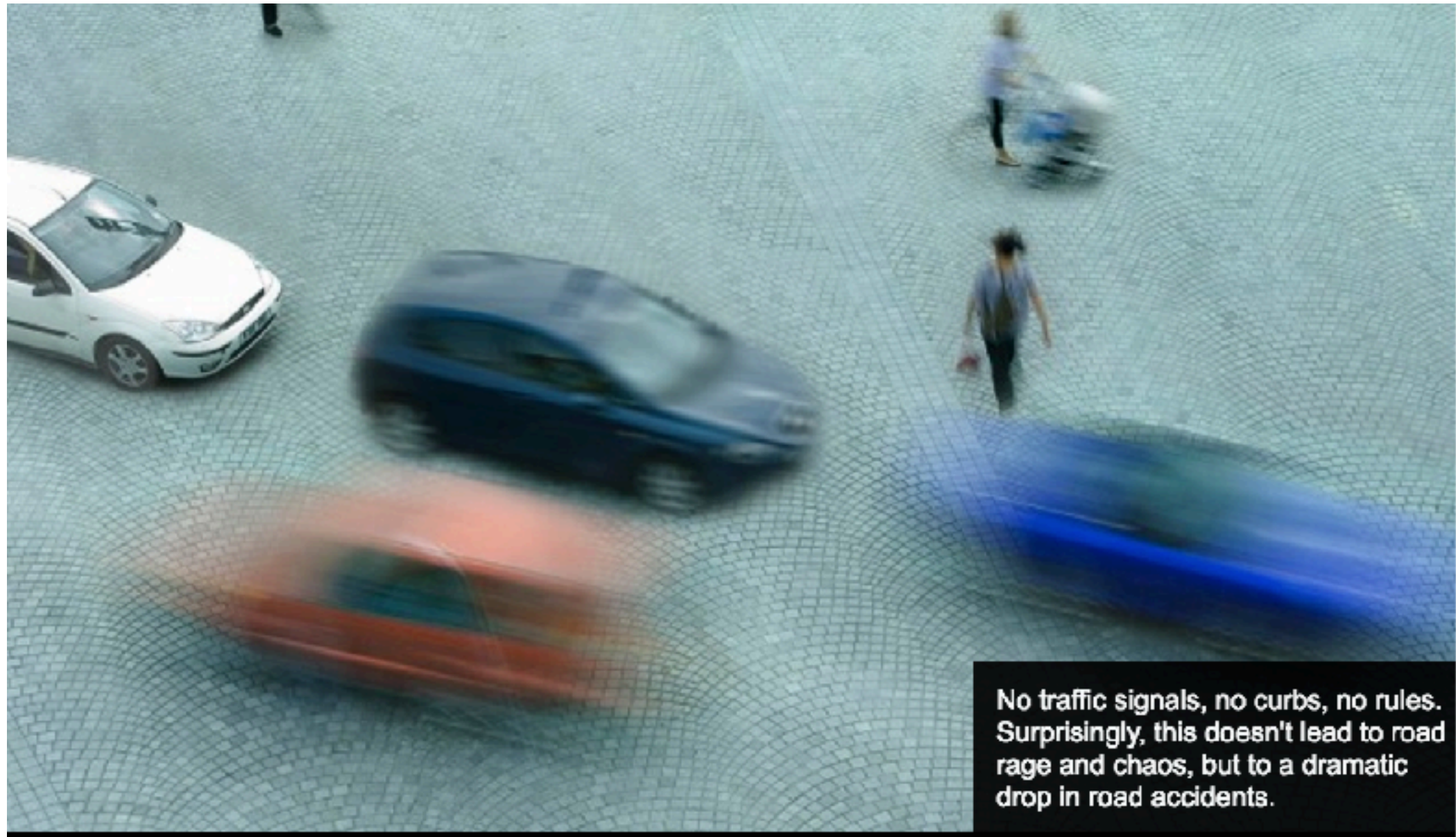
Safety is hard work.  
Where do we want it?



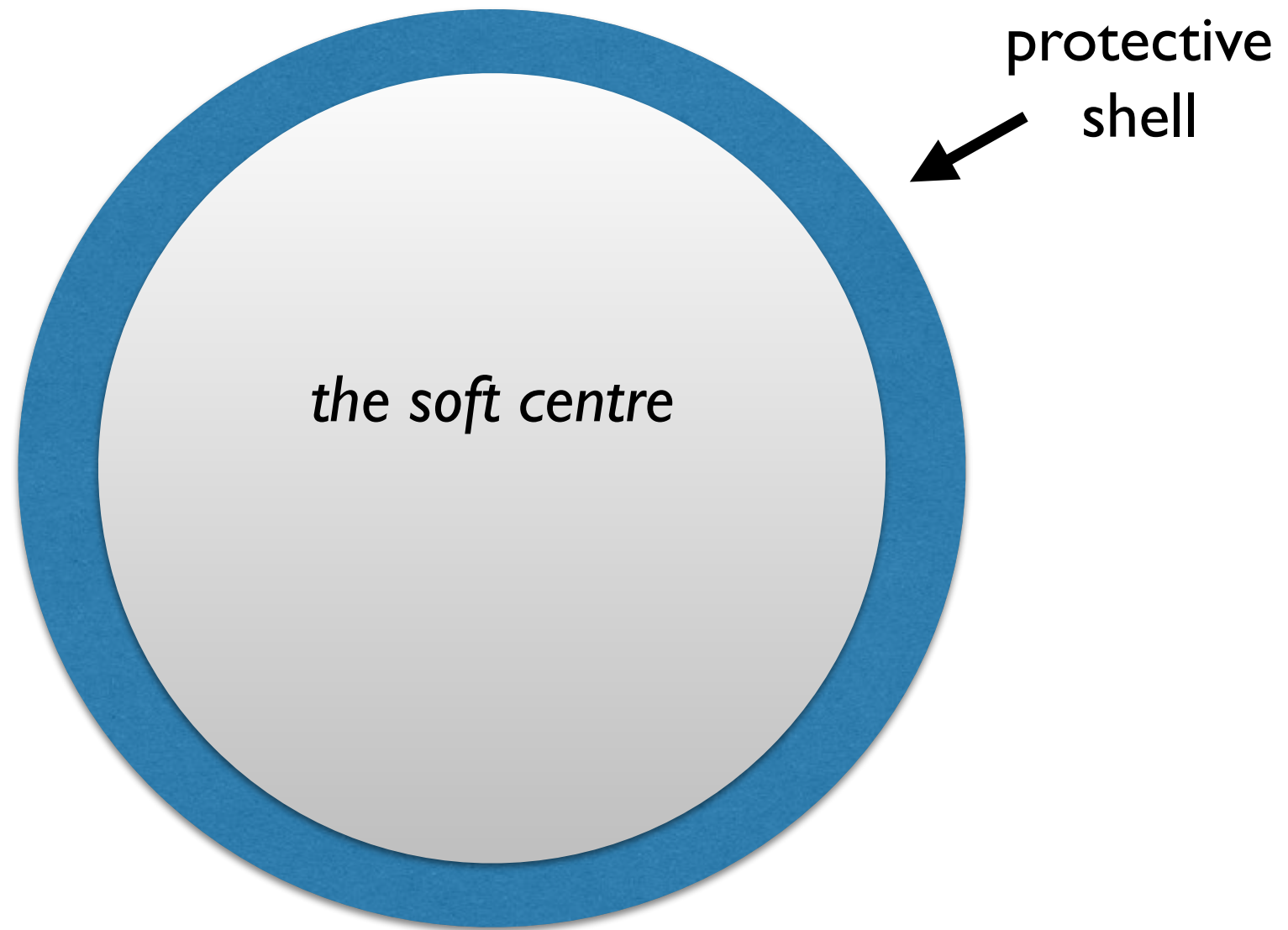
**Bullet-proofing is important when there  
is no supervision**

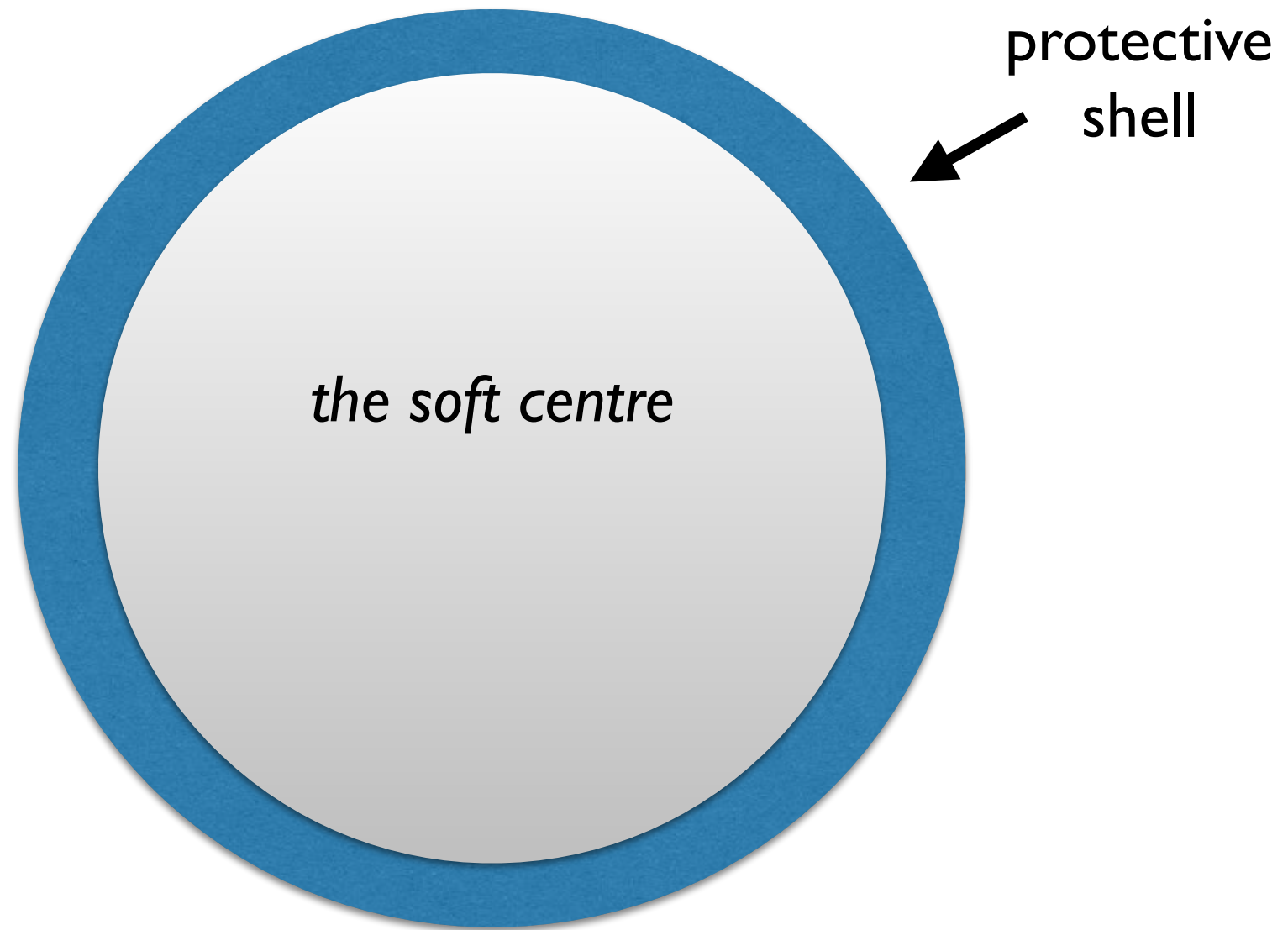
**What is the cost of safety?**





No traffic signals, no curbs, no rules. Surprisingly, this doesn't lead to road rage and chaos, but to a dramatic drop in road accidents.





*where are we?*

Consider ALL cases to make design better

# Challenge Errors



*Please*

**Remember to  
rate this session**

*Thank you!*

