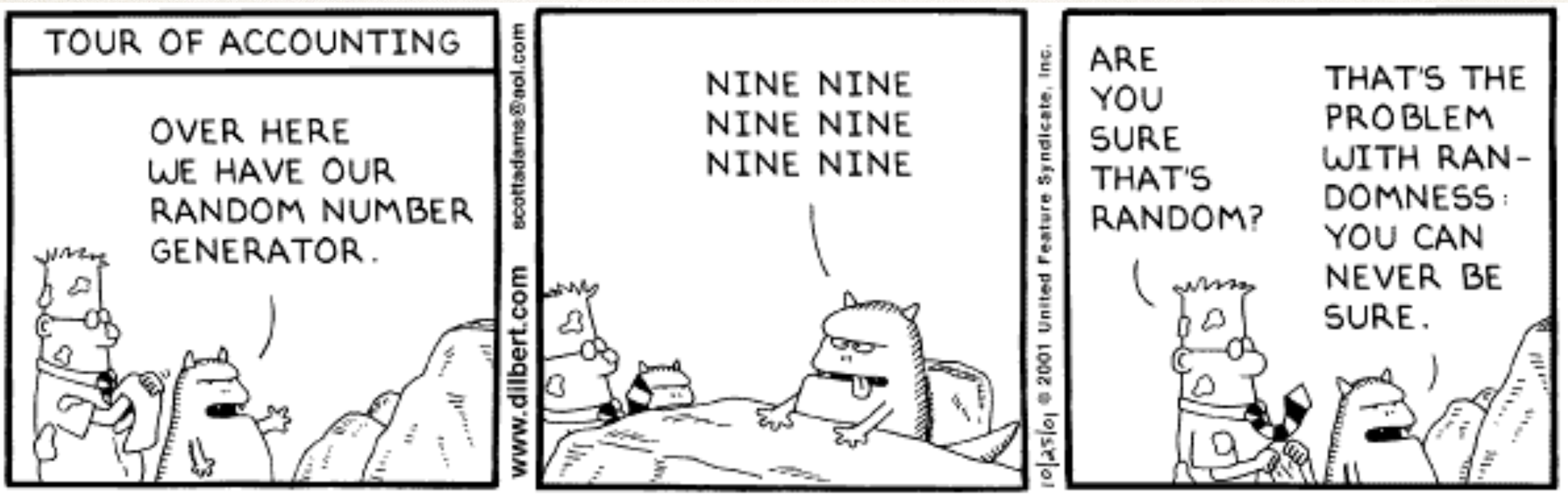


Random Testing

Validation & Verification

COMP0103-A7P
COMP0103-ATU

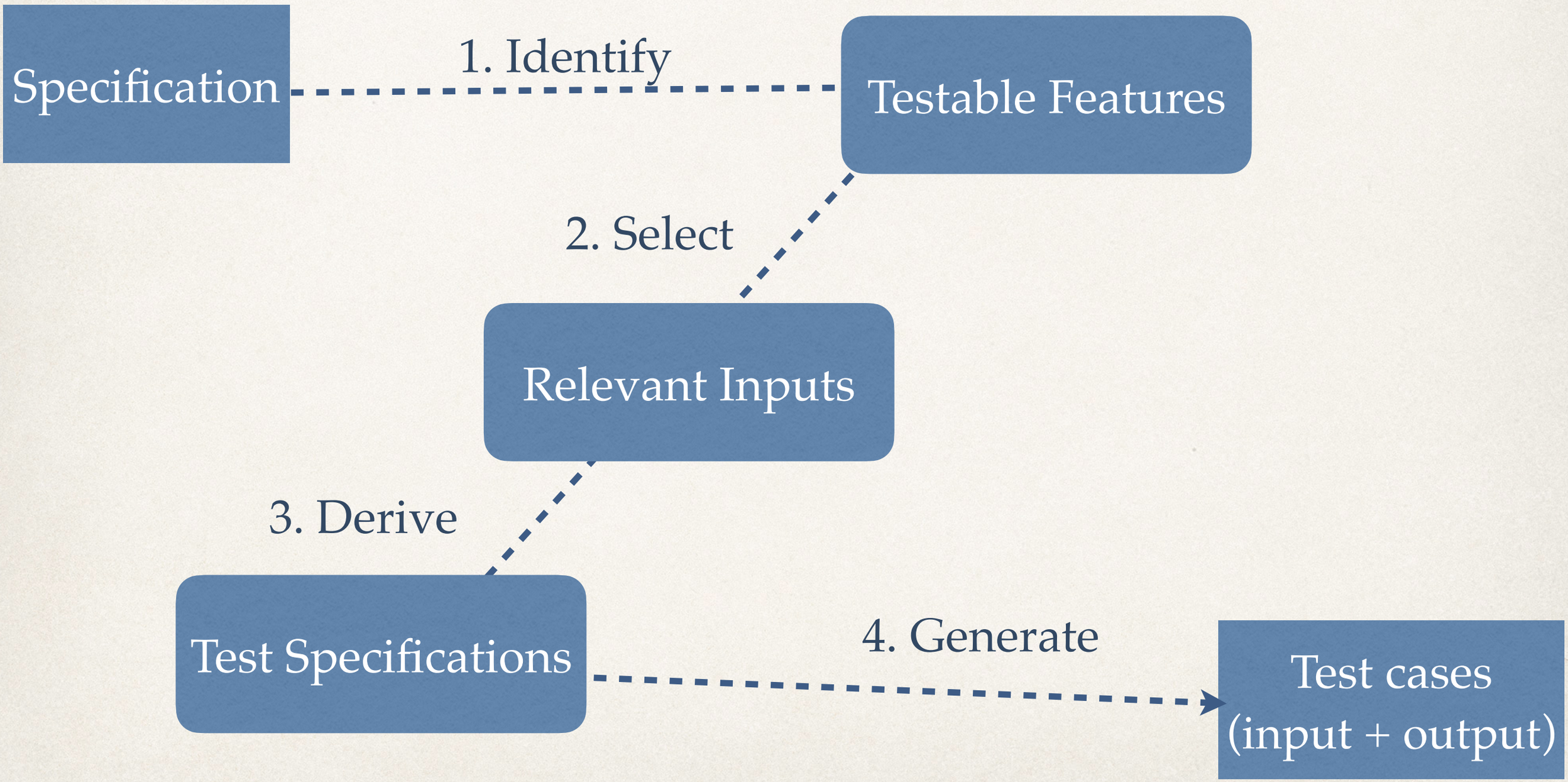
Part of the slides are used with kind permission of Dr Yue Jia

Recap

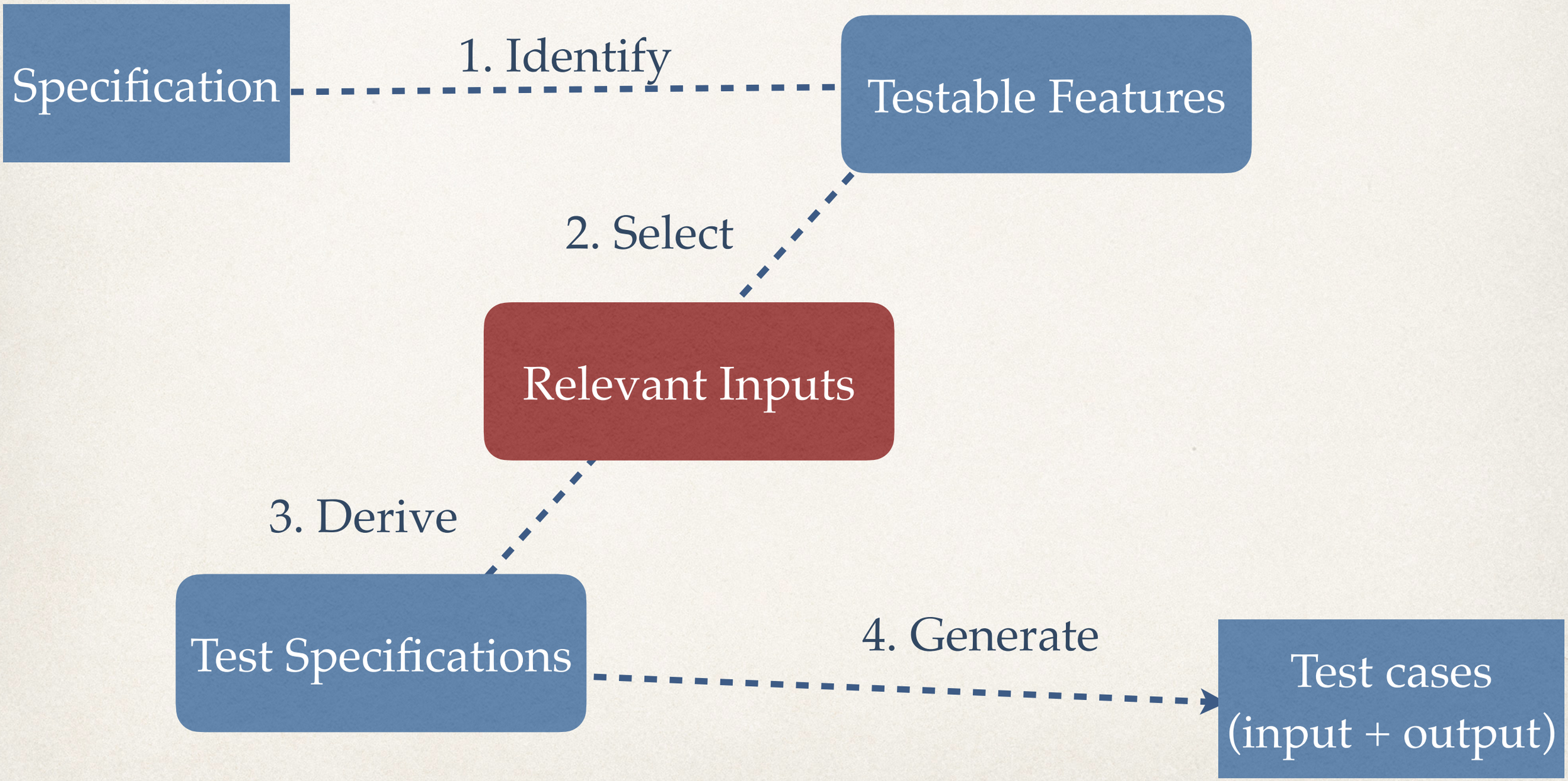
UCL

- Black-Box Testing
- Equivalence Partitioning
- Boundary Value Analysis
- Category Partition Method
- Unit Testing & JUnit

Recap: General Process

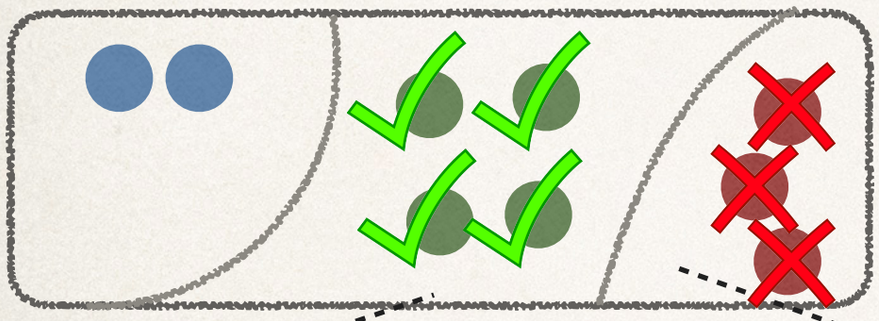


Recap: General Process



Recap: Partitioning Testing & Boundary Value Analysis

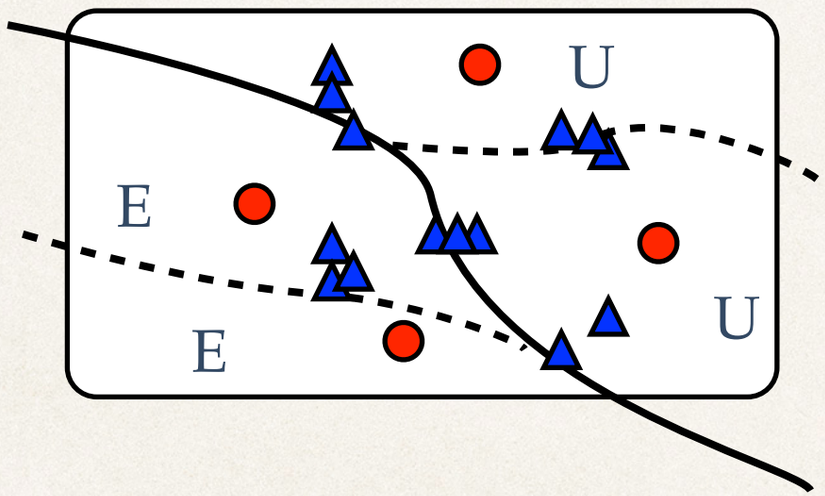
UCL



Input Domain

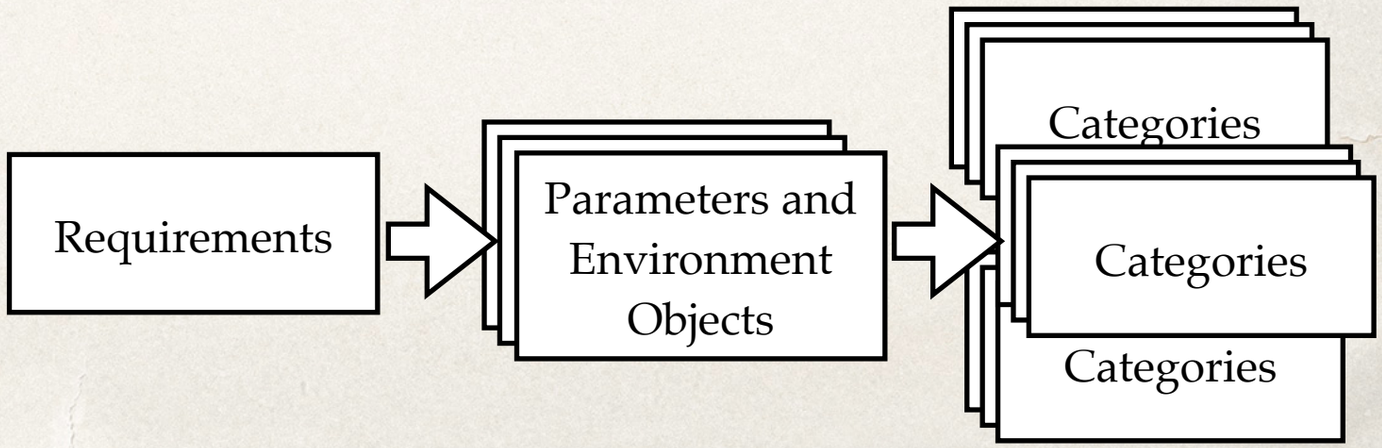
correct behaviour

Incorrect behaviour



Equivalence Partitioning

Boundary Value Analysis



Recap: Category Partition Method

1. **Decompose the specification into independently testable features**
   - for each feature identify *parameters* and *environment elements*
   - for each parameter and environment element identify elementary characteristics (*categories*)
2. **Identify relevant values**
   - for each characteristic (category) identify (classes of) values
   - normal values, boundary values, special values, error values
3. **Introduce constraints**

Recap: Category Partition - Example

UCL

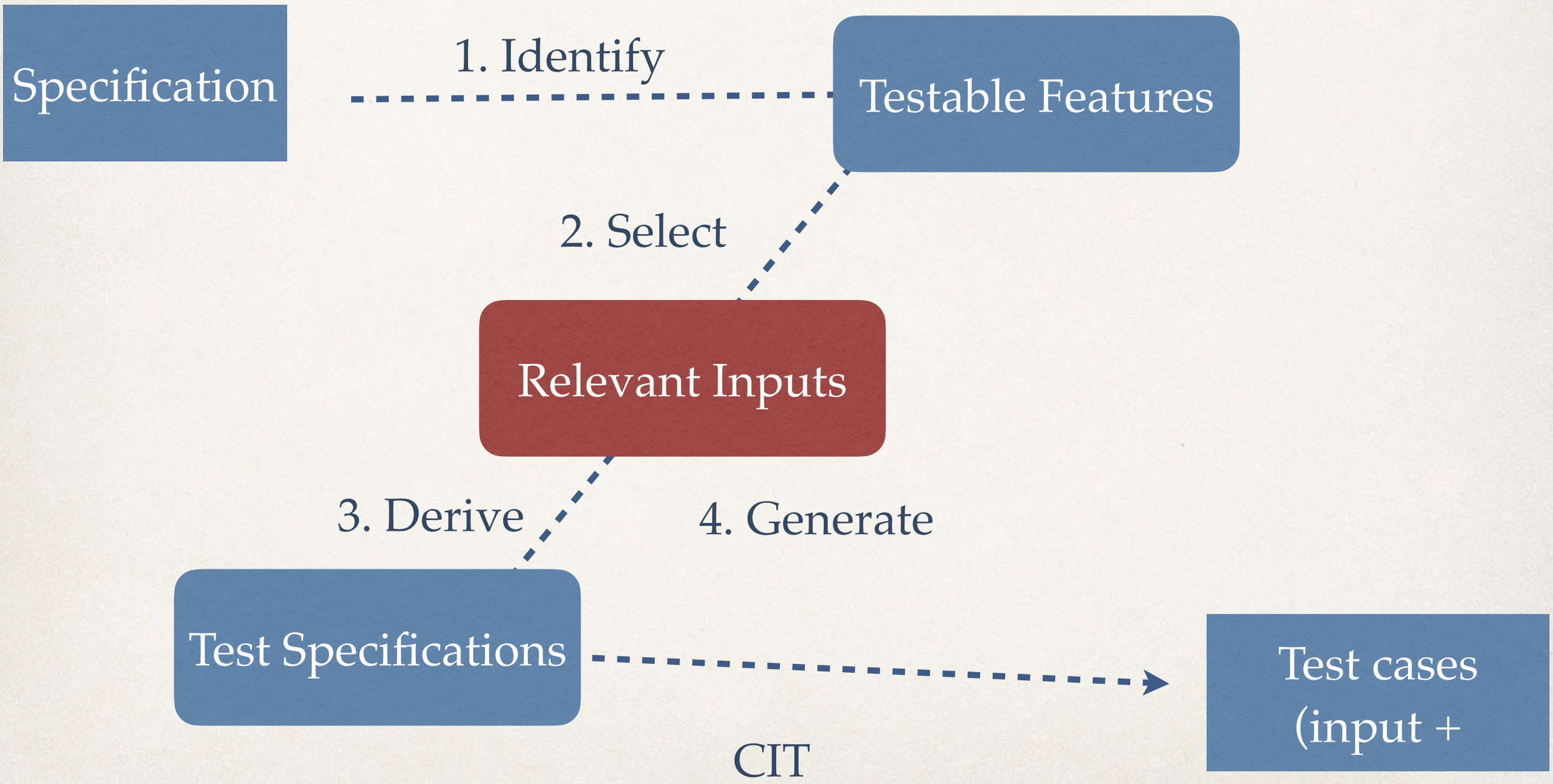
```
:
GREP(1)                    BSD General Commands Manual                    GREP(1)

NAME
     grep, egrep, fgrep, zgrep, zegrep, zfgrep -- file pattern searcher

SYNOPSIS
     grep

                [pattern] [file ...]

     The grep utility searches any given input files, selecting lines that
     match one or more patterns.
```

Recap: Category Partition - Example

UCL

Parameters and environment	Categories	Choices
pattern	filenames	files
# of Patterns:	Number:	Size:
Zero | One | One+	Zero | One | One+	Empty | Non-empty
	Exist:	Pattern occurrence:
	YES | NO	Zero | One | Many

Recap: General Process

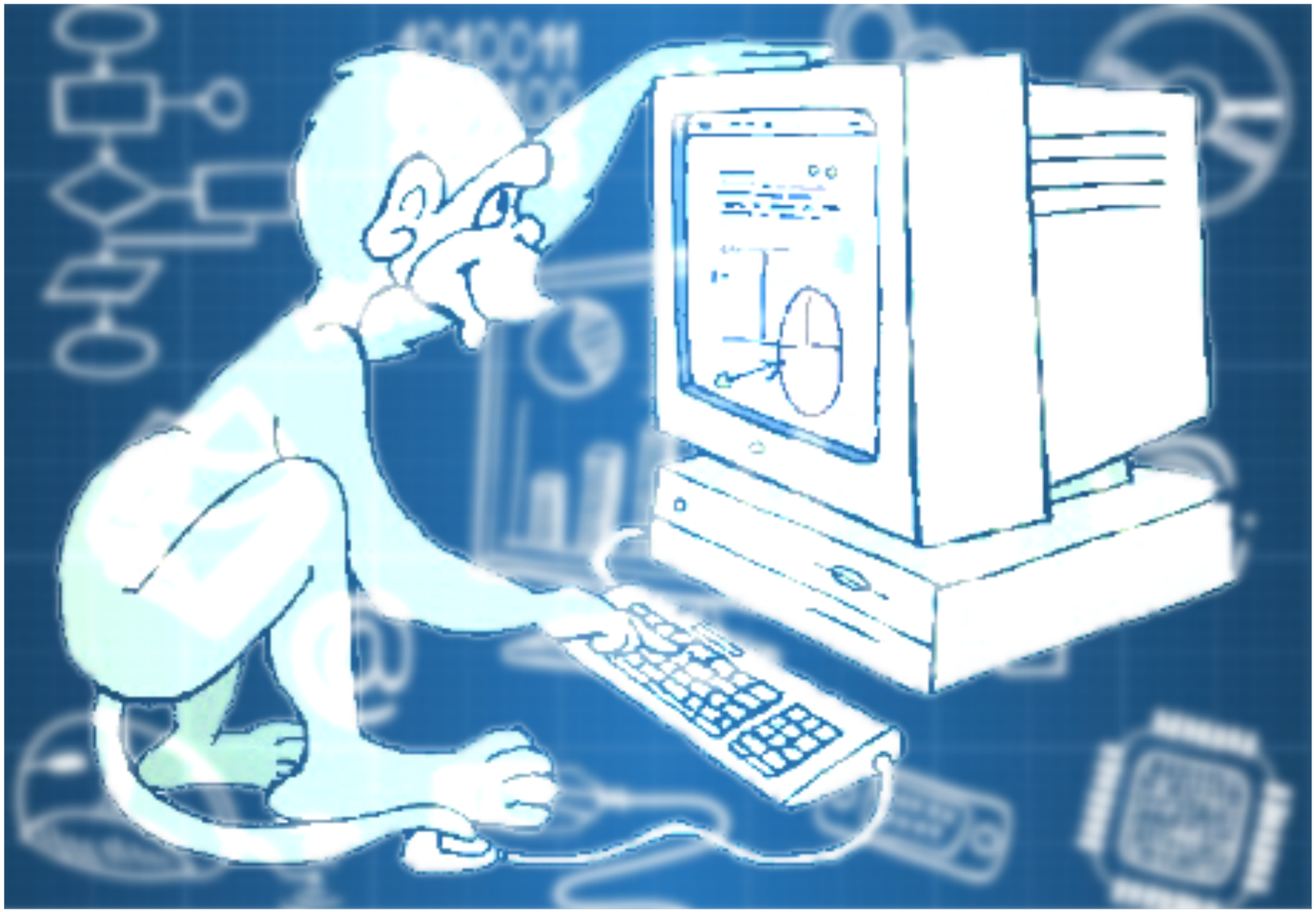


Random Testing Overview

UCL

- Choose test input randomly
- Benefits:
  - Easy to understand
  - Cheap and easy to implement
- It is actually used in the industry

Monkey Testing



f.sarro@ucl.ac.uk

Random Testing

UCL

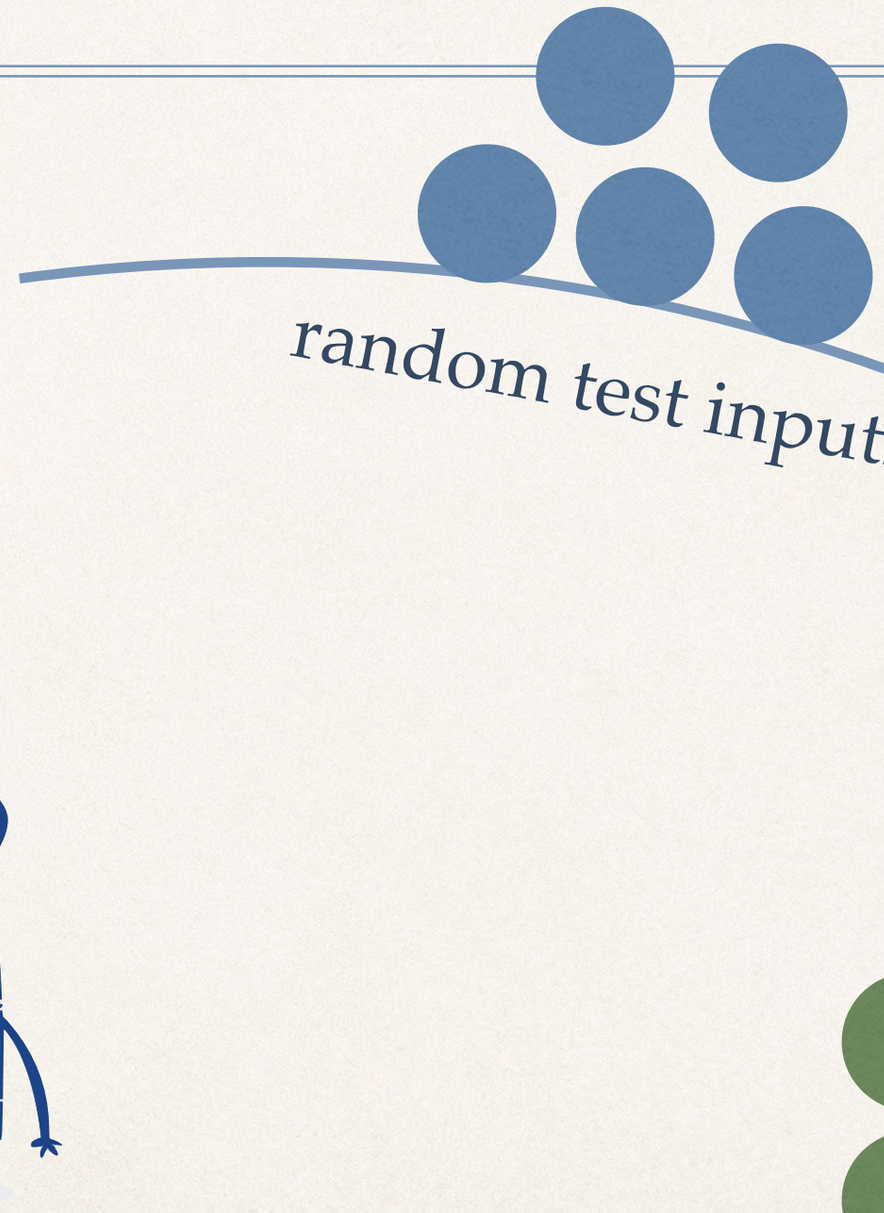
Random case generator

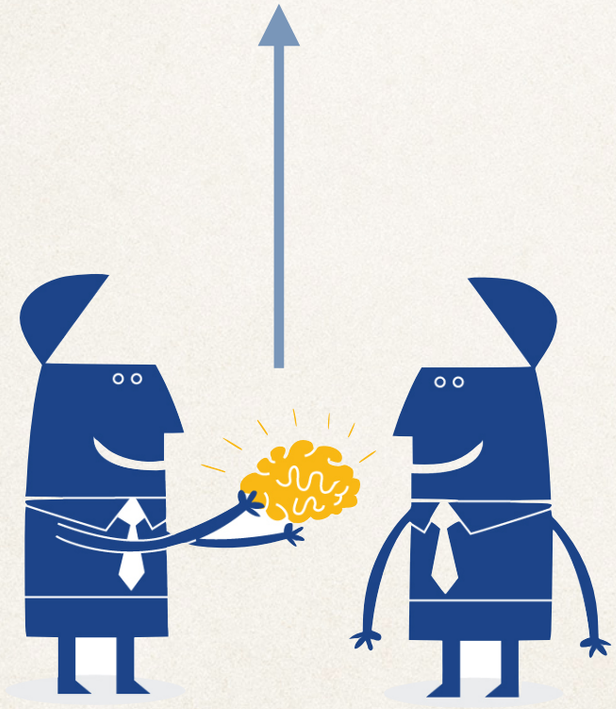
Random Testing

Random case generator

Random Testing

Random case generator





Domain Knowledge

Task: GZIP

https://www.gnu.org/software/gzip/manual/gzip.html

UCL

```
GZIP(1)                    BSD General Commands Manual                    GZIP(1)

NAME
     gzip -- compression/decompression tool using Lempel-Ziv coding (LZ77)

SYNOPSIS
     gzip [-cdfhkLlNnqrtVv] [-S suffix] file [file [...]]
     gunzip [-cfhkLNqrtVv] [-S suffix] file [file [...]]
     zcat [-fhV] file [file [...]]

DESCRIPTION
     The gzip program compresses and decompresses files using Lempel-Ziv coding (LZ77).  If no
     files are specified, gzip will compress from standard input, or decompress to standard out-
     put.  When in compression mode, each file will be replaced with another file with the suf-
     fix, set by the -S suffix option, added, if possible.
```

usage: gzip f → f.gz

gzip -d f.gz → f

Task: GZIP

UCL

1. Generate a random file

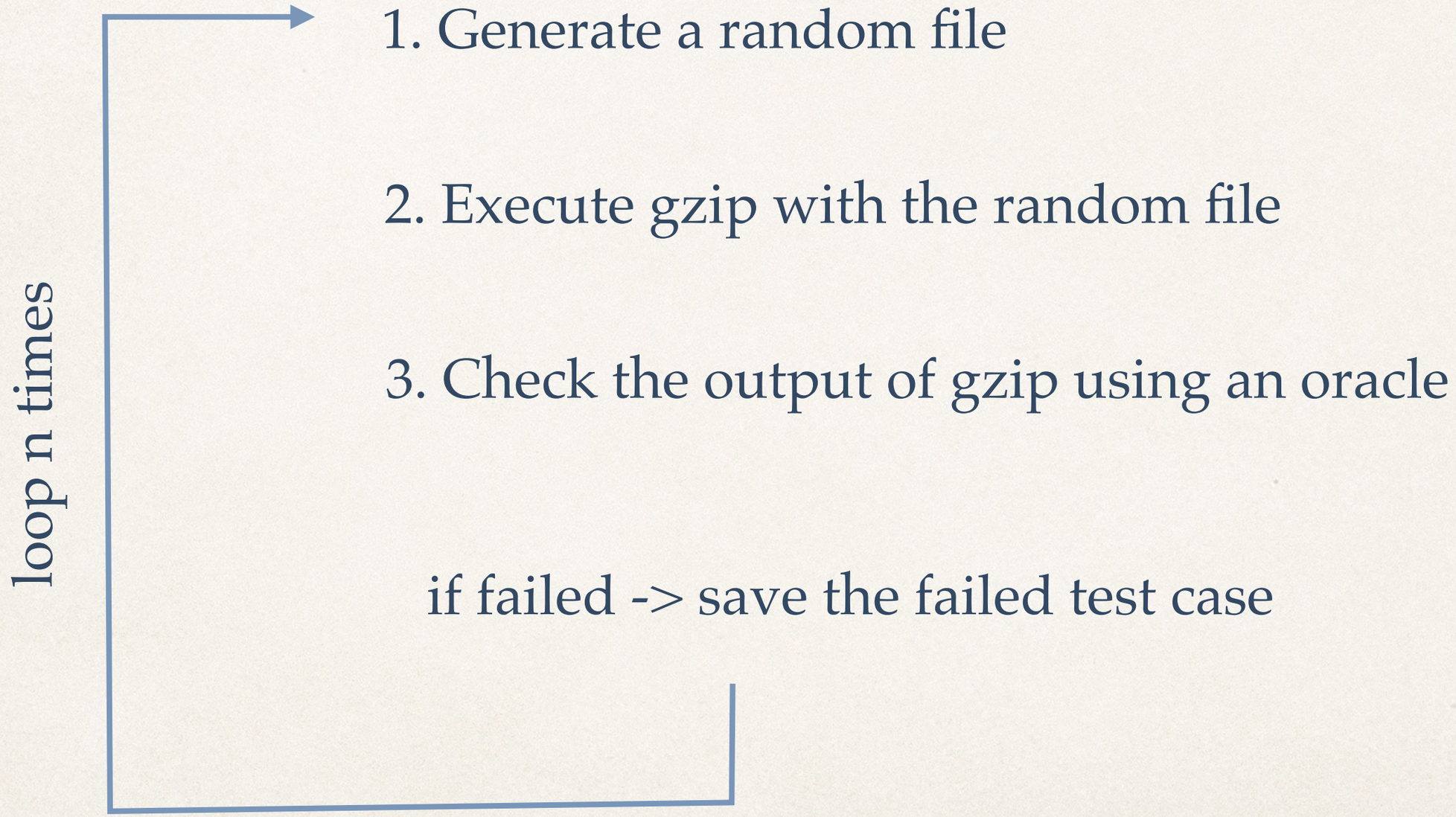
2. Execute gzip with the random file

3. Check the output of gzip using an oracle

if failed -> save the failed test case

Task: GZIP

UCL



Task: GZIP

https://en.wikipedia.org/wiki/Dd_(Unix)
https://en.wikipedia.org/wiki//dev/random
https://en.wikipedia.org/wiki/Cmp_(Unix)

UCL

1. Generate a random file

dd if=/dev/urandom of=randFile bs=512 count=2

2. Execute gzip with the random file

cp randFile originalRandFile;
gzip randFile

3. Check the output of gzip using an oracle

gzip -d randFile

if failed -> save the failed test case

```
if ! cmp --silent randFile originalRandFile;
    then  echo failed; mv originalRandFile failedTest1
    else echo pass;
fi
```

Task: GZIP

https://en.wikipedia.org/wiki/Dd_(Unix)
https://en.wikipedia.org/wiki//dev/random
https://en.wikipedia.org/wiki/Cmp_(Unix)

UCL

1. Generate a random file

   dd if=/dev/urandom of=randFile bs=512 count=2

2. Execute gzip with the random file

   cp randFile originalRandFile;

   gzip randFile

3. Check the output of gzip using an oracle

   gzip -d randFile

   if failed -> save the failed test case

   if ! cmp --silent randFile originalRandFile;

   then echo failed; mv originalRandFile failedTest1

   else echo pass;

   fi

Can we improve this random testing script?

Task: GZIP

https://en.wikipedia.org/wiki/Dd_(Unix)
https://en.wikipedia.org/wiki//dev/random
https://en.wikipedia.org/wiki/Cmp_(Unix)

UCL

1. Generate a random file

   dd if=/dev/urandom of=randFile ***bs=512 count=2***

2. Execute gzip with the random file

   cp randFile originalRandFile;
   gzip randFile

3. Check the output of gzip using an oracle

   gzip -d randFile **Check zipped file size <= original**

   if failed -> save the failed test case

   if ! cmp --silent randFile originalRandFile;
   then echo failed; mv originalRandFile failedTest1
   else echo pass;
   fi

Can we improve this random testing script?

Challenges

- Randomness is expensive
- Less effective when failure rate is low
- More time and resources
- Fully automated oracle

How Random Can We Get?

- True randomness is possible but expensive
- In general, computers use pseudo-random number generators (PRNGs), which generates a long sequence of bits approximating the properties of random bits (the sequence is controlled by a small initial value called *seed*)

java.util.Random

UCL

```
public int nextInt()
{
   return next(32);
}

synchronized protected int next(int bits)
{
   seed = (seed * 0x5DEECE66DL + 0xBL) & ((1L << 48) - 1);
   return (int)(seed >>> (48 - bits));
}
```

Linear Congruential Formula

The period of Java's implementation is 2^{48}

- The pseudo-random bits will repeat themselves, but only at very long intervals - okay for testing purposes

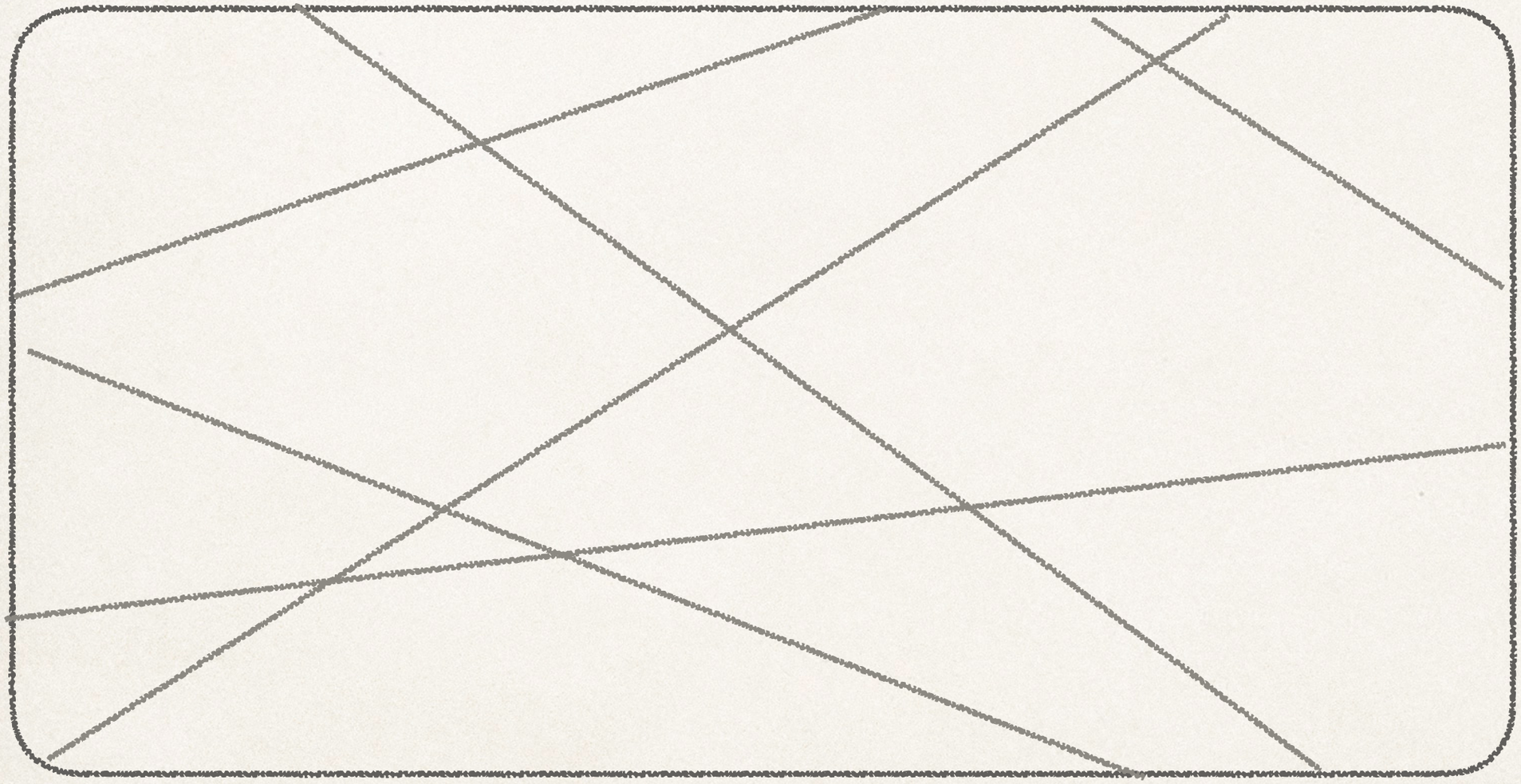
Challenges

UCL

- Randomness is expensive
- Less effective when failure rate is low
- More time and resources
- Fully automated oracle

Failure Rate

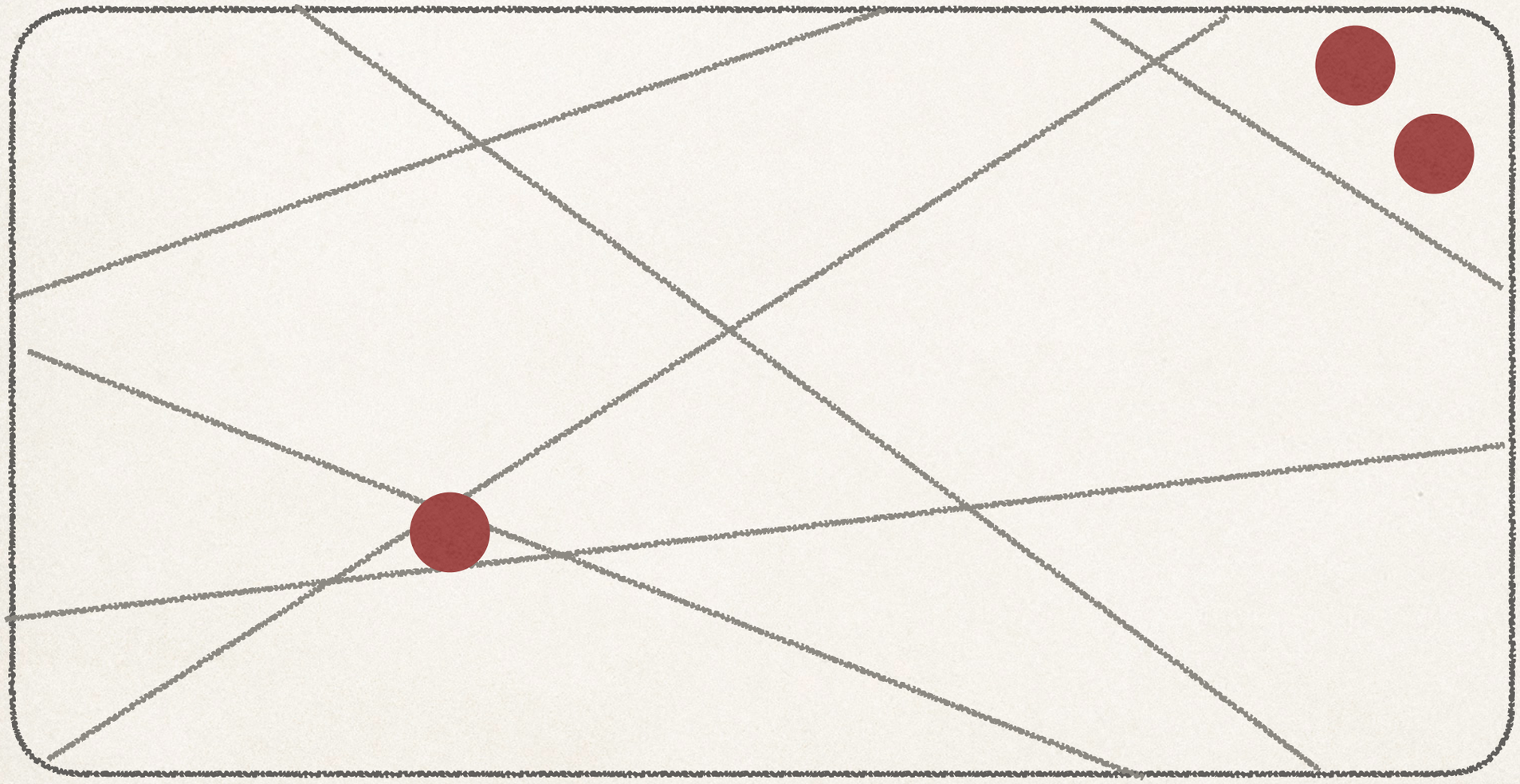
UCL



Input Domain

Failure Rate

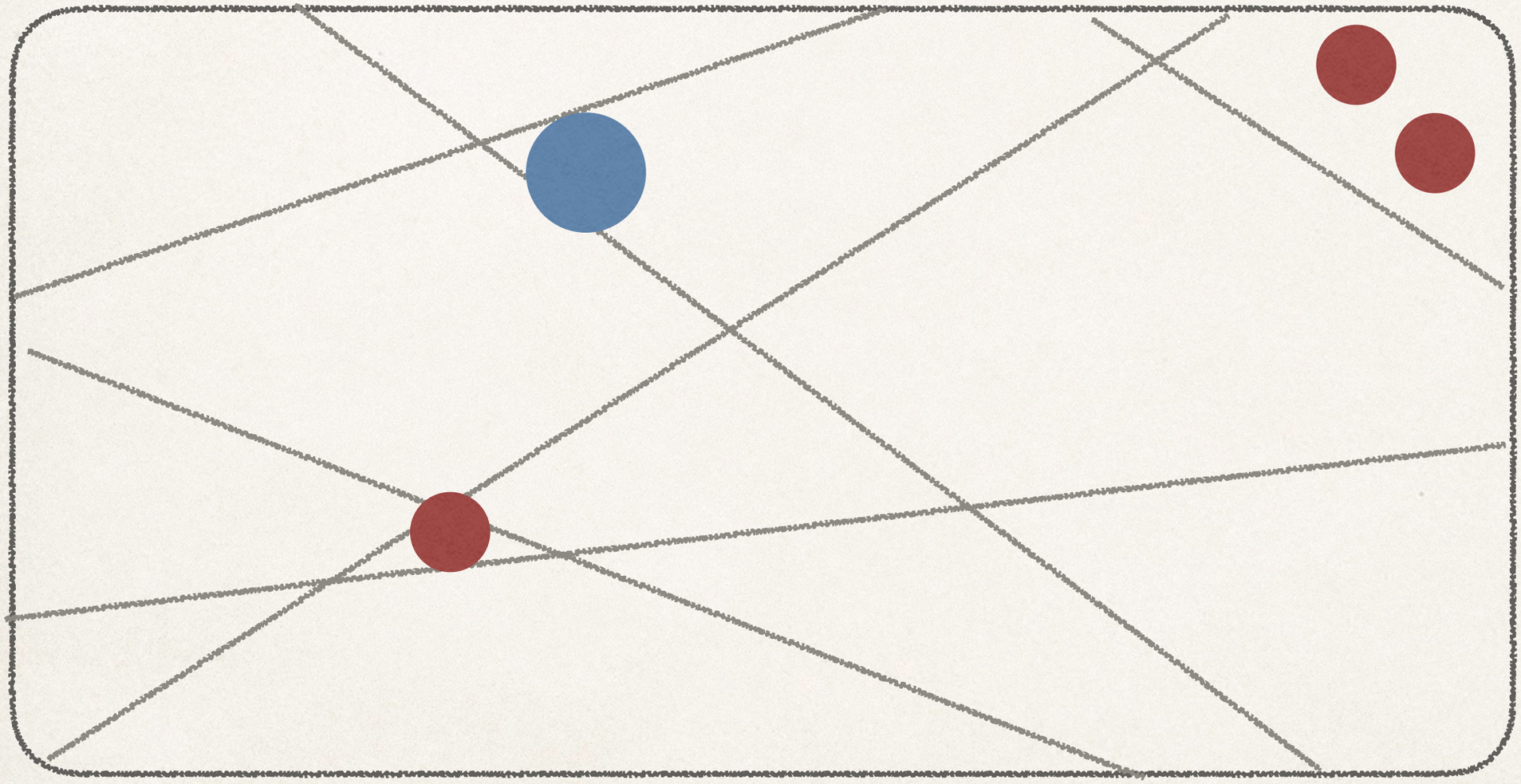
UCL



Input Domain

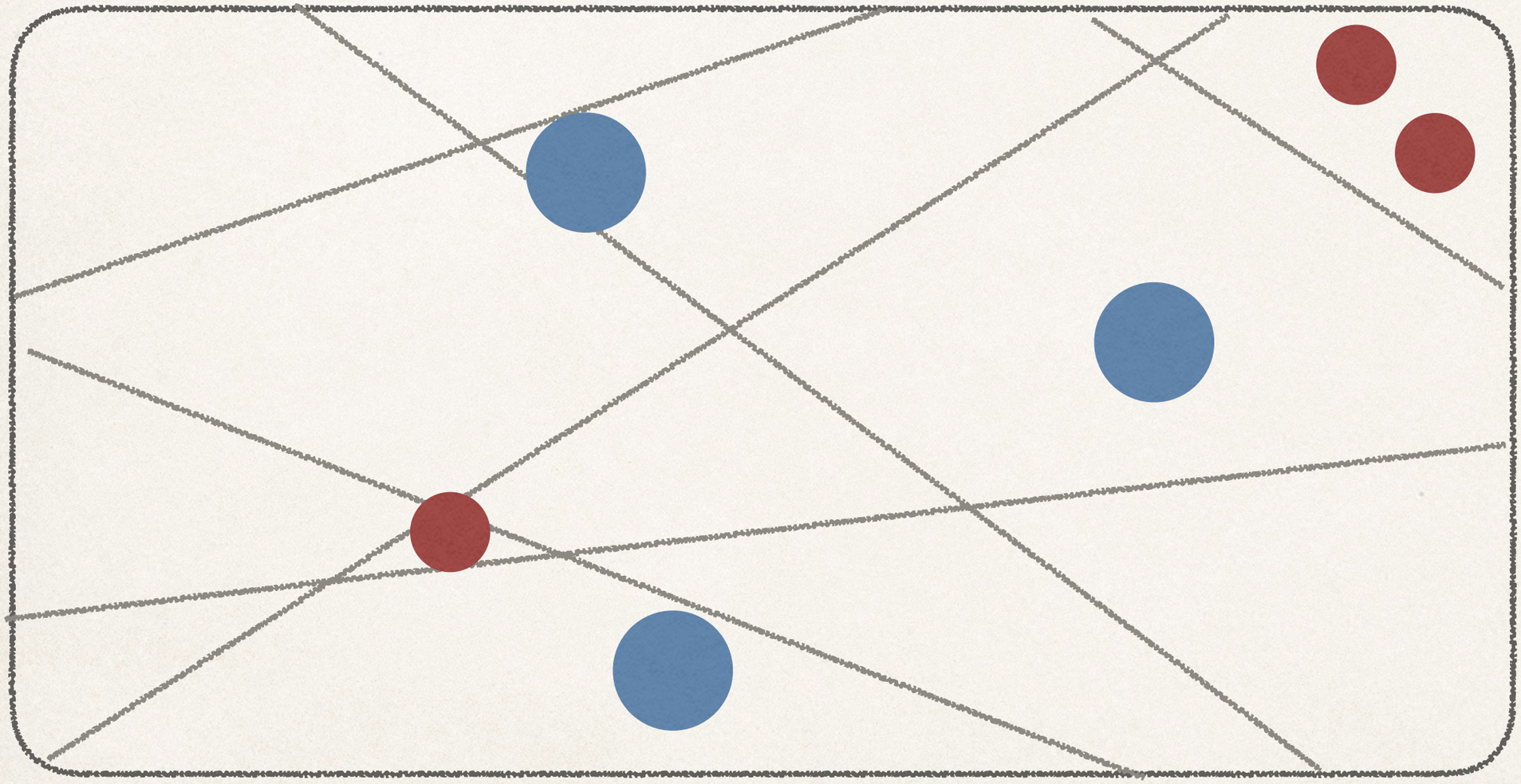
Failure Rate

UCL



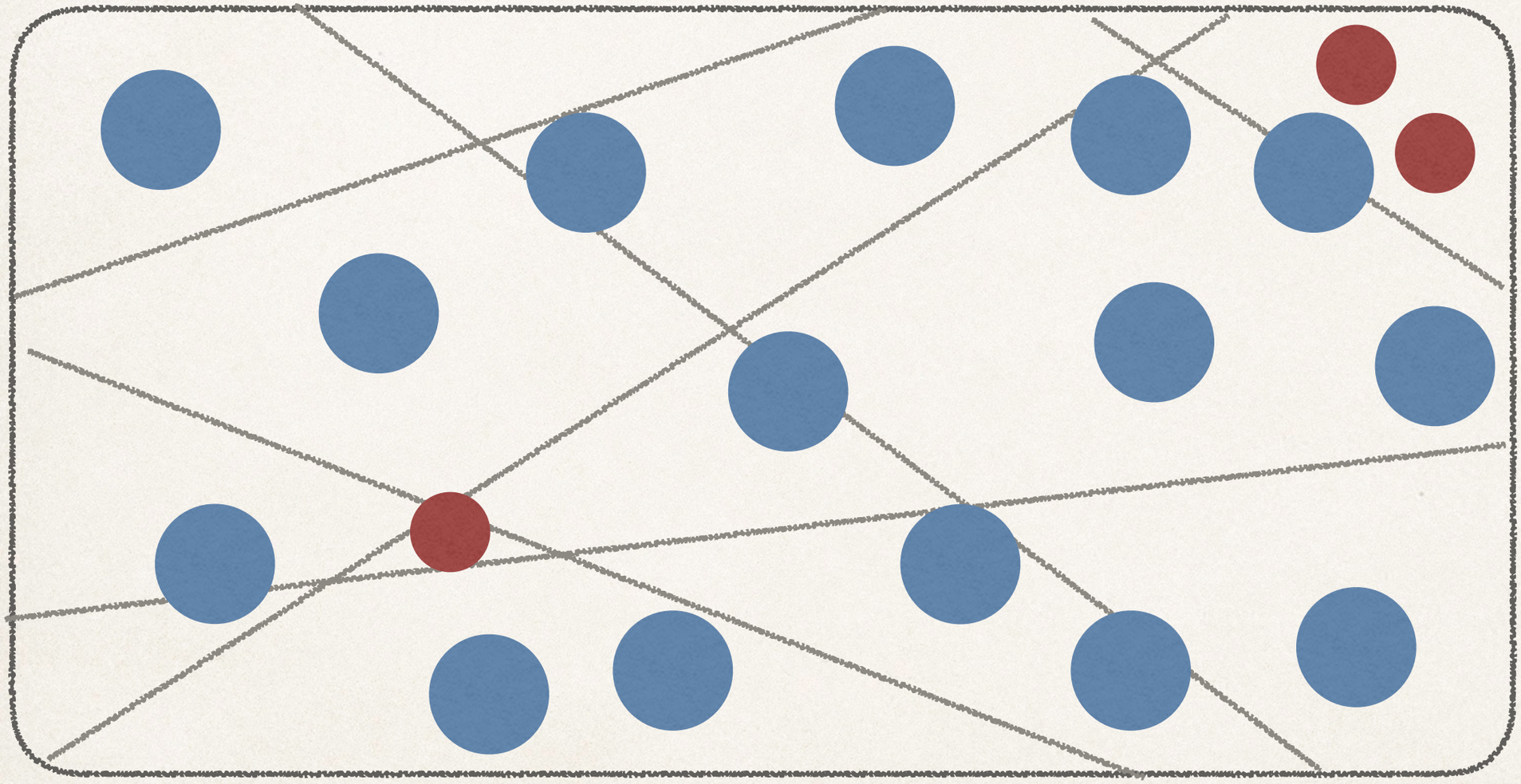
Failure Rate

UCL



Failure Rate

UCL



Input Domain

Some Definitions

- SUT: Software Under Test
- S: set of all possible test inputs for SUT
- |S|: cardinality of S
- F: subset of S - a set of all failing test inputs (not known in advance, of course)
- Failure Rate t = |F| / |S|: the probability that a random test input will fail, if sampled uniformly

What Is the Failure Rate?


```
int abs(int x)
{
  if(x>0) return x;
  else return x;
}
```

```
void foo(int x){
if(x==0){
  y = 3 / x;
}
```

- Failure Rate t = |F| / |S| : the probability that a random test input will fail, if sampled uniformly

What Is the Failure Rate?

UCL

```
int abs(int x)
{
   if(x>0) return x;
   else return x; // should be -x
}
```

What Is the Failure Rate?

UCL

- Failure Rate t ≈ 0.5
- Oracle
  - assertEqual(abs(-5), 5)
  - assertEqual(abs(2), 2)

```
int abs(int x)
{
   if(x>0) return x;
   else return x; // should be -x
}
```

What Is the Failure Rate?

UCL

```
void foo(int x){
 if(x==0){
   y = 3 / x;/* faulty code here */
  }
}
```

What Is the Failure Rate?

- Random testing is particularly weak against the needle in the haystack problem

```
void foo(int x){
 if(x==0){
    y = 3 / x;/* faulty code here */
  }
}
```

- We need, on average, over 4 billion test inputs before triggering this fault

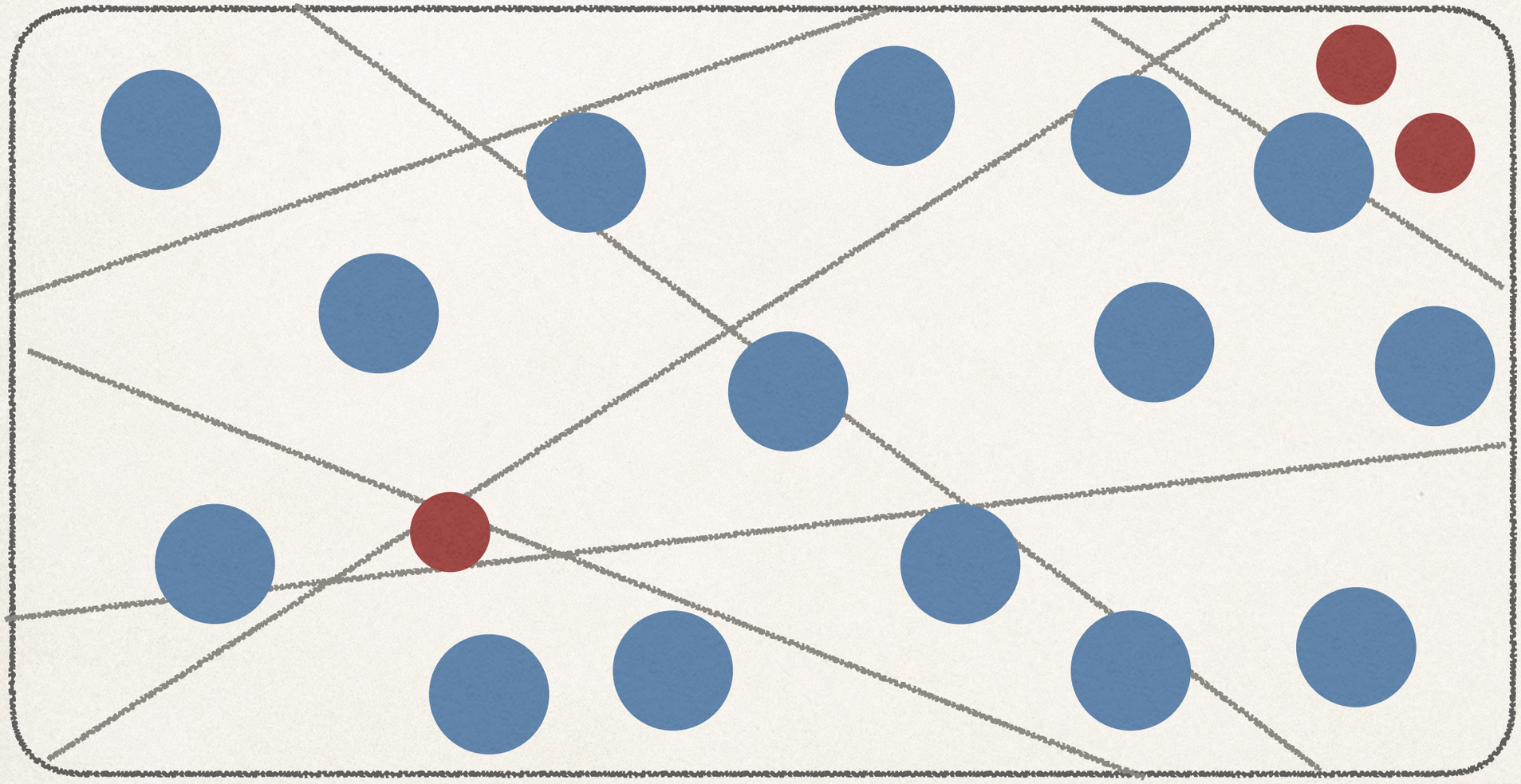
Challenges

UCL

- Randomness is expensive
- Less effective when failure rate is low
- More time and resources
- Fully automated oracle

Failure Rate

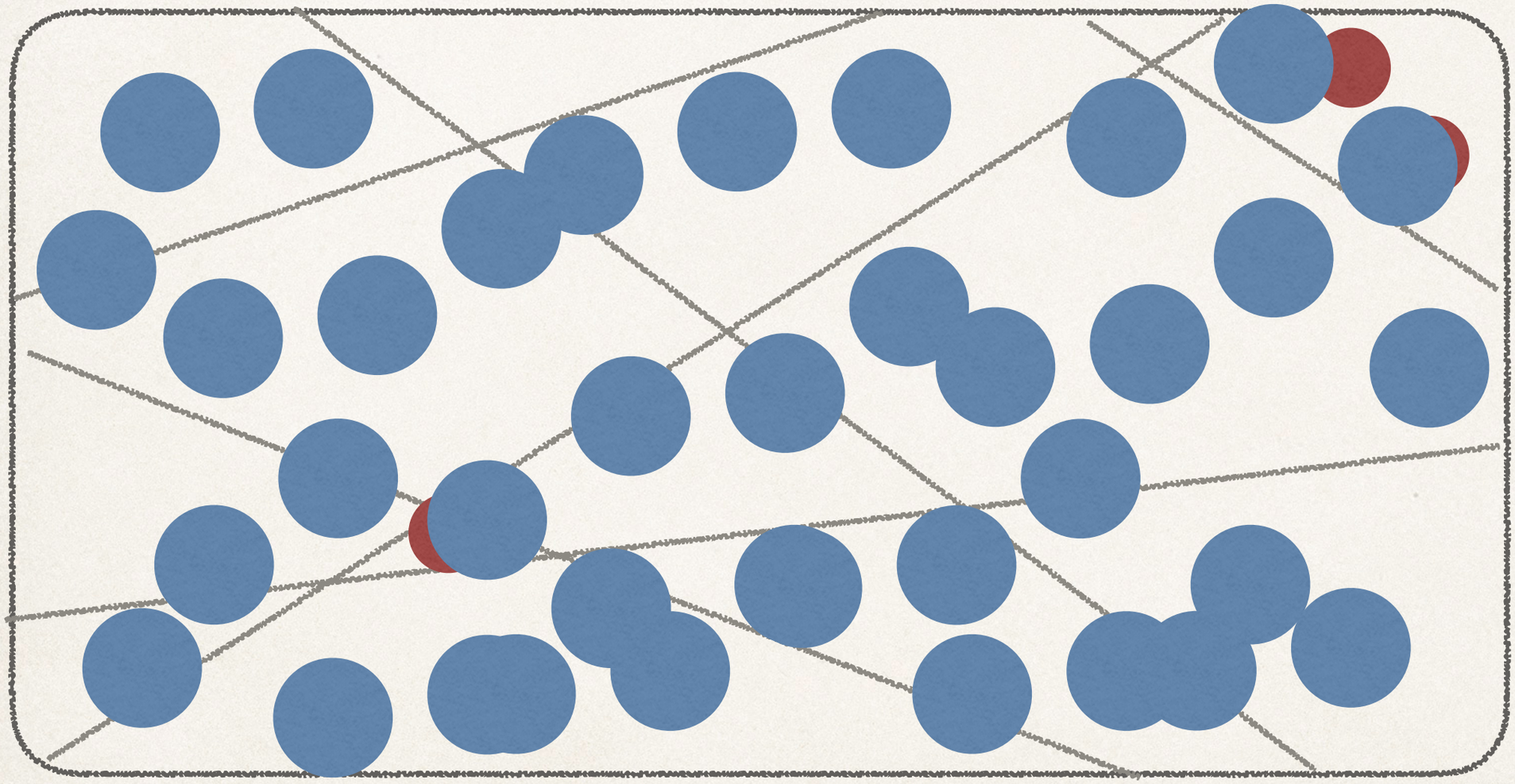
UCL



Input Domain

Failure Rate

UCL



Input Domain

Challenges

UCL

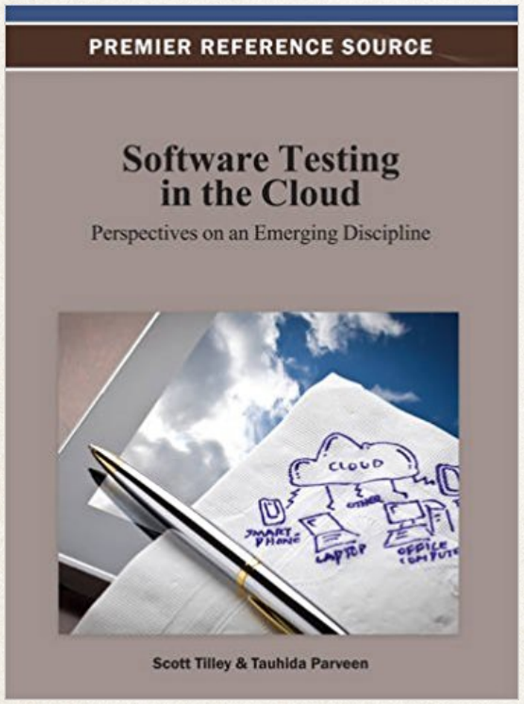
- Randomness is expensive
- Less effective when failure rate is low
- More time and resources
- Fully automated oracle

Challenges: More Time and Resources

UCL

YETI on the Cloud http://ieeexplore.ieee.org/document/5463683/

The York Extensible Testing Infrastructure (YETI) is an automated random testing tool that allows to test programs written in various programming languages



Software Testing in the Cloud: Perspectives on an Emerging Discipline, ICI Global 2013

Challenges

UCL

- Randomness is expensive
- Less effective when failure rate is low
- More time and resources
- Fully automated oracle

Strong Oracles vs. Weak Oracles

- Strong oracles: check behaviour of the SUT
- Expected outputs according to specification
- Checking against other implementations of the SUT
- Checking inverting functions

Strong Oracles vs. Weak Oracles

UCL

- Weak oracles: check general properties
  - Crashes
  - Memory issues
  - Exceptions
  - Execution time

Challenges

UCL

- Randomness is expensive
- **Less effective when failure rate is low**
- More time and resources
- Fully automated oracle

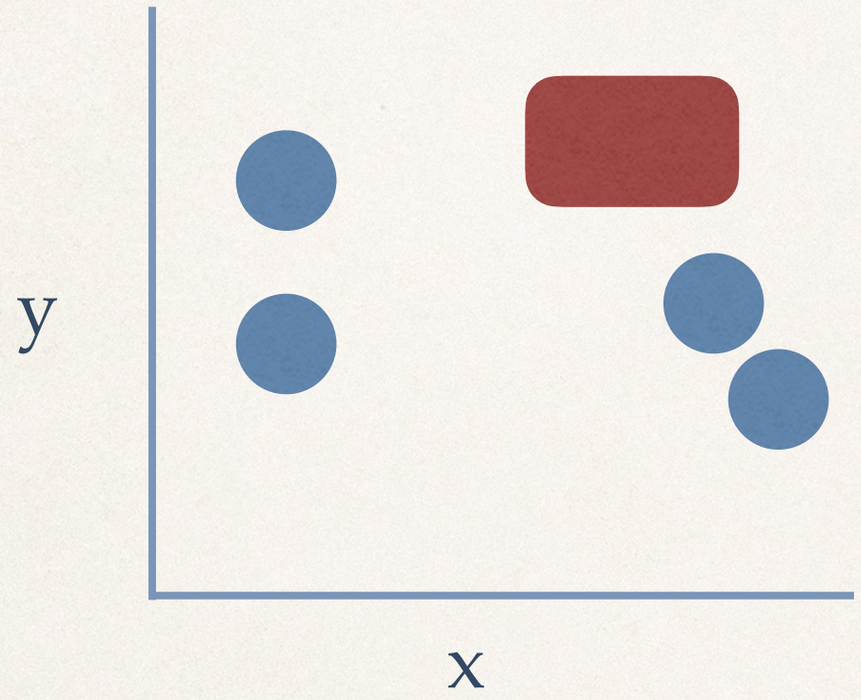
Adaptive Random Testing

UCL

- Often, failing test inputs cluster together
  - What if the fault is under the predicate if(x>=0 && x<=10)
- Let us call these faulty regions in the input space
- Without knowing where they actually are, what is our best strategy?

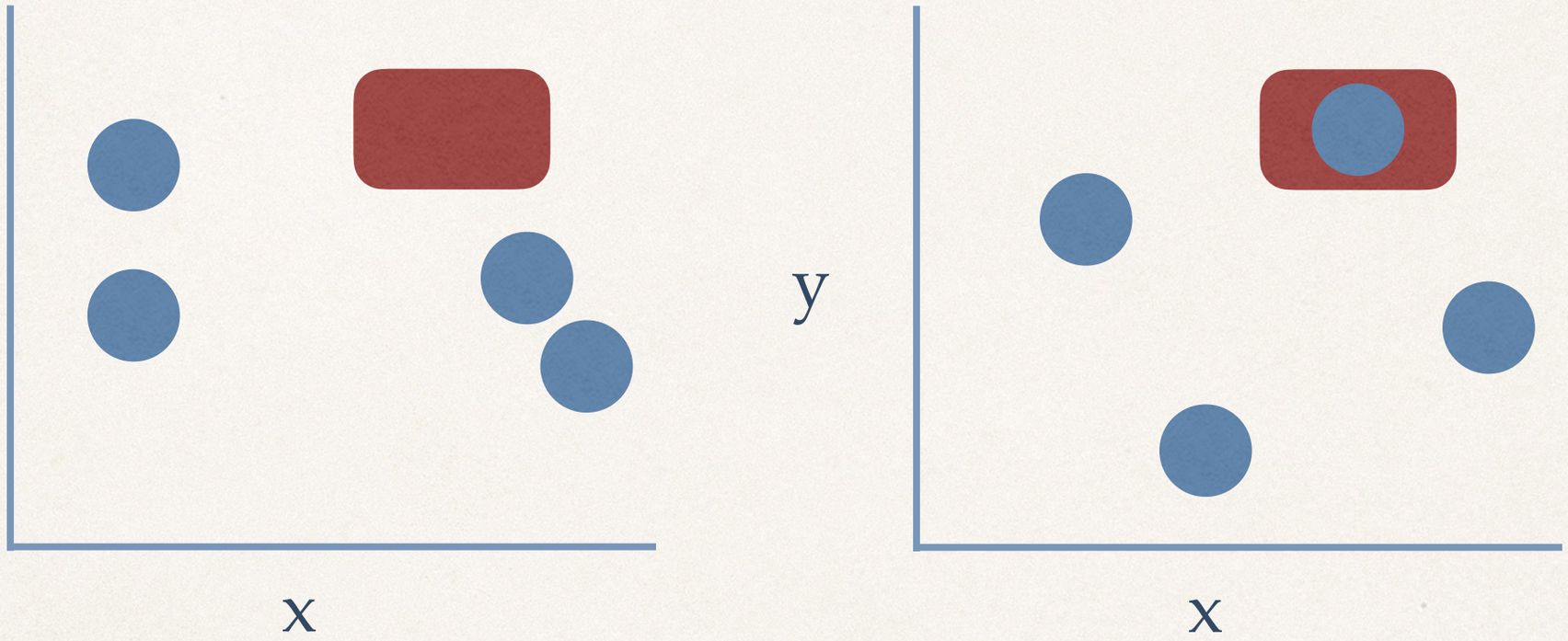
Diversity

UCL



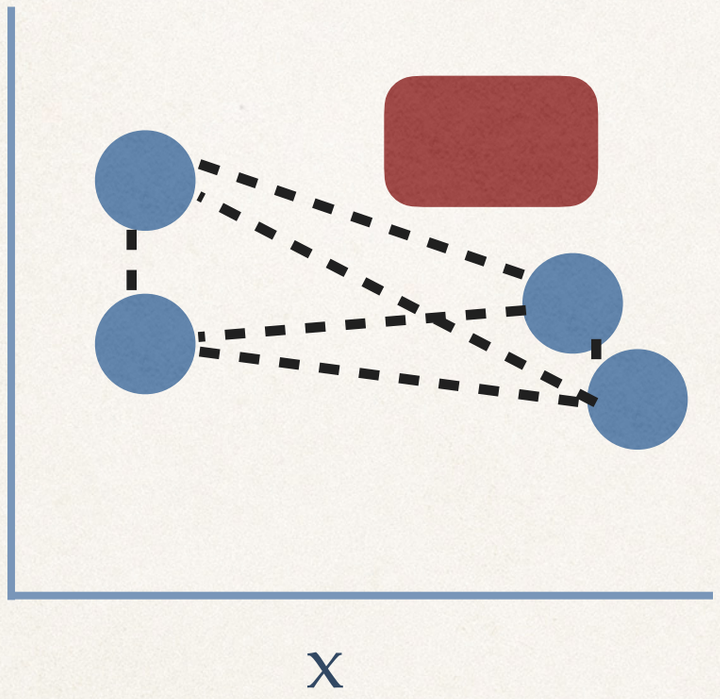
Diversity

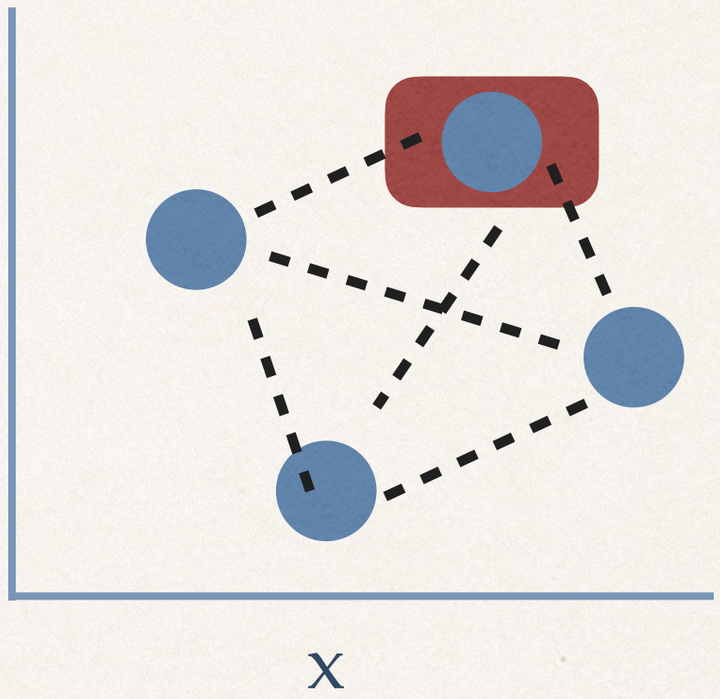
UCL



A more diverse set of inputs will have a higher probability of hitting the faulty region

Diversity





A more diverse set of inputs will have a higher probability of hitting the faulty region

Adaptive Random Testing

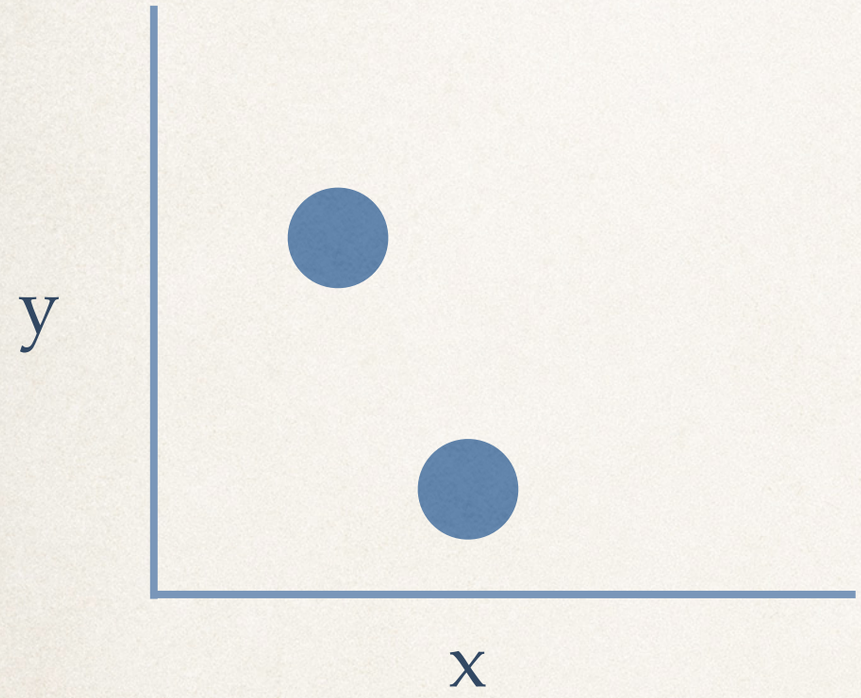
http://www.utdallas.edu/~lxz144130/cs6301-readings/art.pdf

UCL

- Still choose test inputs randomly, but only one at a time, into the set S of executed test cases (initially empty)
- Whenever adding a new input:
  - Sample z inputs randomly and add to Z (set of candidates)
  - Choose z from the set Z so that it is the most diverse against (i.e. *the farthest away from*) the current chosen set S
  - Add z into S

Adaptive Random Testing

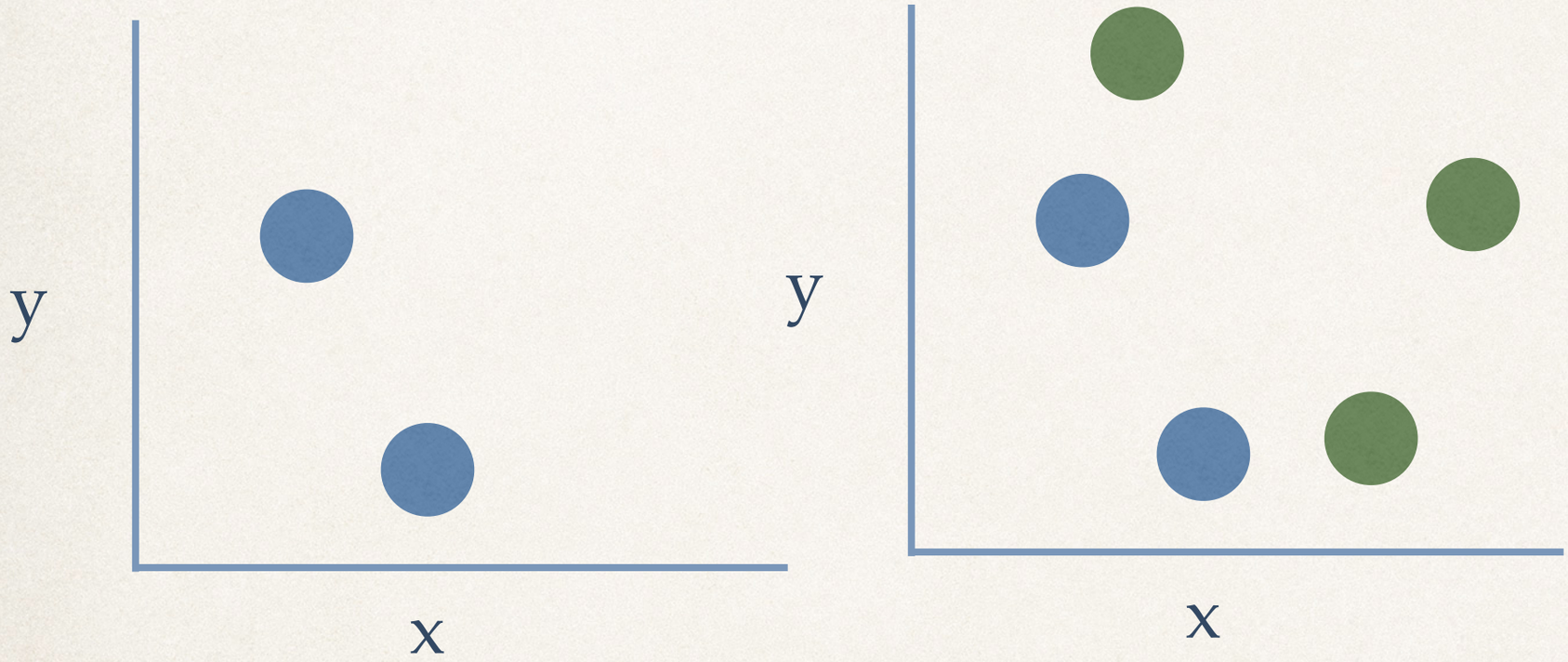
UCL



Executed

Adaptive Random Testing

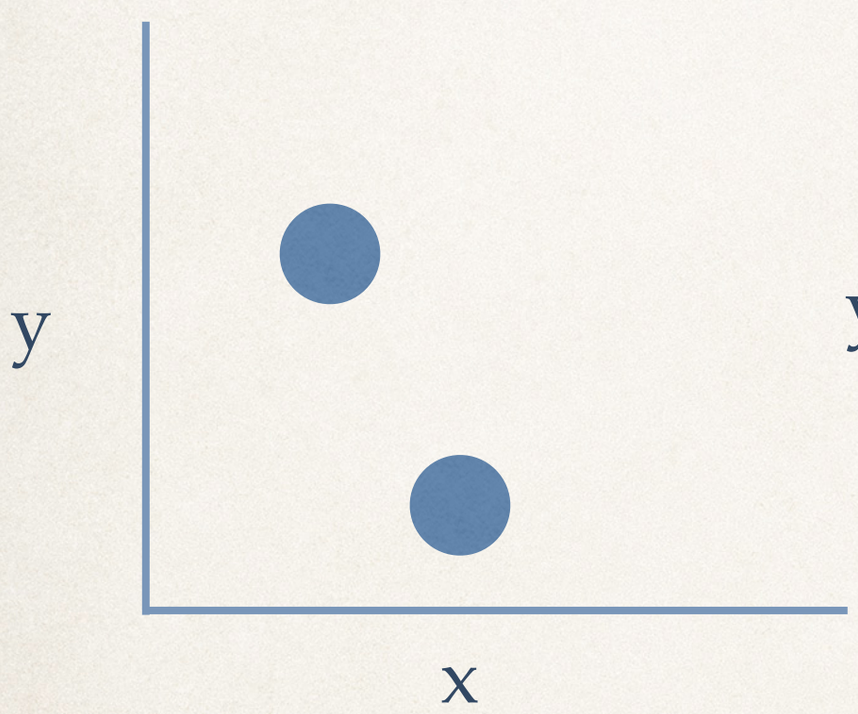
UCL

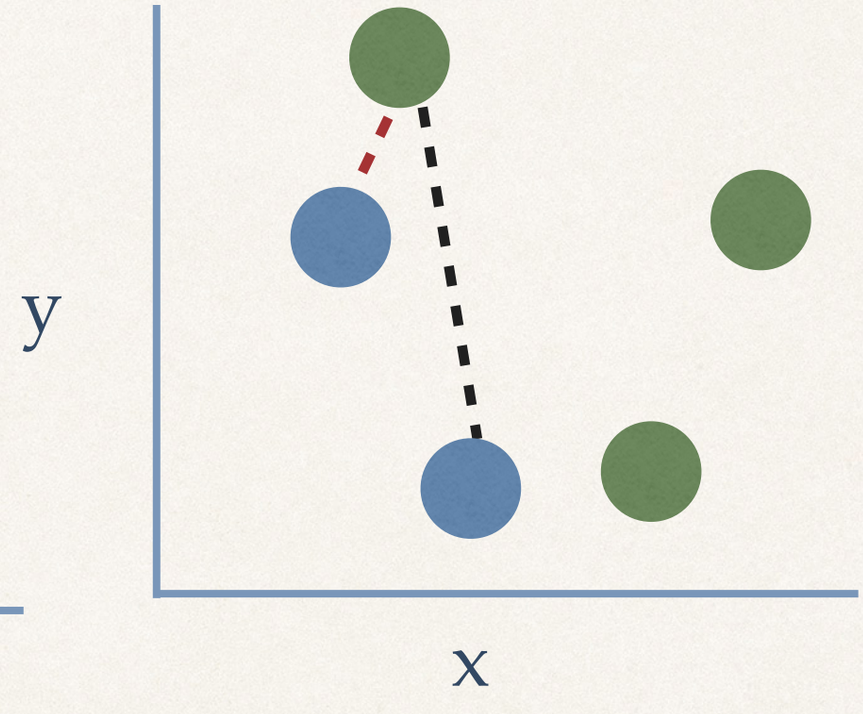


Executed Candidates

Adaptive Random Testing

UCL



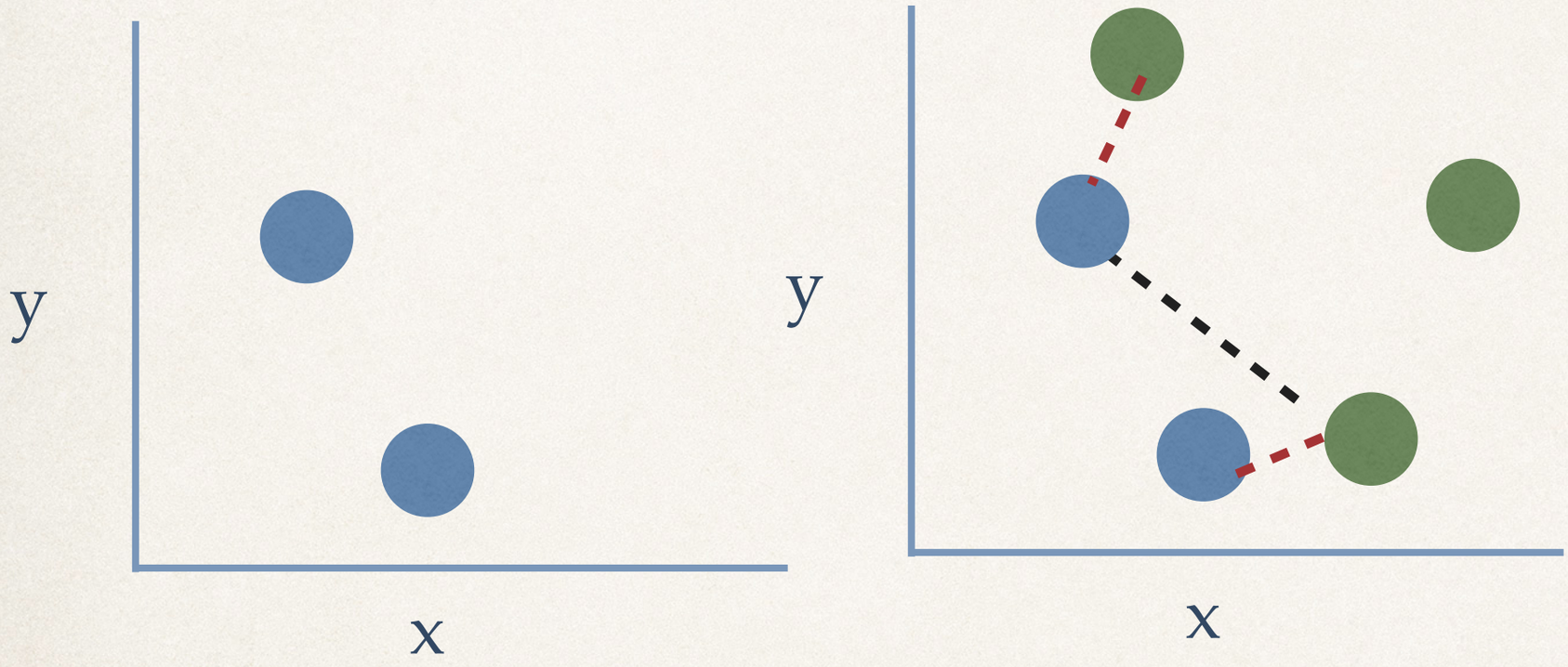


for each z, find minimum distance between z to S

Executed Candidates

Adaptive Random Testing

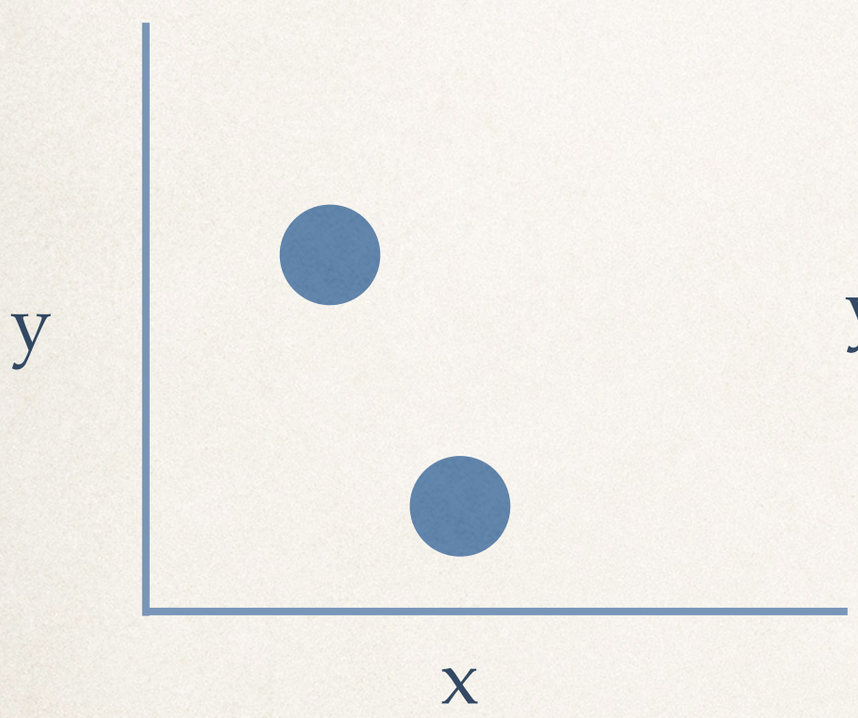
UCL

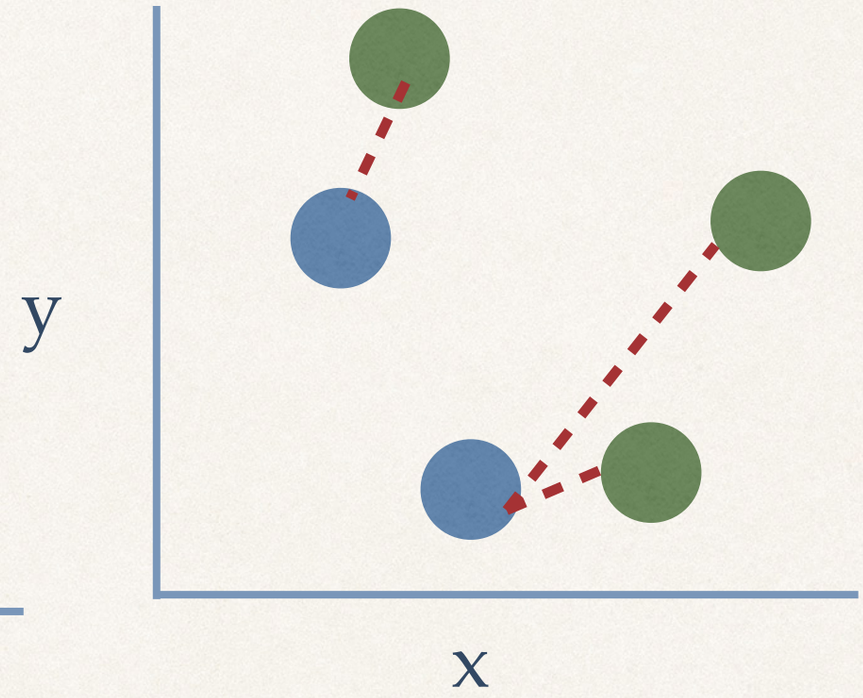


for each z, find minimum distance between z to S

Adaptive Random Testing

UCL

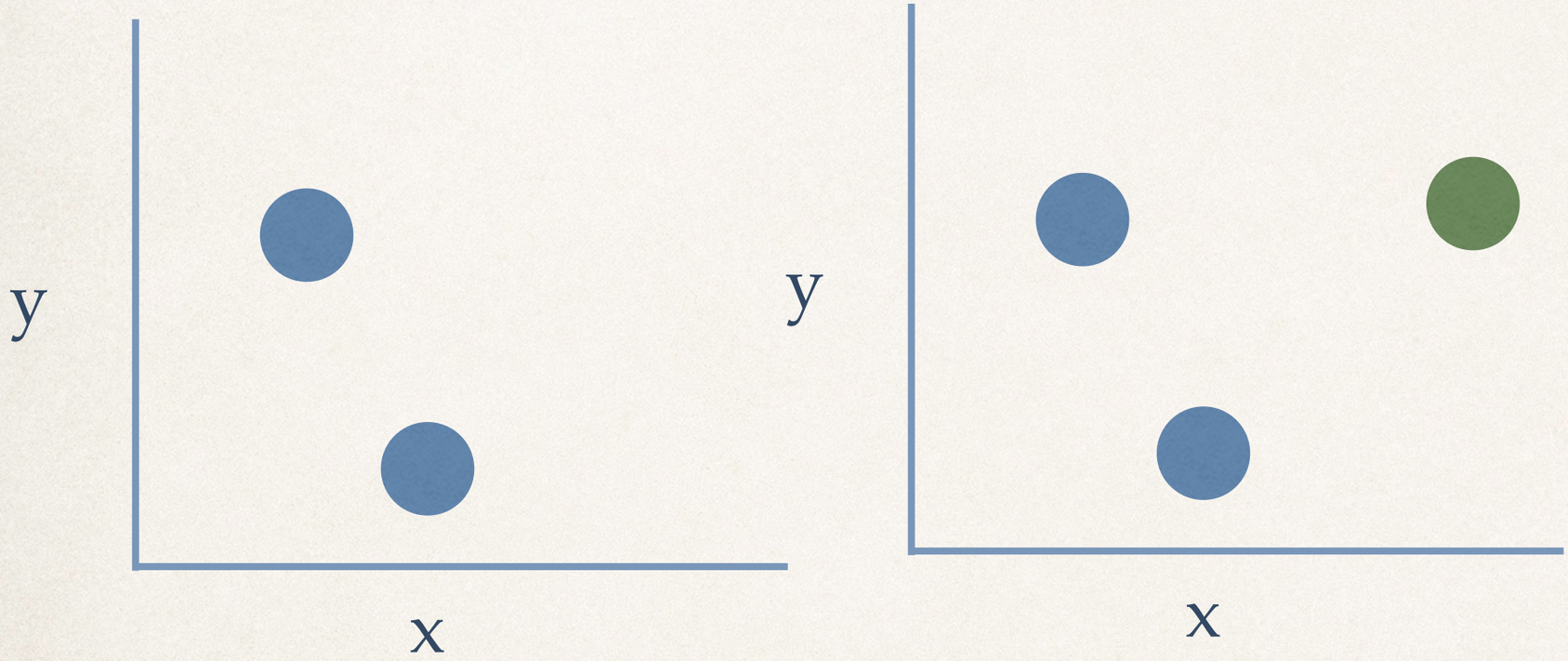




Choose z having the maximum
`selected distance' to S

Adaptive Random Testing

UCL



Adaptive random testing seeks to distribute test cases more evenly within the input space

How Can We Measure Distance?

UCL

- Diversity depends on the distance between test inputs
- If input data is numerical, we can use the Euclidean distance
- But how do we measure distance between complex data types?

Hamming Distance (Edit Distance)

UCL

- Between two strings, Hamming distance is the number of edits that are required to change one string into the other

- s1 = (A,B,C,A,A), s2 = (A,C,C,A,B) : Hamming distance is 2

Object Distance

http://se.inf.ethz.ch/old/people/ciupa/papers/icse08.pdf

UCL

- Elementary distance: a measure of the difference between the direct values of the objects

- Type distance: a measure of the difference between the objects' types

- Field distance: a measure of the difference between the objects' individual fields

$$
\begin{aligned}
&p \leftrightarrow q = combination\ (\\
&\quad elementary\_distance\ (p,q),\\
&\quad type\_distance\ (type\ (p), type\ (q)),\\
&\quad field\_distance\ (\{[p.a \leftrightarrow q.a]\\
&\qquad\quad |a \in Attributes\ (type\ (p), type\ (q))\}))
\end{aligned}
\tag{1}
$$

Adaptive Random Testing: Pros, Cons, and Questions

UCL

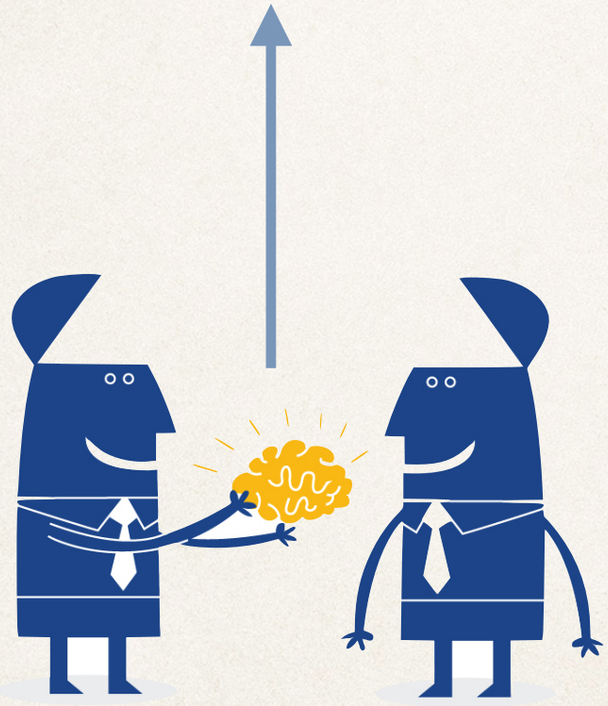
- Distribute test cases more evenly within the input space
- Still very easy to implement
- It may be difficult to choose the right/meaningful distance metric for your input type
- Faulty regions may not apply to all types of faults

Random Testing

Random case generator

SUT

Oracle



Domain Knowledge

Random Files

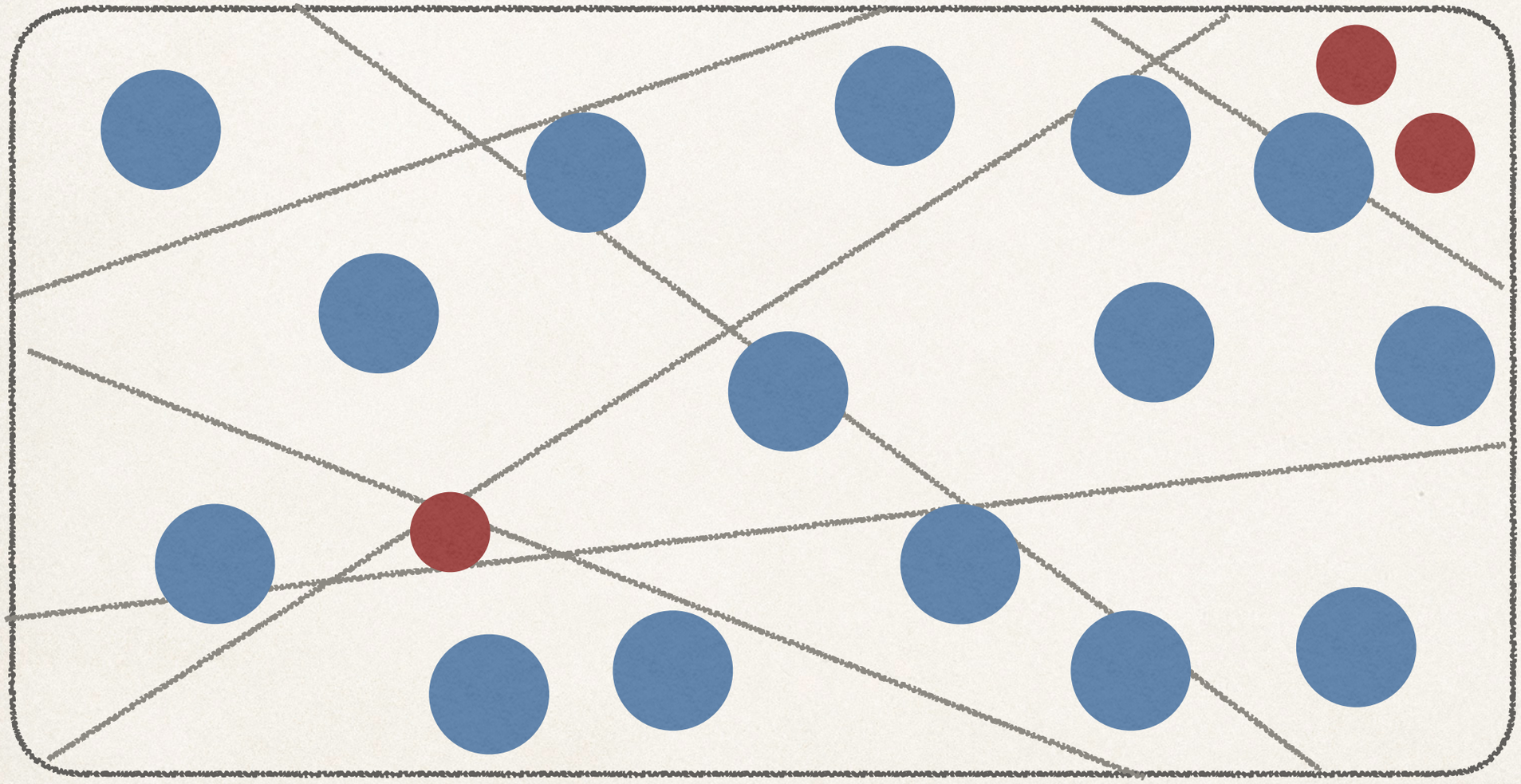
GZIP

Random Files

Web Browsers

Failure Rate

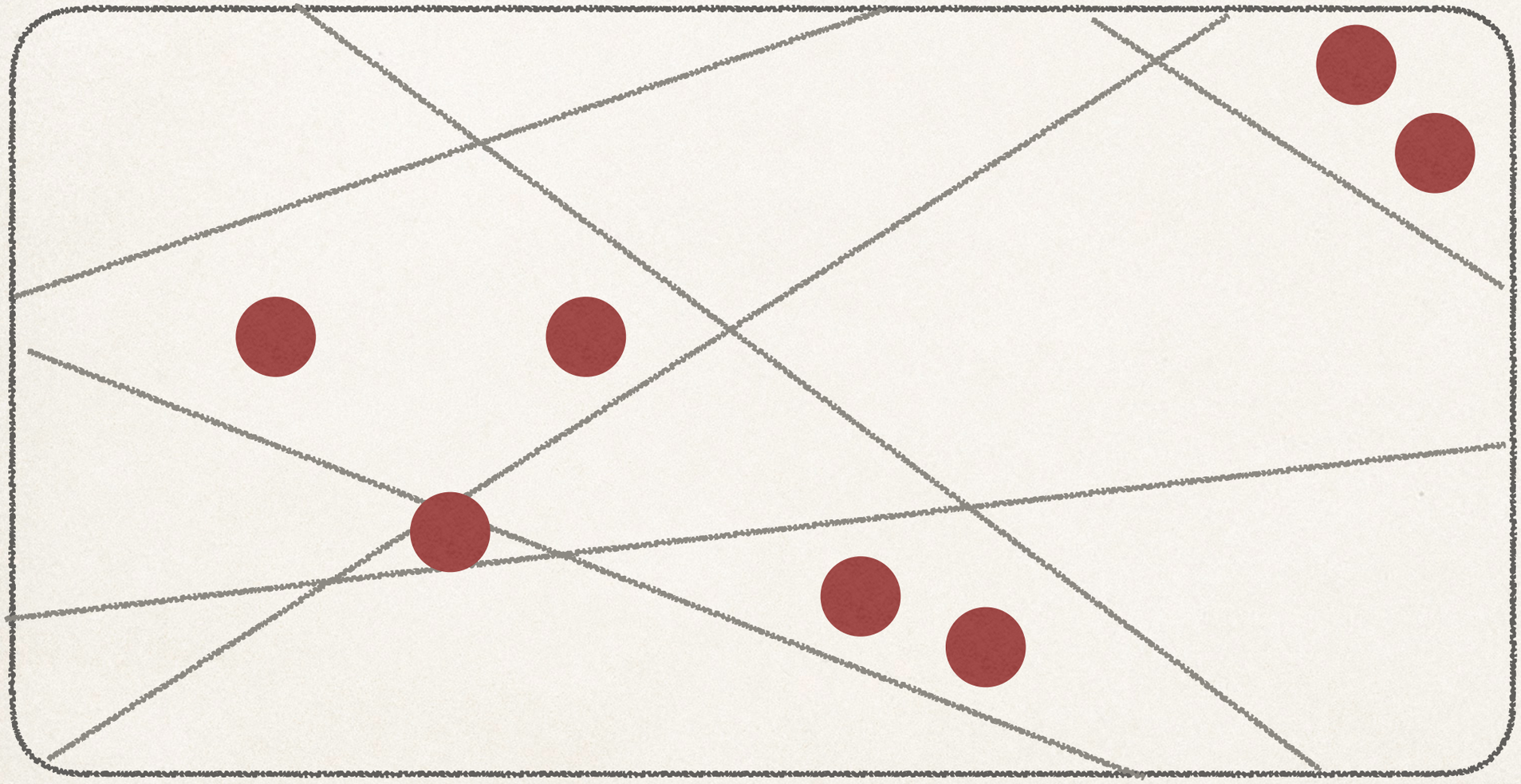
UCL



Input Domain

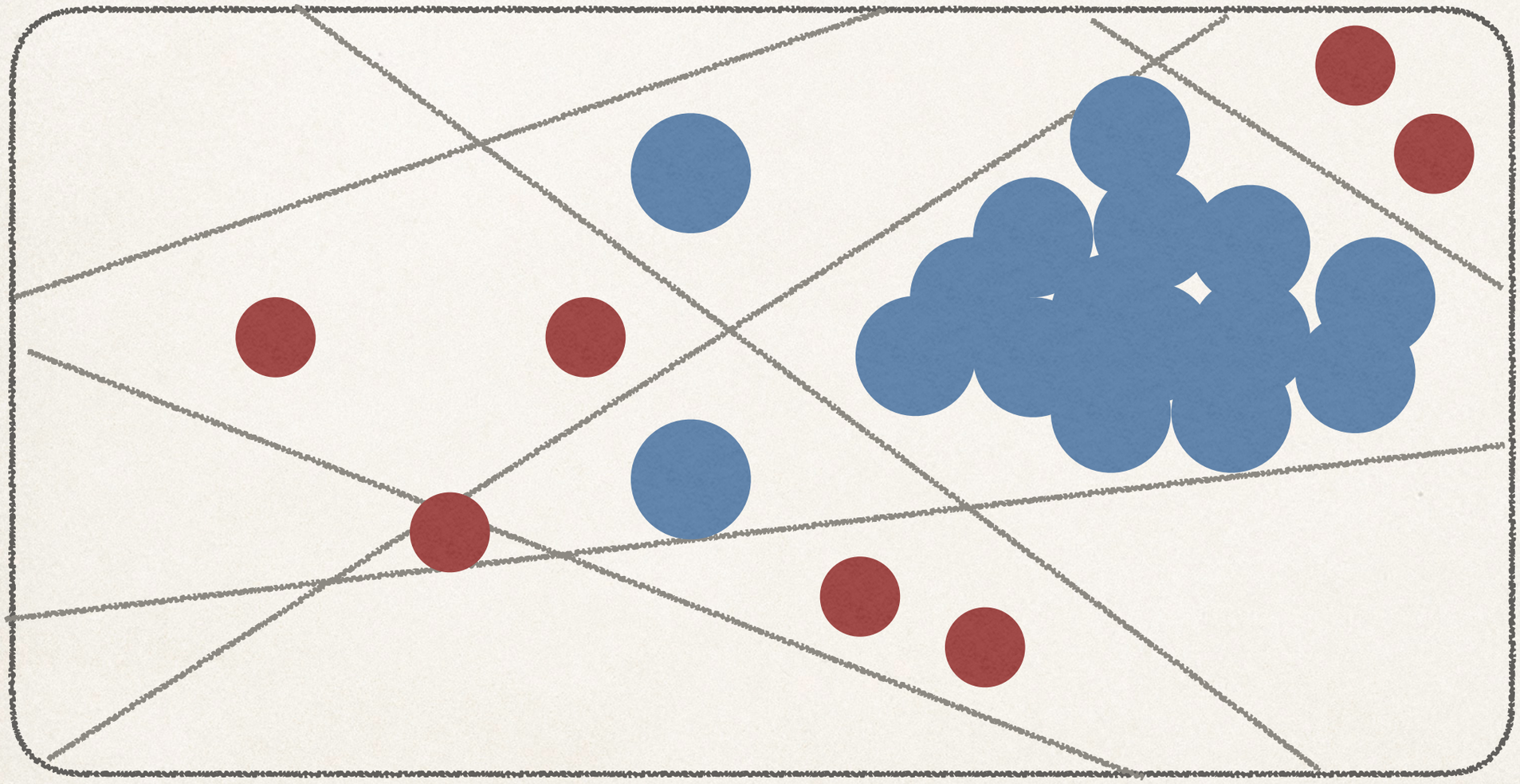
Failure Rate

UCL



Input Domain

Failure Rate



Input Domain

Invalid inputs

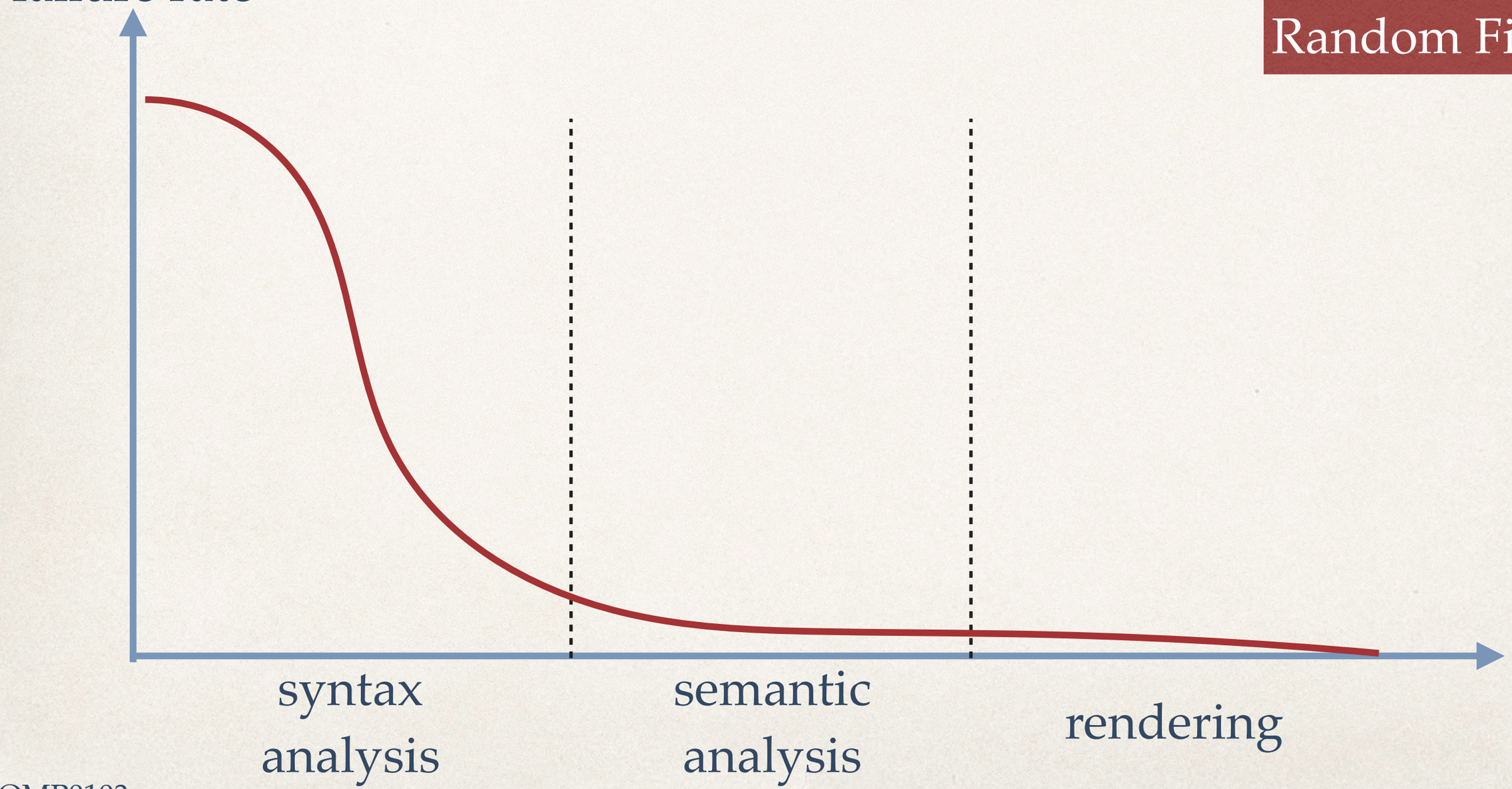
UCL

Random Files

Web Browsers

Invalid Inputs

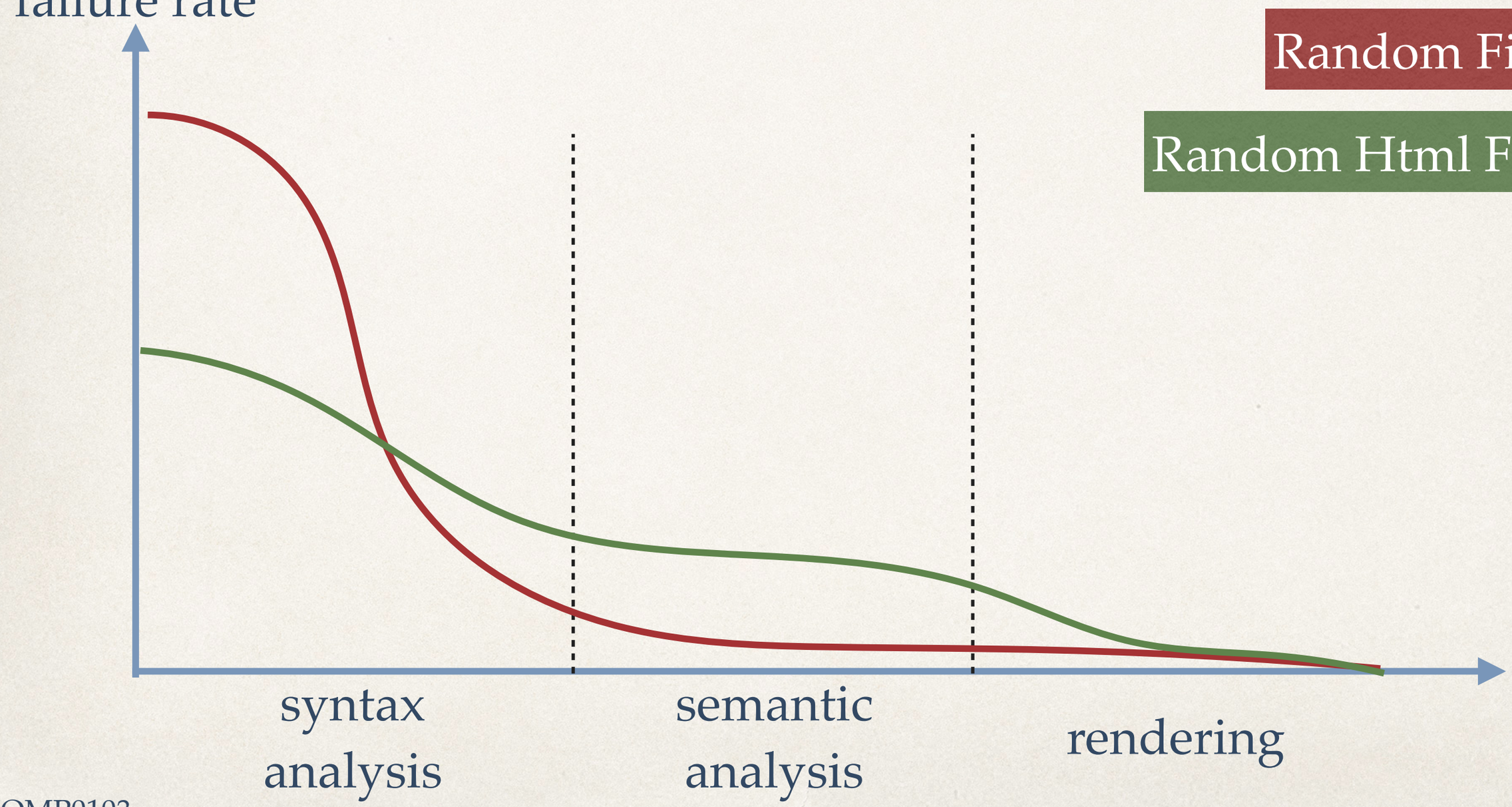
UCL



Random Files

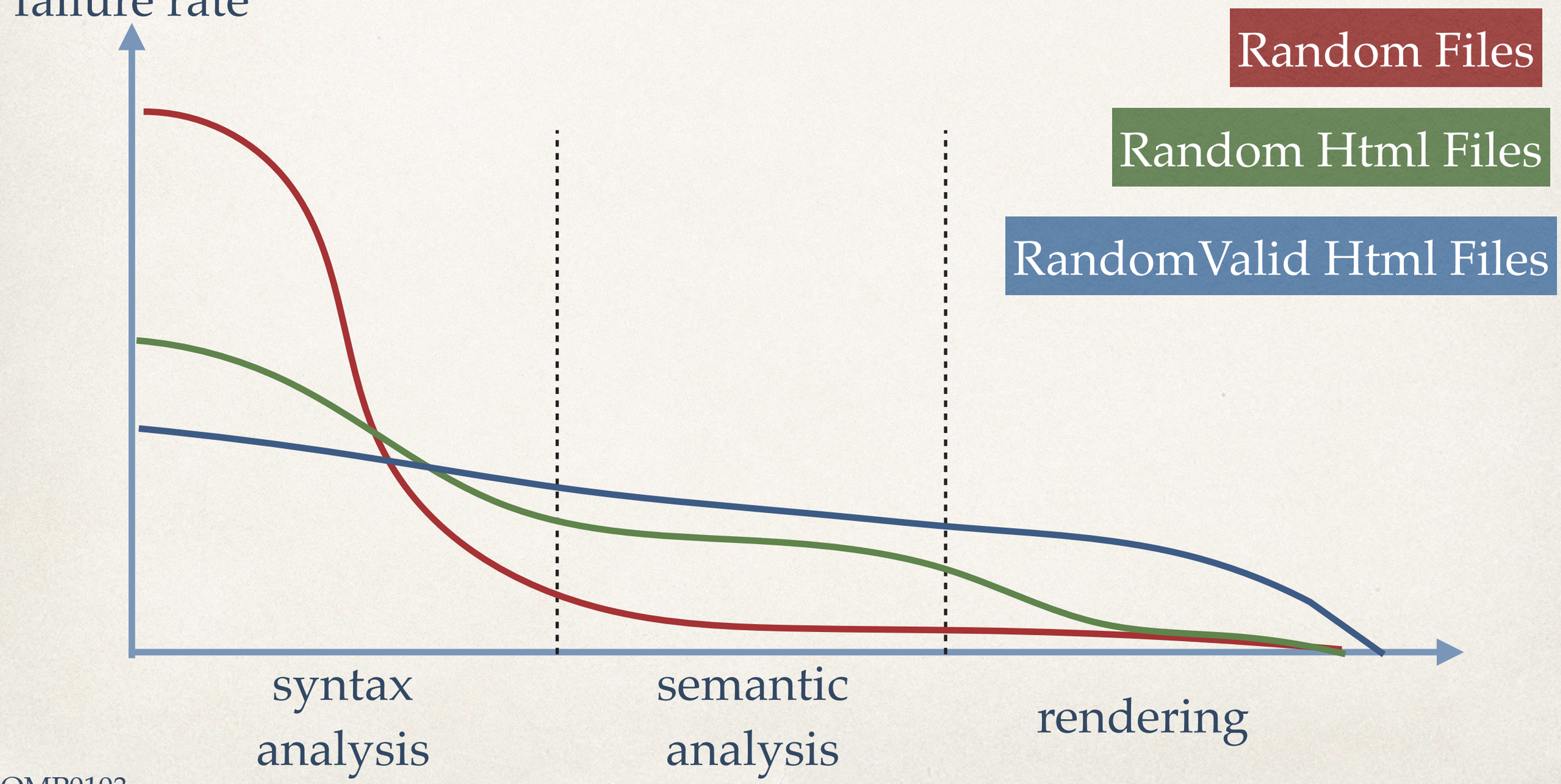
Invalid Inputs

UCL



Invalid Inputs

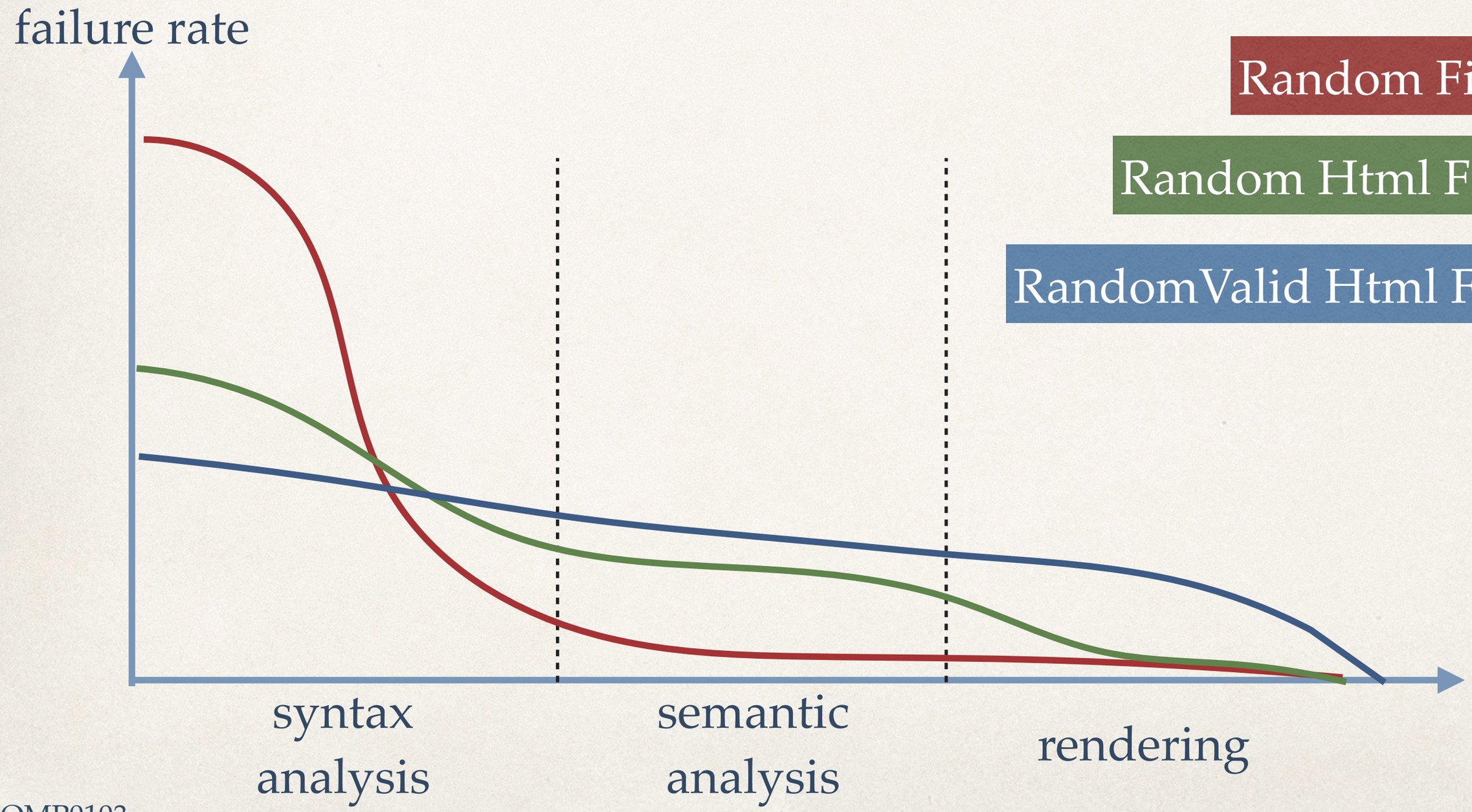
UCL



Invalid Inputs

Which is better?

UCL



Invalid Inputs

UCL

failure rate

How to generate

RandomValid Html Files

syntax analysis

semantic analysis

rendering

Invalid Inputs

UCL

How to generate RandomValid Html Files

Fuzzed Inputs

- Data mutation-based approach
- Starting from a set of valid inputs
- Making random changes to the valid inputs
- Execute SUT with the “fuzzed” inputs

Real World Applications

UCL

- Random Testing is actually used by industry. For example:
  - *Randomized Differential Testing as a Prelude to Formal Verification*, Groce et al, ICSE 2007: random testing for flight control system used in space missions
    - Very complex system, formal verification did not work, random testing found multiple faults with automation

Real World Applications: Netflix ChaosMonkey

- Netflix relies on Amazon Web Service cloud infrastructure to stream videos: the servers scale automatically according to the load
  - Aim: detect instance failure in Auto Scaling Group (ASG)
  - Action: ChaosMonkey just randomly shuts down AWS instances!
  - Oracle: the service should not collapse (implicit)
    - Load balancers should detect the instance failure, re-route requests, and additional instances need to be brought in

http://techblog.netflix.com/2012/07/chaos-monkey-released-into-wild.html