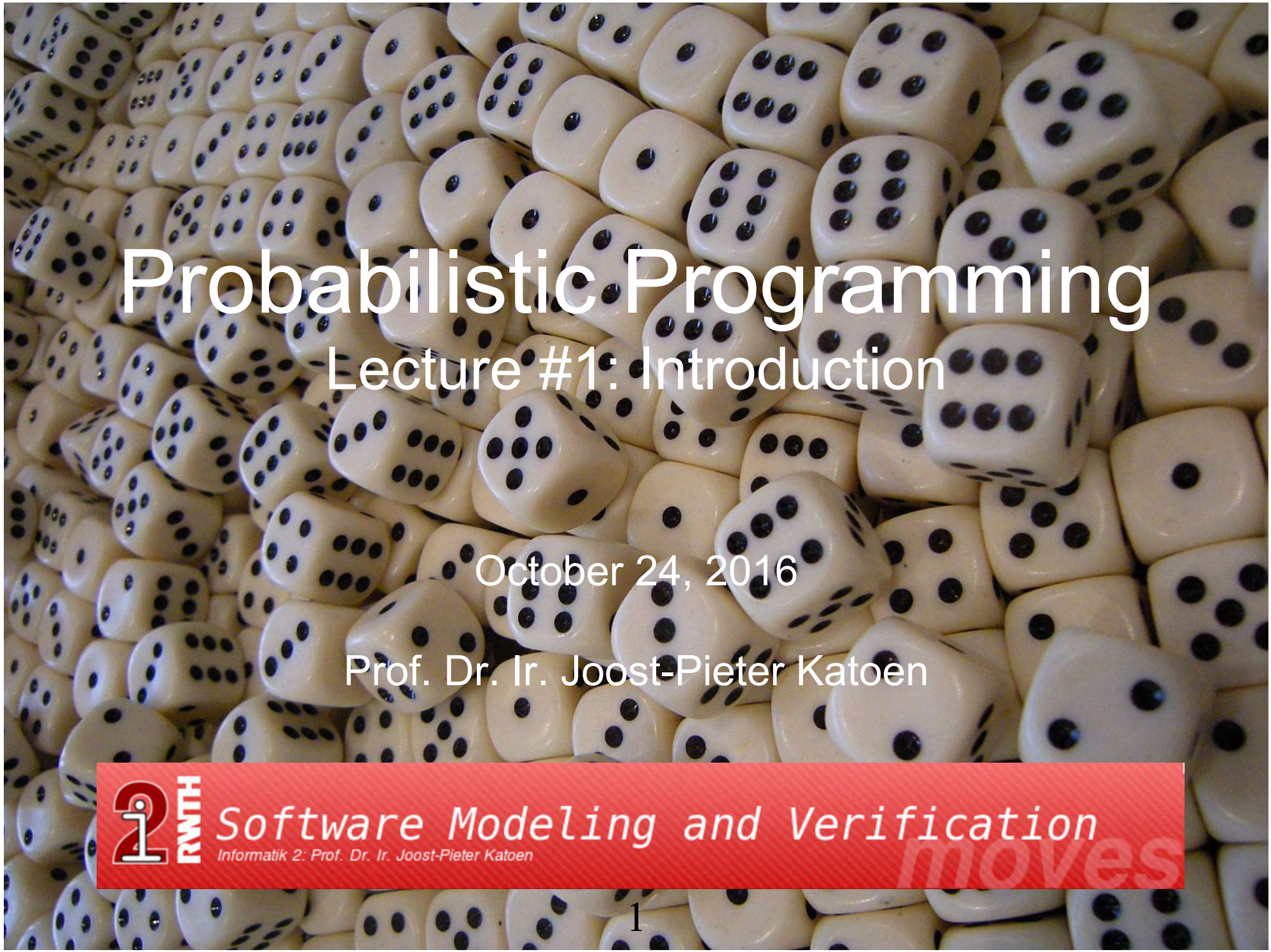


Probabilistic Programming

Lecture #1: Introduction

October 24, 2016

Prof. Dr. Ir. Joost-Pieter Katoen

RWTH i2 Software Modeling and Verification
Informatik 2: Prof. Dr. Ir. Joost-Pieter Katoen
moves

Probabilistic Programming

- What is probabilistic programming?
- What are probabilistic programs good for?
- Why are probabilistic programs intricate?
- What are we going to do in this course?
- What do we expect from you in this course?

Probabilistic Programming

- What is probabilistic programming?
- What are probabilistic programs good for?
- Why are probabilistic programs intricate?
- What are we going to do in this course?
- What do we expect from you in this course?

What is probabilistic programming?

The crux of probabilistic programming
is to consider normal-looking programs
as if they were probability distributions.

The Programming Languages Enthusiast

Michael Hicks, Univ. of Maryland

Probabilistic Programming Languages

Probabilistic-C	C
Figaro	Scala
Church	Scheme
Tabular	Windows Excel
ProbLog	Prolog
Rely	Guarded Command L
Venture	.....
PyMC	Python
webPPL	Javascript
.....	.....

PROBABILISTIC-PROGRAMMING.org

Probabilistic Programming

- What is probabilistic programming?
- What are probabilistic programs good for?
- Why are probabilistic programs intricate?
- What are we going to do in this course?
- What do we expect from you in this course?

Probabilistic Programming: Application Areas

Quantum Computing

Security

Bayesian Networks

Approximate Computing

Randomised Algorithms

Robotics

Probabilistic Programming: Application Areas

Quantum Computing

Security

Bayesian Networks

Approximate Computing

Randomised Algorithms

Robotics

How Statisticians Found Air France Flight 447 Two Years After It Crashed Into Atlantic

MIT Technology Review (2014)

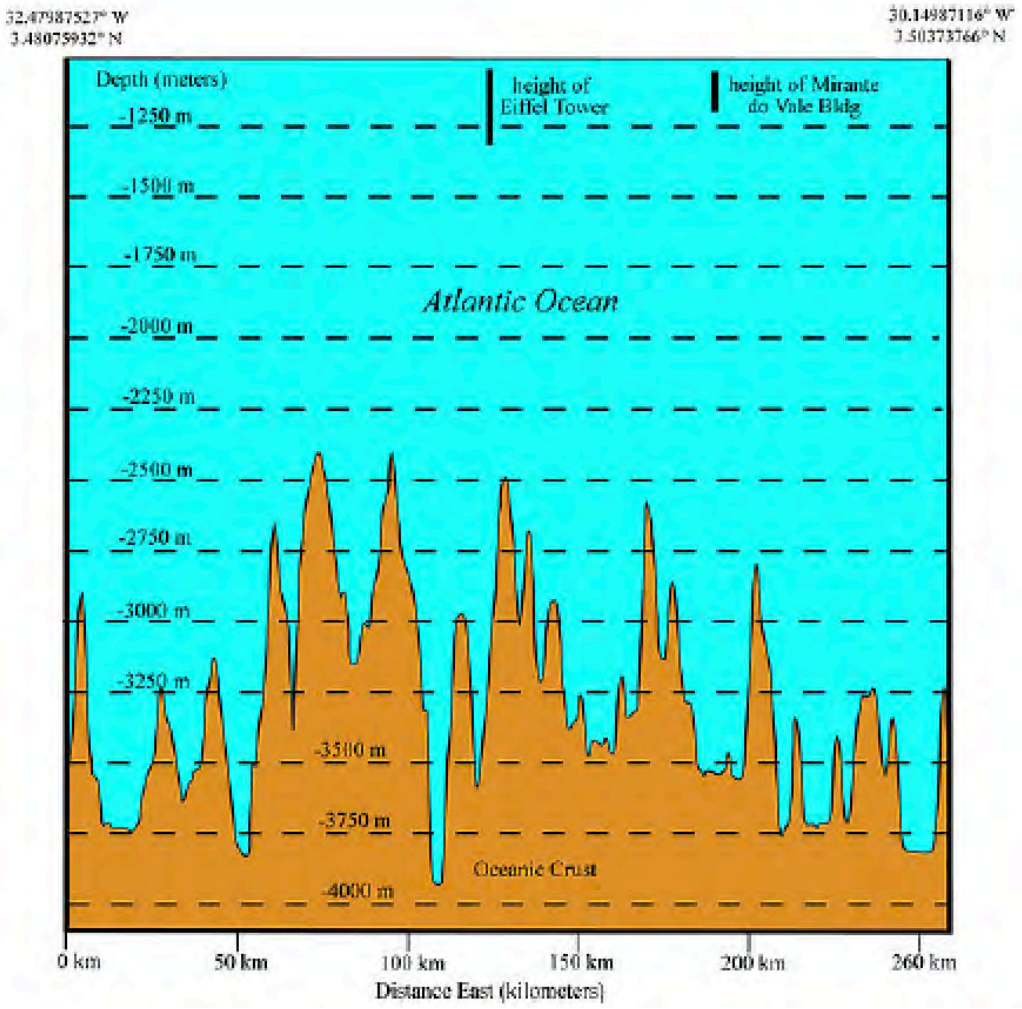
Air France Flight AF 447

Airbus A-330 AF 447

June 1, 2009

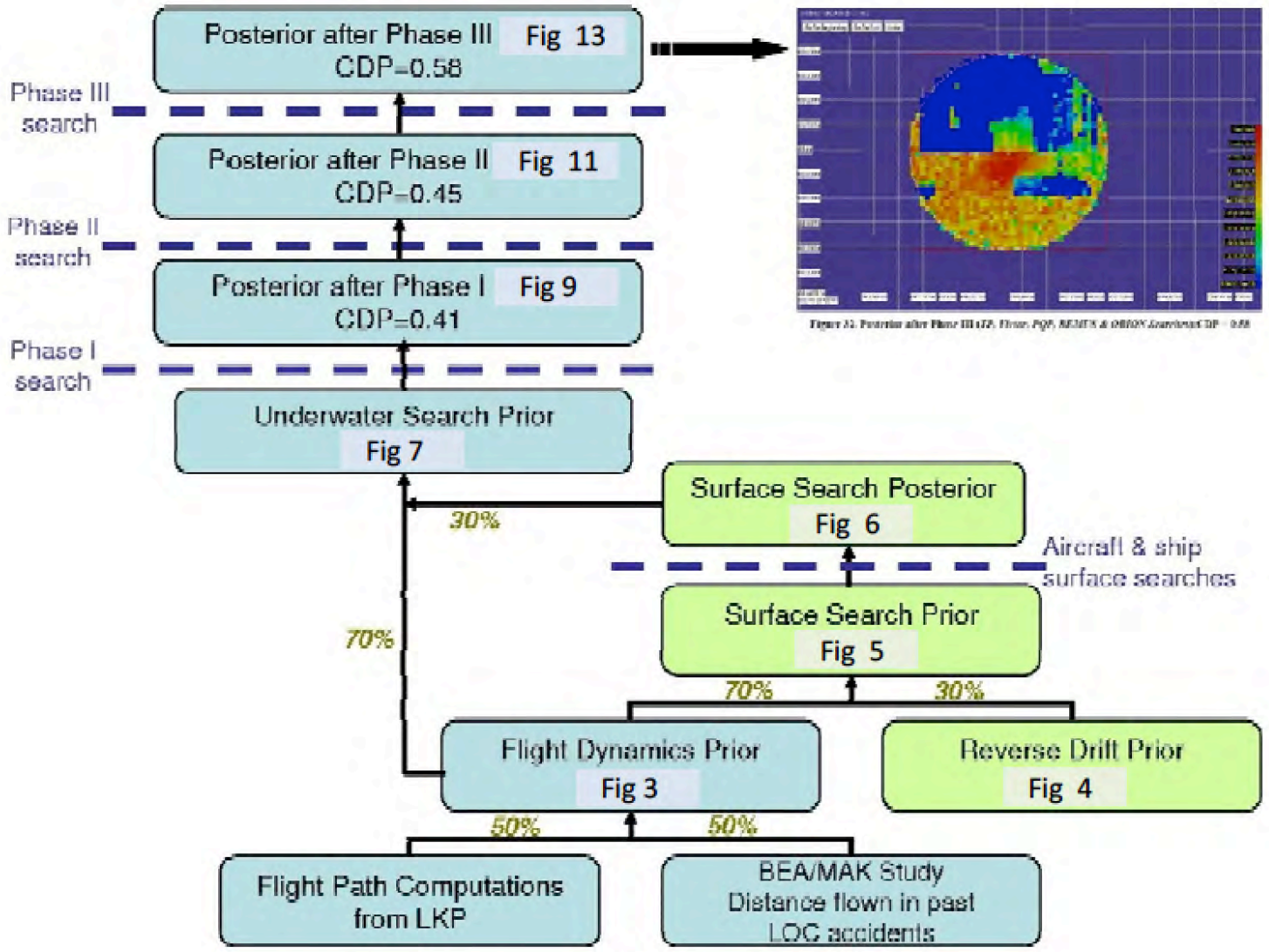
AF 447: First search attempts

AF 447: Where is the wreckage?



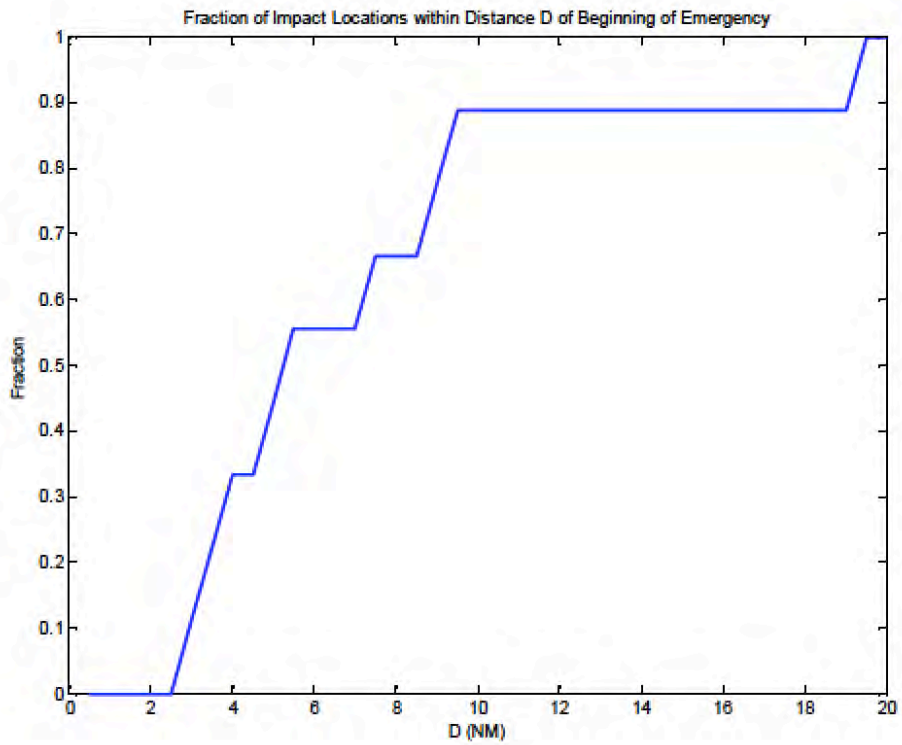
East-west cross section Atlantic

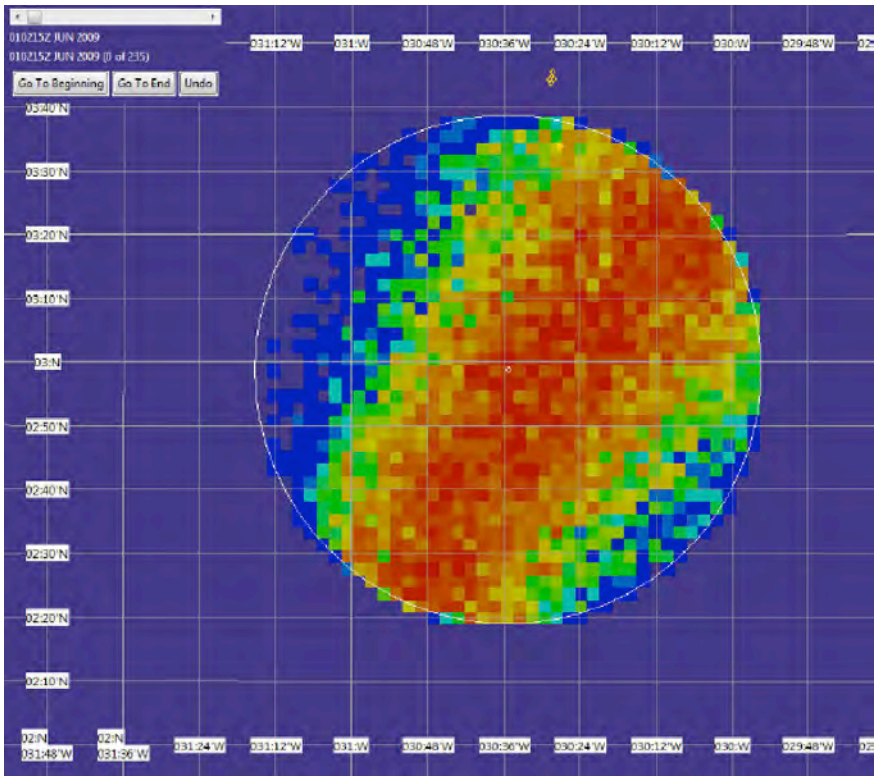
A `Probabilistic Program'



The prior distributions

Probability of possible location
Relative to beginning emergency

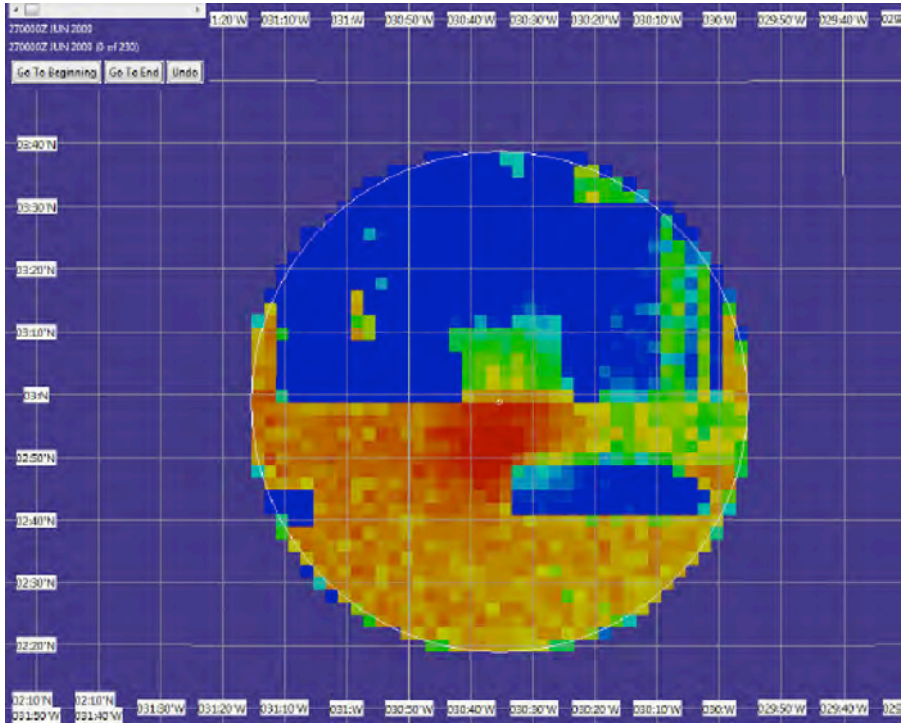


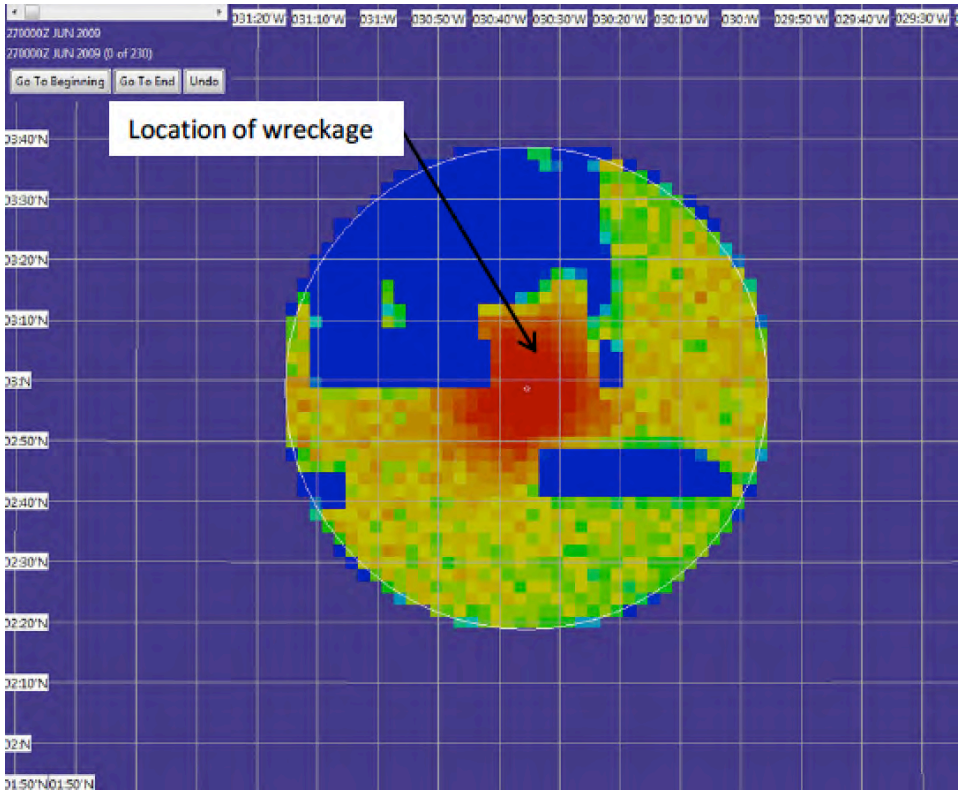


Reverse drift prior
(ocean and wind drift)

The posterior distributions

Pingers black boxes worked





Pingers of black boxes failed

Bayesian Networks

Rethinking the Bayesian Approach

[Daniel Roy, 2011][a]

"In particular, the graphical model formalism that ushered in an era of rapid progress in AI has proven inadequate in the face of [these] new challenges.

A promising new approach that aims to bridge this gap is probabilistic programming, which marries probability theory, statistics and programming languages"

[a] MIT/EECS George M. Sprowls Doctoral Dissertation Award

Probabilistic Programming: Application Areas

Quantum Computing

Security

Bayesian Networks

Approximate Computing

Randomised Algorithms

Robotics

Randomised Algorithms

- A randomised algorithm depends on *random numbers*
  - some decisions are based on random number generations

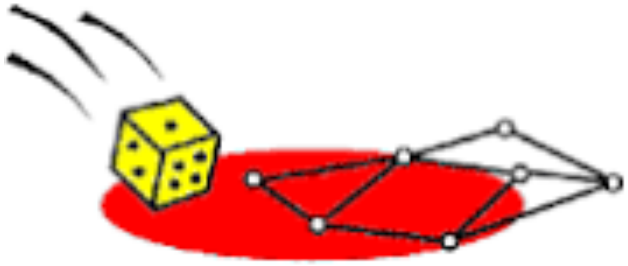
- That is:
  - Generate a random number *k* from some range *{1, …, N}*
  - Make decisions based on the value of *k*

- Instead of guessing whether the order is random,
  a random order is imposed

- Behavior of the algorithm depends on input and
  on the values produced by the random-number generator

Randomised Algorithms

- What? Randomised algorithms depend on random numbers
  - some decisions are based on random number generations



- Why randomised algorithms?
  1. Their conceptual simplicity
  2. Their speed: many are faster than deterministic algorithms
     (no particular input elicits worst-case behavior)
  3. Their existence:
     many solve problems that have no deterministic solution

- Types of randomised algorithms:

  1. Las Vegas: always produces correct results, random run-time
  2. Monte Carlo: may produces errors, deterministic run-time

Sample Randomised Algorithms

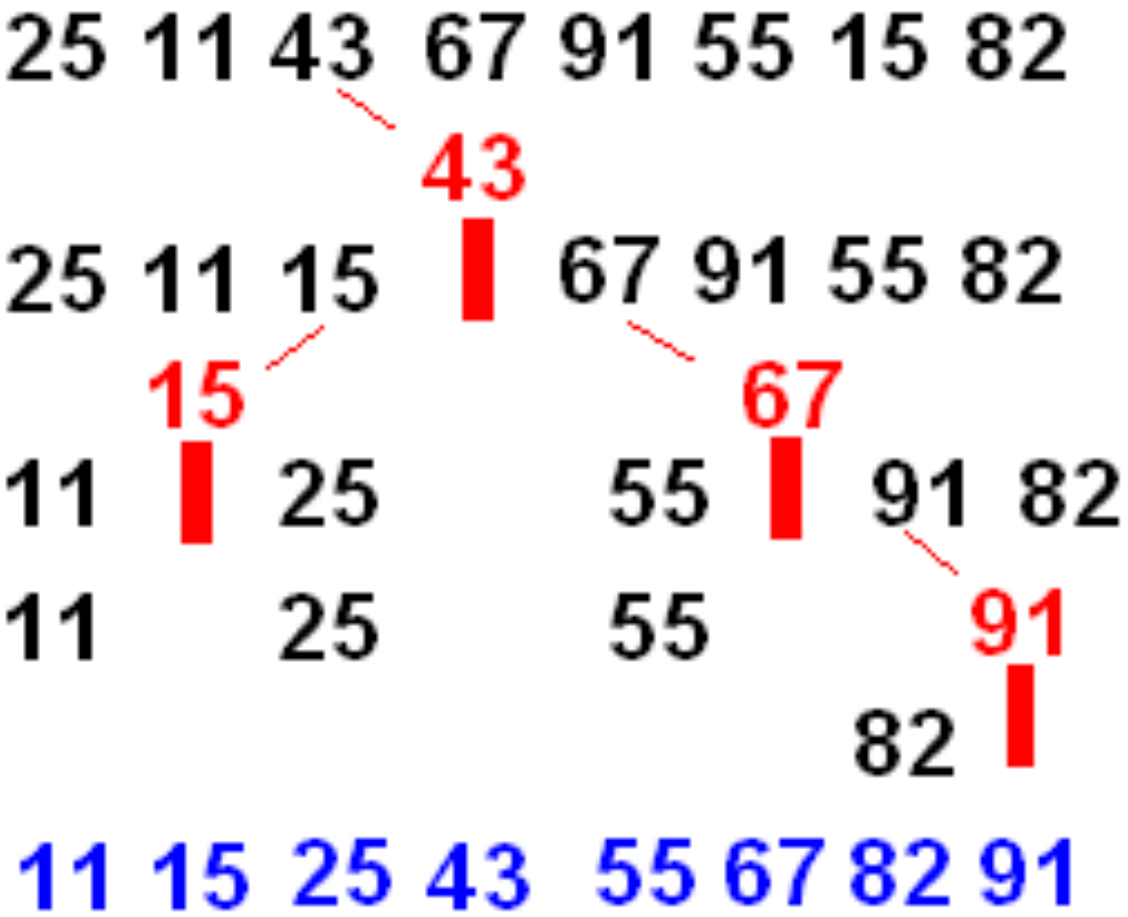
- Randomised *Quicksort*
- Randomised *Binary Search*
- Rabin-Miller's *Primality Test* (1980)
- Freivald's *Matrix Multiplication* (1977)
- Lehmann Rabin's Randomised *Mutual Exclusion* (1981)
- Itah-Rodeh's *Leader Election Protocol* (1990)

Hoare's Quicksort

Pivot 43

Pivots 15, 67

Pivot 91



Pick a pivot deterministically

Hoare's Quicksort

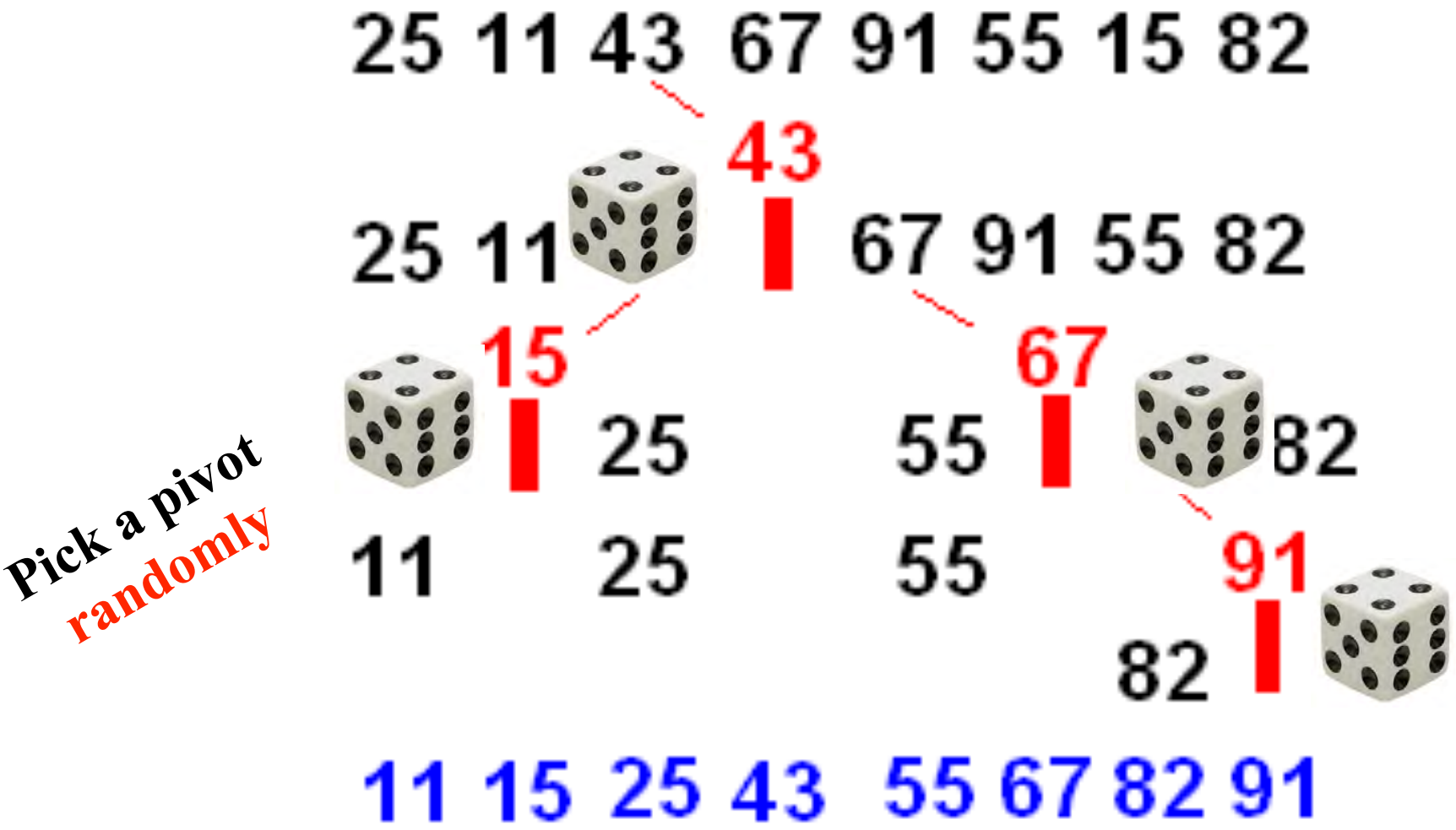
Quicksort:

$$
\begin{array}{l}
\mathrm{QS}(A) \triangleq \\
\quad \texttt{if}\ (|A| \leq 1)\ \texttt{then return}\ (A); \\
\quad i := \lfloor |A|/2 \rfloor; \\
\quad A_< := \{a' \in A \mid a' < A[i]\}; \\
\quad A_> := \{a' \in A \mid a' > A[i]\}; \\
\quad \texttt{return}\ \left(\mathrm{QS}(A_<) +\!\!+ A[i] +\!\!+ \mathrm{QS}(A_>)\right)
\end{array}
$$

Worst case complexity:

$O(n^2)$ comparisons

Randomised Quicksort



Randomised Quicksort:

$$
\begin{array}{l}
\mathsf{rQS}(A) \triangleq \\
\quad \texttt{if}\,(|A| \leq 1)\,\texttt{then return}\,(A); \\
\quad i := \mathsf{rand}[1 \ldots |A|]; \\
\quad A_< := \{a' \in A \mid a' < A[i]\}; \\
\quad A_> := \{a' \in A \mid a' > A[i]\}; \\
\quad \texttt{return}\,\left(\mathsf{QS}(A_<) +\!\!+ A[i] +\!\!+ \mathsf{QS}(A_>)\right)
\end{array}
$$

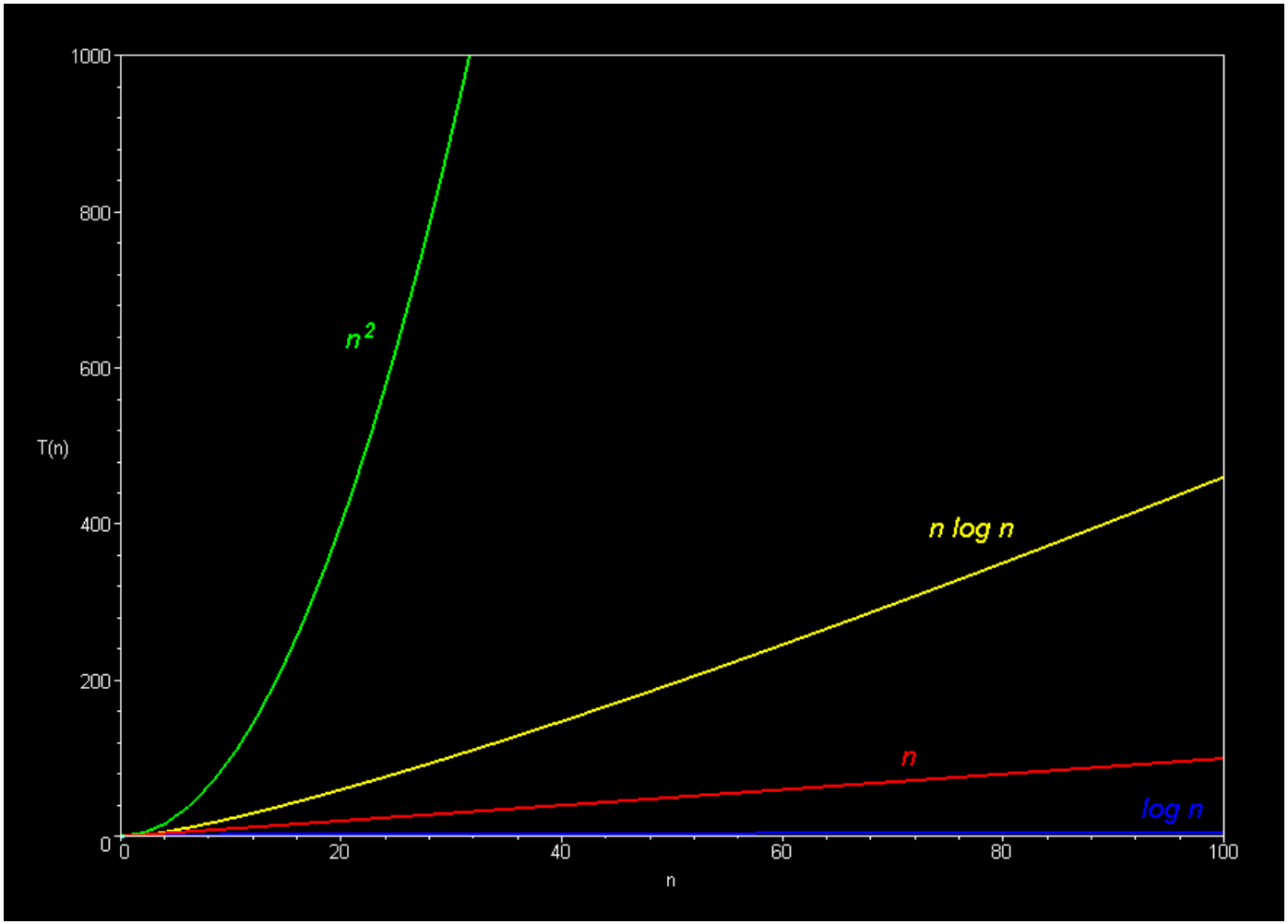
Worst case complexity:

O(n log(n)) expected comparisons

Why: better efficiency!

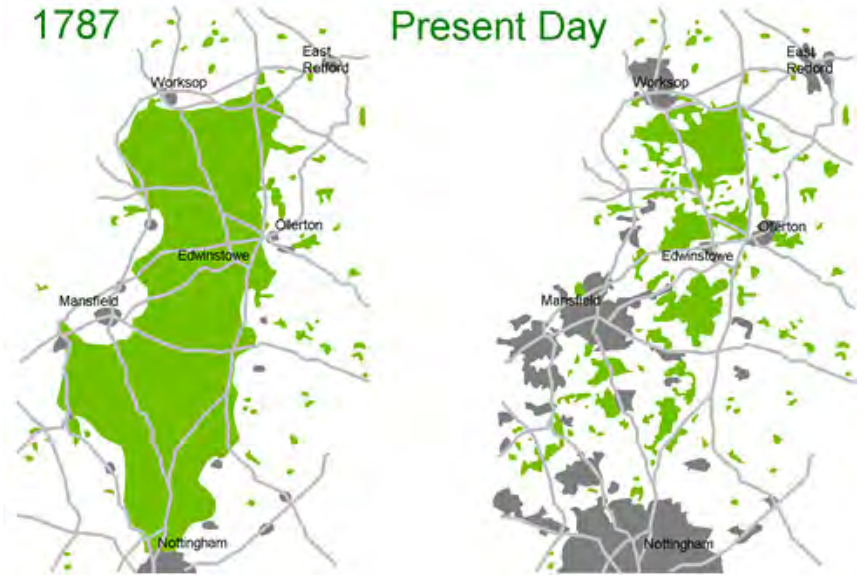
input size	Hoare's quicksort	randomised quicksort
n	$n^2/4$	$n \lg n$
10	25	33
100	2,500	664
1,000	250,000	9,965
10,000	25,000,000	132,877
100,000	2,500,000,000	1,660,960

Randomized versus Hoare's quicksort



Sherwood Algorithms

- Basic idea: randomise to **lower** the worst case complexity and **increase** the best case complexity.
- Like Robin Hood in Sherwood Forest, this approach **gives to the poor** (worst case) and **robs from the rich** (best case).
- E.g.: *randomised quicksort* and *randomised binary search*



Matrix Multiplication

Input: three $\mathcal{O}(N^2)$ square matrices A, B, and C

Output: yes, if $A \times B = C$; no, otherwise

Time complexity:

- until end 1960s: *cubic*
- 1969: *2.808*
- 1978: *2.796*
- 1979: *2.78*
- 1981: *2.522*
- 1982: *2.527*
- 1984: *2.496*
- 1986: *2.479*
- 1989: *2.376*
- 2014: *2.373*

Freivald's Matrix Multiplication (1977)

Input: three $\mathcal{O}(N^2)$ square matrices A, B, and C

Output: yes, if $A \times B = C$; no, otherwise

Deterministic: compute $A \times B$ and compare with C

Complexity: in $\mathcal{O}(N^3)$, best known complexity $\mathcal{O}(N^{2.37})$

Randomised:

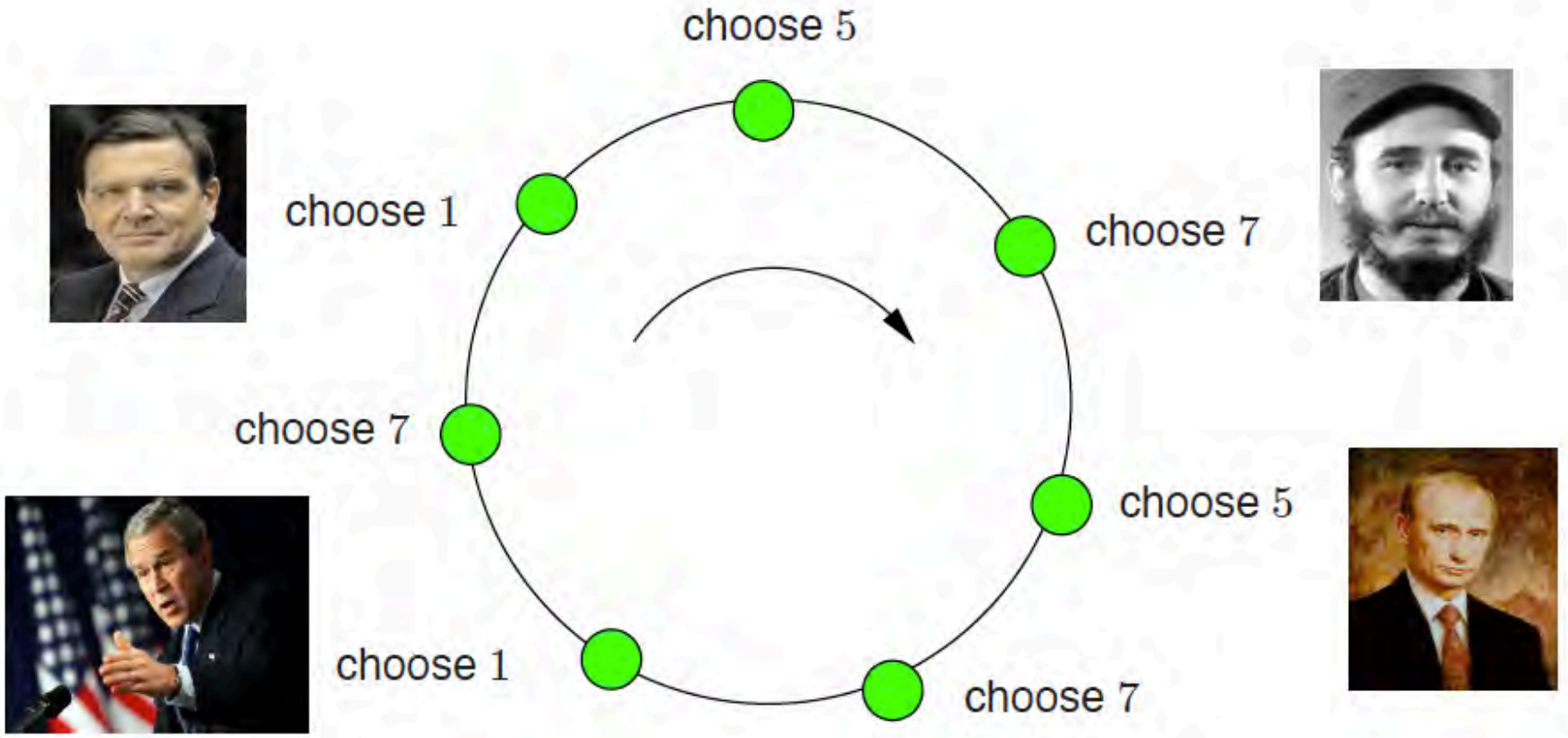
1. take a random bit-vector $\vec{x}$ of size N
2. compute $A \times (B\,\vec{x}) - C\,\vec{x}$
3. output yes if this yields the null vector; no otherwise
4. repeat these steps k times

Complexity: in $\mathcal{O}(k \cdot N^2)$, with false positive with probability $\leqslant 2^{-k}$

Randomised Leader Election

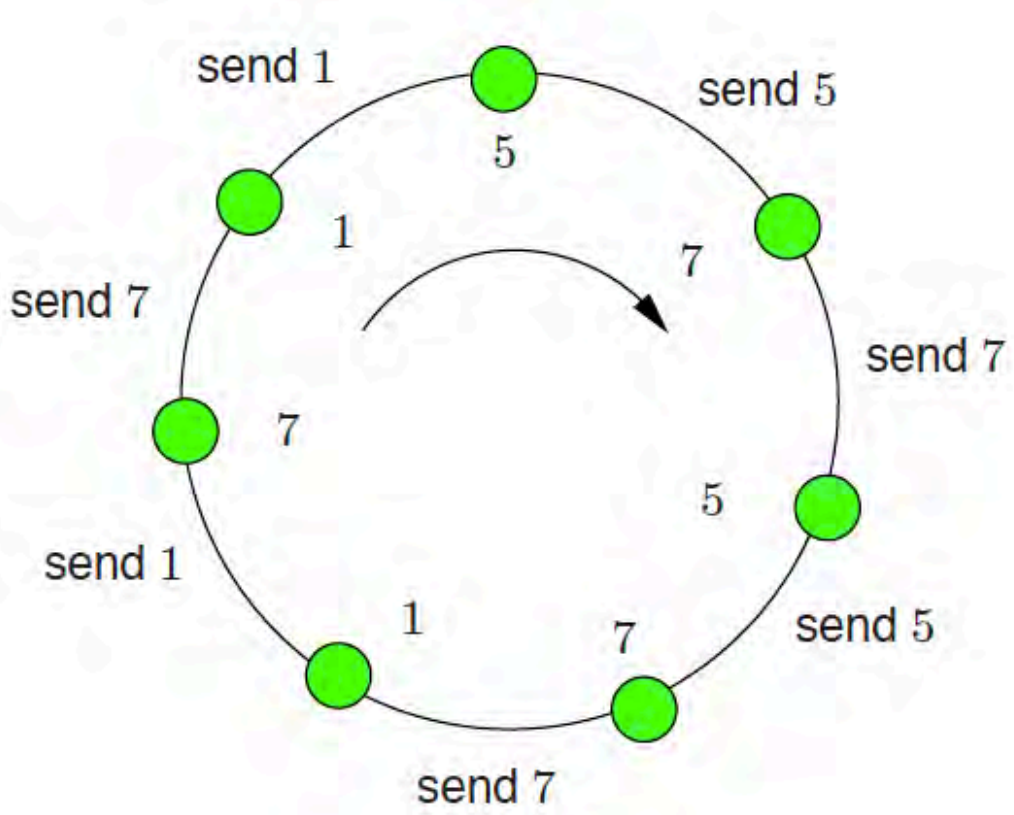
- A unidirectional ring network of *N* stations
  - Each node proceeds in a lock-step fashion
  - Each time-slot: read message + process it + send message
- Aim: elected a unique designated leader
- Each round starts by each station randomly selecting its id
  - According to rand(*1, …, K*) with *K* << *N*
- Stations pass their selected id around the ring
- Leader := station with highest unique id, if present

Randomised Leader Election



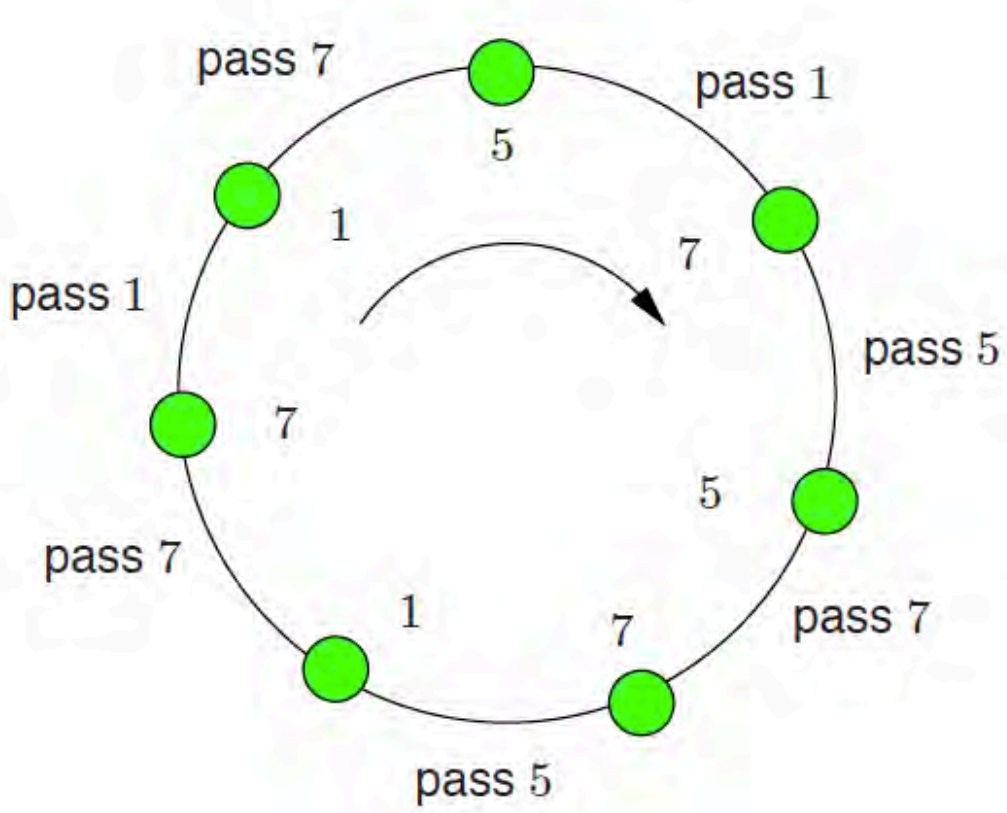
probabilistically choose an id from $[1 \ldots K]$

Randomised Leader Election



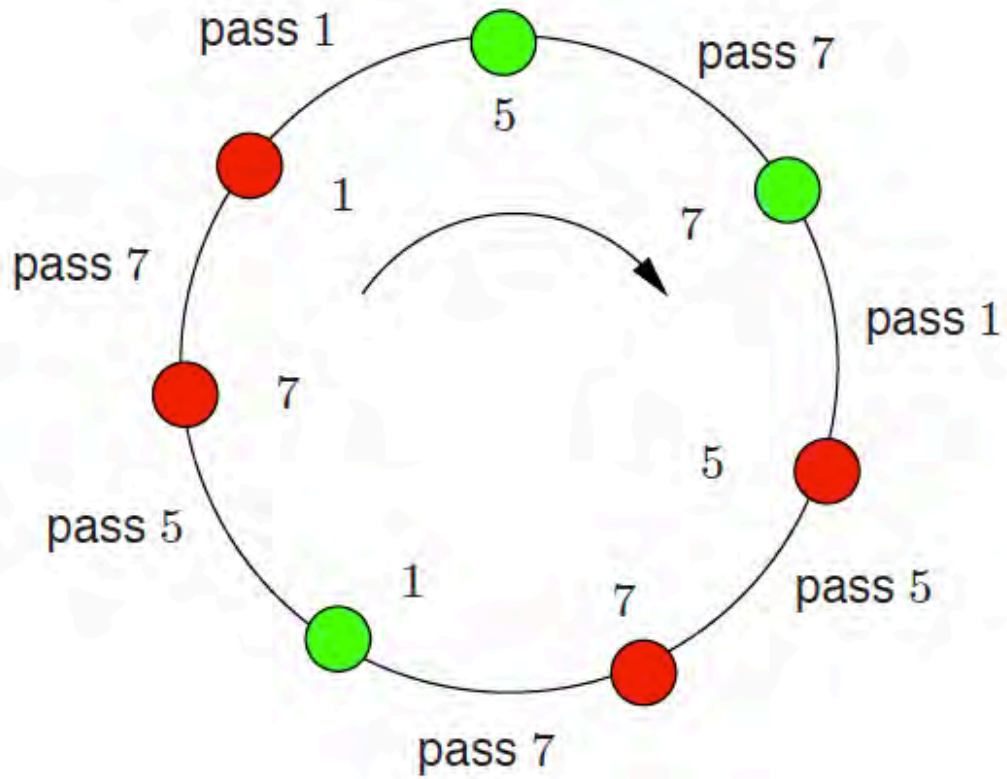
send your selected id to your neighbour

Randomised Leader Election



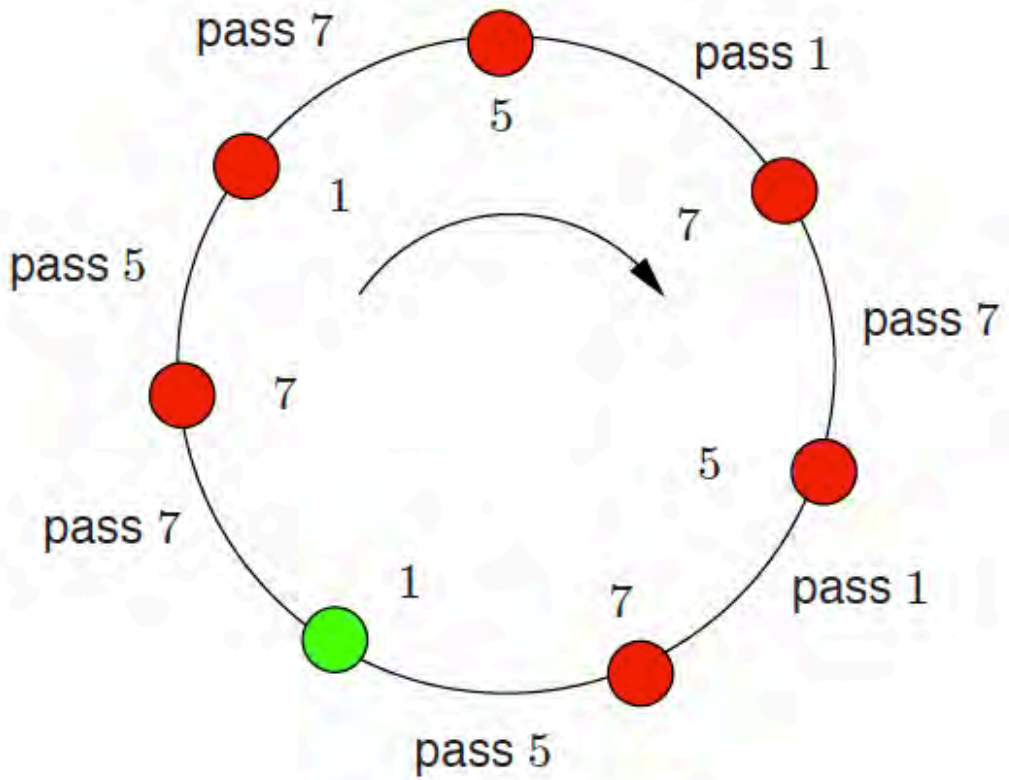
pass the received id, and check uniqueness own id

Randomised Leader Election



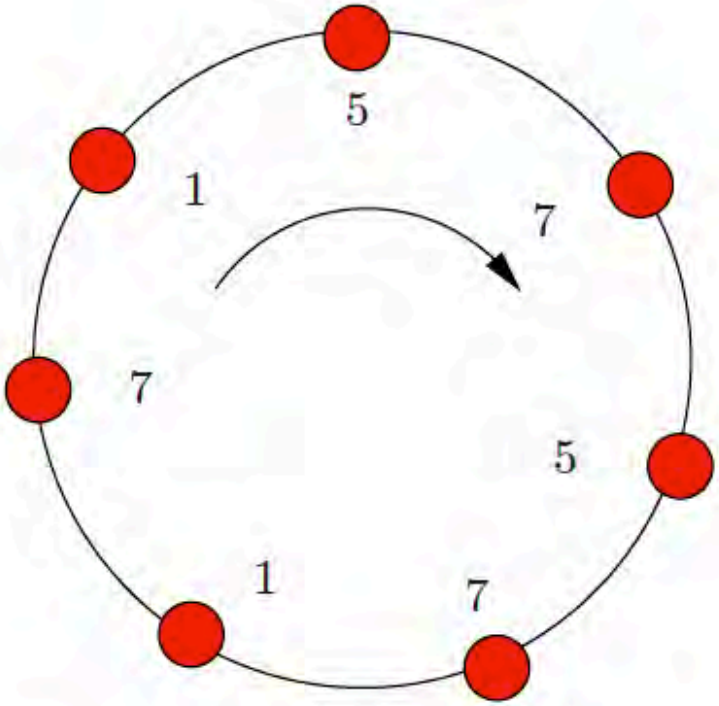
pass the received id, and check uniqueness own id

Randomised Leader Election



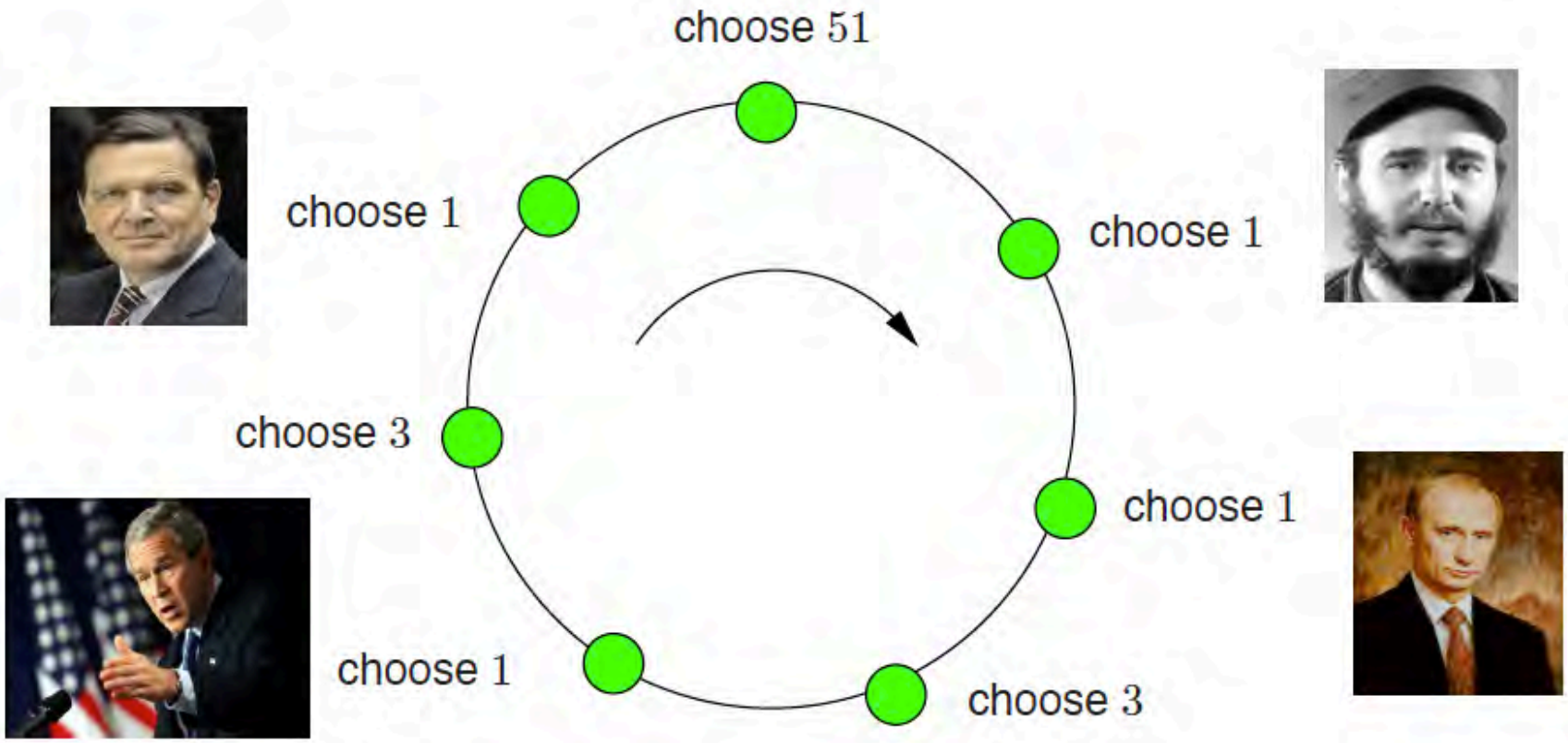
pass the received id, and check uniqueness own id

End of First Election Round



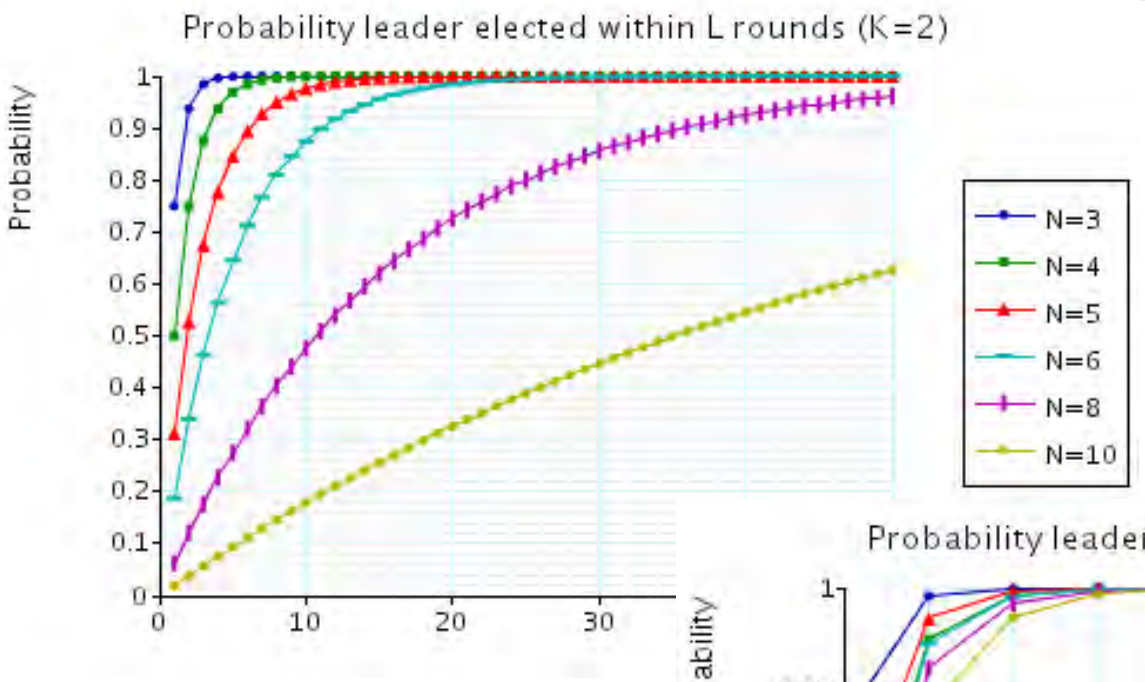
no unique leader has been elected

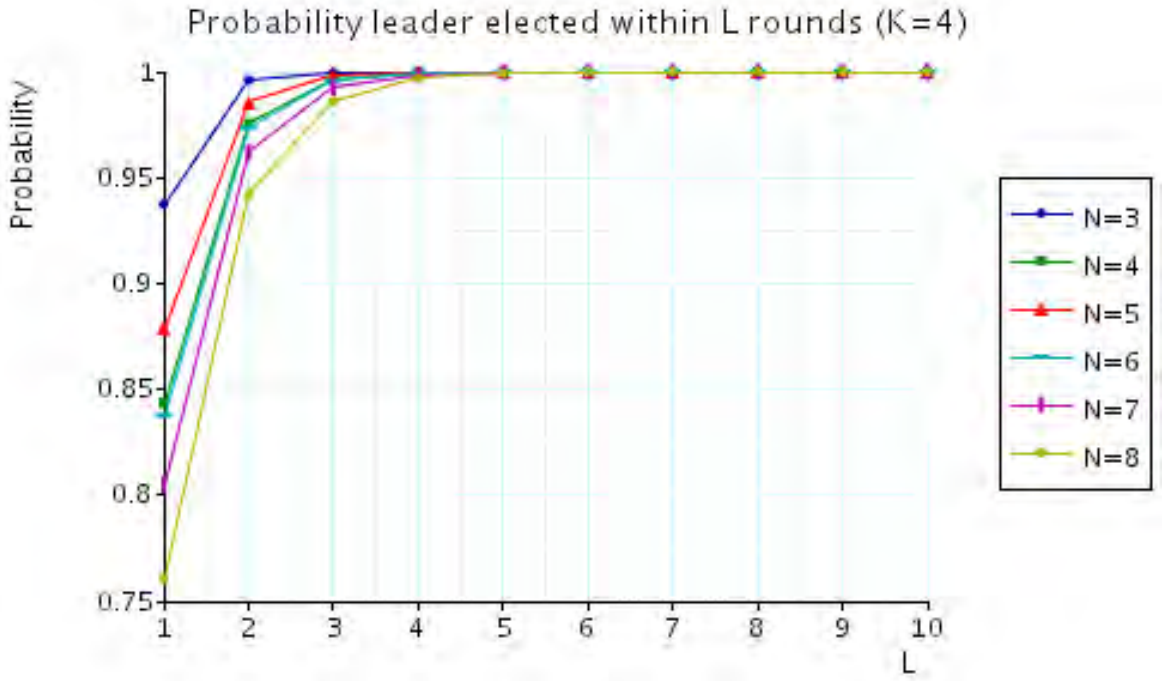
New Election Round



new round and new chances!

Probability to End Elections





Probabilistic Programming: Application Areas

Quantum Computing

Security

Bayesian Networks

Approximate Computing

Randomised Algorithms

Robotics

Security

Turing Award Winners 2013

"Goldwasser and Micali proved (1982) that encryption schemes must be **random** rather than deterministic […] an insight that revolutionised the study of encryption and laid the foundation for the theory of cryptographic security."

used in almost all communication protocols,
Internet transactions and cloud computing

The Famous RSA-OAEP Protocol

Oracle $\mathrm{Enc}_{pk}(m)$:
$r \xleftarrow{\$} \{0,1\}^{k_0}$;
$s \leftarrow G(r) \oplus (m \| 0^{k_1})$;
$t \leftarrow H(s) \oplus r$;
return $f_{pk}(s \| t)$

Oracle $\mathrm{Dec}_{sk}(c)$:
$(s,t) \leftarrow f_{sk}^{-1}(c)$;
$r \leftarrow t \oplus H(s)$;
if $[s \oplus G(r)]_{k_1} = 0^{k_1}$ then return $[s \oplus G(r)]^n$
else return $\perp$

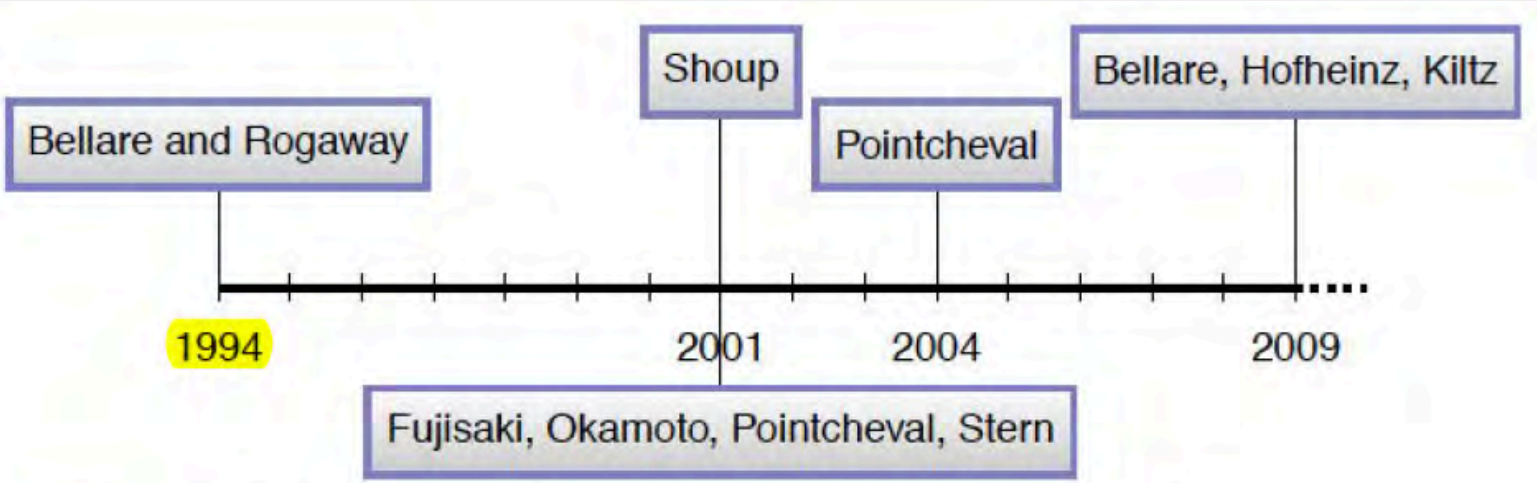
Oracle $G(x)$:
if $x \notin \mathrm{dom}(L_G)$ then $L_G[x] \xleftarrow{\$} \{0,1\}^{n+k_1}$;
return $L_G[x]$

Oracle $H(x)$:
if $x \notin \mathrm{dom}(L_H)$ then $L_H[x] \xleftarrow{\$} \{0,1\}^{k_0}$;
return $L_H[x]$

Game IND-CCA2 :
$(sk, pk) \leftarrow \mathcal{KG}(\,)$;
$(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1(pk)$;
$b \xleftarrow{\$} \{0,1\}$;
$c^* \leftarrow \mathrm{Enc}(pk, m_b)$;
$b' \leftarrow \mathcal{A}_2(pk, c^*, \sigma)$;
return $b = b'$

Game POW :
$(sk, pk) \leftarrow \mathcal{KG}()$;
$y \xleftarrow{\$} \{0,1\}^{n+k_1}$;
$z \xleftarrow{\$} \{0,1\}^{k_0}$;
$y' \leftarrow \mathcal{I}(f_{pk}(y \| z))$;
return $y = y'$

Its Correctness Proof Took Very Long



1994 Purported proof of chosen-ciphertext security

2001 Proof establishes a weaker security notion, but desired security can be achieved

1. ...for a modified scheme, or
2. ...under stronger assumptions

2004 Filled gaps in Fujisaki et al. 2001 proof

2009 Security definition needs to be clarified

2010 Filled gaps and improved bounds from 2004 proof

2012 Improved bound from 2010 proof

Probabilistic Programming

- What is probabilistic programming?
- What are probabilistic programs good for?
- Why are probabilistic programs intricate?
- What are we going to do in this course?
- What do we expect from you in this course?

Three Core Issues

1. Program Correctness
2. Termination
3. Run-Time

Issue 1: Program Correctness

- Traditional programs:
  - A program is correct with respect to a (formal) specification
    "for input array A, the output array B is sorted and contains all elements contained in A"
  - It refers to the deterministic input-output relation of a program
    on a given input, always the same output is provided
  - Partial correctness: if an output is produced, it is correct
  - Total correctness: in addition, the program terminates

- Probabilistic programs:
  - They do not always generate the same output
  - They generate a probability distribution over possible outputs

Let us start simple

```
x := 0 [0.5] x := 1;
y := -1 [0.5] y := 0
```

This program admits four runs and yields the outcome:

$$Pr[x=0, y=0] = Pr[x=0, y=-1] = Pr[x=1, y=0] = Pr[x=1, y=-1] = 1/4$$

Loops

For p an arbitrary probability:

```
bool c := true;
int i : = 0;
while (c) {
    i := i + 1;
    (c := false [p] c := true)
}
```

This models a geometric distribution with parameter p.

$$Pr[i = N] = (1-p)^{N-1} \cdot p \quad \text{for } N > 0$$

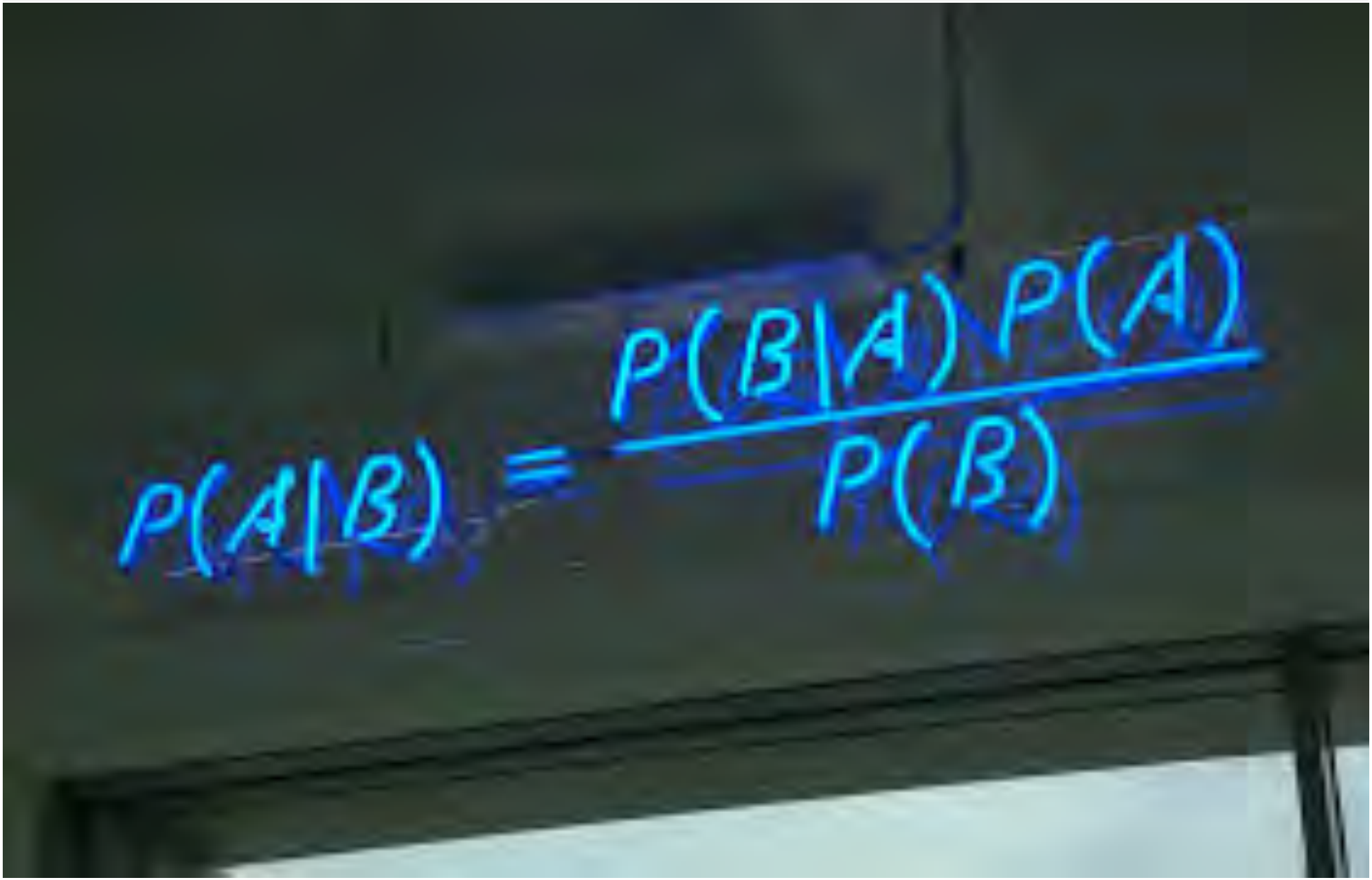
Program Equivalence

```
int XminY1(float p, q){
  int  x, f := 0, 0;
  while (f = 0) {
    (x +:= 1 [p] f := 1);
  }
  f := 0;
  while (f = 0) {
    (x -:= 1 [q] f := 1);
  }
  return x;
}
```

```
int XminY2(float p, q){
  int x, f := 0, 0;
  (f := 0 [0.5] f := 1);
  if (f = 0) {
    while (f = 0) {
      (x +:= 1 [p] f := 1);
    }
  } else {
    f := 0;
    while (f = 0) {
      x -:= 1;
      (skip [q] f := 1);
    }
  }
return x;
}
```

For which p and q are these two programs equivalent?

Conditioning



Conditioning

```
x := 0 [0.5] x := 1;
y := -1 [0.5] y := 0;
observe (x+y = 0)
```

This program blocks two runs as they violate `x+y = 0`. Outcome:

$$Pr[x=0, y=0] = Pr[x=1, y=-1] = 1/2$$

Observations thus normalize the probability of the "feasible" program runs

The Piranha Puzzle [Tijms 2004]

One fish is contained within the confines of an opaque fishbowl. The fish is equally likely to be a piranha or a goldfish. A sushi lover throws a piranha into the fish bowl alongside the other fish. Then, immediately, before either fish can devour the other, one of the fish is blindly removed from the fishbowl. The fish that has been removed from the bowl turns out to be a piranha. What is the probability that the fish that was originally in the bowl by itself was a piranha?

The Piranha Puzzle Program

```
f1 := gf [0.5] f1 := pir;
f2 := pir;
s := f1 [0.5] s := f2;
observe (s = pir)
```

What is the probability that the original fish was a piranha?

$$\mathbb{E}(\texttt{f1 = pir} \mid P \text{ terminates}) = \frac{1 \cdot 1/2 + 0 \cdot 1/4}{1 - 1/4} = \frac{1/2}{3/4} = \frac{2}{3}.$$

Issue 2: Program Termination

- Traditional programs:
  - They terminate (on a given/all inputs), or they do not
  - If they terminate, it takes finitely many steps to do so
  - Showing program termination is undecidable (halting problem)

- Probabilistic programs:
  - They terminate (or not) with a certain likelihood
  - They may have diverging runs whose likelihood is zero
  - They may take infinitely many steps (on average) to terminate
  - Showing “probability one” termination is “more” undecidable

Termination

```
bool c := true;
int i : = 0;
while (c) {
    i := i + 1;
    (c := false [p] c := true)
}
```

This program does not always terminate.
It terminates with probability one.
In finite time, on average.

When does it terminate?

```
bool c := true;
int nrflips := 0;
while (c) {
      nrflips := nrflips + 1;
      (c := false [0.5] c := true);
}
```

Expected runtime (integral over the bars):

The nrflips-th iteration takes place with probability $1/2^{\texttt{nrflips}}$.

Consider the one-dimensional (symmetric) random walk:

```
int x : = 10;
while (x > 0) {
    (x := x - 1 [0.5] x := x + 1)
}
```

This program almost surely terminates
but requires an infinite expected time to do so.

Issue 3: Program Efficiency

- Traditional programs:
  - They have a deterministic, fixed run-time for a given input
  - Run-times of terminating programs in sequence are compositional
    if *P* and *Q* terminate in *n* and *k* steps, then *P;Q* halts in *n*+*k* steps
  - Analysis techniques: recurrence equations, tree analysis, etc.

- Probabilistic programs:
  - Every run-time has a probability; their run-time is a distribution
  - Run-times of "probability one" terminating programs in sequence are not compositional
  - Analysis techniques: involve reasoning about expected values etc.

Consider the two probabilistic programs:

```
int x := 1;
bool c := true;
while (c) {
    c := false [0.5] c := true;
    x := 2*x
}
```

Finite expected termination time

```
while (x > 0) {
    x : = x - 1
}
```

Finite termination time


```
int x := 1;
bool c := true;
while (c) {
    c := false [0.5] c := true;
    x := 2*x
}
```

Finite expected termination time

```
while (x > 0) {
    x : = x - 1
}
```

Finite termination time

How long does it take (on average) to run the programs in sequence?

Probabilistic Programming

- What is probabilistic programming?
- What are probabilistic programs good for?
- Why are probabilistic programs intricate?
- What are we going to do in this course?
- What do we expect from you in this course?

Probabilistic Programming Organisation

- This course is a block course

- Two lectures/week in December 2016 and January 2017
  - Mondays 16:15-17:45, room 9U10
  - Tuesdays 10:15-11:45, room 9U10
  - Weeks **49**, 50, 51, 2, 3, and **5**

- One exercise class/week December 2016 – February 2017
  - Fridays 14:15-15:45, room 9U10
  - Weeks **49**, 50, 2, 3, 5, and 6
  - Instructors: *Federico Olmedo and Christoph Matheja*

Probabilistic Programming Material

- Lecture material = the slides + the lectures + the exercises
  web page: moves.rwth-aachen.de/teaching/ws-1617/

- **`webPPL`** website (webppl.org) and its accompanying book (on dippl.org):
  Noah Goodman and Andreas Stuhlmüller:
  The Design and Implementation of Probabilistic Programming Languages, 2016

- Course is based on (very recent) literature and the book:
  Annabelle McIver and Carroll Morgan:
  Abstraction, Refinement and Proof for Probabilistic Systems
  Springer, 2005.

Probabilistic Programming Topics

- Probabilistic programming in **`webPPL`**
  examples, recursion, plots, conditioning

- The probabilistic guarded command language **`pGCL`**
  examples, syntax, semantics (Markov chains), conditioning, non-determinism, [recursion]

- Formal reasoning about probabilistic programs
  weakest pre-conditions, loop invariants, post-conditions

- Almost-sure termination

- Run-time analysis of probabilistic programs

Milestones in Program Verification

- 1969: Program annotations

Tony Hoare

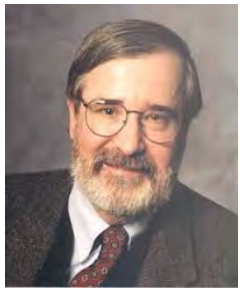
- 1970: Weakest precondition

Edsger Dijkstra

- 1977: Modal logic

Amir Pnueli

- 1981: Model checking



E. Clarke

A. Emerson

J. Sifakis

Milestones in Verifying Probabilistic Programs

- 1979: Program semantics

Dexter Kozen

- 1983: Dynamic logic

A. McIver
C. Morgan

- 1997: Weakest precondition
- 2006: New programming languages

Microsoft Research

A. Pfeffer D. Roy N. Goodman

Probabilistic Programming

- What is probabilistic programming?
- What are probabilistic programs good for?
- Why are probabilistic programs intricate?
- What are we going to do in this course?
- What do we expect from you in this course?

Probabilistic Programming: You

- You are expected to hand in homework exercises
  - working in groups of max. two students
  - 50% of the points are required for exam qualification
  - at least 90% of the point: bonus point for exam
  - **`webPPL`** programming exercises + (mostly) theory exercises

- Dates
  - Exercise class: Fridays 14:15-15:45, Weeks 49-50, 2-3, 5-6
  - First exercise series available: December 2, 2016
  - First exercise hand-in deadline: December 9, 2016
  - Written exam: February (week 7) and March 2017 (week 12)

- Reward: **4 ECTS**

Probabilistic Programming is …….