Perfect Failure Detection
with Very Few Bits

Pierre Fraigniaud 1  Sergio Rajsbaum 2  C. Travers 3
Petr Kuznetsov 4  Thibault Rieutord 4

1IRIF, Paris  2UNAM, Mexico  3LaBRI, Bordeaux
4ParisTech, Paris

ANR Descartes, Chasseneuil, Octobre 2017
• Distributed device
• Give (unreliable) information on failures
Modular Distributed Computing

- Protocol
  - Failure Detector
  - Communication Primitives
- Network
Modular Distributed Computing

Protocol

Failure Detector  Communication Primitives

Network

Asynchrony Failures
Failure detector $D$ is \textit{the weakest for task }$T$

1. There is a protocol for $T$ using $D$
2. Any f.d. $D'$ that can be used to solve $T$ can emulate $D$
Relative Hardness of Distributed Task

Failure detector $D$ is \textit{the weakest for task $T$}

\begin{align*}
1. & \text{ There is a protocol for } T \text{ using } D \\
2. & \text{ Any f.d. } D' \text{ that can be used to solve } T \text{ can emulate } D
\end{align*}

Minimum information on failures required to solve $T$
• Local failure detection module at each proc.
Failure Detectors

- Local failure detection module at each proc.
- Provide information on other proc. failures
Perfect failure detector

\[ \{p_1, p_4\} \]

- Provide each proc. with a list of proc ids.
- No false alarm
- Eventually outputs the set of non-faulty processes

Perfect failure detector \( P \)
Perfect Failure Detector

\{p_1, p_4\}

$\{p_1, p_4\}$

Perfect failure detector $P$

- Provide each proc. with a list of proc ids.

$p_1$ $p_2$ $p_3$ $p_4$ $p_5$
Perfect failure detector $P$

- Provide each proc. with a list of proc ids.
- No false alarm
Perfect Failure Detector

Perfect failure detector \( P \)

- Provide each proc. with a list of proc ids.
- No false alarm
- Eventually outputs the set of non-faulty processes
Failure Detector $\phi$

- Provide each proc. with an integer
- Lower bound on the number of failures
- Eventually tight
Failure detector $\phi$

- Provide each proc. with an *integer*
Failure detector $\phi$

- Provide each proc. with an integer
- Lower bound on the number of failures
Failure detector $\phi$

- Provide each proc. with an integer
- Lower bound on the number of failures
- Eventually tight
In a $n$-process system

\begin{itemize}
\item \textbf{P}
\item \textbf{φ}
\end{itemize}

And yet:

\textbf{Theorem (Mostefaoui, Raynal, T.)}

\textbf{P} and \textbf{φ} are equivalent: any task that can be solved using \textbf{P} (resp. \textbf{φ}) can also be solved using \textbf{φ} (resp. \textbf{P})
In a $n$-process system

$\mathbf{P}$
- **List** of proc ids.

$\mathbf{\phi}$
- **integer** $f$, $0 \leq f \leq n$

And yet:

Theorem (Mostefaoui, Raynal, T.) $\mathbf{P}$ and $\mathbf{\phi}$ are equivalent: any task that can be solved using $\mathbf{P}$ (resp. $\mathbf{\phi}$) can also be solved using $\mathbf{\phi}$ (resp. $\mathbf{P}$).
In a $n$-process system

$\mathbf{P}$

- **List** of proc ids.
- $n$ bits per process

$\mathbf{\phi}$

- integer $f$, $0 \leq f \leq n$
- $\log n$ bits per process

And yet:

Theorem (Mostefaoui, Raynal, T.) $\mathbf{P}$ and $\mathbf{\phi}$ are equivalent: any task that can be solved using $\mathbf{P}$ (resp. $\mathbf{\phi}$) can also be solved using $\mathbf{\phi}$ (resp. $\mathbf{P}$).
In a $n$-process system

$P$
- List of proc ids.
- $n$ bits per process

$\phi$
- integer $f$, $0 \leq f \leq n$
- $\log n$ bits per process

And yet:

Theorem (Mostefaoui, Raynal, T.)

$P$ and $\phi$ are equivalent: any task that can be solved using $P$ (resp. $\phi$) can also be solved using $\phi$ (resp. $P$)
How many bits per proc. are needed to achieve perfect failure detection?
How many bits per proc. are needed to achieve perfect failure detection?

Theorem (upper bound)
There exists a failure detector $\mu P$ as powerful as $P$ that outputs $O(Ack^{-1}(n))$ bits per proc.
How many bits per proc. are needed to achieve perfect failure detection?

**Theorem (upper bound)**

There exists a failure detector $\mu P$ as powerful as $P$ that outputs $O(Ack^{-1}(n))$ bits per proc.

**Theorem (lower bound)**

No failure detector outputting a constant number of bits per proc. can emulate $P$. 
- Message passing
• Message passing
• Asynchronous
- Message passing
- Asynchronous
- \( n \) processes
- Message passing
- Asynchronous
- $n$ processes
- Crash failures
- Message passing
- Asynchronous
- $n$ processes
- Crash failures
- Unique ids
Distributed Encoding of the Integers

[Fraigniaud, Rajsbaum, T. LATIN’16]
Counting the Stars
Counting with Distributed Certificates

- $id_1$
- $id_2$
- $id_3$
- $id_4$
- $id_5$

Verify ($id_1$, 01 11 00 01 01) → YES

Verify ($id_3$, 11 00 01 01) → NO
Counting with Distributed Certificates

\textbullet{} \text{verify}(5, 01100010101) \rightarrow \text{YES}

\textbullet{} \text{verify}(3, 110001) \rightarrow \text{NO}
Counting with Distributed Certificates

• \( \text{verify}(5, \begin{array}{c} 01 \ 11 \ 00 \ 01 \ 01 \end{array}) \) \rightarrow \text{YES}
Counting with Distributed Certificates

- $\text{verify}(5, \begin{array}{cccc} 01 & 11 & 00 & 01 \end{array}) \rightarrow \text{YES}
- \text{verify}(3, \begin{array}{ccc} 11 & 00 & 01 \end{array}) \rightarrow \text{NO}$
Distributed Encoding of the Integers

- $\mathcal{A}$ alphabet

$f : \mathcal{A} \rightarrow \{\text{YES}, \text{NO}\}$ such that for each $n \in \mathbb{N}$ there exists a code $c_n \in \mathcal{A}$:

1. $f(c_n) = \text{YES}$
2. For every sub-word $c'_n$ of $c_n$, $f(c'_n) = \text{NO}$
• \( \mathcal{A} \) alphabet
• \( f : \mathcal{A}^* \rightarrow \{\text{YES, NO}\} \)
• \( A \) alphabet

• \( f : A^* \rightarrow \{\text{YES, NO}\} \)

such that for each \( n \in \mathbb{N} \)
there exists a \underline{code of} \( n \) \( c_n \in A^n : \)

\[ f(c_n) = \text{YES} \quad \text{and} \quad \]
• $A$ alphabet
• $f : A^* \rightarrow \{\text{YES, NO}\}$

such that for each $n \in \mathbb{N}$ there exists a code of $n$ $c_n \in A^n$:

1. $f(c_n) = \text{YES}$ and
2. For every sub-word $c'$ of $c_n$, $f(c') = \text{NO}$
Simple Distributed Encoding

distributed code of \( n \)

\[
C(n) = n, n, \ldots, n
\]

\[\text{n times}\]
Simple Distributed Encoding

distributed code of $n$

$$C(n) = \underbrace{n, n, \ldots, n}_{n \text{ times}}$$

$$f(x_1, \ldots, x_\ell) = \text{YES} \iff x_1 = x_2 = \ldots = x_\ell = \ell$$
distributed code of \( n \)

\[
C(n) = n, n, \ldots, n
\]

n times

\[
f(x_1, \ldots, x_\ell) = \text{YES} \iff x_1 = x_2 = \ldots = x_\ell = \ell
\]

Alphabet of \( N \) symbols to encode the first \( N \) integers
Simple Distributed Encoding

distributed code of \( n \) 

\[ C(n) = n, n, \ldots, n \]

\( n \) times

\[ f(x_1, \ldots, x_\ell) = \text{YES} \iff x_1 = x_2 = \ldots = x_\ell = \ell \]

Alphabet of \( N \) symbols to encode the first \( N \) integers

**Challenge:** Compact encoding
0000 code of 4
0000  code of 4
00110  code of 5
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>code of 4</td>
</tr>
<tr>
<td>00110</td>
<td>code of 5</td>
</tr>
<tr>
<td>011010</td>
<td>code of 6</td>
</tr>
</tbody>
</table>
0000  code of 4
00110  code of 5
011010 code of 6
1101010
10101011
010101111
1111110010
11111001011
11100101111
111001011111
:       :
:       :
11111111111111111111111111111111
code of \(2^{257} - 2\)
Diagonal Sequence

0000 code of 4
00110 code of 5
011010 code of 6
1101010
10101011
010101111
1111110010
11111001011
11110010111
111001011111
::
111111111111 code of $2^{257} - 2$
Diagonal Sequence

0000  code of 4
00110  code of 5
011010  code of 6
1101010
10101011
010101111
111110010
11111001011
111100101111
111001011111

... ...

1111111111111111   1   code of $2^{257} - 2$

H(4)-1
Aside: Well Quasi-order

Let $w, w' \in \{0, 1\}^*$

$w \preceq \star w' \iff w$ is a sub-word of $w'$

$w \quad w' \quad \begin{array}{c} 1010 \\ 00111011110 \end{array}$
Aside: Well Quasi-order

Let \( w, w' \in \{0, 1\}^* \)

\[ w \preceq_\ast w' \iff w \text{ is a sub-word of } w' \]

\[ \begin{array}{c}
w & 1010 \\
w' & 001110111110 \\
\end{array} \]

Bad Sequence

A sequence \( w_1, w_2, \ldots, w_\ell \) of words of \( \{0, 1\}^* \) is \textcolor{red}{bad} iff for every \( i < j \), \( w_i \not\preceq_\ast w_j \).
Higman’s lemma

$\langle \{0, 1\}^*, \preceq \rangle$ is a well-quasi order
Higman’s lemma

\((\{0, 1\}^*, \leq_{\ast})\) is a well-quasi order

That is, every bad sequence over \(\{0, 1\}^*\) is finite
Higman’s lemma

$(\{0, 1\}^*, \preceq_*)$ is a well-quasi order

That is, every bad sequence over $\{0, 1\}^*$ is **finite**

Length Function Theorem [Schmitz et al., ICALP’11]

Bad sequences $w_1, w_2, \ldots, w_\ell$ over $\{0, 1\}^*$ with

- $|w_1| \leq d$
- $|w_i| \leq i$

have length bounded by $L(d)$ where $L$ is a function of

*Ackermannian growth*
<table>
<thead>
<tr>
<th>Sequence</th>
<th>Not a sub-word</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>00110</td>
<td></td>
</tr>
<tr>
<td>011010</td>
<td></td>
</tr>
<tr>
<td>110100</td>
<td></td>
</tr>
<tr>
<td>1010111</td>
<td></td>
</tr>
<tr>
<td><strong>1010111</strong></td>
<td></td>
</tr>
<tr>
<td>010101111</td>
<td></td>
</tr>
<tr>
<td>1111110010</td>
<td></td>
</tr>
<tr>
<td>111110010111</td>
<td></td>
</tr>
<tr>
<td>111100101111</td>
<td></td>
</tr>
<tr>
<td><strong>111001011111</strong></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td><strong>111111111111</strong></td>
<td><strong>1</strong></td>
</tr>
</tbody>
</table>
Multi diagonal sequence

$D_1$

$D_2$

$D_3$
Multi diagonal sequence

$D_1^{H(1)}$

$D_2$

$D_3$

$H(H(1))$

$H(H(H(1)))$
Multi diagonal sequence

- \( D_1 H(1) \)
- \( D_2 \)
- \( H(H(1)) \)

Encode integers \( H(1) \ldots H(H(1)) - 1 \)
Multi diagonal sequence

d\_1 \ H(1)

D_2

\ H(H(1))

D_3

\ H(H(H(1)))

encode integers \( H(1) \ldots H(H(1)) - 1 \)

encode integers \( H(H(1)) \ldots H(H(H(1))) - 1 \)
Encoding from Multi Diagonal Sequence

\[ A = \{0, 1\} \times \mathbb{N} \]

\[ \text{Code}(n) : \]

\[ 010\ldots0101 \]

\[ D_i \]

\[ n \]
Encoding from Multi Diagonal Sequence

\[ A = \{0, 1\} \times \mathbb{N} \]

\[ \text{Code}(n) : (0, i), (1, i), (0, i), \ldots, (1, i), (0, i), (1, i) \]
Encoding from Multi Diagonal Sequence

\[ A = \{0, 1\} \times \mathbb{N} \]

\[ \text{Code}(n) : (0, i), (1, i), (0, i), \ldots, (1, i), (0, i), (1, i) \]

\[
\begin{array}{c}
D_i \\
n \\
010\ldots0101
\end{array}
\]

\[ f((b_1, d_1), (b_2, d_2), \ldots, (b_n, d_n)) = \text{YES} \iff \]

1. \[ d_1 = d_2 = \ldots = d_n = i \]
2. \[ b_1, \ldots, b_n \text{ is the sequence of length } n \text{ in } D_i \]
How many bits to distributively encode the first $n$ integers?

$$Code(n) : (0, i), (1, i), (0, i), \ldots, (1, i), (0, i), (1, i)$$
How many bits to distributively encode the first $n$ integers?

$Code(n) : (0, i), (1, i), (0, i), \ldots, (1, i), (0, i), (1, i)$

$\implies 1 + \log(i)$ bits
How many bits to distributively encode the first $n$ integers?

$$Code(n) : (0, i), (1, i), (0, i), \ldots, (1, i), (0, i), (1, i)$$

$\implies 1 + \log(i)$ bits

where $i = \min\{j : n < H^j(1)\} \leq \text{Ack}^{-1}(n)$
Perfect Failure Detection from Distributed Encoding
Perfect failure detection from distributed encoding

Failure detector $\mu P$:
- Encode an upper bound on the number of alive processes
- Eventually converge to the (code of the) number of non-faulty processes
Due to space considerations, the diagram is not included here. However, the textual content is as follows:

- Constant fd output at each proc in each epoch
- $w_1 w_2 w_4 w_5 \leq \text{code}(a_i)$, where $\# \text{alive}(\text{epoch } i) \leq a_i$
At most $n$ epochs

$a_1 \geq a_2 \geq \ldots \geq a_\ell$
• At most $n$ epochs
• $a_1 \geq a_2 \geq \ldots \geq a_\ell$
• $a_\ell = \# \text{ alive(last epoch)} = \# \text{ correct procs}$
Let $Q = \{q_1, \ldots, q_4\} \subseteq \{p_1, \ldots, p_n\}$

Recall: $w \preceq^* \text{code}(a_i)$ and $|\text{Alive}(\text{epoch } i)| \leq a_i$
Let \( Q = \{q_1, \ldots, q_4\} \subseteq \{p_1, \ldots, p_n\} \)

- Recall: \( w \preceq_* \text{code}(a_i) \) and \( |\text{Alive}(\text{epoch } i)| \leq a_i \).
- Code def.: \( w \preceq_* \text{code}(a_i) \implies f(w) = false \) and \( f(\text{code}(a_i)) = true \).
Let $Q = \{q_1, \ldots, q_4\} \subseteq \{p_1, \ldots, p_n\}$

Recall: $w \preceq code(a_i)$ and $|Alive(\text{epoch } i)| \leq a_i$

Code def.: $w \prec code(a_i) \implies f(w) = false$ and $f(code(a_i)) = true$

Hence, if $f(w) = true$ then $\{p_1, \ldots, p_n\} \setminus Q \subseteq \text{Faulty}$
Let \( Q = \{q_1, \ldots, q_4\} \subseteq \{p_1, \ldots, p_n\} \)

- \( w_1, w_2, w_3, w_4 \) sampled in \( \neq \) epochs
Let $Q = \{q_1, \ldots, q_4\} \subseteq \{p_1, \ldots, p_n\}$

- $w_1$, $w_2$, $w_3$, $w_4$ sampled in $\neq$ epochs
- $f(w) = true$ ?? $f(w) = false$ ??
At most \( n \) epochs
At most \( n \) epochs

\[ \Rightarrow \]

In a sequence of \( n \) collects, at least one is **clean**
• Collect $i$ is *successful* if (1) terminates and (2) $f(w_i) = true$
• Collect $i$ is *successful* if (1) terminates and (2) $f(w_i) = true$

• If for some set $Q$, there are $n$ *successful* collects

$P$ output $= \{p_1, \ldots, p_n\} \setminus Q$
Failure detector $\mu P$

- Outputs $O(\log \text{Ack}^{-1}(n))$ bits per processes
- Can emulate the perfect failure detector $P$
Failure detector $\mu P$

- Outputs $O(\log \text{Ack}^{-1}(n))$ bits per processes
- Can **emulate** the perfect failure detector $P$
- ($P$ can also emulate $\mu P$ – see the paper)
Failure detector $\mu P$

- Outputs $O(\log \text{Ack}^{-1}(n))$ bits per processes
Failure detector $\mu P$

- Outputs $O(\log \text{Ack}^{-1}(n))$ bits per processes

Is there a f.d. $D$ that

1. can emulate $P$
2. outputs less than $\log \text{Ack}^{-1}(n)$ bits per process?
Failure detector $\mu P$

- Outputs $O(\log \text{Ack}^{-1}(n))$ bits per processes

Is there a f.d. $D$ that

1. can emulate $P$
2. outputs less than $\log \text{Ack}^{-1}(n)$ bits per process?

**Theorem**

*No failure detector with constant-size output can emulate* $P$
Assume for contradiction $D$ f.d. such that
- Constant range $R$, (independant of $n$)
Assume for contradiction $D$ f.d. such that

- Constant range $R$, (independant of $n$)
- $T_{D \rightarrow P}$ (can emulate $P$)
Assume for contradiction $D$ f.d. such that
- Constant range $R$, (independant of $n$)
- $T_{D\rightarrow P}$ (can emulate $P$)

Ingredients
- Ramsey’s theorem
- Well quasi-order theory
Construct two executions $e$ and $e'$:

- *indistinguishable* for some non-faulty processes
- with $\text{Correct}(e) \subsetneq \text{Correct}(e')$
Construct two executions $e$ and $e'$:

- *indistinguishable* for some non-faulty processes
- with $\text{Correct}(e) \subsetneq \text{Correct}(e')$

$\implies$

in $e'$ $T_{D \rightarrow P}$ erroneously outputs a non-faulty process
Let $e$ an (infinite) execution

$q \quad d_1 \quad d_2 \quad d_3 \quad d_4 \quad d_5 \quad d_6 \quad d_7$

- As $R_D$ is finite, $\exists d \in D$ output infinitely many times at $q$
Let $e$ an (infinite) execution

$q \xrightarrow{d} d_2 \xrightarrow{d_3} d \xrightarrow{d_5} d \xrightarrow{d_7}$

- As $R_D$ is finite, $\exists d \in D$ output infinitely many times at $q$
Let \( e \) an (infinite) execution

\[
\begin{array}{cccccccc}
q & d & d_2 & d_3 & d & d_5 & d & d_7 \\
\end{array}
\]

- As \( R_D \) is finite, \( \exists d \in D \) output infinitely many times at \( q \)

Execution \( \tilde{e} \)

\[
\begin{array}{cccc}
q & d & d & d \\
\end{array}
\]

- Constant failure detector output
Let $e$ an (infinite) execution in which crashes are initial

$q_1 \rightarrow \cdots \rightarrow q_5 \in R^*$

Constant f.d. output ($d_i$) at each non-faulty process $q_i$
Let $e$ an (infinite) execution in which crashes are initial.

Constant f.d. output ($d_i$) at each non-faulty process $q_i$. 
Let e an (infinite) execution in which crashes are initial

\[ q_1 \quad d_1 \quad d_1 \quad q_2 \quad \times \quad q_3 \quad d_3 \quad d_3 \quad q_4 \quad \times \quad q_5 \quad d_5 \quad d_5 \quad d_5 \]

Constant f.d. output (\( d_i \)) at each non-faulty process \( q_i \)

\[ e \rightarrow w_e = d_1 d_3 d_5 \in R_D^* \]
Towards Indistinguishable Executions

<table>
<thead>
<tr>
<th>execution</th>
<th>associated word $\in R_D^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_1$</td>
<td>$w_1$</td>
</tr>
<tr>
<td>$e_2$</td>
<td>$w_2$</td>
</tr>
<tr>
<td>$e_3$</td>
<td>$w_3$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$e_L$</td>
<td>$w_L$</td>
</tr>
</tbody>
</table>

$|w_i| = i$
Towards Indistinguishable Executions

<table>
<thead>
<tr>
<th>execution</th>
<th>associated word $\in R^*_D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_1$</td>
<td>$W_1$</td>
</tr>
<tr>
<td>$e_2$</td>
<td>$W_2$</td>
</tr>
<tr>
<td>$e_3$</td>
<td>$W_3$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$e_L$</td>
<td>$W_L$</td>
</tr>
</tbody>
</table>

$|w_i| = i$

- (Higman’s Lemma) $(R^*_D \preceq_*)$ is a wqo
**Towards Indistinguishable Executions**

<table>
<thead>
<tr>
<th>execution</th>
<th>associated word $\in R_D^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_1$</td>
<td>$w_1$</td>
</tr>
<tr>
<td>$e_2$</td>
<td>$w_2$</td>
</tr>
<tr>
<td>$e_3$</td>
<td>$w_3$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$e_L$</td>
<td>$w_L$</td>
</tr>
</tbody>
</table>

| $|w_i| = i$ |

- (Higman’s Lemma) $(R_D^* \preceq_*)$ is a wqo

$\implies$ For large enough $L$,

$\exists \ i, j : 1 \leq i < j \leq L$ and $w_i$ subword of $w_j$
Towards Indistinguishable Executions

execution $e$

$w = abc$

$q_1$:

\begin{itemize}
  \item a
  \item a
  \item a
  \item a
\end{itemize}

$q_2$:

\begin{itemize}
  \item b
  \item b
  \item b
  \item b
\end{itemize}

$q_3$:

\begin{itemize}
  \item c
  \item c
  \item c
  \item c
\end{itemize}

execution $e'$

$w' = xaby\text{c}$

$q'_1$:

\begin{itemize}
  \item x
  \item x
  \item x
  \item x
\end{itemize}

$q'_2$:

\begin{itemize}
  \item a
  \item a
  \item a
  \item a
\end{itemize}

$q'_3$:

\begin{itemize}
  \item b
  \item b
  \item b
  \item b
\end{itemize}

$q'_4$:

\begin{itemize}
  \item y
  \item y
  \item y
  \item y
\end{itemize}

$q'_5$:

\begin{itemize}
  \item c
  \item c
  \item c
  \item c
\end{itemize}

a (or b, c) may be output at processes with distinct ids in $e$ and $e'$. Rely on Ramsey's Theorem to get rid of ids.
Towards Indistinguishable Executions

Execution $e$

$w = abc$

$q_1$: a a a a a
$q_2$: b b b b b
$q_3$: c c c c c

Execution $e'$

$w' = xaby c$

$q'_1$: x x x x x
$q'_2$: a a a a a
$q'_3$: b b b b b
$q'_4$: y y y y y
$q'_5$: c c c c c

a (or b, c) may be output at processes with distinct ids in $e$ and $e'$. 

Rely on Ramsey's Theorem to get rid of ids.
Towards Indistinguishable Executions

Execution $e$

\[ w = abc \]

Processes:

- $q_1$: a a a a a
- $q_2$: b b b b b
- $q_3$: c c c c c

Execution $e'$

\[ w' = xabyc \]

Processes:

- $q_1'$: x x x x x
- $q_2'$: a a a a a
- $q_3'$: b b b b b
- $q_4'$: y y y y y
- $q_5'$: c c c c c

A (or b, c) may be output at processes with distinct ids in $e$ and $e'$. Rely on Ramsey’s Theorem to get rid of ids.
Summary:

- Perfect failure detection with $O(Ack^{-1}(n))$ bits per process
- Perfect failure detection with constant output is impossible
Summary:

- Perfect failure detection with $O(Ack^{-1}(n))$ bits per process
- Perfect failure detection with constant output is impossible
- Applications of wqo theory to distributed computing
Summary:

- Perfect failure detection with $O(Ack^{-1}(n))$ bits per process
- Perfect failure detection with constant output is impossible
- Applications of wqo theory to distributed computing

Future work:
Summary:

- Perfect failure detection with $O(Ack^{-1}(n))$ bits per process
- Perfect failure detection with constant output is impossible
- Applications of wqo theory to distributed computing

Future work:

- Close the gap between lower and upper bounds
Summary:

- Perfect failure detection with $O(Ack^{-1}(n))$ bits per process
- Perfect failure detection with constant output is impossible
- Applications of wqo theory to distributed computing

Future work:

- Close the gap between lower and upper bounds
- Failure detector as (distributed) encoder: Relation between output size and failure detector power
Summary:

- Perfect failure detection with $O(Ack^{-1}(n))$ bits per process
- Perfect failure detection with constant output is impossible
- Applications of wqo theory to distributed computing

Future work:

- Close the gap between lower and upper bounds
- Failure detector as (distributed) encoder: Relation between output size and failure detector power
- Other application of the distributed encoding of the integers
Thanks!
\( \mathbf{c} \) assigns a color to each subset of processes
\[
\mathbf{c}(Q = \{q_1, \ldots, q_k\}) \in R_D^k
\]

\( q_1 \)
\( q_2 \)
\( q_3 \)
\( p \neq q_i \)
$c$ assigns a color to each subset of processes

$c(Q = \{ q_1, \ldots, q_k \}) \in R_D^k$
$c$ assigns a color to each subset of processes

$c(Q = \{q_1, \ldots, q_k\}) \in R_D^k$

\[
c(\{q_1, \ldots, q_3\}) = (d_1, d_2, d_3)
\]
\[ c : Q = \{ q_1, \ldots, q_k \} \rightarrow R_D^k \]
\( c : Q = \{q_1, \ldots, q_k\} \rightarrow R_D^k \)

**Ramsey’s Theorem**

- For any \( m, k \)
c : Q = \{ q_1, \ldots, q_k \} \rightarrow R_D^k

Ramsey’s Theorem

- For any \( m, k \)
- There exists \( n = g(m, k) \) such that
\begin{align*}
  \mathbf{c} : Q = \{ q_1, \ldots, q_k \} &\to R^k_D \\
\end{align*}

Ramsey’s Theorem

- For any \( m, k \)
- There exists \( n = g(m, k) \) such that
- There exists a \( m \)-subset \( S \) of the \( n \) procs such that
\[ \mathbf{c} : Q = \{q_1, \ldots, q_k\} \rightarrow R_D^k \]

**Ramsey’s Theorem**

- For any \( m, k \)
- There exists \( n = g(m, k) \) such that
- There exists a \( m \)-subset \( S \) of the \( n \) procs such that
- Every \( k \)-subset of \( S \) has the same color
\[ c : Q = \{ q_1, \ldots, q_k \} \to R^k_D \]

Ramsey’s Theorem

- For any \( m, k \)
- There exists \( n = g(m, k) \) such that
- There exists a \( m \)-subset \( S \) of the \( n \) procs such that
- Every \( k \)-subset of \( S \) has the same color

Intuitively

For any \( q \in S \),

- For failure pattern with \( k \) correct procs and initial crashes
- F.d output at \( q \) depends only on the rank of its id
A failure detector $D$

- Outputs symbols in some range $R_D$
- Is defined with respect to failure patterns
Failure Pattern

$p_1 \rightarrow \{p_1, \ldots, p_n\} \rightarrow p_2 \rightarrow \{p_2, \ldots, p_n\} \rightarrow p_3 \rightarrow \{p_3, \ldots, p_n\} \rightarrow p_4 \rightarrow \{p_4, \ldots, p_n\}$
$\mathcal{F} : \mathbb{N} \rightarrow 2^{\{p_1, \ldots, p_n\}}$
For each failure pattern, $D$ defines which outputs are valid

- History: $H(p, t)$ is the output at process $p$ at time $t$
- $D(\mathcal{F}) = \text{valid histories for failure pattern } \mathcal{F}$
Failure detectors $D$ and $D'$ are equivalent \iff 

There exist two asynchronous, crash-resilient protocols

- $T_{D \rightarrow D'}$ that emulates $D'$ using $D$
- $T_{D' \rightarrow D}$ that emulates $D$ using $D'$