In the Beginning …

a computer was just a Turing machine …
Today

Computing is co-ordination and communication
No, distributed computations are static mathematical objects!

Distributed computations unfold in time!

Operational versus combinatorial approaches ...
There are Many Models

Communication?
There are Many Models

Communication?

Failures?
There are Many Models

Communication?

Timing?

Failures?
Local Views

Each process has a 3-bit *local view*
Multiple Local Views

local views differ by 1 bit
Multiple Local Views

local views differ by 1 bit
but no process knows which one
Multiple Local Views

Each view is represented by a labeled vertex.
Global States

compatible views represented by an edge
All possible global states

(where blue has 111)
Communication

Each process sends local view to the other
Communication

Each process sends local view to the other but at most one message may be lost!
One Communication Round

110
?

111
110

110
111

111
?
One Communication Round

\[ 110 \quad ? \]
\[ 111 \quad 110 \]
\[ 110 \quad 111 \]
\[ 111 \quad ? \]

1 Lost
One Communication Round

110
?

111
110

110
111

111
?

1 Lost

none Lost

Distributed Computing though
Combinatorial Topology
One Communication Round

110 ?

111 110

110 111

111 ?

1 Lost

none Lost

1 Lost
One Communication Round

![Diagram showing a network with nodes labeled with binary values.]

Distributed Computing though Combinatorial Topology
All possible global states after one round unreliable communication
Informally ....

Unreliable communication does not change "topology" of global states
Reliable Communication?

![Diagram showing nodes with binary values and question marks.](https://via.placeholder.com/150)
Reliable Communication?
A Classic Distributed Problem

Muddy Children

Distributed Computing through Combinatorial Topology
Muddy Children
Muddy Children

At least one of you is dirty!

Combinatorial Topology
Muddy Children

You may not communicate!

Combinatorial Topology
When you realize you are dirty, confess on the hour!
Muddy Children

(silence ...)

1:00
Muddy Children

Distributed Computing through
Combinatorial Topology
Operational Explanation
Others are clean, so I must be dirty.
Operational Explanation

Others are clean, so I must be dirty.

Me!
Operational Explanation
He was quiet, so I must be dirty.

Distributed Computing though
Combinatorial Topology
Combinatorial Explanation
Each process has its own input
Each process has its own input.
Combinatorial Explanation

Global State

12:00
all dirty
all dirty

Distributed Computing though Combinatorial Topology
all dirty

Distributed Computing though Combinatorial Topology
Informal Task Definition

Processes start with input values …

They communicate …

They halt with output values …

legal for those inputs.
Consensus: Each Thread has a Private Input
They Communicate
They Agree on One Thread’s Input
Colorless Tasks

The set of input values …

determines the set of output values.

Number and identities irrelevant…

for both input and output values
Road Map

Operational Model

Combinatorial Model

Main Theorem
Processes

A process is a state machine

Could be Turing machines or more powerful
Processes

A process’s state is called its view

Process names taken from a domain $\Pi$

Each process has a unique name (color)

$P_i \in \Pi$
Processes

Each process “knows” its own name

But not the names of the other processes
Processes

Often, $P_i$ is just $i$

Sometimes $P_i$ and $i$ are distinct, and the process “knows” $P_i$ but not $i$
Processes

There are $n+1$ processes
For now, they communicate via reading & writing shared memory.
Immediate Snapshot

Individual reads & writes are too low-level …

A snapshot = atomic read of all memory

We will use immediate snapshot …
Immediate Snapshot

write view to memory

take snapshot

adjacent steps!
Immediate Snapshot

Concurrent steps

write view to memory

take snapshot

write view to memory

take snapshot
Immediate Snapshot

\begin{verbatim}
immediate
  mem[i] := view;
  snap := snapshot(mem[*])
\end{verbatim}
<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>Q</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>write</td>
<td></td>
<td></td>
</tr>
<tr>
<td>write</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>snap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>snap</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>{p}</td>
<td>{p,q}</td>
<td>{p,q,r}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>Q</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>write</td>
<td></td>
<td></td>
</tr>
<tr>
<td>write</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>snap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>snap</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>{p}</td>
<td>{p,q,r}</td>
<td>{p,q,r}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>Q</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>write</td>
<td></td>
<td></td>
</tr>
<tr>
<td>write</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>snap</td>
<td></td>
<td></td>
</tr>
<tr>
<td>snap</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>{p,q,r}</td>
<td>{p,q,r}</td>
<td>{p,q,r}</td>
</tr>
</tbody>
</table>
Realistic?

**No!**

My laptop reads only a few contiguous memory words at a time

**Yes!**

Simpler lower bounds: if it’s impossible with IS, it’s impossible on your laptop.

Can implement IS from read-write
Crashes

Processes may halt without warning

as many as $n$ out of $n+1$
Asynchronous
Asynchronous Failures

detection impossible
Configurations

\[ C = \{s_0, \ldots, s_n\} \]

set of *simultaneous* process states

*initial* configurations

*final* configurations
Executions

Processes that communicate

\[ C_0, S_0, C_1, S_1, \ldots, S_r, C_{r+1} \]

Initial configuration

Next configuration

Final configuration
triple is a concurrent step

Processes in $S_0$ said to participate in step

Only $P_i \in S_0$ can change between $C_0$ and $C_1$

state change result of communication
Executions

**finite**
\[ C_0, S_0, C_1, S_1, \ldots, S_r, C_{r+1} \]

**infinite**
\[ C_0, S_0, C_1, S_1, \ldots, \]

**partial**
\[ C_0, S_0, C_1, S_1, \ldots, S_r, C_{r+1} \]
Crashes are Implicit

\[ C_0, S_0, C_1, S_1, \ldots, S_r, C_{r+1} \]

If \( P_i \)'s state is not final in the final configuration, then \( P_i \) has crashed.

Crash cannot be detected in finite time.
Colorless Tasks

\((\mathcal{I}, \emptyset, \Delta)\)

(colorless) input assignment

carrier map \(\Delta: \mathcal{I} \rightarrow 2^\emptyset\)

(colorless) output assignment
Example: Binary Consensus

\[ \mathcal{I} = \{\{0\}, \{1\}, \{0,1\}\} \]

- All start with 0
- All start with 1
- Start with both
Example: Binary Consensus

\[ I = \{\{0\}, \{1\}, \{0,1\}\} \]

\[ O = \{\{0\}, \{1\}\} \]

All decide 0

All decide 1
Example: Binary Consensus

\[ \Delta(\{0\}) = \{\{0\}\} \]

All start with 0, all decide 0
Example: Binary Consensus

\[ \Delta(\{0\}) = \{\{0\}\} \]

\[ \Delta(\{1\}) = \{\{1\}\} \]

All start with 1, all decide 1
Example: Binary Consensus

\[ \Delta(\{0\}) = \{\{0\}\} \]
\[ \Delta(\{1\}) = \{\{1\}\} \]
\[ \Delta(\{0,1\}) = \{\{0\}, \{1\}\} \]

with mixed inputs, all decide 0, or all decide 1
Colorless Layered Protocol

shared mem array 0..N-1,0..n of Value
view := input
for l := 0 to N-1 do
  immediate
    mem[l][i] := view;
  snap := snapshot(mem[l][*])
view := set of values in snap
return δ(view)
Colorless Layered Protocol

shared mem array 0..N-1,0..n of Value

view := input
for j := 0 to N-1 do
    immediate
    mem[j][i] := view;
    snap := Snapshot(mem[j][*])
    view := set of values in snap
return δ(view)

2-dimensional memory array

row is clean per-layer memory

column is per-process word
shared mem array 0..N-1,0..n of Value

view := input

for j := 0 to N-1 do

    mem[j][i] := view;

    snap := snapshot(mem[j][*])

    view := set of values in snap

return δ(view)

initial view is input value
shared mem array 0..N-1,0..n of Value
view := input
for l := 0 to N-1 do
  immediate
  mem[j][l] := view;
  snap := snapshot(mem[j][*])
  view := set of values in snap
run for N layers
return δ(view)
shared mem array 0..N-1 0..n of Value

view := input
for j := 0 to N-1 do
    mem[l][i] := view;
    snap := snapshot(mem[l][*])
    view := set of values in snap
return δ(view)
Colorless Layered Protocol

shared mem array 0..N-1,0..n of Value
view := input
for j := 0 to N-1 do
    immediate
    mem[j][i] := view;
    snap := snapshot(mem[j][*])
view := set of values in snap
return δ(view)

new view is set of values seen
shared mem array 0..N-1,0..n of Value
view := input
for j := 0 to N-1 do
  immediate
  mem[j][i] := view;
snap := snapshot(mem[j][*])
view := set of values in snap
return \( \delta(\text{view}) \)
finally apply decision value to final view
Colorless configurations for processes $P, Q, R$, inputs $p, q, r$, final configurations in black.
Road Map

Operational Model

Combinatorial Model

Main Theorem
Vertex = Process State

Process ID (color)

Value (input or output)
Simplex = Global State
Complex = Global States
Input Complex for Binary Consensus

All possible initial states

Processes: red, green, blue

Independently assigned 0 or 1
Output Complex for Binary Consensus

All possible final states

Output values all 0 or all 1

Two disconnected simplexes
Carrier Map for Consensus

All 0 inputs

All 0 outputs
Carrier Map for Consensus

All 1 inputs

All 1 outputs
Carrier Map for Consensus

Mixed 0-1 inputs

All 0 outputs

All 1 outputs
Task Specification

$$(I, O, \Delta)$$

Input complex

Output complex

Carrier map

$\Delta: I \rightarrow 2^O$$
Colorless Tasks

\[(\mathcal{I}, \mathcal{P}, \Xi)\]

(colorless) input complex

(strict carrier map)

\[\Xi: \mathcal{I} \rightarrow 2^P\]

(colorless) protocol complex
Protocol Complex

**Vertex**: process name, view
all values read and written

**Simplex**: compatible set of views
Each execution defines a simplex
Example: Synchronous Message-Passing
Failures: Fail-Stop

Partial broadcast
Single Input: Round Zero

No messages sent

View is input value

Same as input simplex
Round Zero Protocol Complex

- No messages sent
- View is input value
- Same as input complex
Single Input: Round One

Distributed Computing through Combinatorial Topology
no one fails

Distributed Computing through Combinatorial Topology
Single Input: Round One

no one fails

Distributed Computing through Combinatorial Topology
Single Input: Round One

red fails

blue fails

no one fails

green fails
Protocol Complex: Round One
Protocol Complex: Round Two
Protocol Complex Evolution

zero

one

two
Summary

protocol complex

input complex

Δ

output complex

E

δ
Decision Map

Protocol complex

Output complex

Simplicial map, sending simplexes to simplexes

\( \delta \)
Lower Bound Strategy

Find topological “obstruction” to this simplicial map

Protocol complex

Output complex
Consensus Example

Protocol

Subcomplex of all-0 inputs

Must map here

Output

\[ \delta \]
Consensus Example

Protocol

Subcomplex of all-1 inputs

Must map here

Output

Distributed Computing through Combinatorial Topology
Consensus Example

Path from “all-0” to “all-1”

Image under $\delta$ must start here..

and end here

Output

$\delta$
Consensus Example

Distributed Computing through Combinatorial Topology
Consensus Example

Path from “all-0” to “all-1”

But this “hole” is an obstruction

Image under $\delta$ must start here..

and end here

Distributed Computing through Combinatorial Topology
Conjecture

A protocol cannot solve consensus if its complex is *path-connected*. Model-independent!
If Adversary keeps Protocol Complex path-connected …

Forever …

Consensus is *impossible*

For *r* rounds …

*A round-complexity* lower bound

For time *t* …

*A time-complexity* lower bound
Barycentric Subdivision
Barycentric Subdivision

Geometric Definition

barycenter

barycenter

barycenter

bary $\sigma$
Barycentric Subdivision

Each vertex of \(\text{Bary } \sigma\) is a face of \(\sigma\)

Simplex = set of faces ordered by inclusion

\[\text{bary } \sigma\]
Barycentric Agreement
Barycentric Agreement

\[(\mathcal{I}, \text{bary } \mathcal{I}, \text{bary}(\cdot))\]

- arbitrary input complex
- subdivided output complex
- subdivision as carrier map
If There are $n+1$ Processes

$$(I, \text{bary skel}^n I, \text{bary skel}^n I)$$

Inputs only from $n$-skeleton of input complex
Theorem

A one-layer immediate snapshot protocol solves the $n$-process barycentric agreement task $(\mathcal{I}, \text{bary skel}^n \mathcal{I}, \text{bary skel}^n)$

Proof

All input simplices belong to $\text{skel}^n \mathcal{I}$

Immediate snapshot returns

Set of input vertices (face of input simplex)

Faces ordered wrt one another
Barycentric Agreement

Snapshots are ordered

{\{v_0\}} \quad \{v_0, v_1, v_2\} \quad \{v_0, v_2\}

\[
\begin{array}{ccc}
V_0 & V_1 & V_2 \\
\end{array}
\]

Snapshots are ordered
Barycentric Agreement Protocol

Each view is a face of $\sigma$
Barycentric Agreement Protocol

Each view is a face of $\sigma$
Barycentric Agreement Protocol

Each face of $\sigma$ is a vertex of Bary $\sigma$
Ordered faces $\Rightarrow$ simplex of Bary $\sigma$
Iterated Barycentric Agreement

\[ \text{skel}^n I \rightarrow \text{bary}^N \text{skel}^n I \]
Iterated Barycentric Agreement

$$(\mathcal{I}, \text{bary}^N \text{skel}^n \mathcal{I}, \text{bary}^N \text{skel}^n)$$
One-round IS executions
One-Layer Immediate Snapshot Protocol Complex

Distributed Computing through Combinatorial Topology
Compare Views

Operational view

Combinatorial view

Distributed Computing through Combinatorial Topology
Compositions

Given protocols

\[(I, P, \Xi) \quad (I', P', \Xi')\]

where \(P \subseteq I\)

their composition is

\[(I, P'', \Xi'')\]

where \(\Xi'' = \Xi' \circ \Xi\) and \(P'' = \Xi''(I)\)
Theorem

The protocol complex for a single-layer IS protocol \((\mathcal{I}, \mathcal{P}, \mathcal{E})\) is \(\text{Bary} \mathcal{I}\)
Theorem

The protocol complex for an $N$-layer IS protocol $(\mathcal{I}, \mathcal{P}, \Xi)$ is $\text{Bary}^N \mathcal{I}$
Corollary

The protocol complex for an $N$-layer IS protocol $(I, P, E)$ is $n$-connected
Road Map

Operational Model

Combinatorial Model

Main Theorem
Fundamental Theorem

$$(\mathcal{I}, \mathcal{O}, \Delta) \text{ has a wait-free (} n+1 \text{)-process layered IS protocol iff there is a continuous map}$$

$$f: |\text{ske}l^n \mathcal{I}| \rightarrow |\mathcal{O}|...$$

carried by $\Delta$
Lemma

If ...

there is a WF layered protocol for 
\((I, O, \Delta) \) ...

then ...

there is a continuous

\( f: |\text{ske}l^n I| \rightarrow |O| \) carried by \( \Delta \).
Map $\Rightarrow$ Protocol

Hypothesis

Continuous

$Bary^n\mathcal{I}$
Map $\Rightarrow$ Protocol

Bary$^n\mathcal{I}$  \(\phi\)  Simplicial approximation
Map $\Rightarrow$ Protocol

Run barycentric Agreement

Apply $\phi$

QED
There is a WF-RW protocol for \((\mathcal{I}, \mathcal{O}, \Delta)\) if and only if there is a continuous function \(f: |\text{skel}^n I| \to |\mathcal{O}|\) carried by \(\Delta\).
Inductive construction
\[ g_d: \lfloor \text{skel}^d \mathcal{I} \rfloor \rightarrow |\Xi(\mathcal{I})|. \]

Base \( d = 0 \)

Define \( g_0: \lfloor \text{skel}^0 \mathcal{I} \rfloor \rightarrow |\Xi(\mathcal{I})| \) ...

Let \( g_0(v) \) be any vertex in \( \Xi(\{v\}) \)
Protocol $\Rightarrow$ Map

**Induction Hypothesis**

$g_{d-1} : |\text{skel}^{d-1} \mathcal{I}| \rightarrow |\Xi(\mathcal{I})|$

For all $\sigma$ in $\text{skel}^{d-1} \mathcal{I}$

$g_{d-1}$ sends $|\text{skel}^{d-1} \sigma| \rightarrow |\Xi(\sigma)|$

But $\Xi(\sigma)$ is $(d-1)$-connected (earlier theorem)

Can extend to $g_d : |\sigma| \rightarrow |\Xi(\sigma)|$

Yielding $g_d : |\text{skel}^d \mathcal{I}| \rightarrow |\Xi(\mathcal{I})|$
Protocol $\Rightarrow$ Map

**Simplicial decision map**

$g: |\text{skel}^n \mathcal{I}| \rightarrow |\Xi(\mathcal{I})|$

$\delta: \Xi(\text{skel}^n \mathcal{I}) \rightarrow \mathcal{O}$

$|\delta|: |\Xi(\text{skel}^n \mathcal{I})| \rightarrow |\mathcal{O}|$

**Composition**

$f = |\delta| \cdot g$ yields $f: |\text{skel}^n \mathcal{I}| \rightarrow |\mathcal{O}|$ carried by $\Delta$

QED
Combinatorial Reasoning

http://commons.wikimedia.org/wiki/File:Blake_ancient_of_days.jpg
Combinatorial Reasoning

Model-independent properties …
Combinatorial Reasoning

Model-independent properties …

… restricted model-dependent reasoning
This work is licensed under a Creative Commons Attribution-ShareAlike 2.5 License.

• You are free:
  – to Share — to copy, distribute and transmit the work
  – to Remix — to adapt the work
• Under the following conditions:
  – Attribution. You must attribute the work to “Distributed Computing Through Combinatorial Topology” (but not in any way that suggests that the authors endorse you or your use of the work).
  – Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.
• For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to
  – http://creativecommons.org/licenses/by-sa/3.0/.
• Any of the above conditions can be waived if you get permission from the copyright holder.
• Nothing in this license impairs or restricts the author's moral rights.