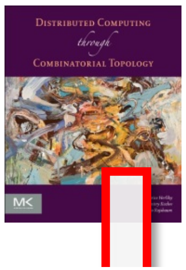
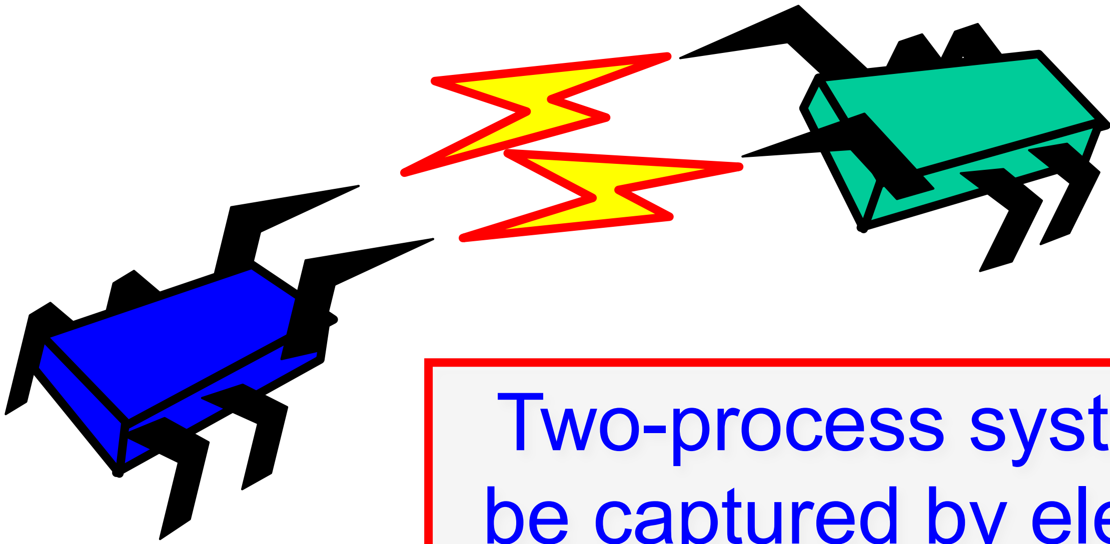
Two-process systems

MITRO207, P4, 2019

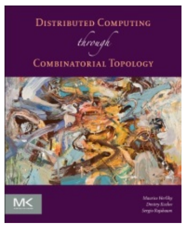


Two-Process Systems



Two-process systems can be captured by elementary ***graph theory***

gentle introduction to more general structures needed later for larger systems



Road Map

Elementary Graph Theory

Tasks

Models of Computation

Approximate Agreement

Task Solvability

Road Map

Elementary Graph Theory

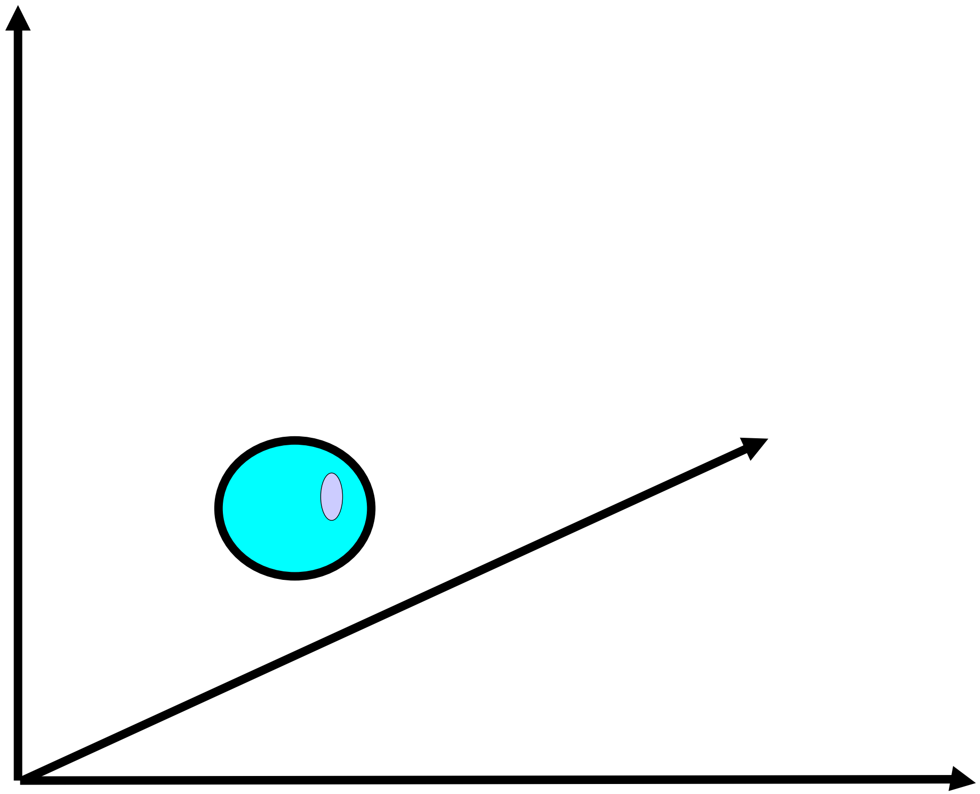
Tasks

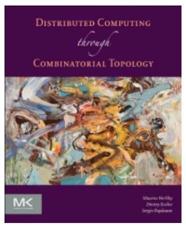
Models of Computation

Approximate Agreement

Task Solvability

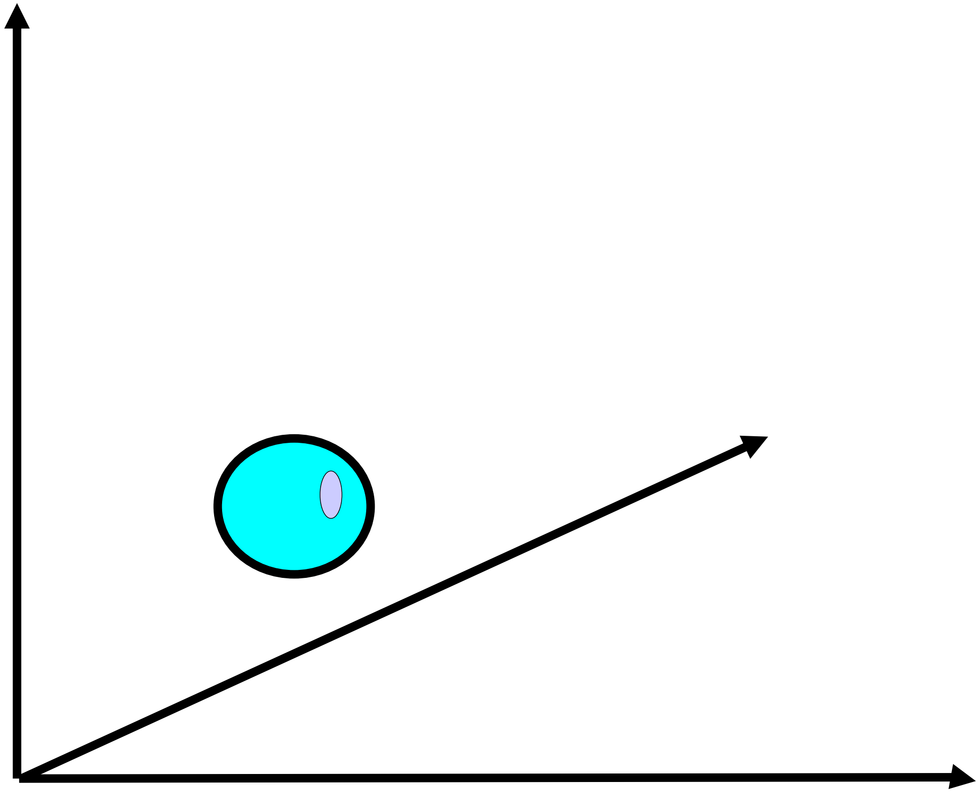
A Vertex

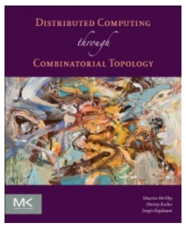




A Vertex

Combinatorial: an element of a set.

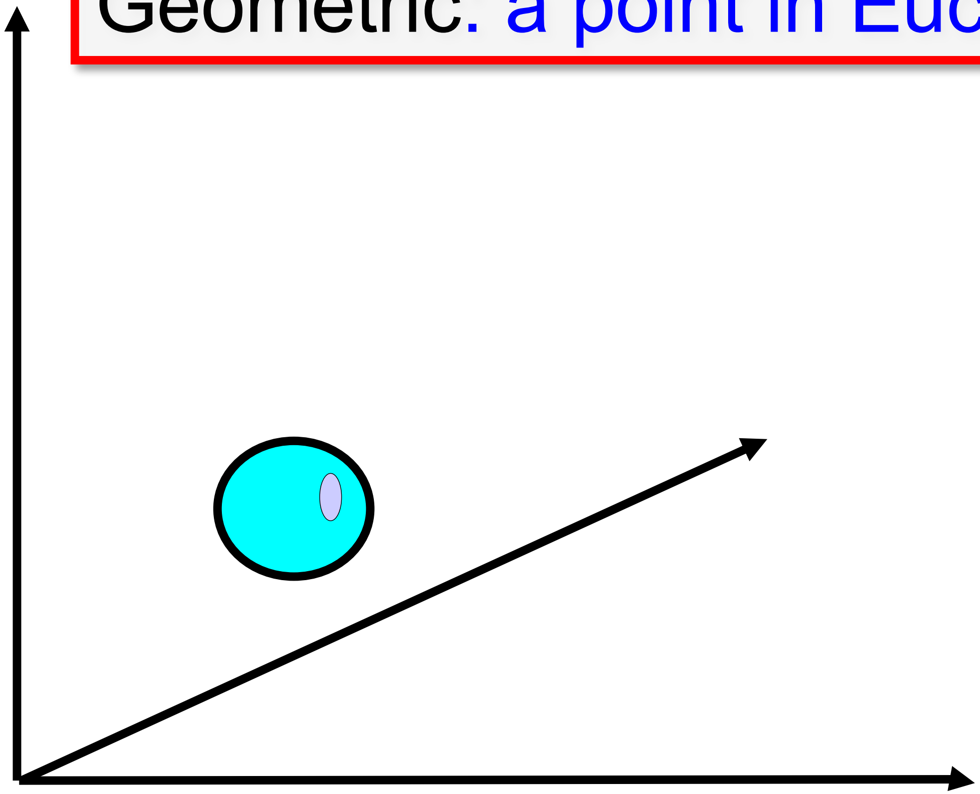


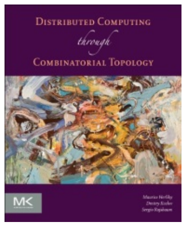


A Vertex

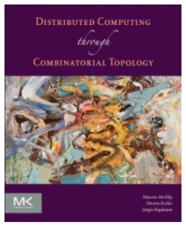
Combinatorial: an element of a set

Geometric: a point in Euclidean Space



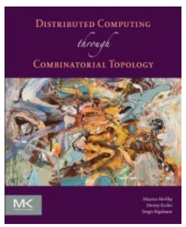


An Edge



An Edge

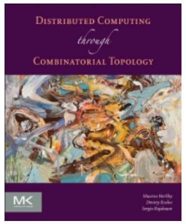
Combinatorial: a set of two vertexes.



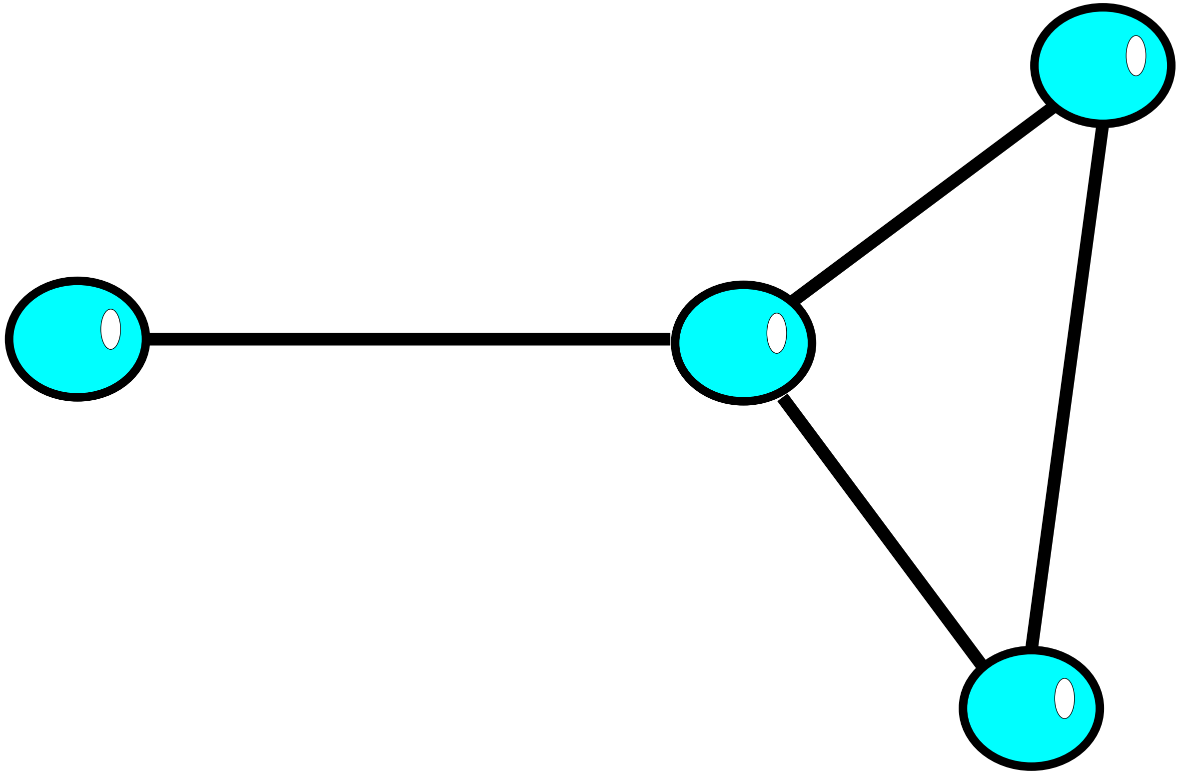
An Edge

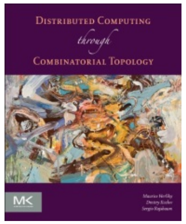
Combinatorial: a set of two vertexes

Geometric: line segment joining two points



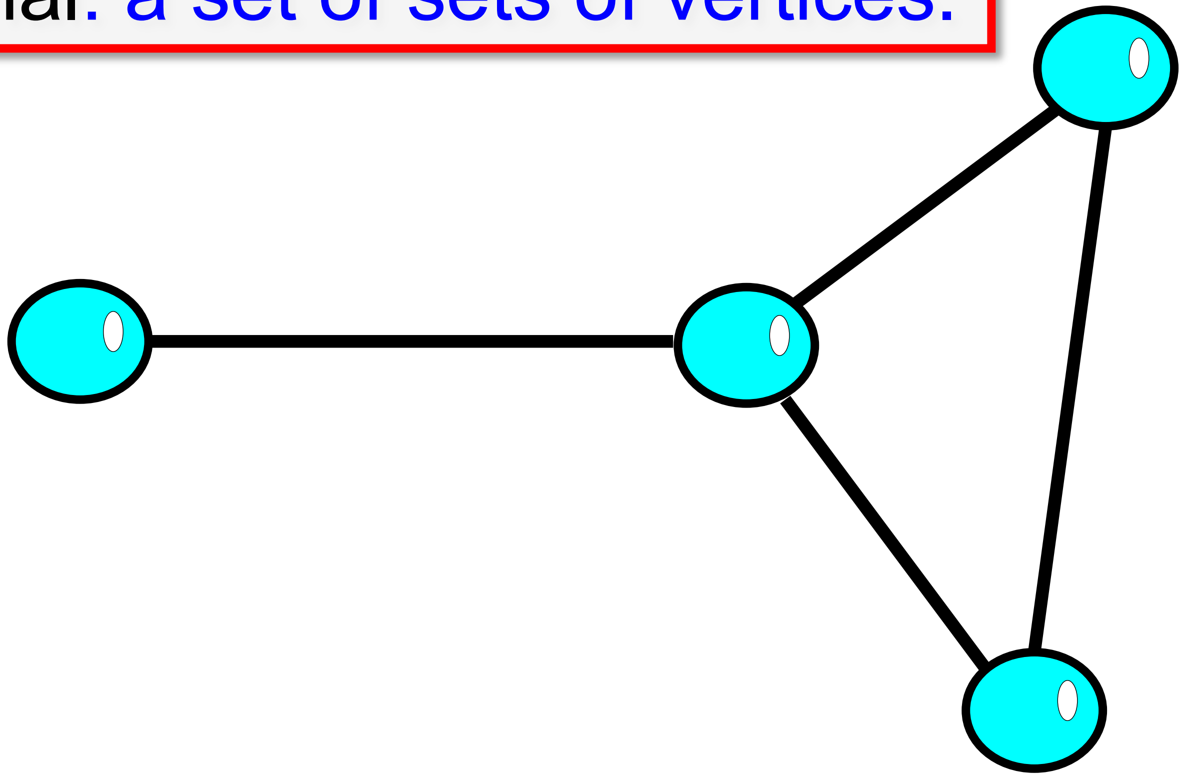
A Graph

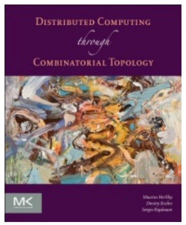



Distributed Computing
through
Combinatorial Topology
MK

A Graph

Combinatorial: a set of sets of vertices.

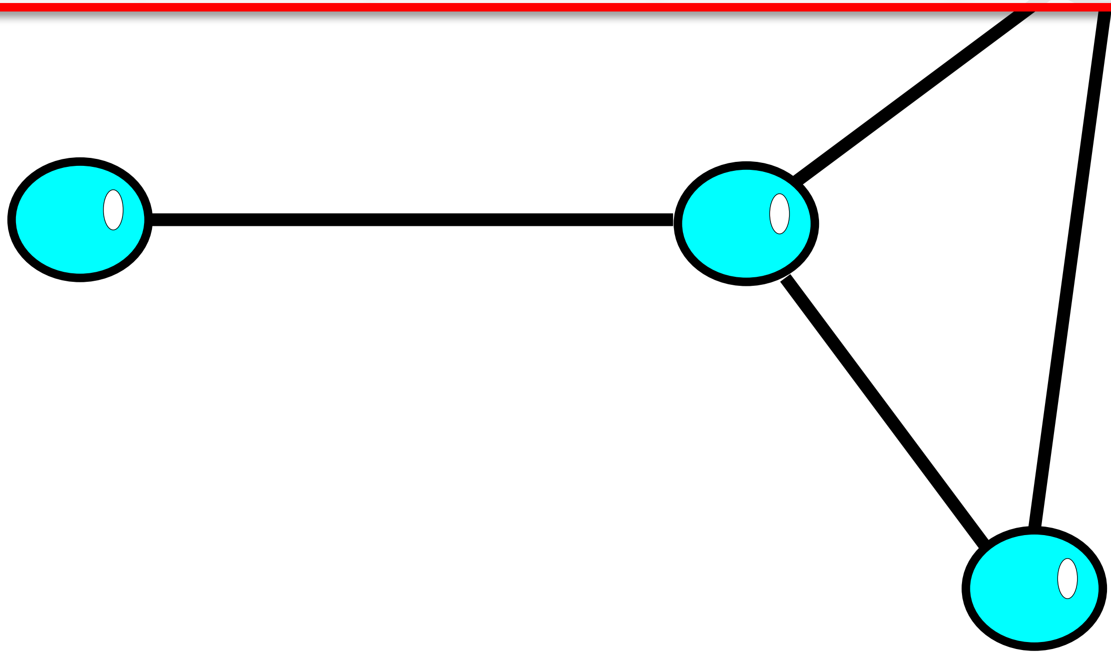


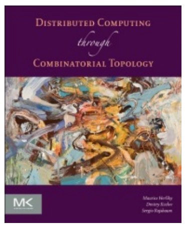


A Graph

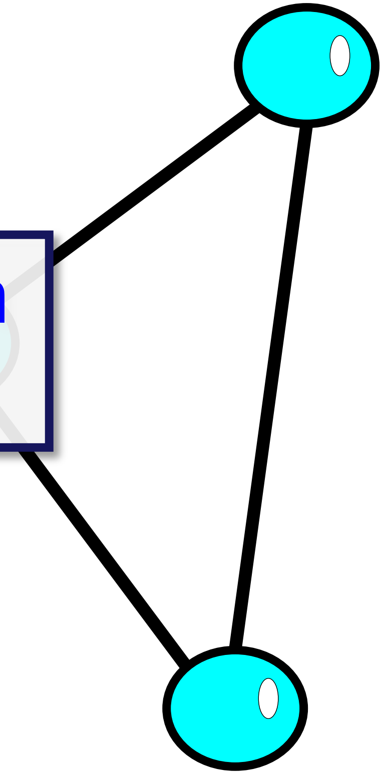
Combinatorial: a set of sets of vertices

Geometric: points joined by line segments

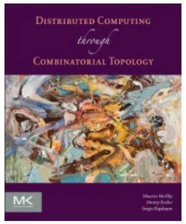




Graphs



finite set V with a collection $\mathcal{G}$ of subsets of V,

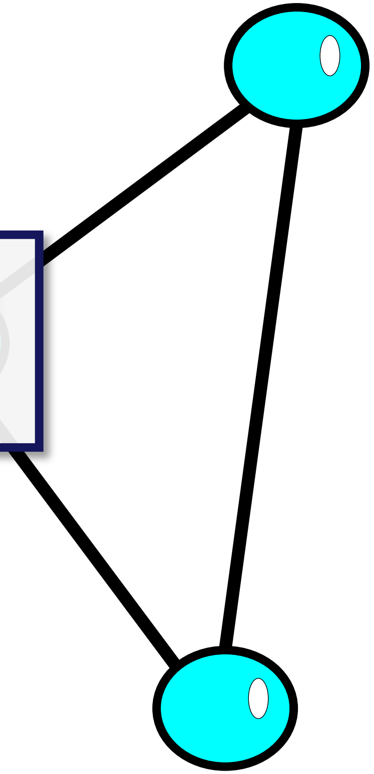


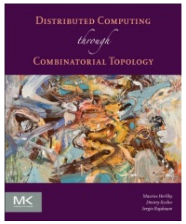
Graphs

vertices

finite set V with a collection $\mathcal{G}$ of subsets of V,

simplices
(singular: simplex)

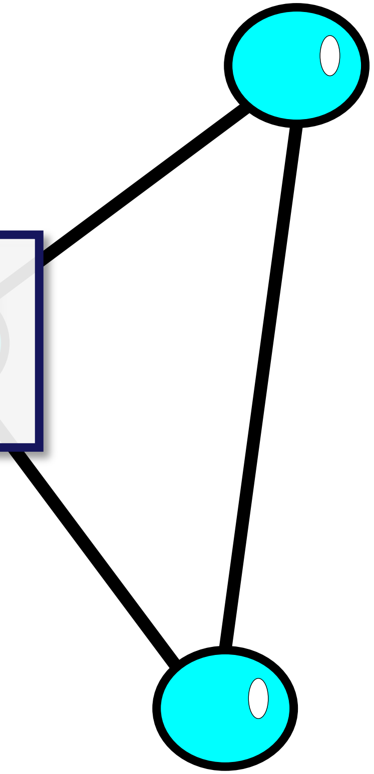


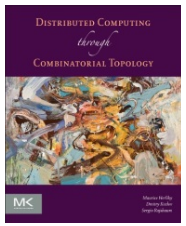


Graphs

finite set V with a collection $\mathcal{G}$ of subsets of V,

(1) If $X \in \mathcal{G}$, then $|X| \leq 2$



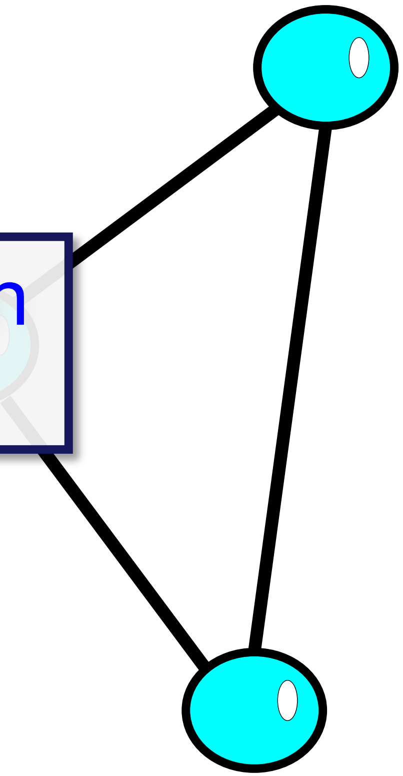


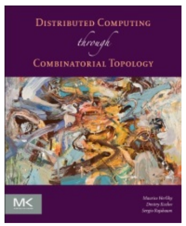
Graphs

finite set V with a collection $\mathcal{G}$ of subsets of V,

(1) If $X \in \mathcal{G}$, then $|X| \leq 2$

vertex: $|X| = 1$
edge: $|X| = 2$



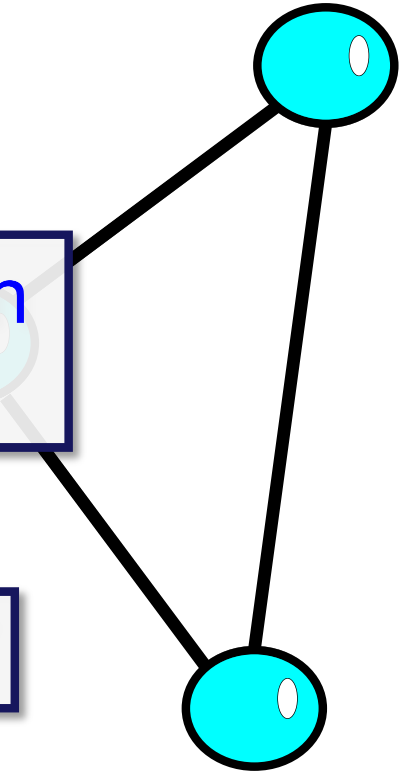


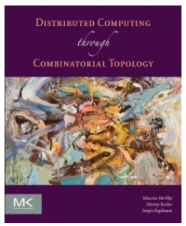
Graphs

finite set V with a collection $\mathcal{G}$ of subsets of V,

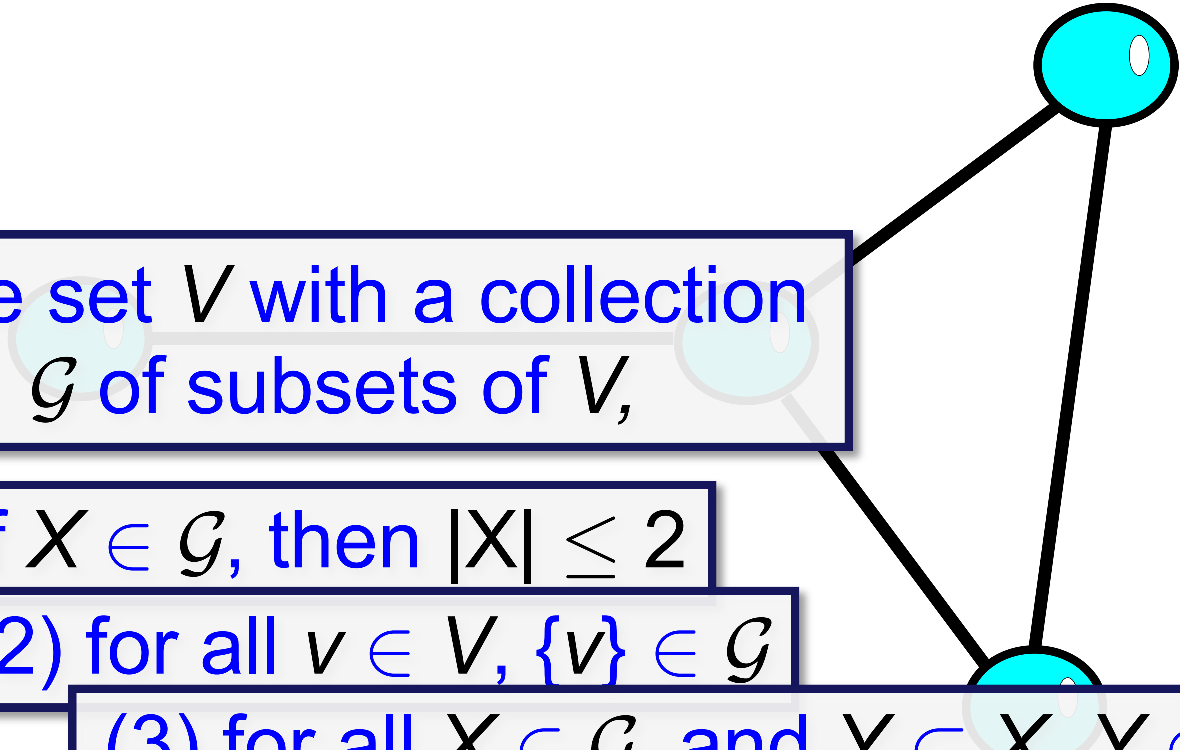
(1) If $X \in \mathcal{G}$, then $|X| \leq 2$

(2) for all $v \in V$, $\{v\} \in \mathcal{G}$





Graphs

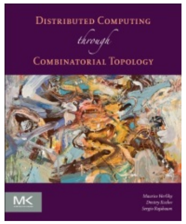


finite set V with a collection $\mathcal{G}$ of subsets of V,

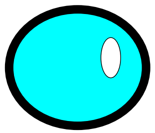
(1) If $X \in \mathcal{G}$, then $|X| \leq 2$

(2) for all $v \in V$, $\{v\} \in \mathcal{G}$

(3) for all $X \in \mathcal{G}$, and $Y \subset X$, $Y \in \mathcal{G}$



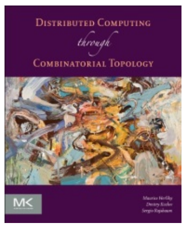
Dimension



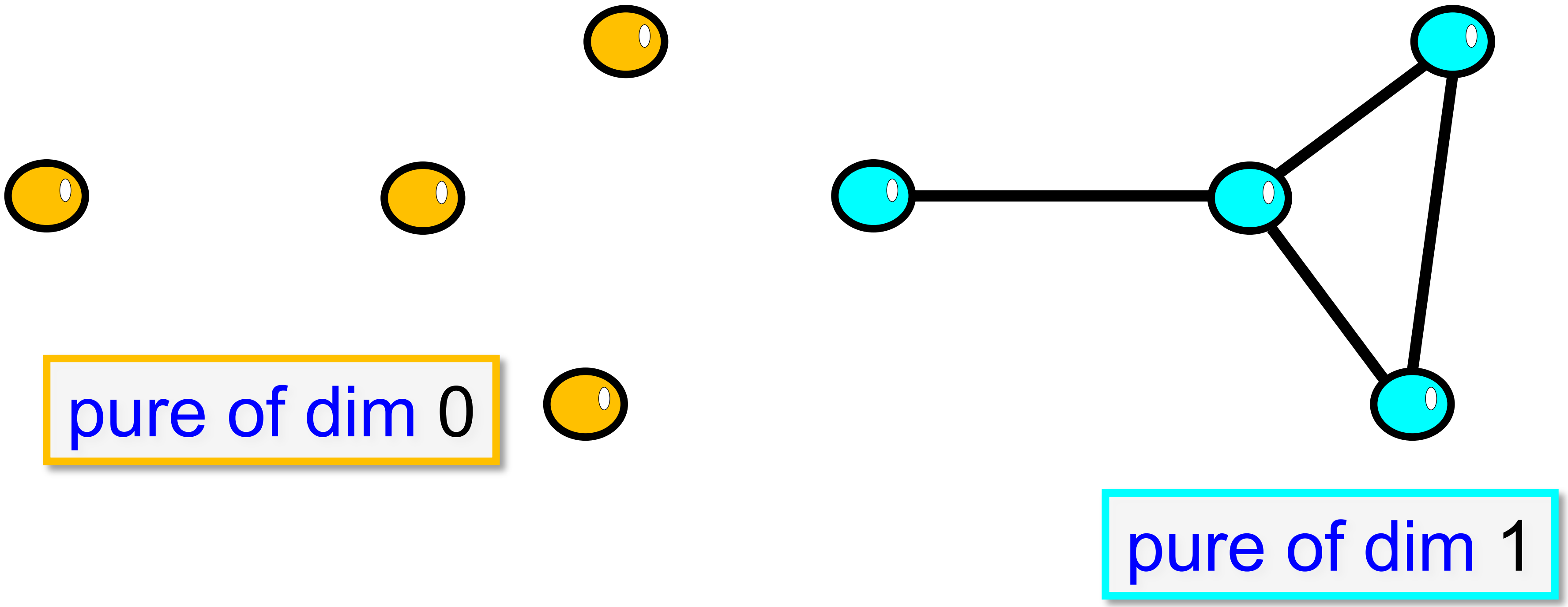
dimension 0

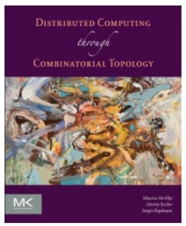
dimension 1

$\dim(X) = |X| - 1.$

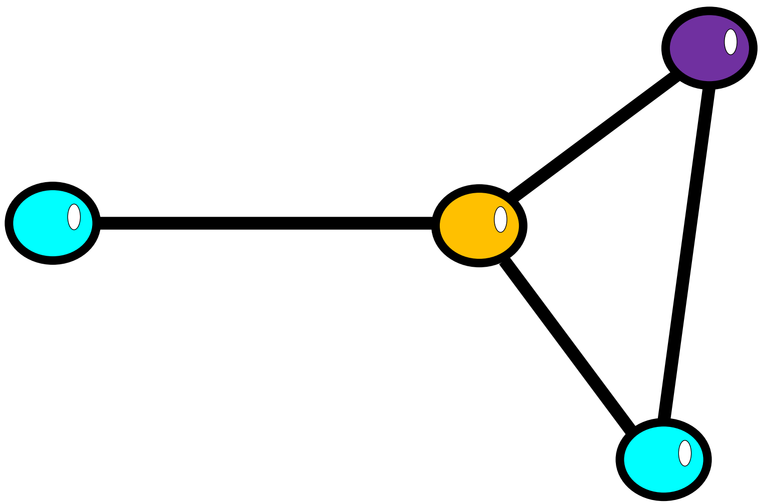


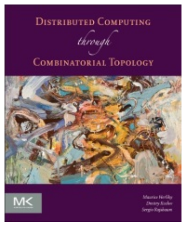
Pure Graphs


pure of dim 0
pure of dim 1

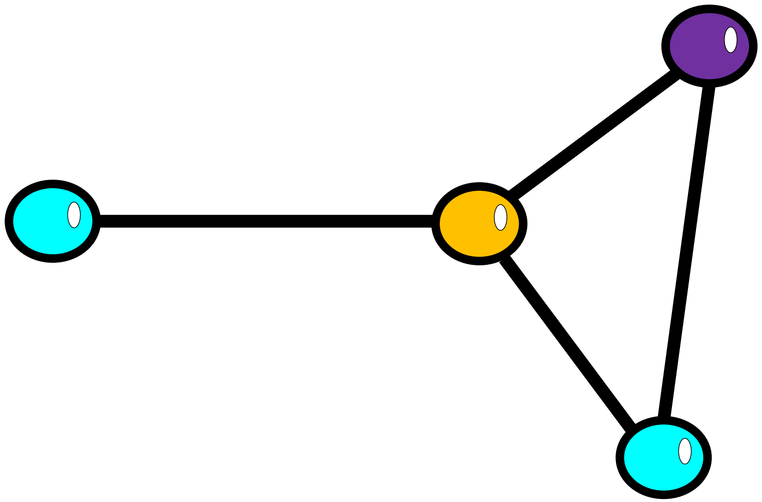

Distributed Computing
through
Combinatorial Topology

Graph Coloring

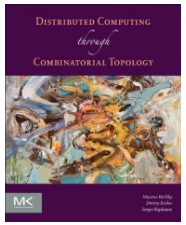




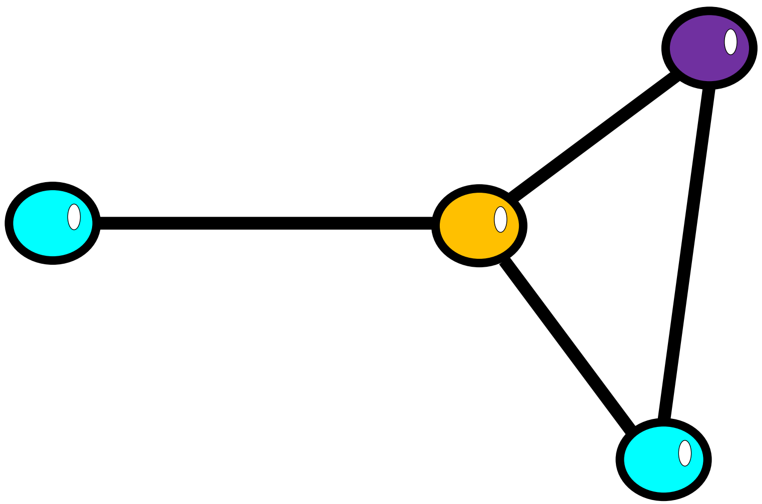
Graph Coloring



$$\chi: \mathcal{G} \to C$$

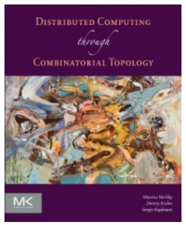


Graph Coloring

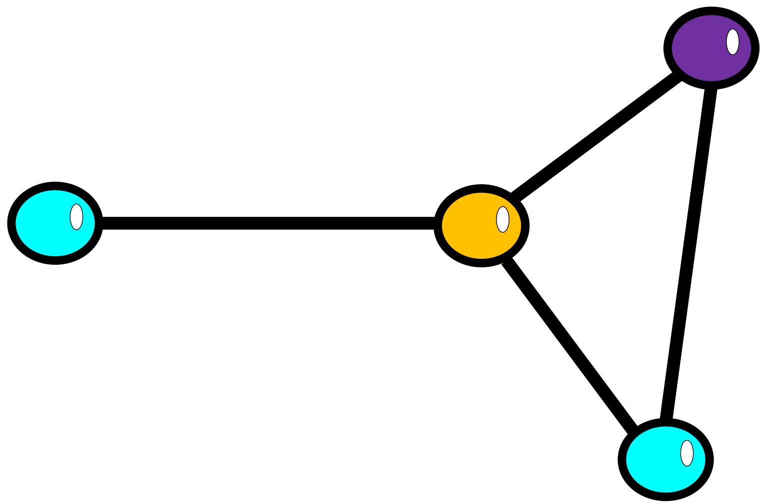


$\chi: \mathcal{G} \to C$

for each edge $(s_0, s_1) \in \mathcal{G}$, $\chi(s_0) \neq \chi(s_1)$.



Graph Coloring

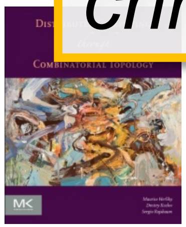


$\chi: \mathcal{G} \rightarrow C$

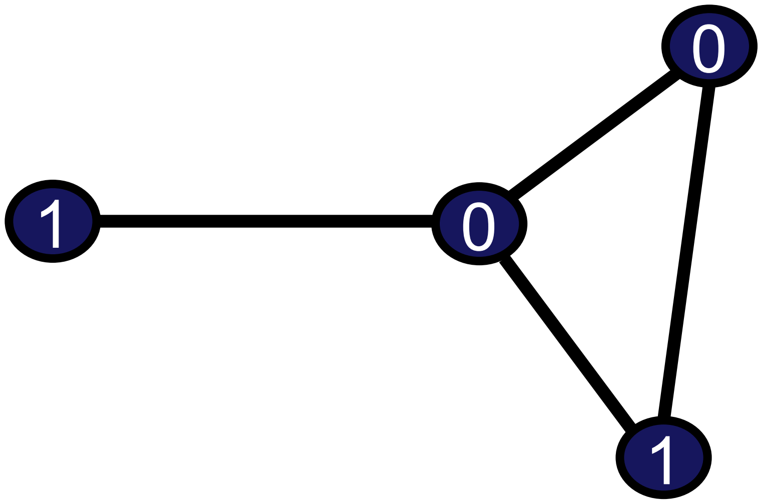
for each edge $(s_0, s_1) \in \mathcal{G}$, $\chi(s_0) \neq \chi(s_1)$.

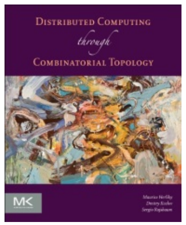
chromatic graphs

usually process names

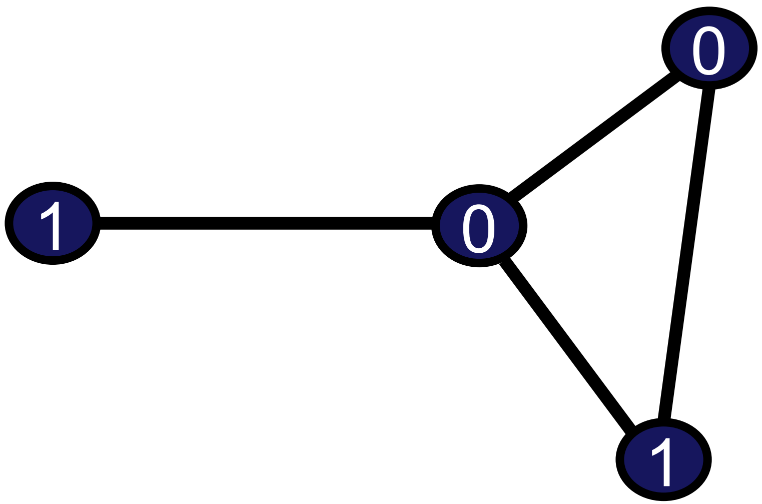


Graph Labeling

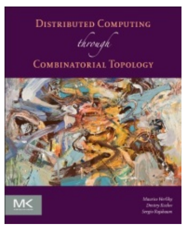




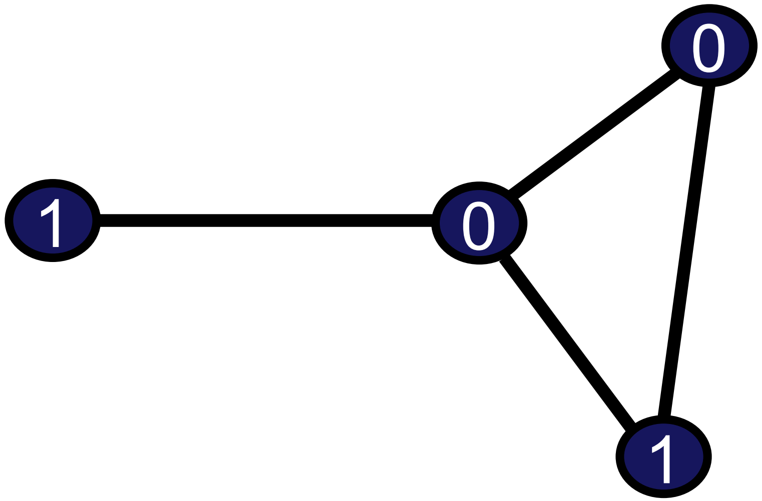
Graph Labeling



$f: \mathcal{G} \rightarrow L$

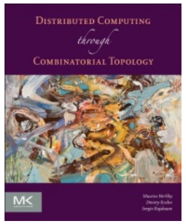


Graph Labeling

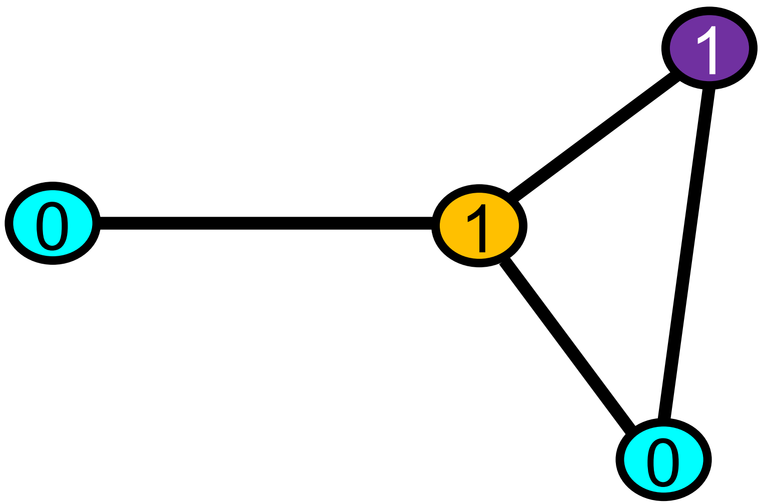


$f: \mathcal{G} \rightarrow L$

usually values from some domain

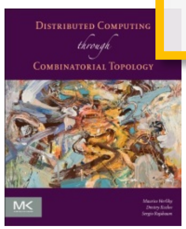


Labeled Chromatic Graph

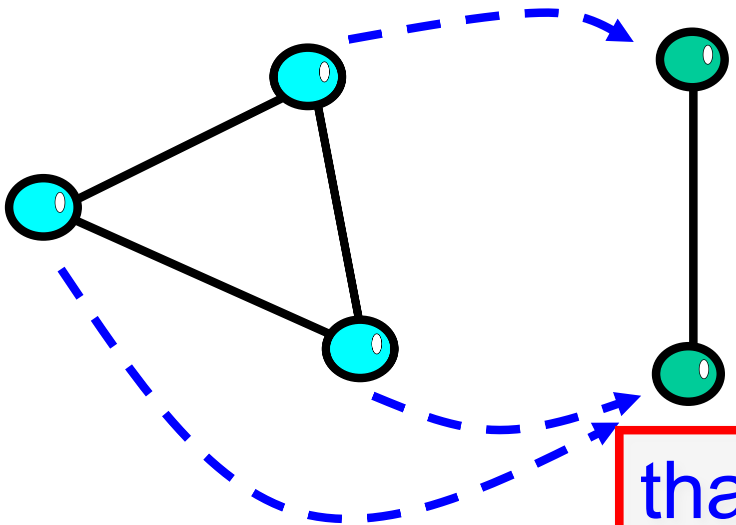


name(s) = $\chi(s)$

view(s) = $f(s)$

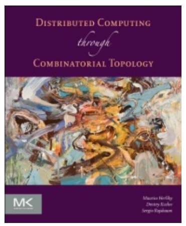


Simplicial Maps

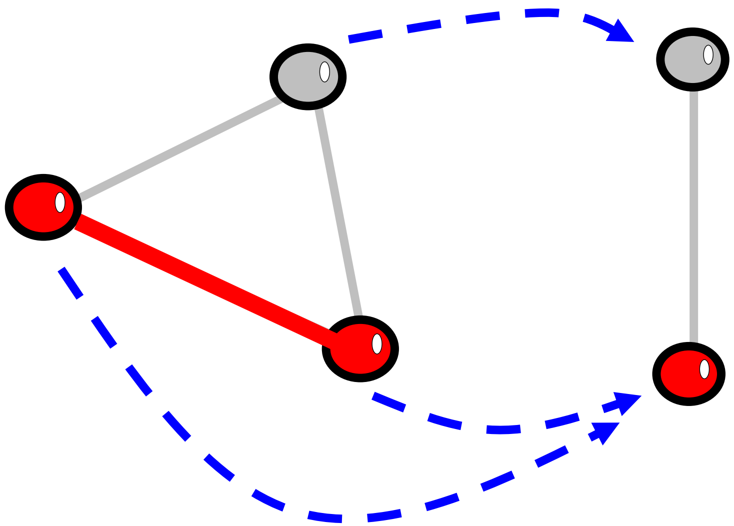


Vertex-to-vertex map …

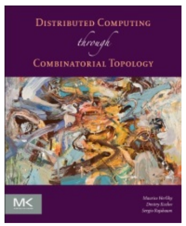
that also sends edges to edges.



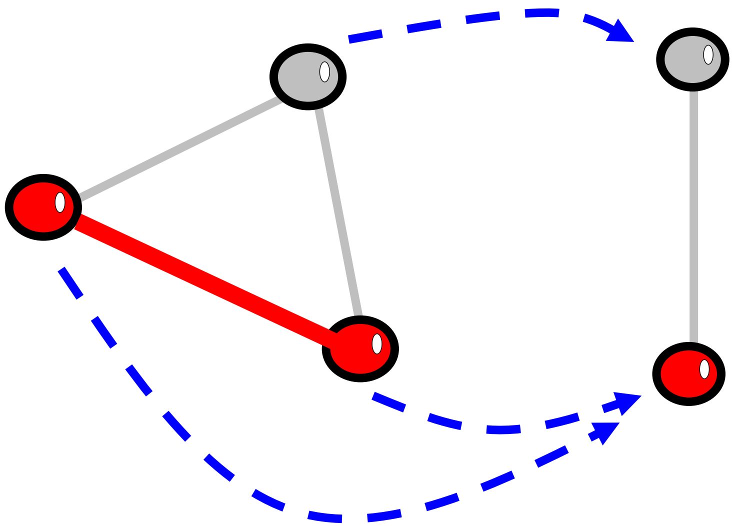
Rigid Simplicial Maps



A simplicial map can send an edge to a vertex …

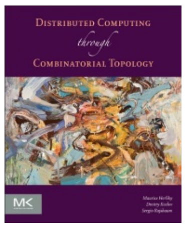


Rigid Simplicial Maps

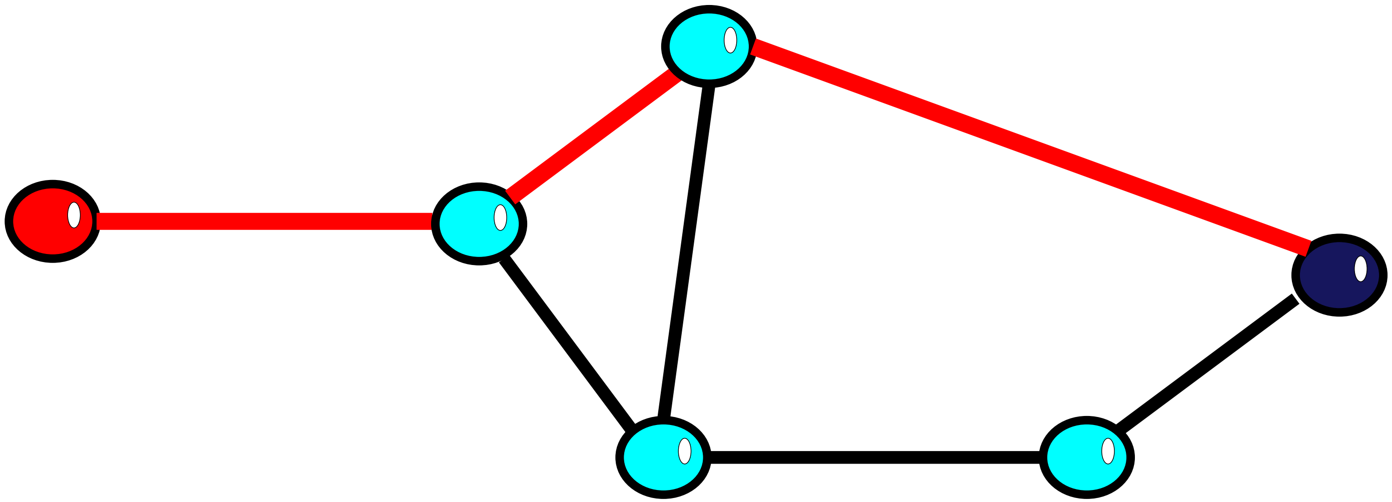


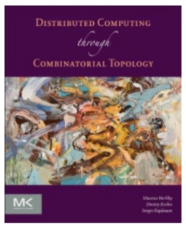
A simplicial map can send an edge to a vertex …

A simplicial map that sends an edge to an edge is *rigid*.

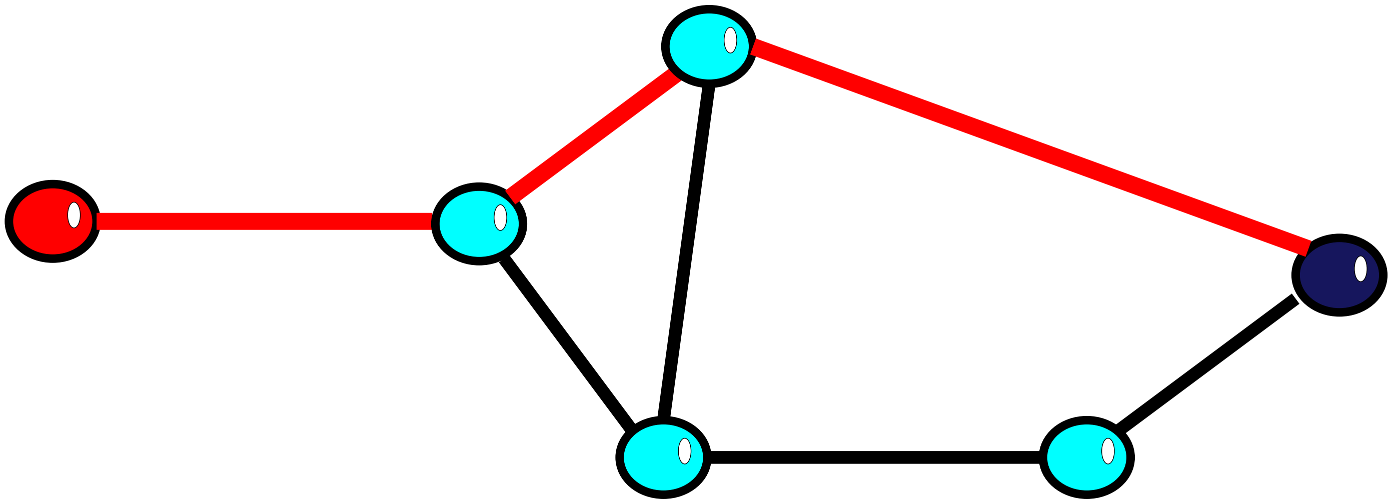


A Path Between two Vertices

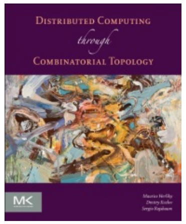




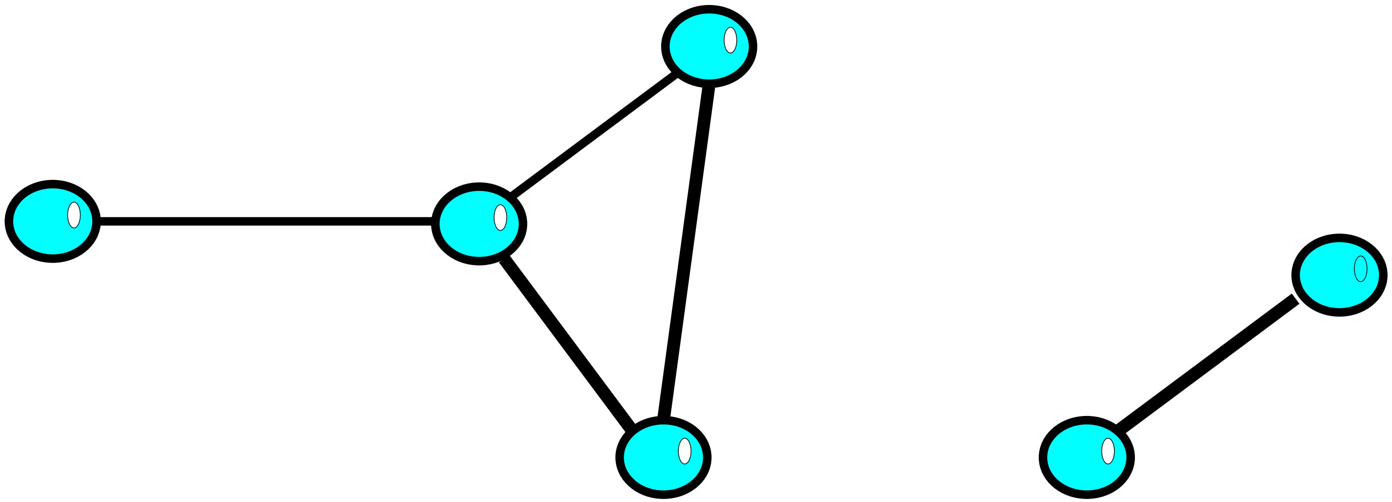
A Path Between two Vertices



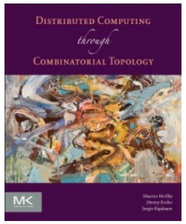
A graph is *connected* if there is a path between every pair of vertices



Not Connected



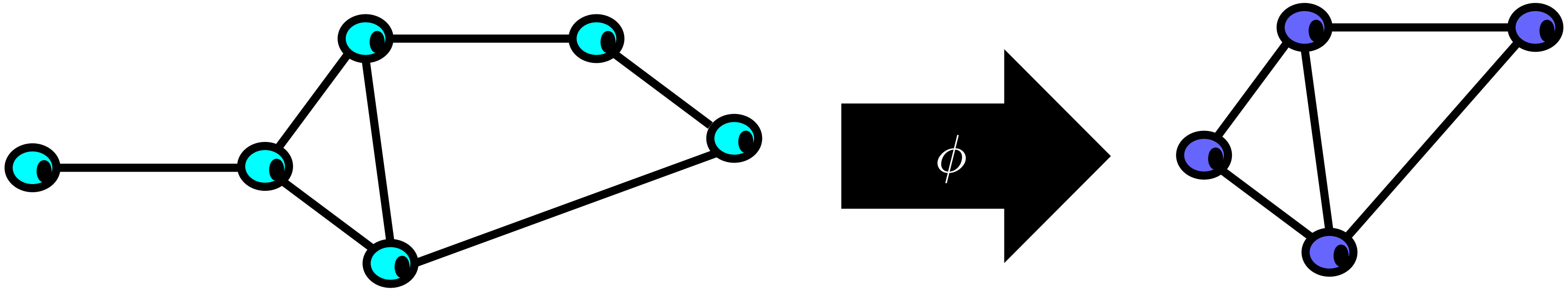
A graph is *connected* if there is a path between every pair of vertices

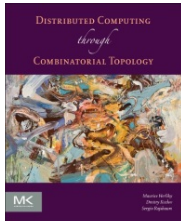


Theorem

Theorem

The image of a connected graph under a simplicial map is connected.

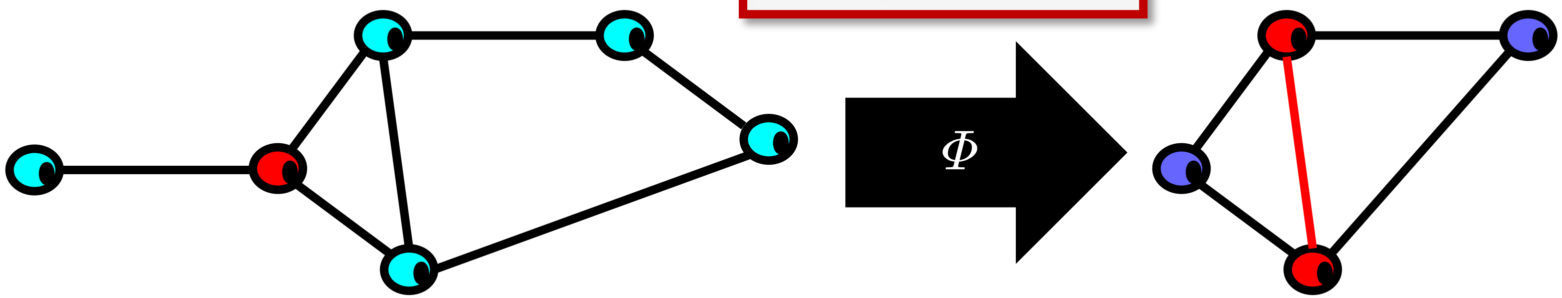




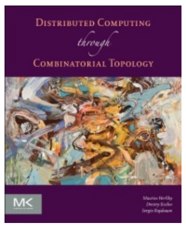
Carrier Maps

For graphs $\mathcal{G}$, $\mathcal{H}$, a *carrier map*

$\Phi: \mathcal{G} \to 2^{\mathcal{H}}$



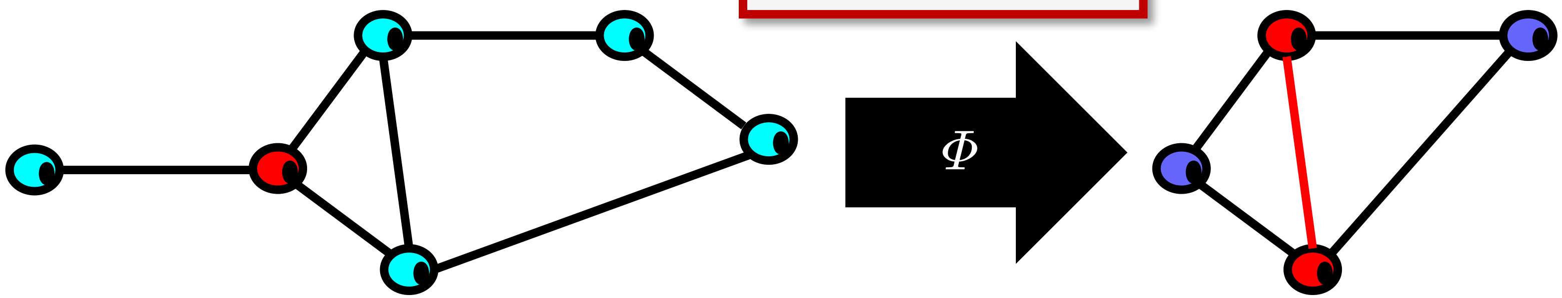
Carries each simplex of $\mathcal{G}$ to a *subgraph* of $\mathcal{H}$ …



Carrier Maps

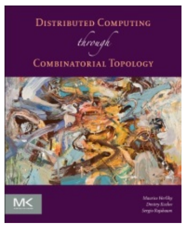
For graphs $\mathcal{G}$, $\mathcal{H}$, a *carrier map*

$\Phi: \mathcal{G} \to 2^{\mathcal{H}}$



Carries each simplex of $\mathcal{G}$ to a *subgraph* of $\mathcal{H}$ …

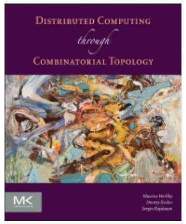
satisfying *monotonicity*:
for all $\sigma,\tau\in\mathcal{G}$, if $\sigma\subseteq\tau$, then $\Phi(\sigma)\subseteq\Phi(\tau)$.



Strict Carrier Maps

Monotonicity

For all $\sigma, \tau \in \mathcal{G}$, if $\sigma \subseteq \tau$, then $\Phi(\sigma) \subseteq \Phi(\tau)$.



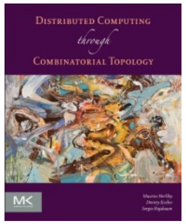
Strict Carrier Maps

Monotonicity

For all $\sigma,\tau \in \mathcal{G}$, if $\sigma \subseteq \tau$, then $\Phi(\sigma) \subseteq \Phi(\tau)$.

Equivalent to …

$$\Phi(\sigma \cap \tau) \subseteq \Phi(\sigma) \cap \Phi(\tau)$$



Strict Carrier Maps

Monotonicity

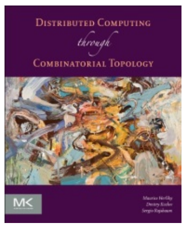
For all $\sigma,\tau\in\mathcal{G}$, if $\sigma\subseteq\tau$, then $\Phi(\sigma)\subseteq\Phi(\tau)$.

Equivalent to …

$\Phi(\sigma\cap\tau) \subseteq \Phi(\sigma) \cap \Phi(\tau)$

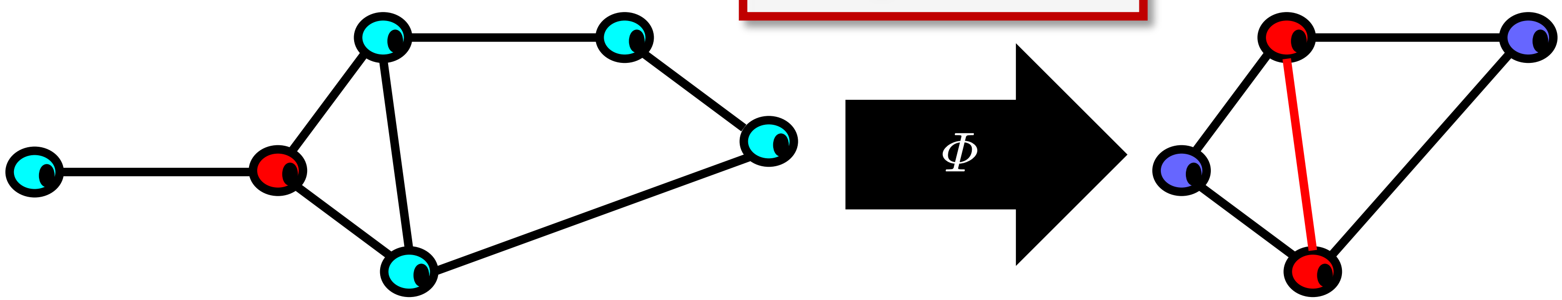
Definition

Φ is *strict* if $\Phi(\sigma\cap\tau) = \Phi(\sigma) \cap \Phi(\tau)$



Connected Carrier Maps

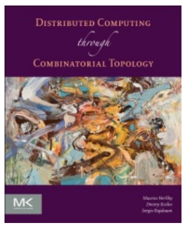
$\Phi: \mathcal{G} \rightarrow 2^{\mathcal{H}}$



Carrier map $\Phi: \mathcal{G} \rightarrow 2^{\mathcal{H}}$ is connected if …

For each vertex $s \in \mathcal{G}$, $\Phi(s)$ is non-empty and …

for each edge $\sigma \in \mathcal{G}$, $\Phi(\sigma)$ is *connected*.



Road Map

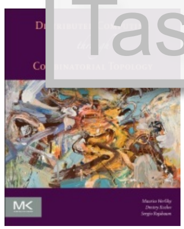
Elementary Graph Theory

Tasks

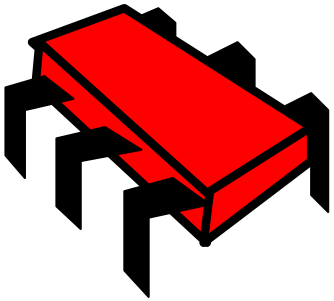
Models of Computation

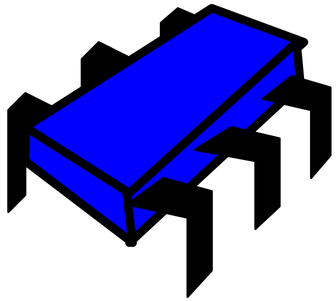
Approximate Agreement

Task Solvability



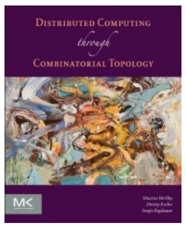
Two Processes





Hello! I'm
Alice

Hello! I'm
Bob



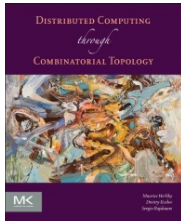
Informal Task Definition

Processes start with input values …

They communicate …

They halt with output values …

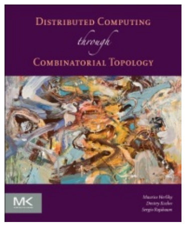
legal for those inputs.



Formal Task Definition

Input graph $\mathcal{I}$

all possible assignments of input values



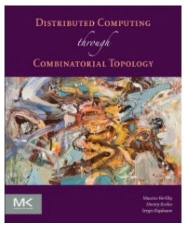
Formal Task Definition

Input graph $\mathcal{I}$

all possible assignments of input values

Output graph $\mathcal{O}$

all possible assignments of output values



Formal Task Definition

Input graph $\mathcal{I}$

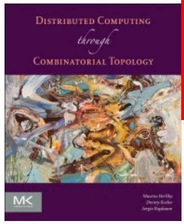
all possible assignments of input values

Output graph $\mathcal{O}$

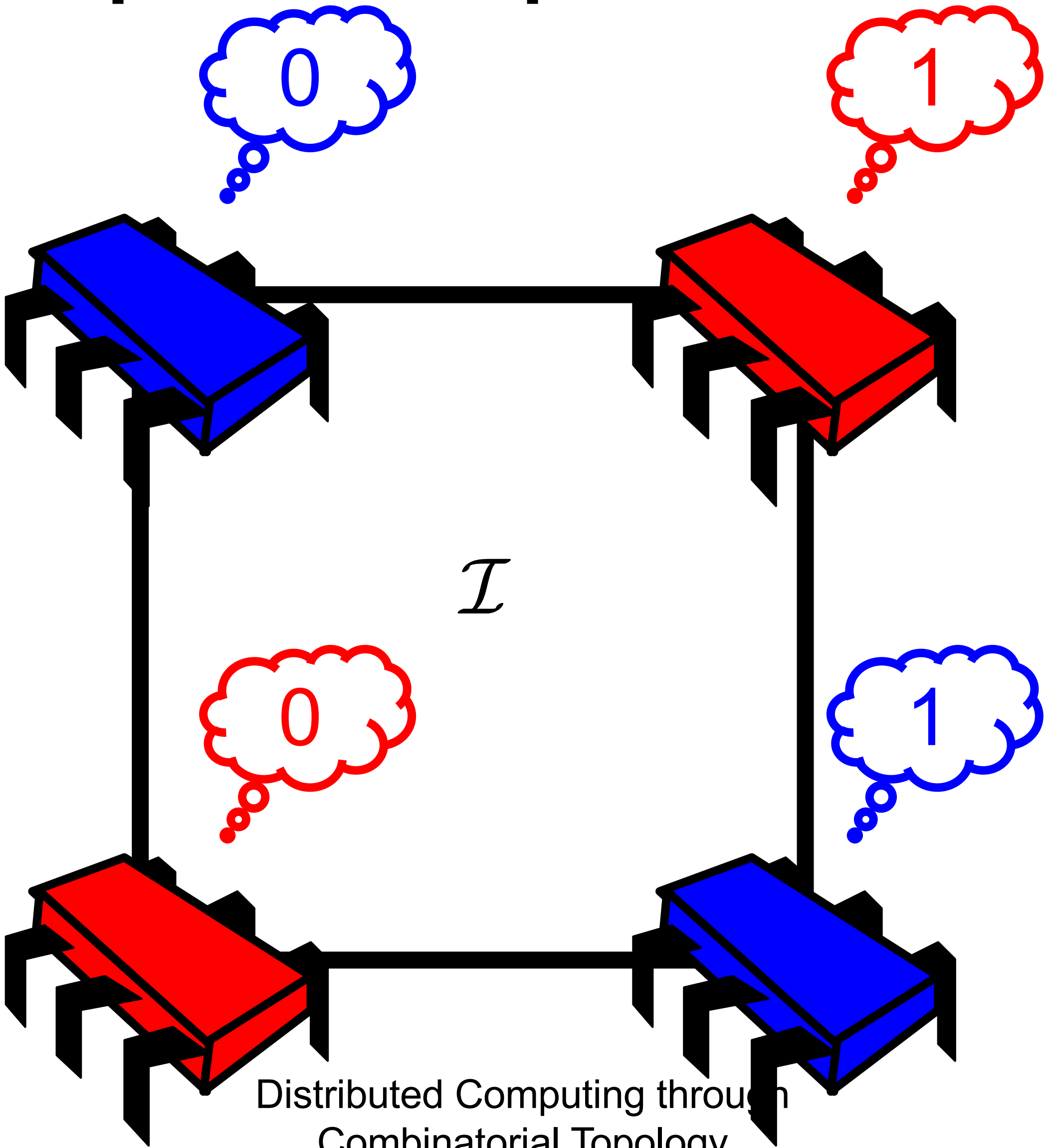
all possible assignments of output values

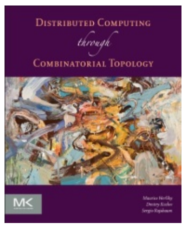
Carrier map $\Delta: \mathcal{I} \to 2^{\mathcal{O}}$

all possible assignments of output values for each input

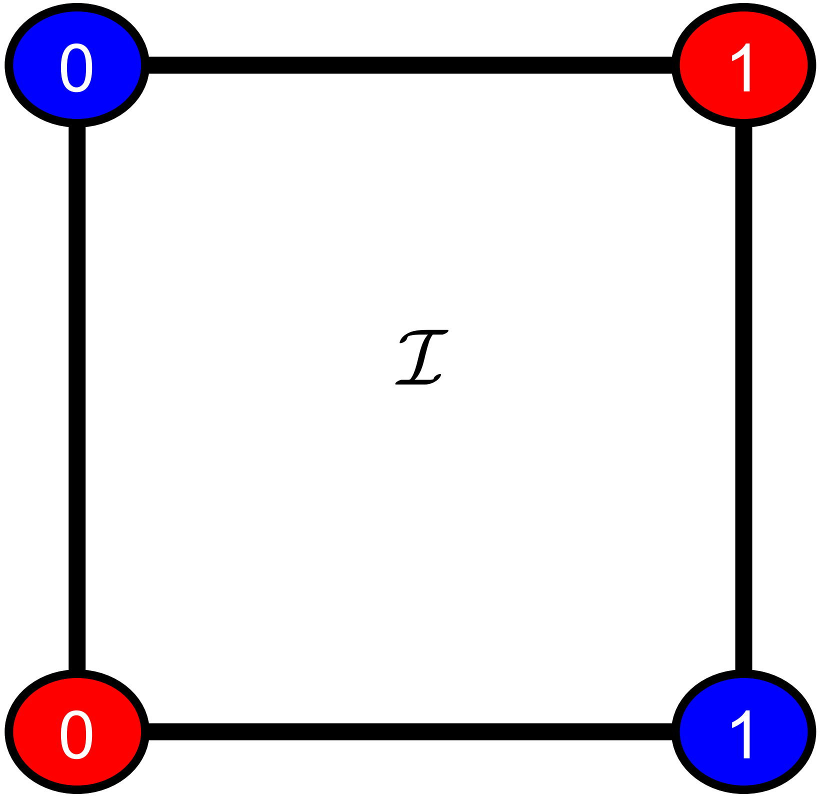


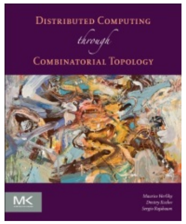
Task Input Graph: Consensus



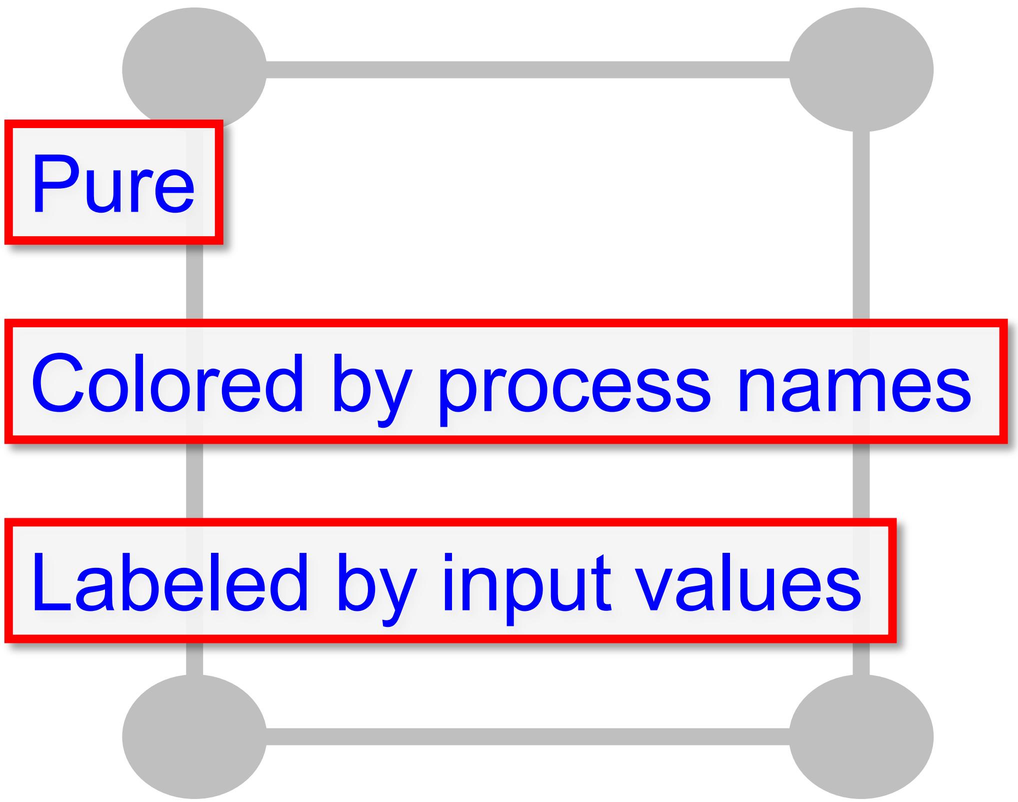


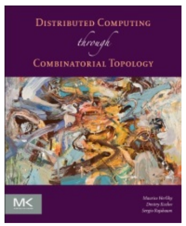
Task Input Graph



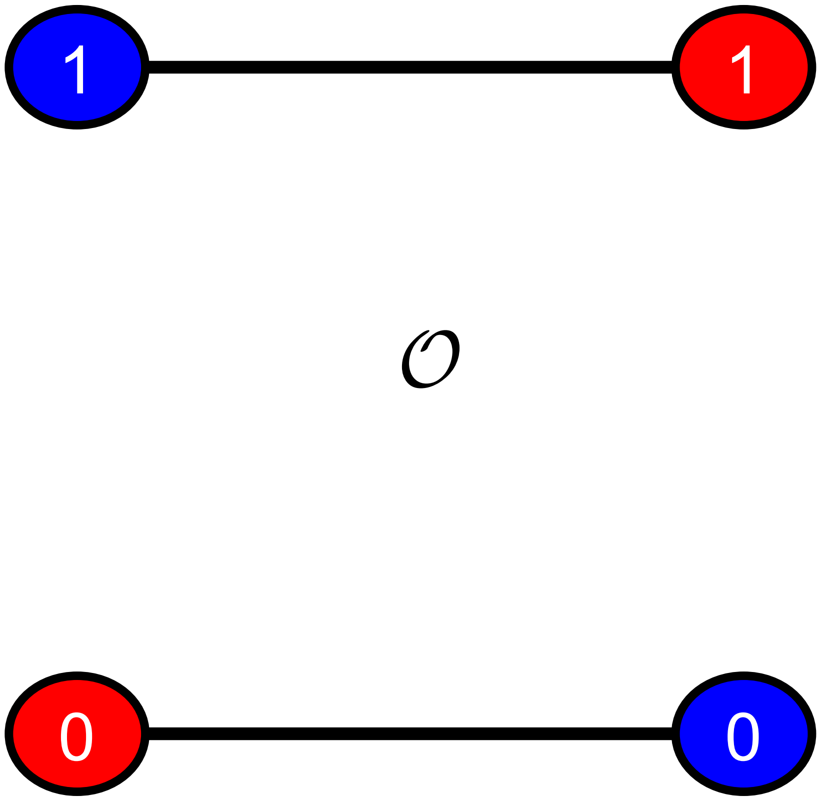


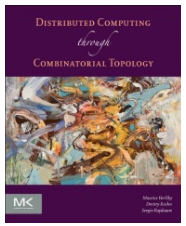
Task Input Graph



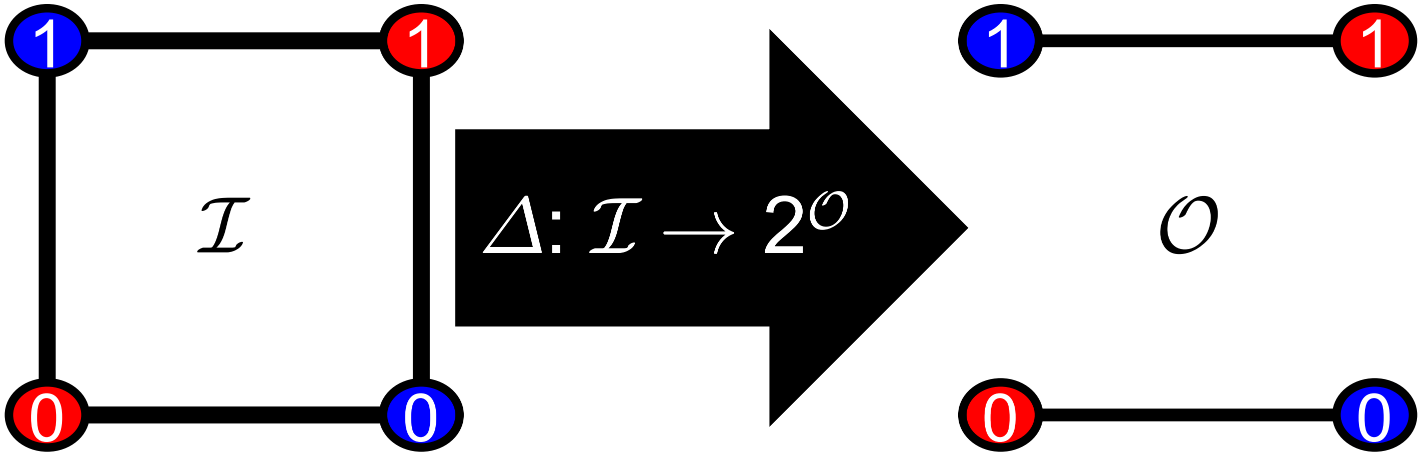


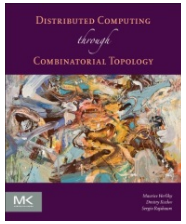
Task Output Graph



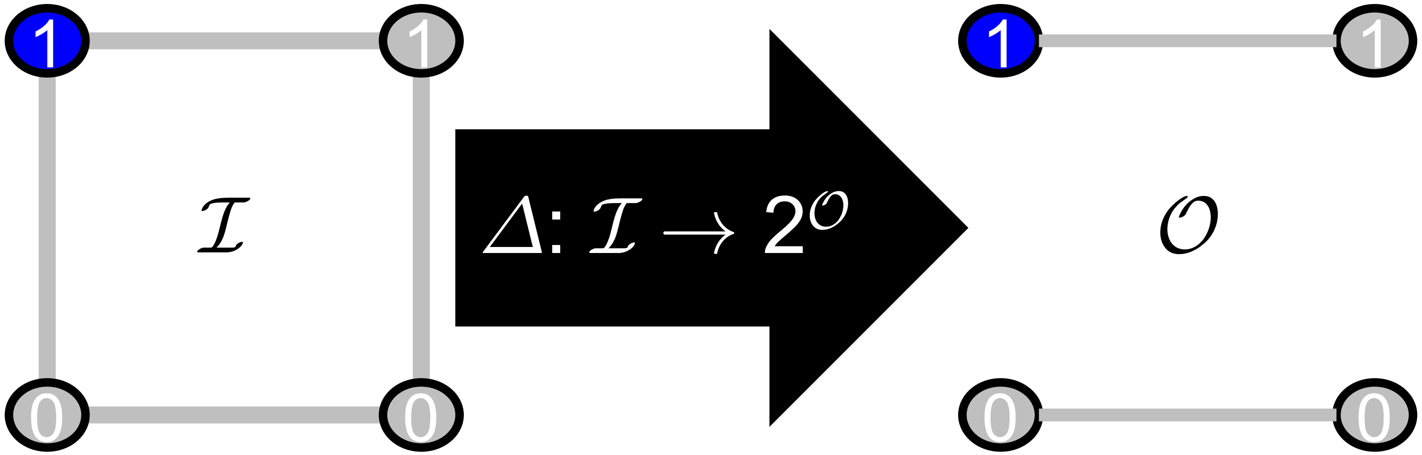


Task Carrier Map



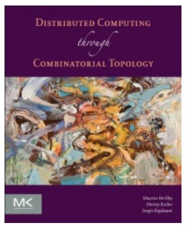


Task Carrier Map

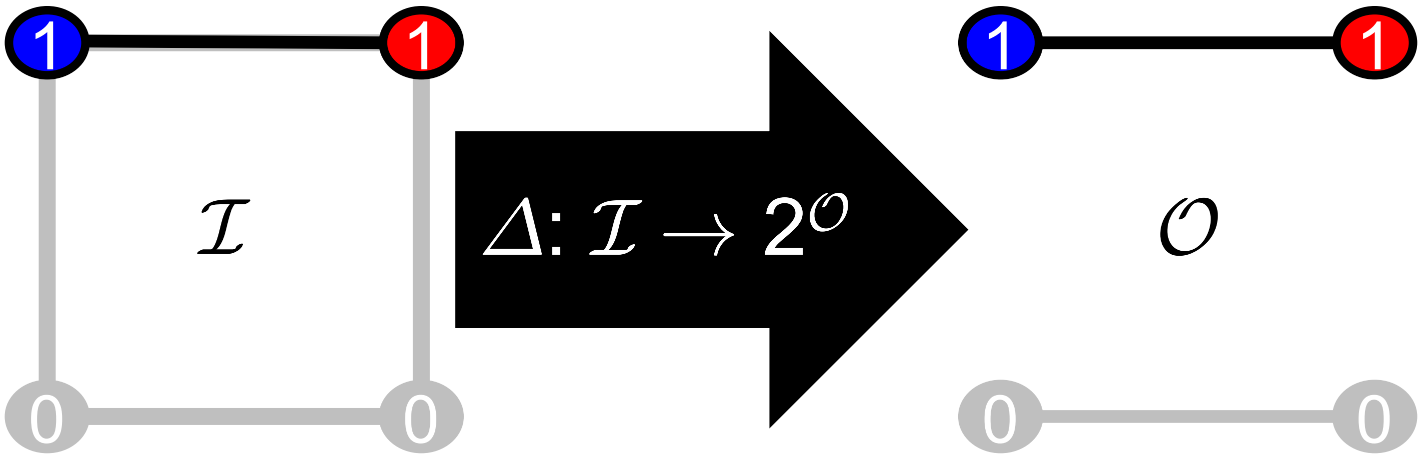


If Bob runs alone with input 1 …

then he decides output 1.

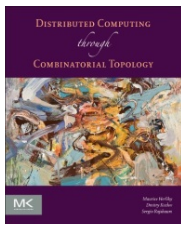


Task Carrier Map

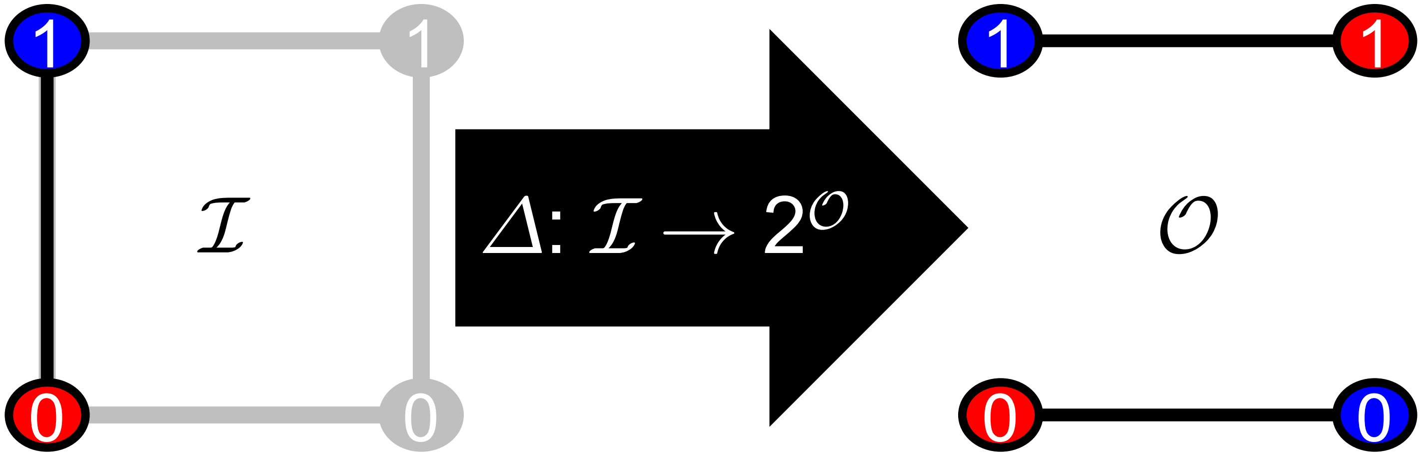


If Bob and Alice both have input 1 …

then they both decide output 1.

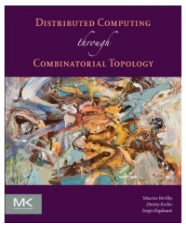


Task Carrier Map

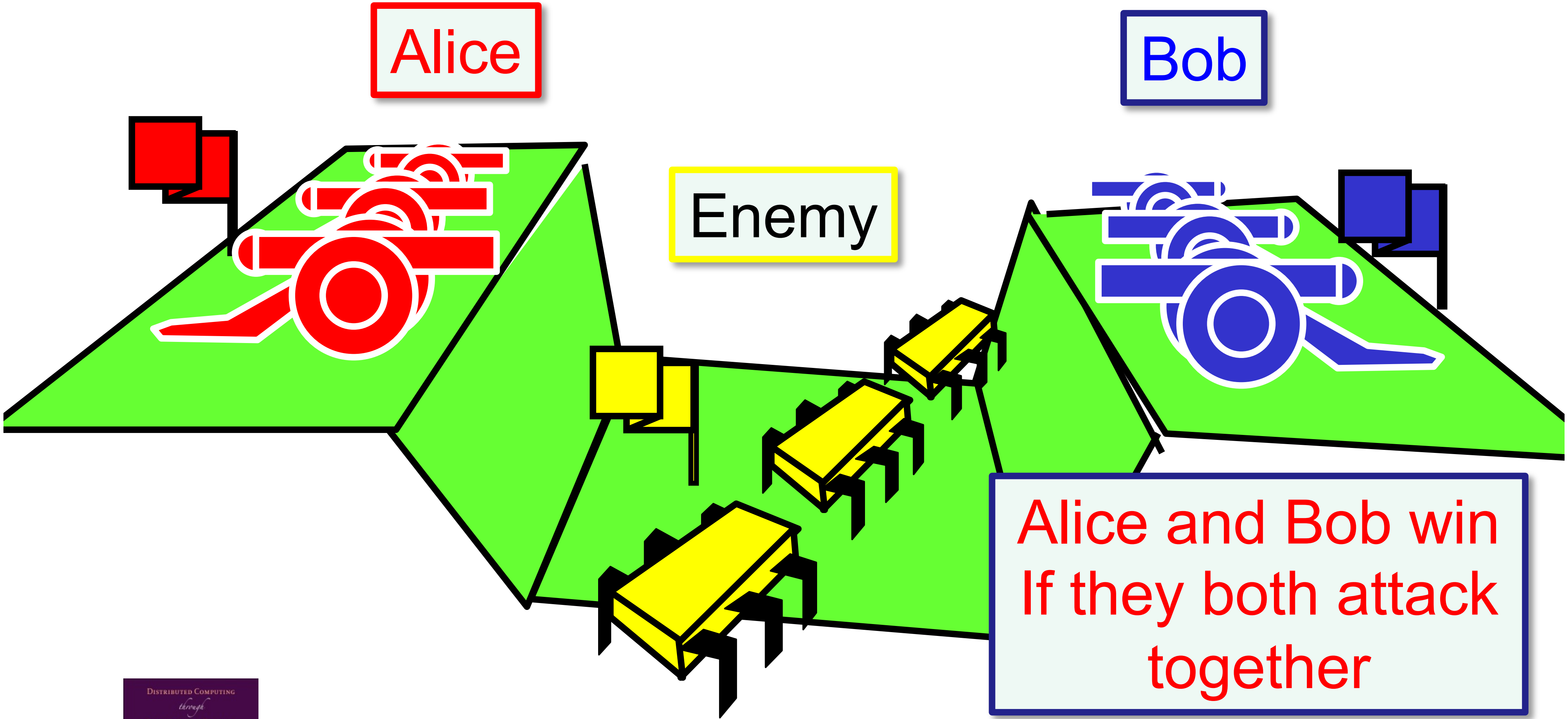


If Bob has 1 and Alice 0 …

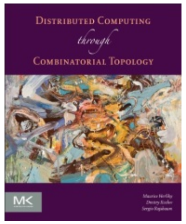
then they must agree, on either one.



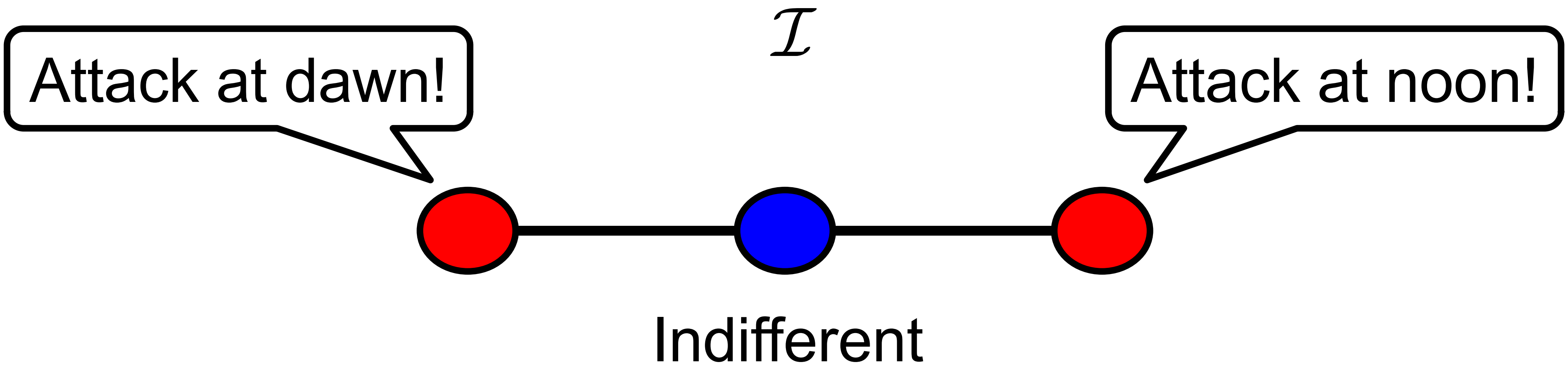
Example: Coordinated Attack

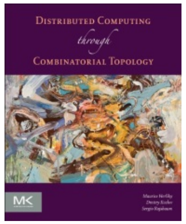


Alice and Bob win
If they both attack
together

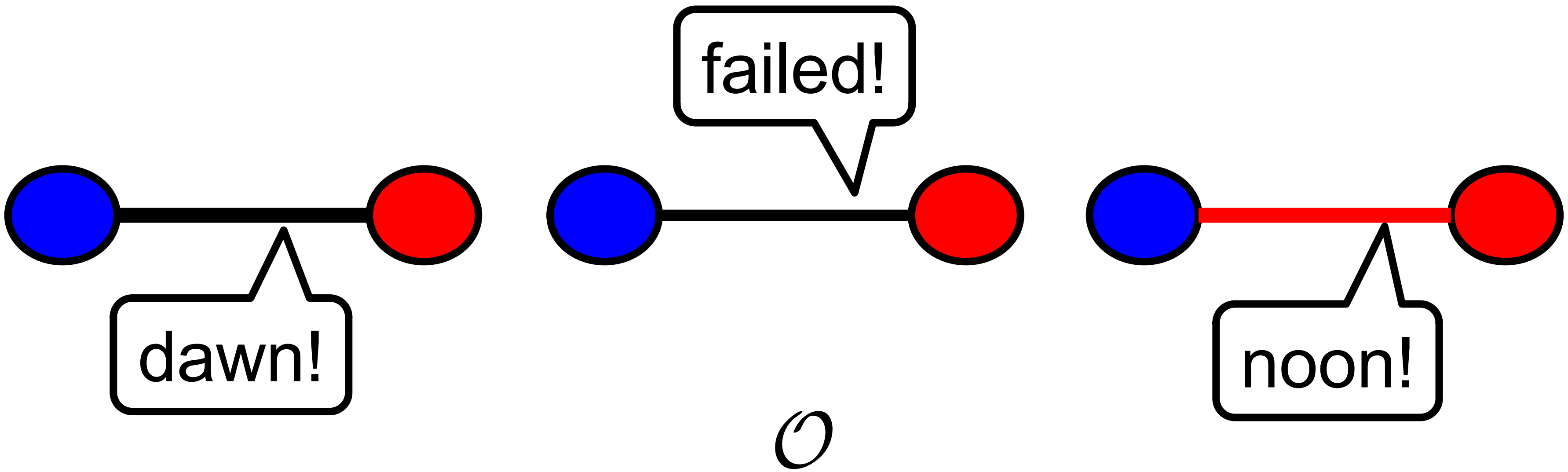


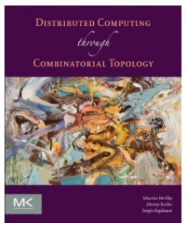
Input Graph


$\mathcal{I}$
Attack at dawn!
Attack at noon!
Indifferent

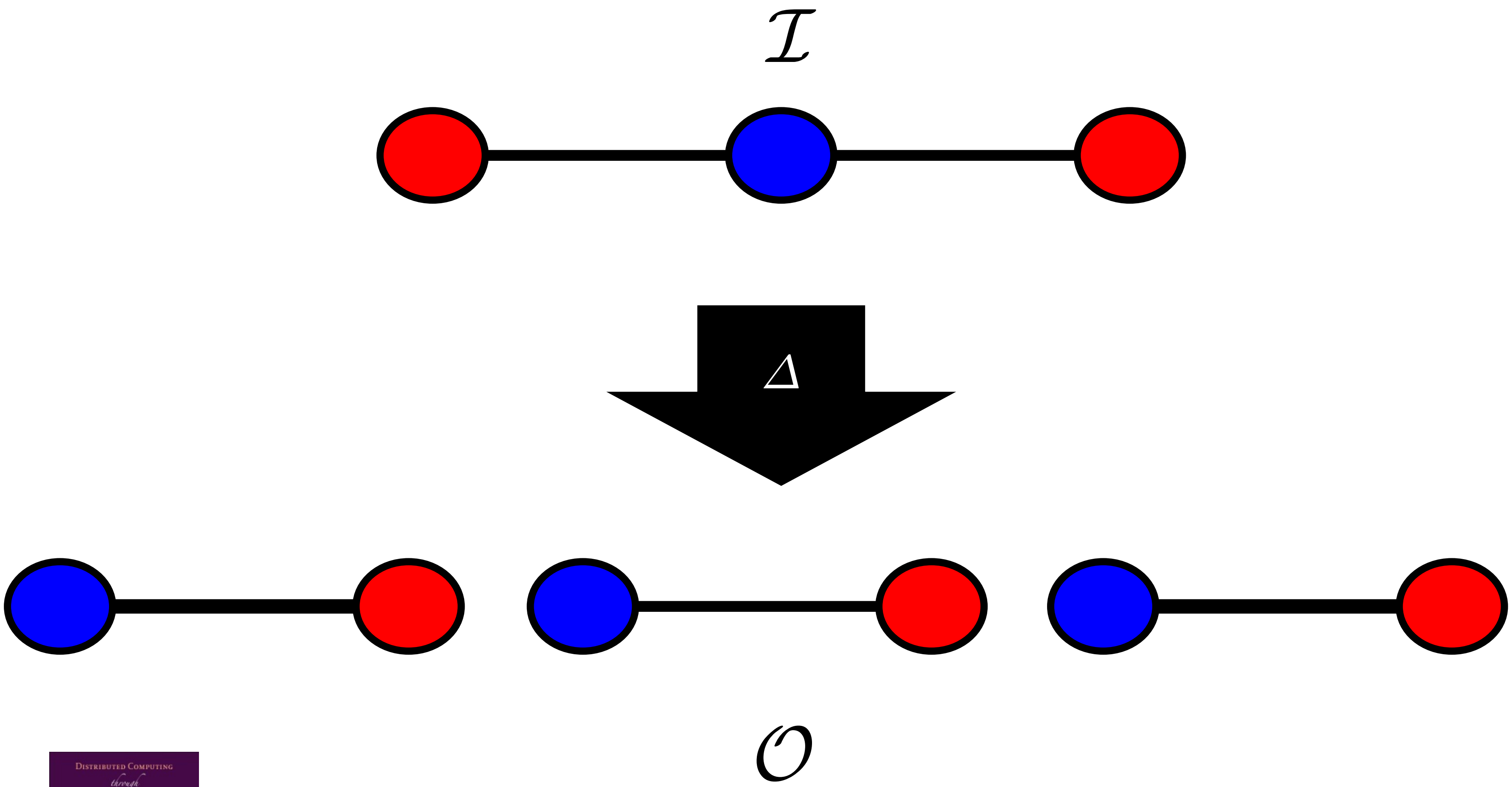

Distributed Computing through Combinatorial Topology

Output Graph

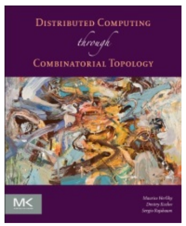




Carrier Map

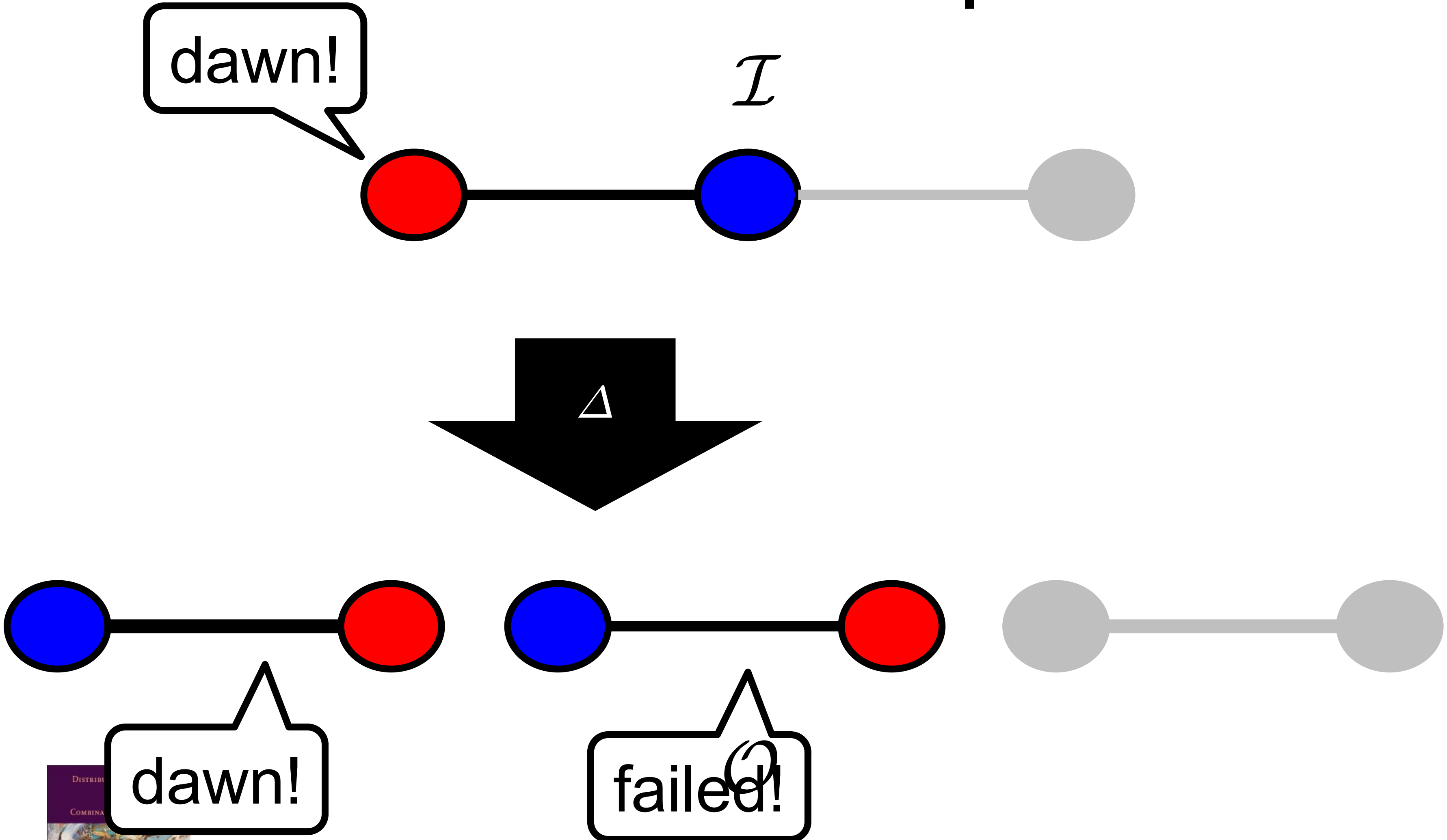


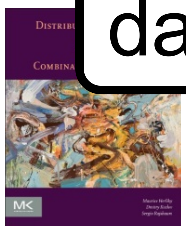
$\mathcal{I}$
Δ
$\mathcal{O}$



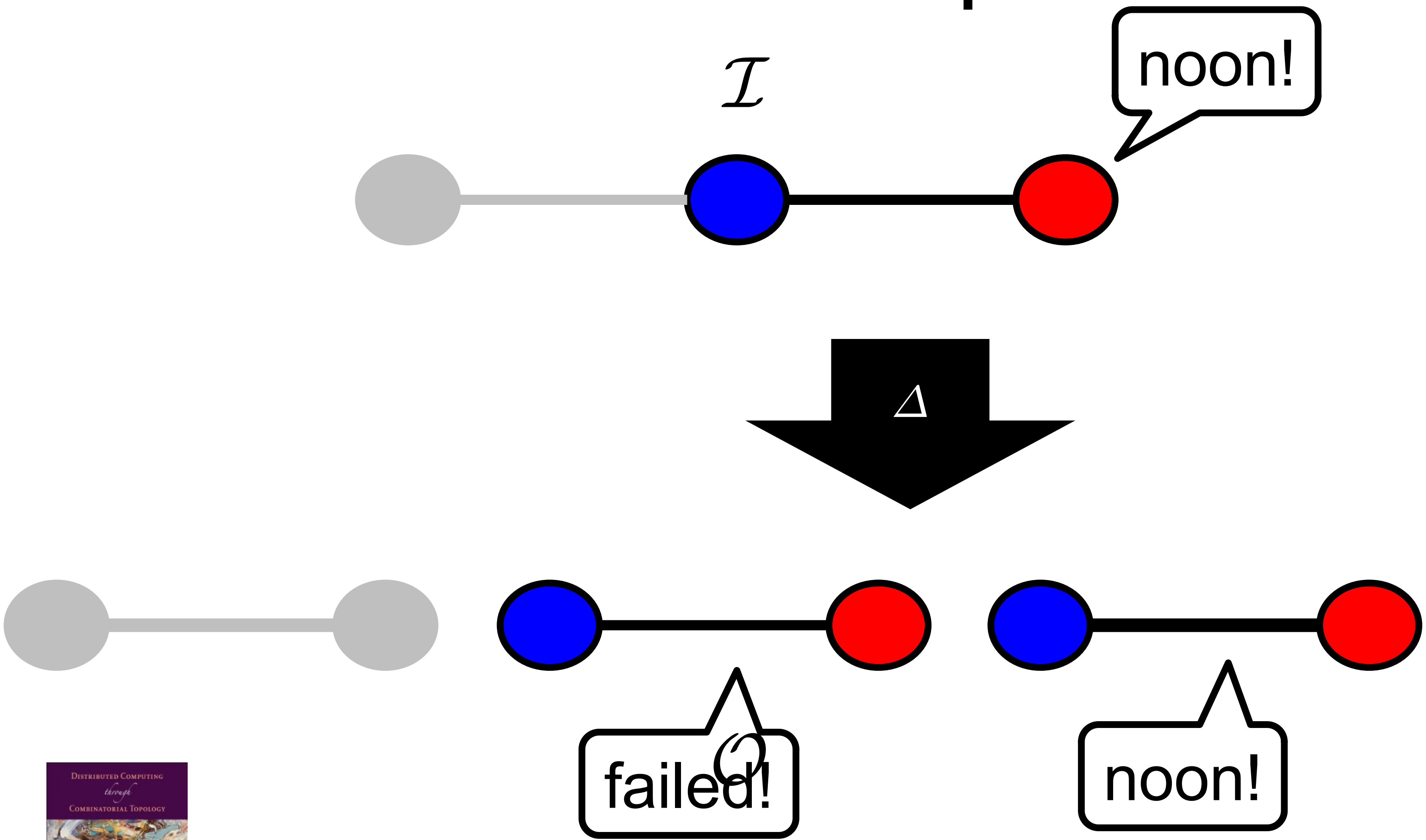
Distributed Computing
through
Combinatorial Topology

Carrier Map

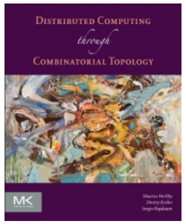




Carrier Map

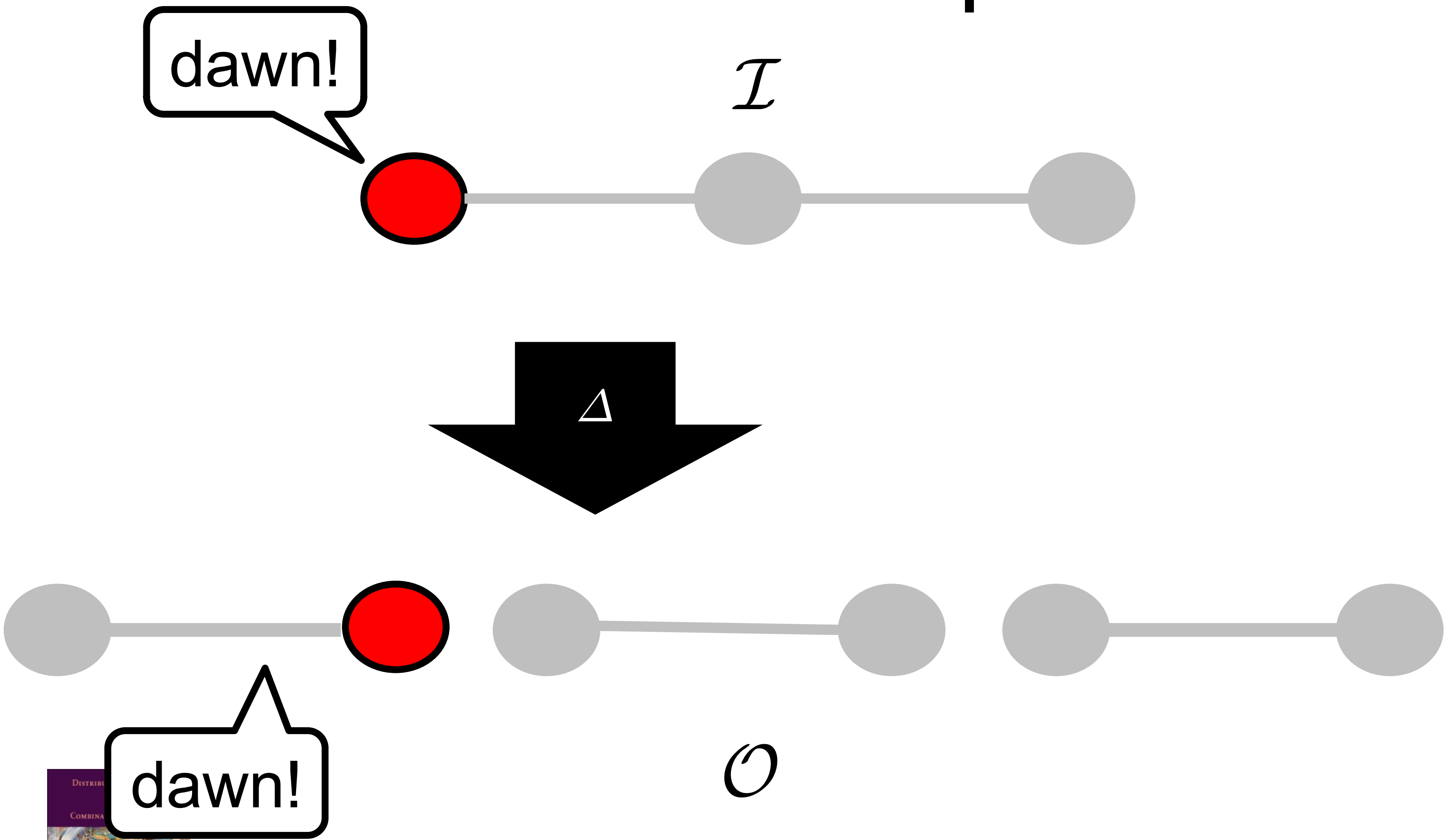


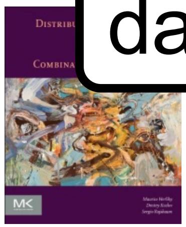
$\mathcal{I}$
noon!
Δ
failed!
$\mathcal{O}$
noon!



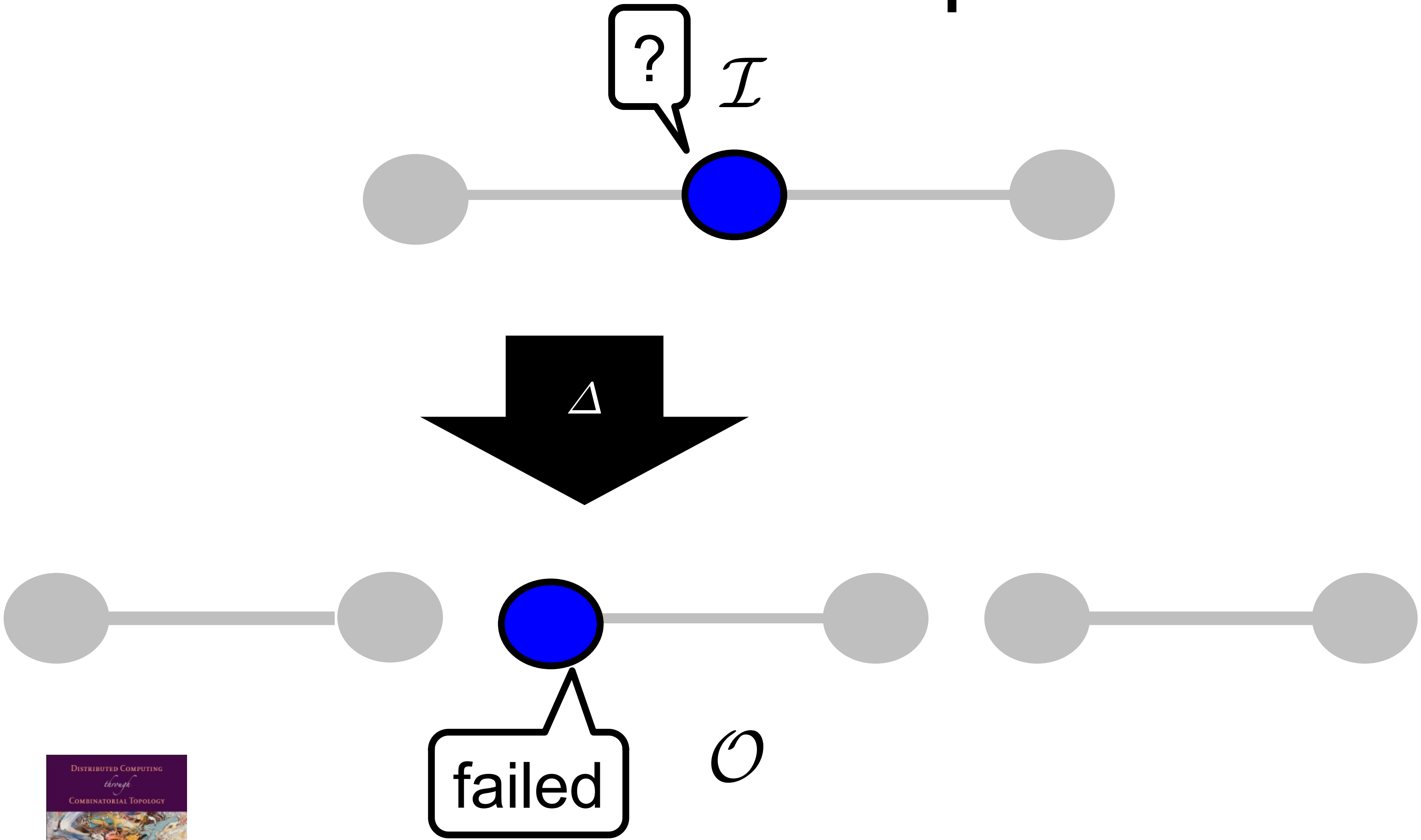
Distributed Computing
through
Combinatorial Topology
MK

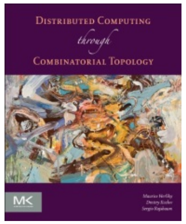
Carrier Map



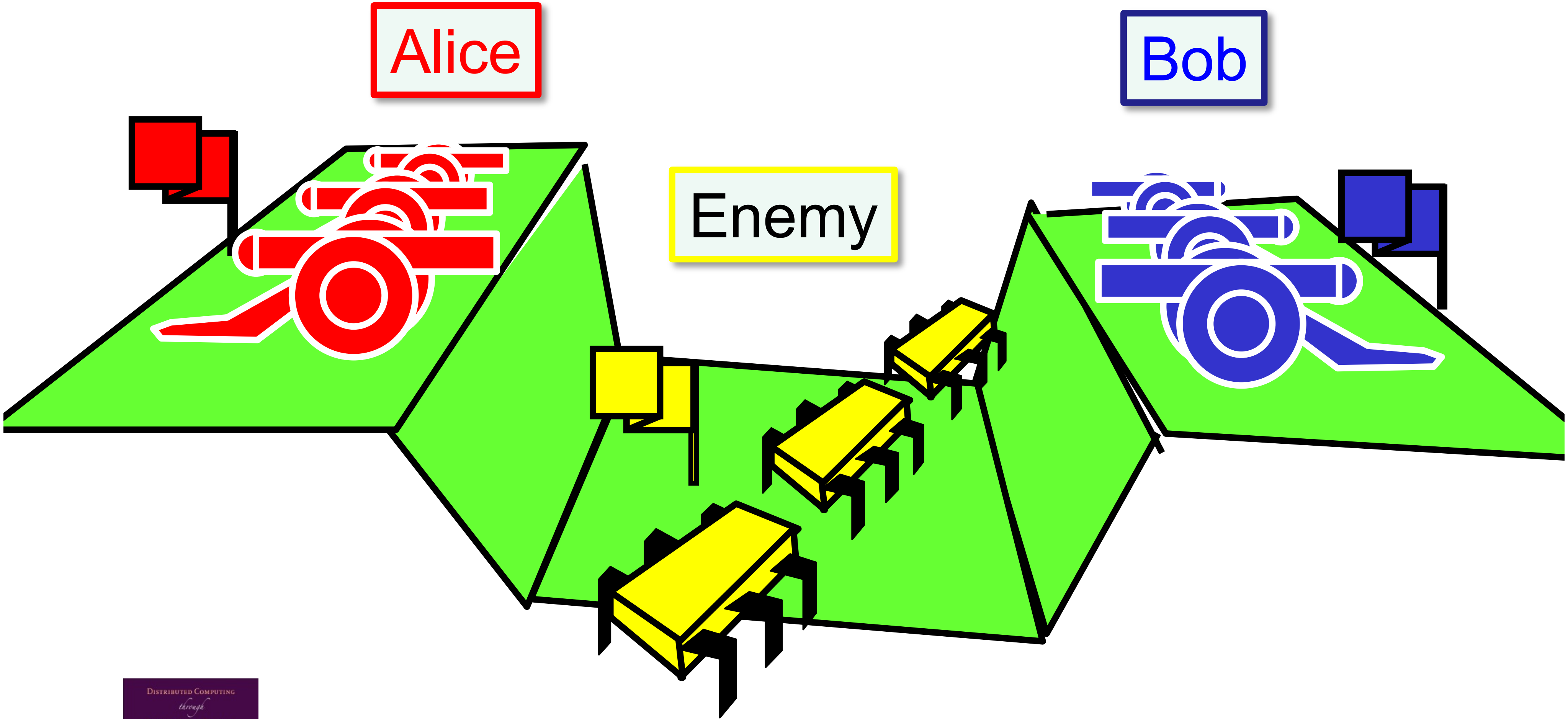


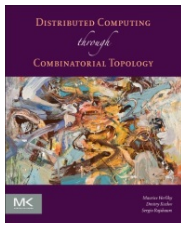
Carrier Map



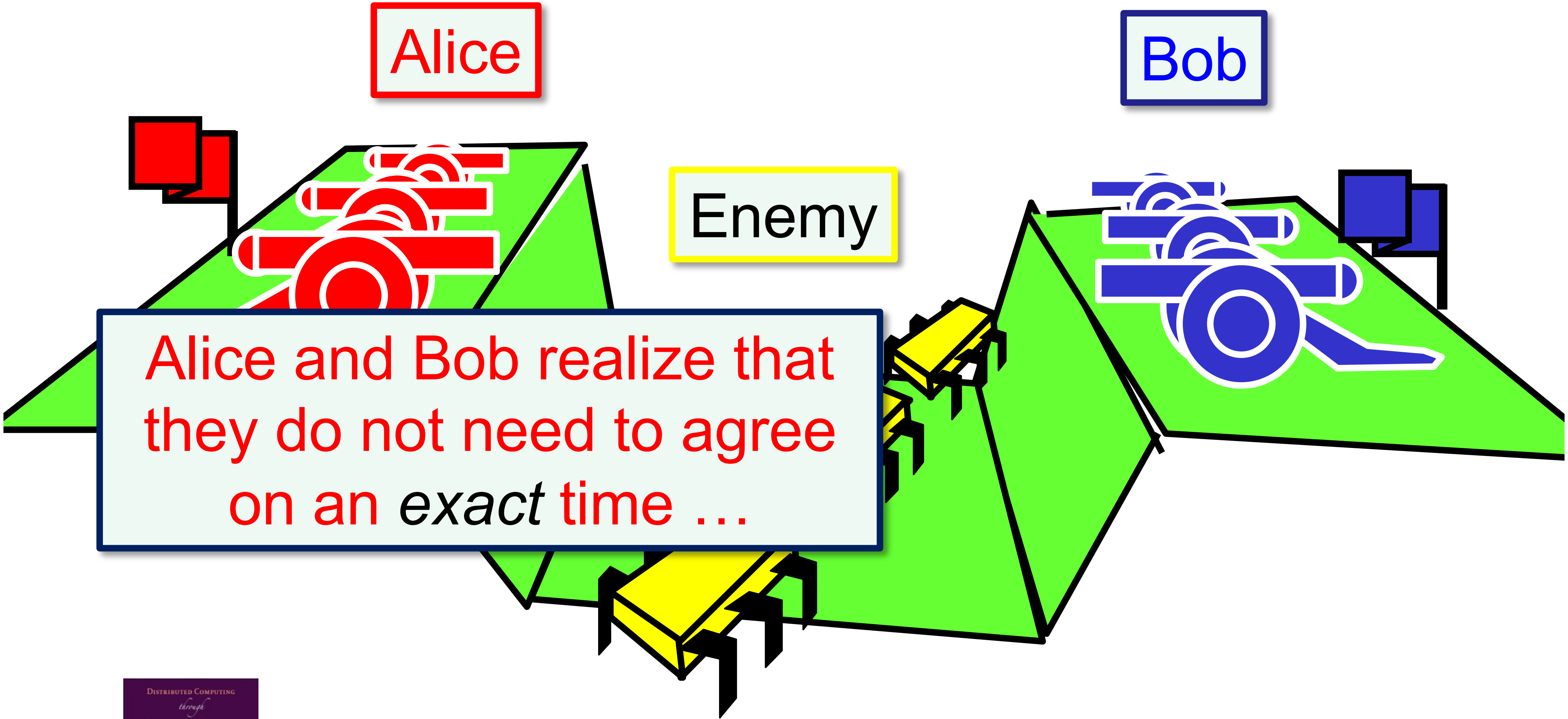


Example: Coordinated Attack

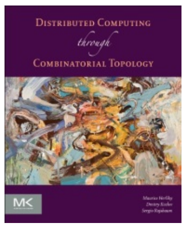




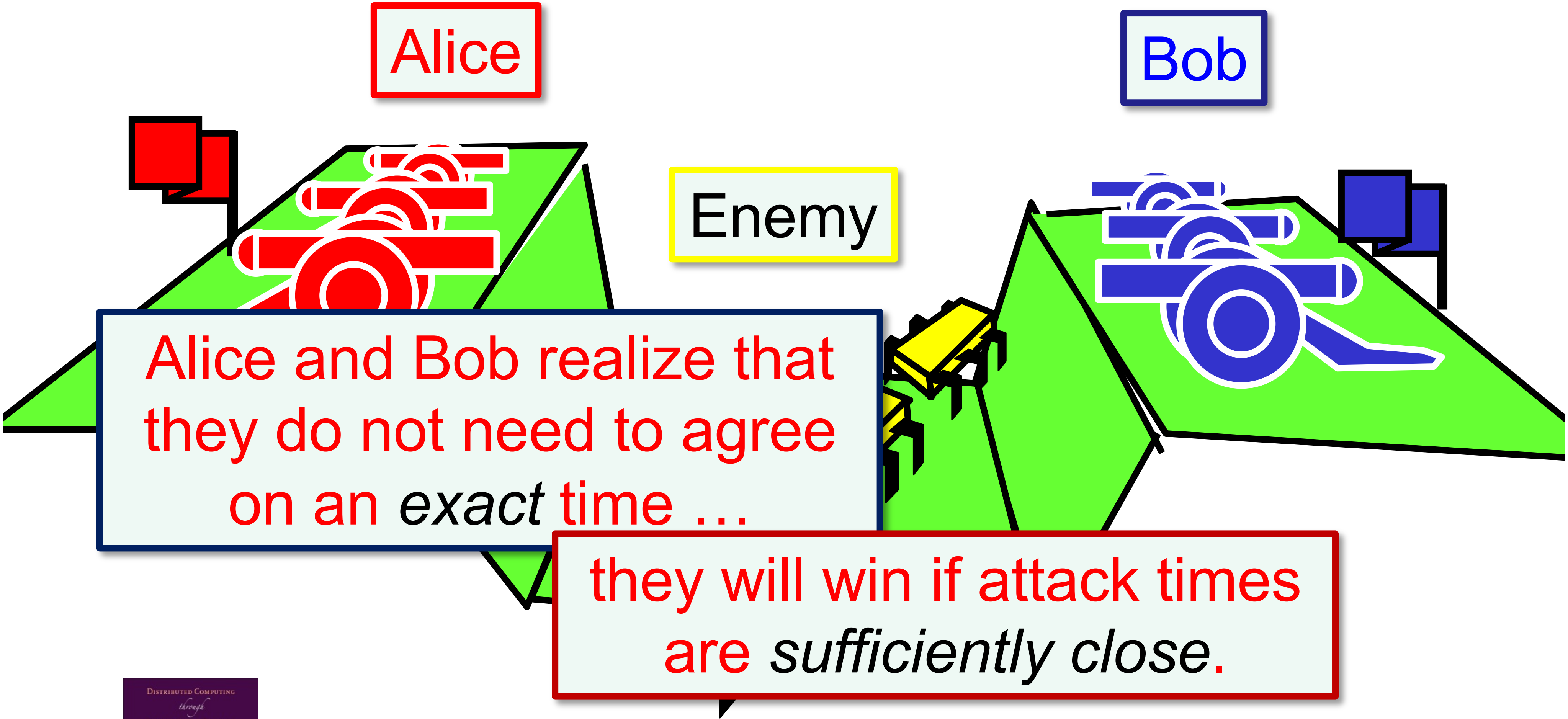
Example: Coordinated Attack



Alice and Bob realize that they do not need to agree on an *exact* time …

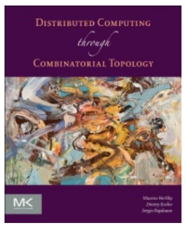


Example: Coordinated Attack

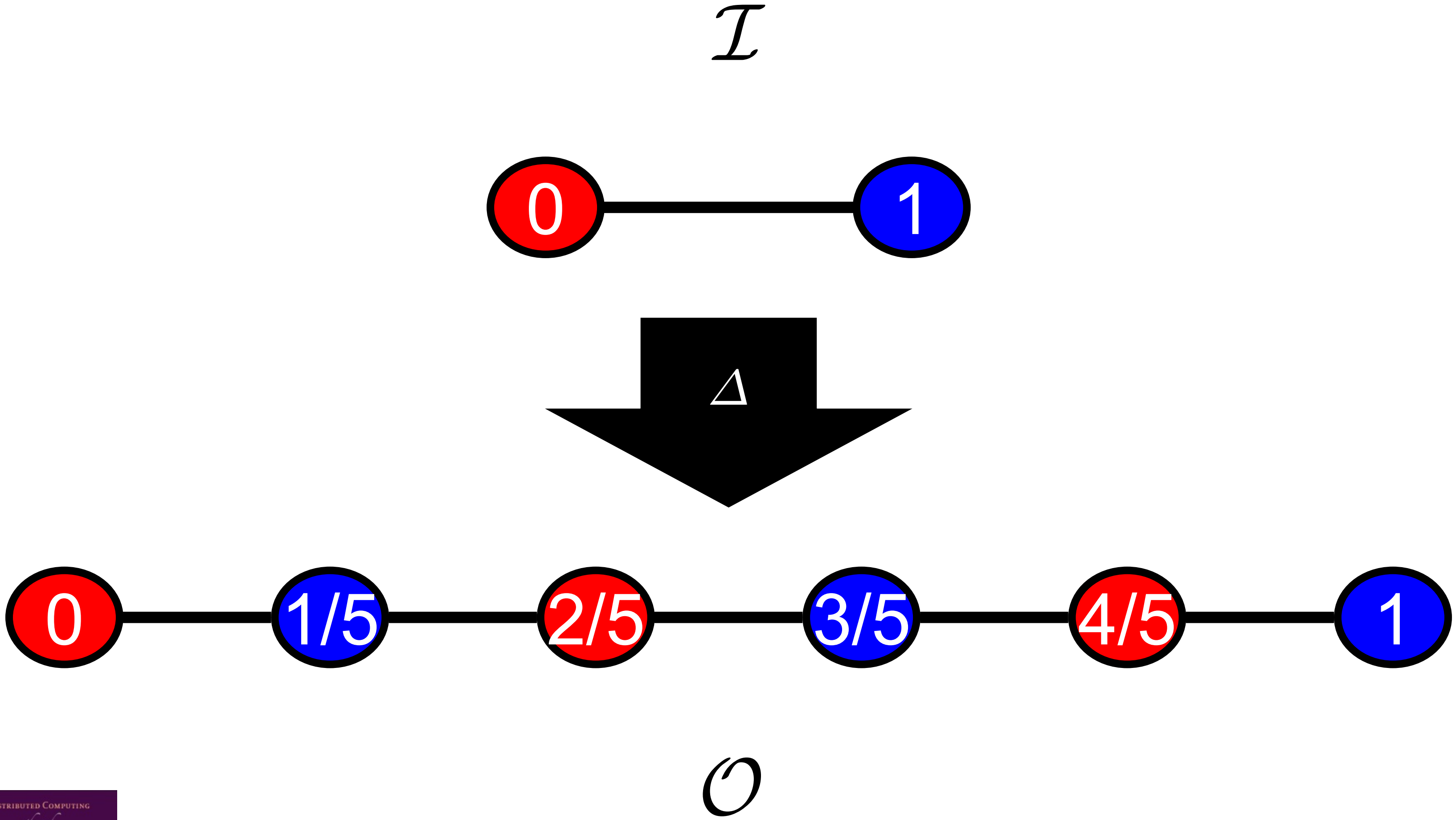


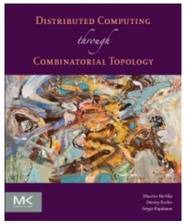
Alice and Bob realize that they do not need to agree on an *exact* time …

they will win if attack times are *sufficiently close*.

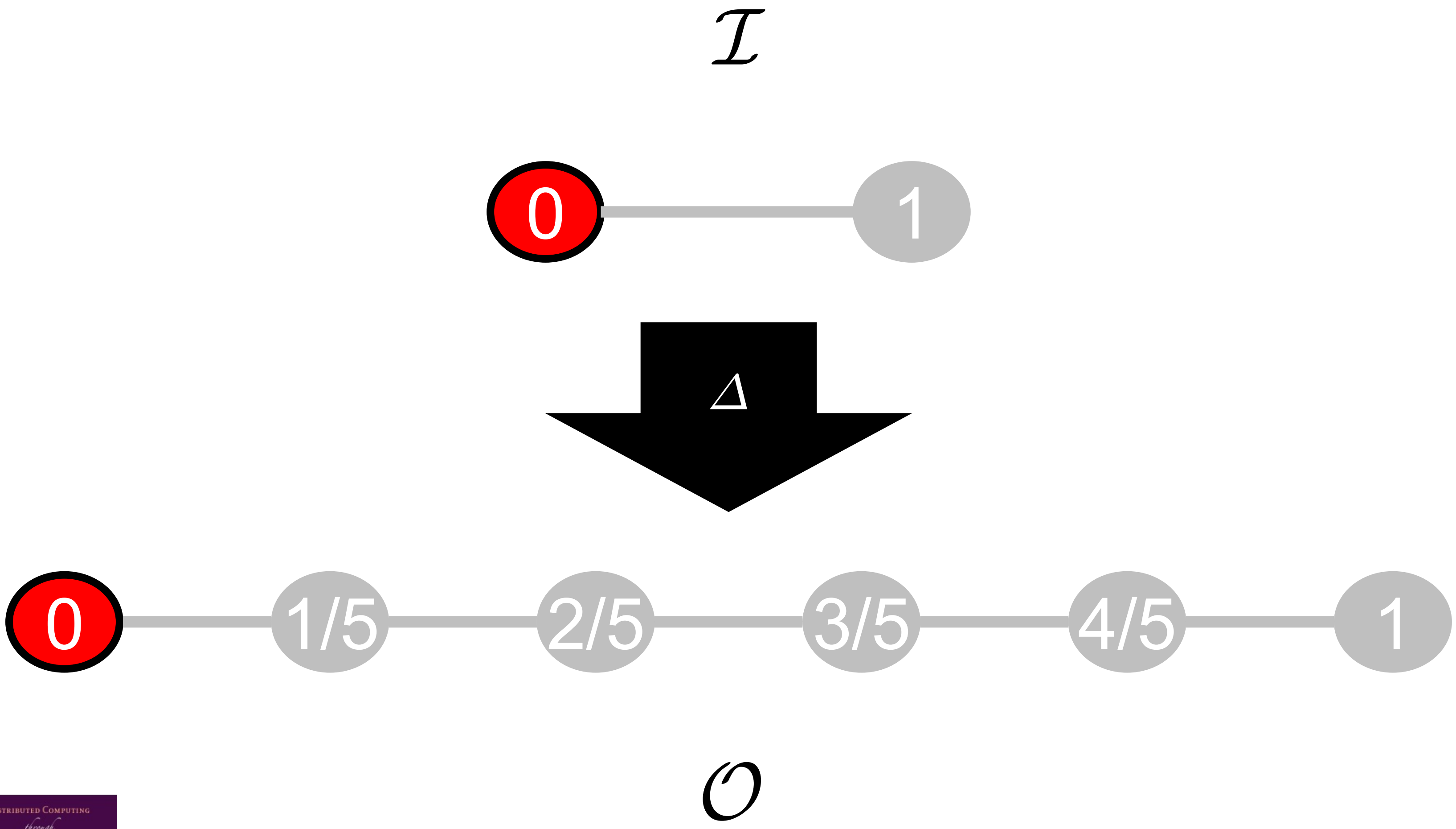


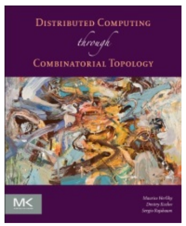
Coordinated Attack Graphs



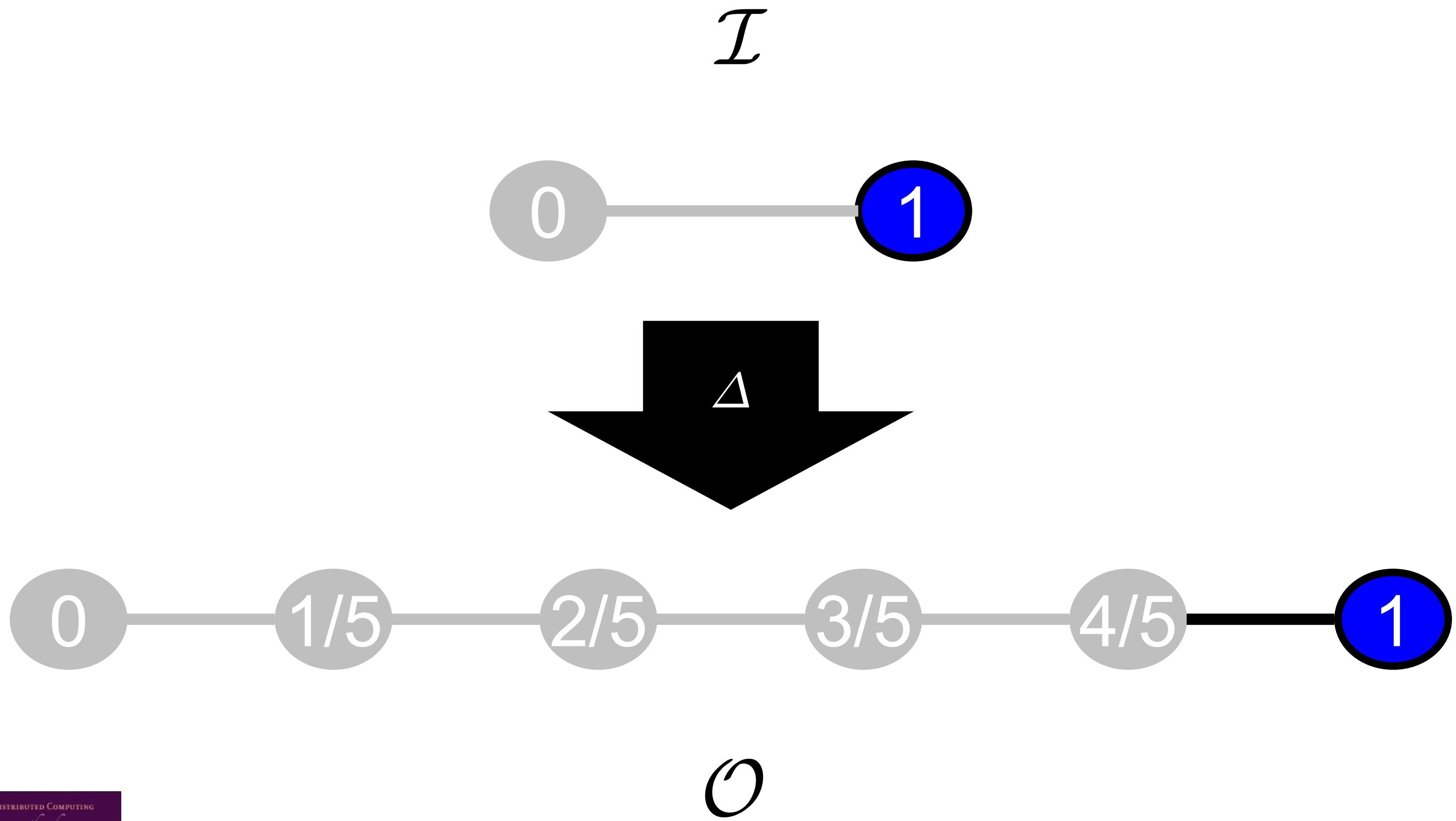


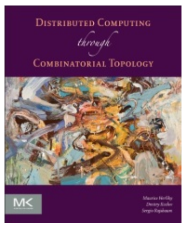
Coordinated Attack Graphs



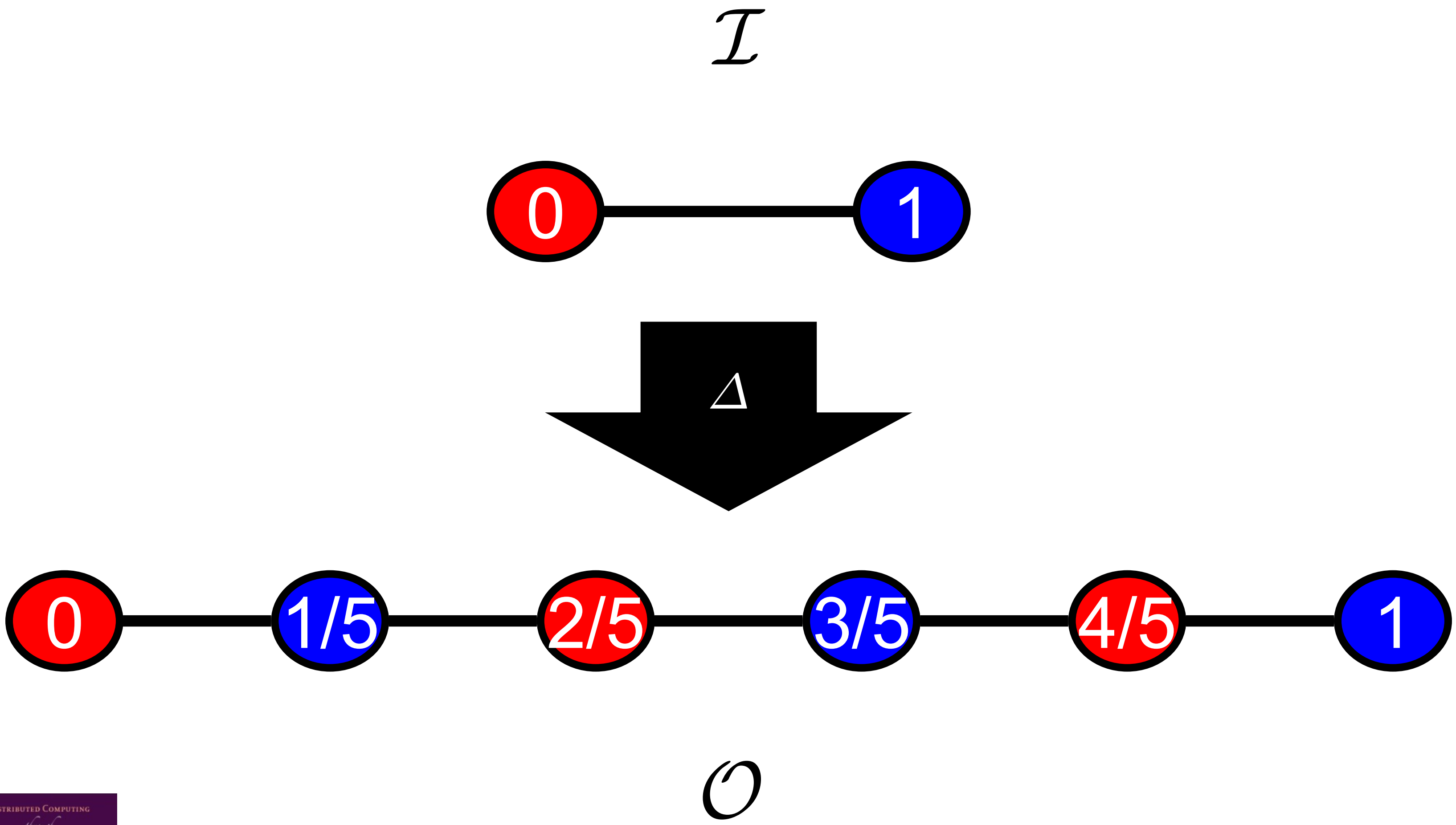


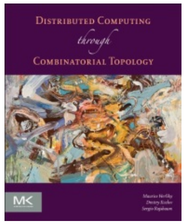
Coordinated Attack Graphs





Coordinated Attack Graphs





Road Map

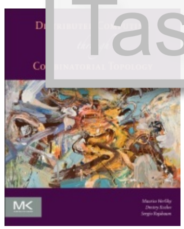
Elementary Graph Theory

Tasks

Models of Computation

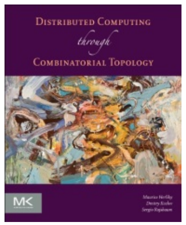
Approximate Agreement

Task Solvability



Protocols

Models of Computation

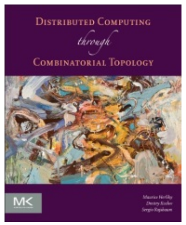


Alice's Protocol

```
shared mem array 0..1 of Value
view: Value := my input value;
for i: int := 0 to L do
   mem[A] := view;
   view := view + mem[A] + mem[B];
return δ(view)
```

Finite program

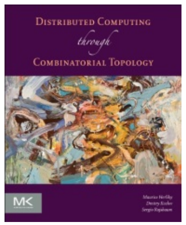
Bob's protocol is symmetric



Alice's Protocol

```
shared mem array 0..1 of Value
v                                    value;
for i: int := 0 to L do
  mem[A] := view;
  view := view + mem[B];
return δ(view)
```

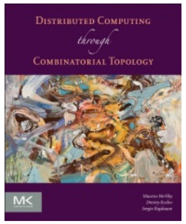
shared two-element memory



Alice's Protocol

```
shared mem array 0..1 of Value
view: Value := my input value;
for i: int i := 0 to 1 do
  mem[A] := view;
  view := view + mem[B];
return δ(view)
```

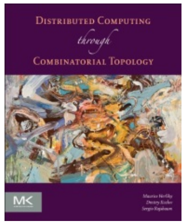
Start with input value



Alice’s Protocol

```
shared mem array 0..1 of Value
view: Value := my input value;
for i: int := 0 to L do
   mem[A] := view;
   view := view + mem[B];
return δ(view)
```

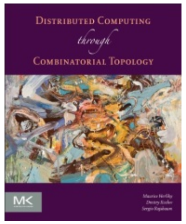
Run for **L** rounds



Alice's Protocol

```
shared mem array 0..1 of Value
view: Value := my input value;
for i: int := 0 to L do
   mem[A] := view;
   view := view + mem[B];
return
```

Alice writes her value, read Bob's value, and concatenate it to her view

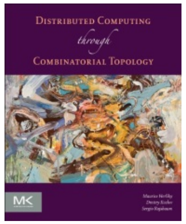


Alice's Protocol

```
shared mem array 0..1 of Value
view: Value := my input value;
for i: int := 0 to L do
   mem[A] := view;
   view := view + mem[B];
return
```

Alice writes her value, read Bob's value, and concatenate it to her view

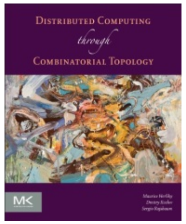
(full-information protocol)



Alice's Protocol

```
shared mem array 0..1 of Value
view: Value := my input value;
for i: int := 0 to L do
  mem[A] := view;
  view := view + mem[B];
return δ(view)
```

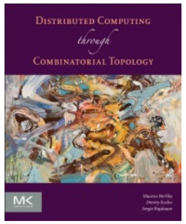
finally, apply task-specific decision map to view



Formal Protocol Definition

Input graph $\mathcal{I}$

all possible assignments of input values



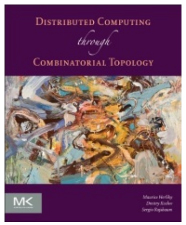
Formal Protocol Definition

Input graph $\mathcal{I}$

all possible assignments of input values

Protocol graph $\mathcal{P}$

all possible process views after execution



Formal Protocol Definition

Input graph $\mathcal{I}$

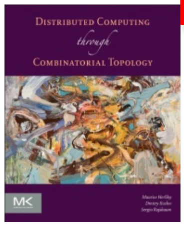
all possible assignments of input values

Protocol graph $\mathcal{P}$

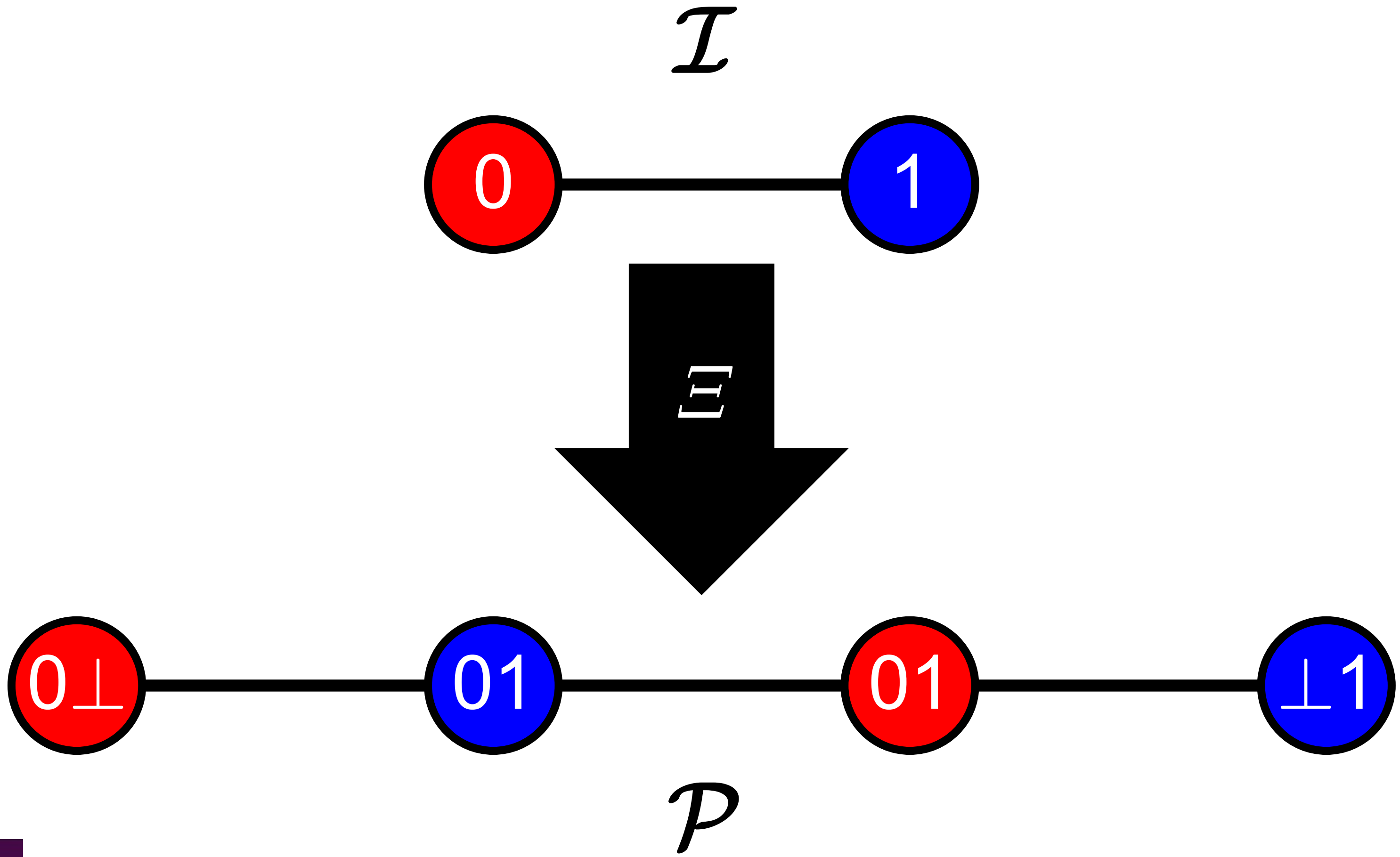
all possible process views after execution

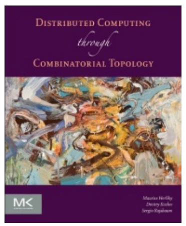
Carrier map Ξ: $\mathcal{I} \to 2^{\mathcal{P}}$

all possible assignments of views

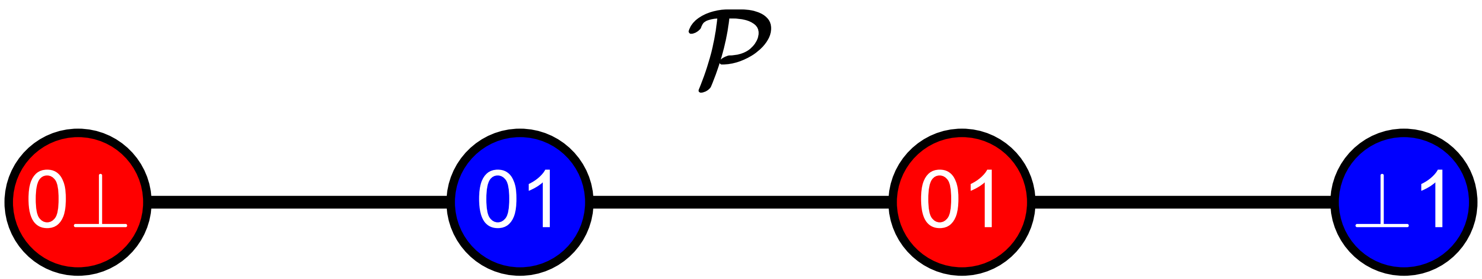


One-Round Protocol Graph



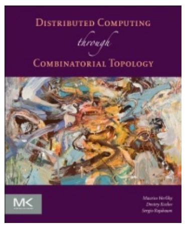


One-Round Protocol Graph

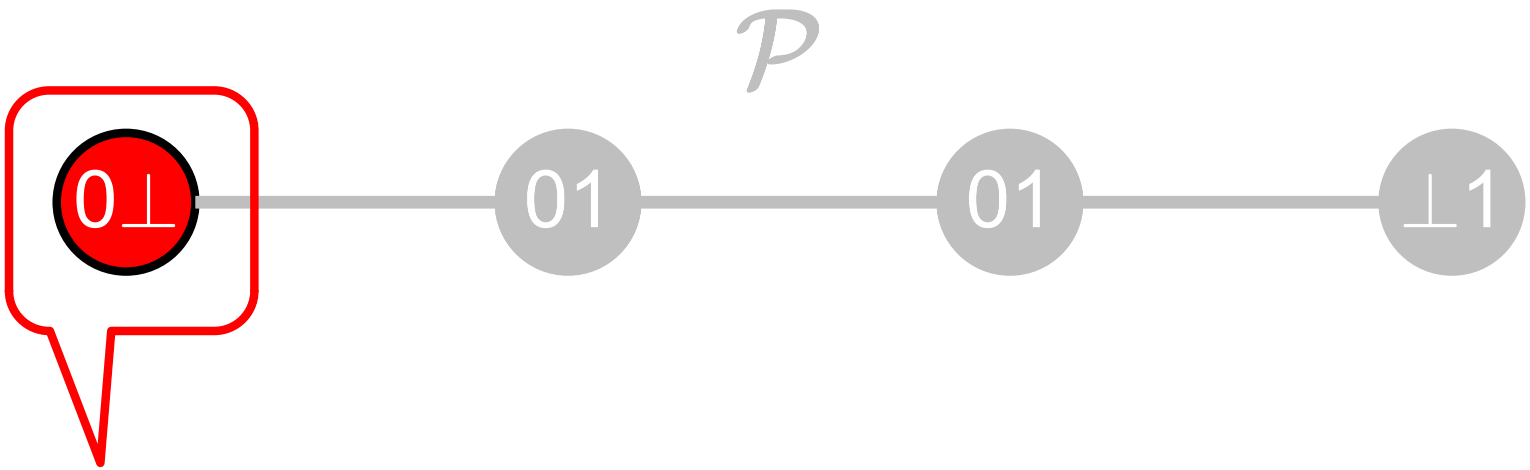


Colored by process names

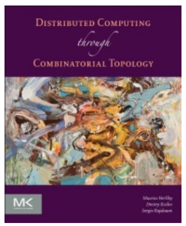
Labeled with final views



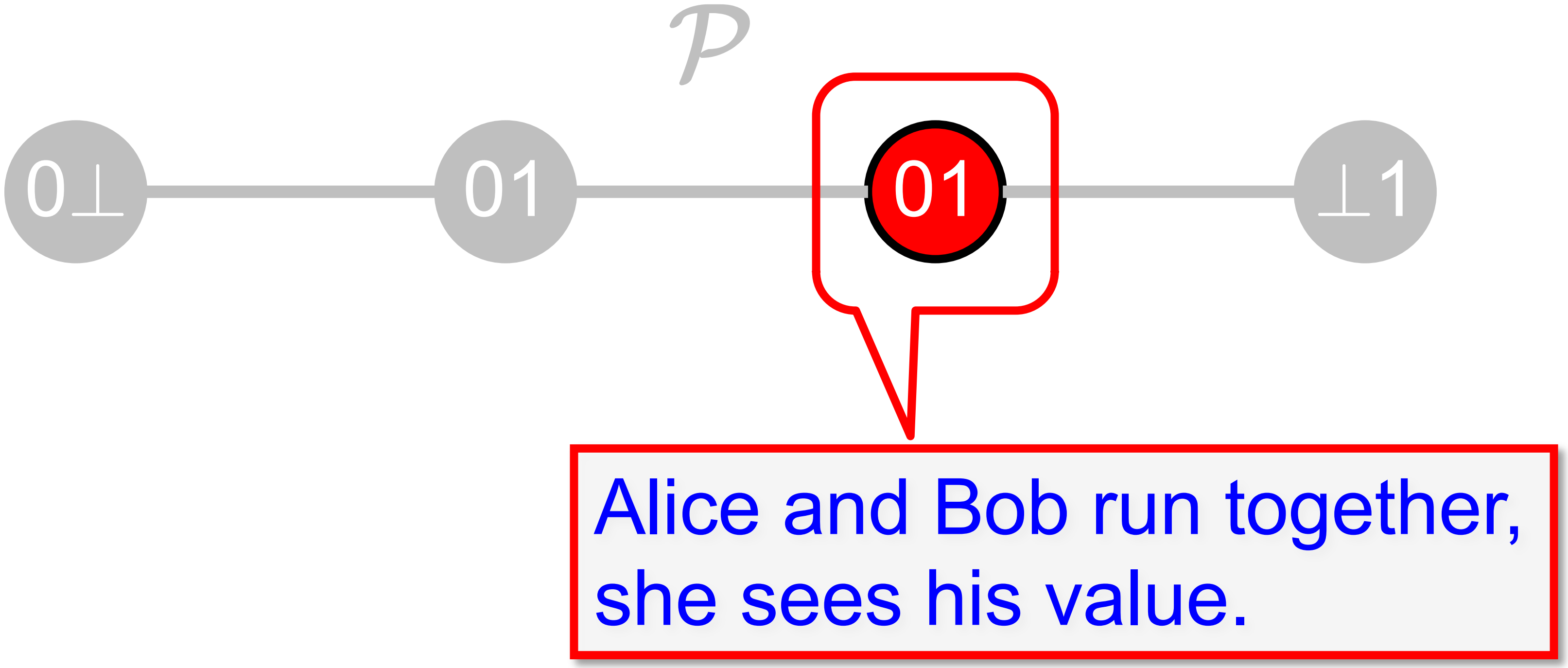
One-Round Protocol Graph

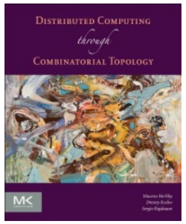


Alice finishes before Bob starts, doesn’t see his value

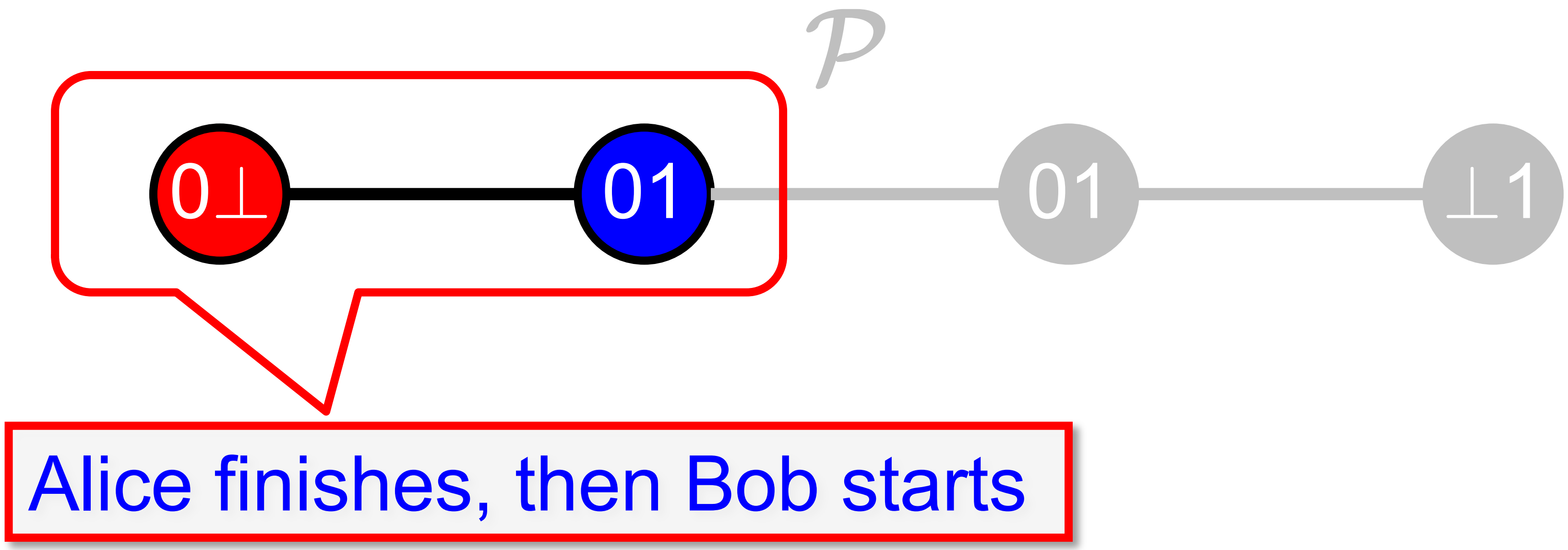


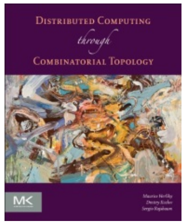
One-Round Protocol Graph



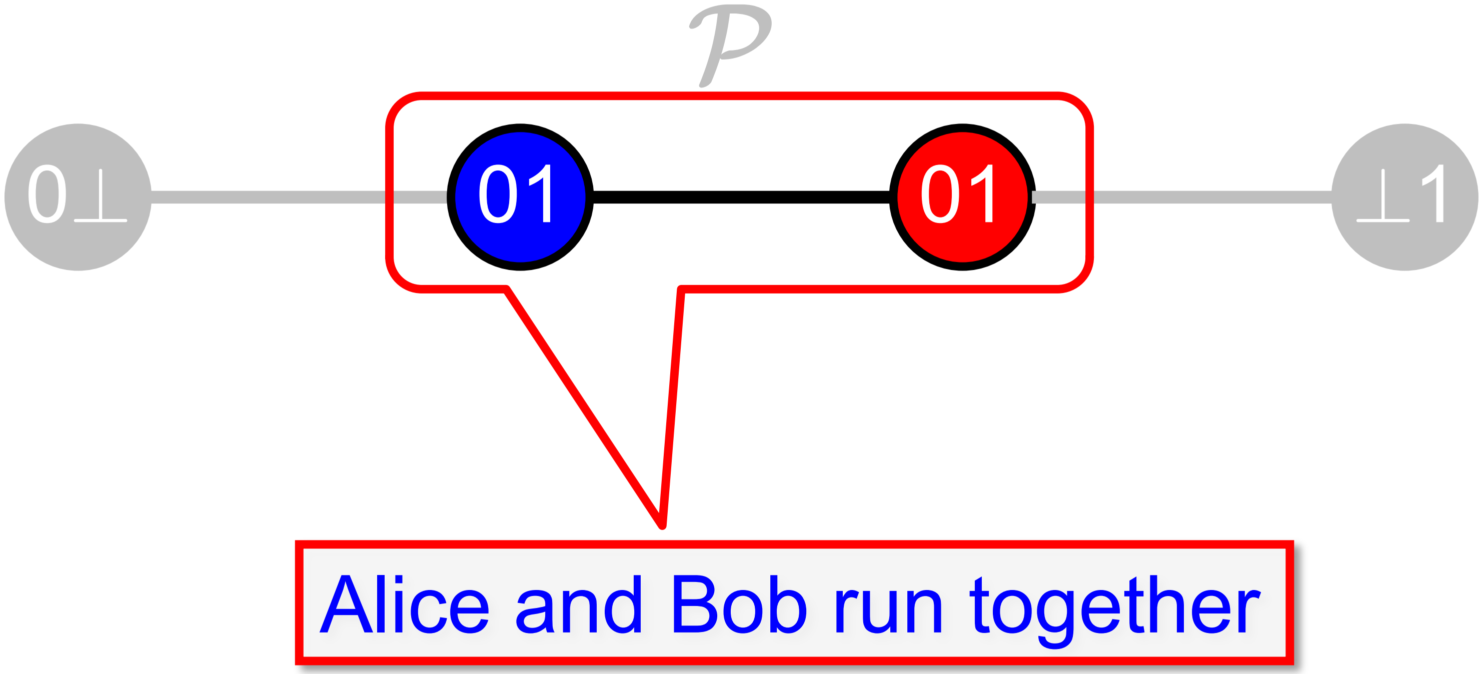


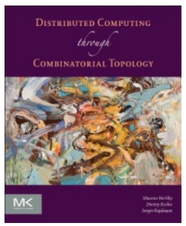
One-Round Protocol Graph



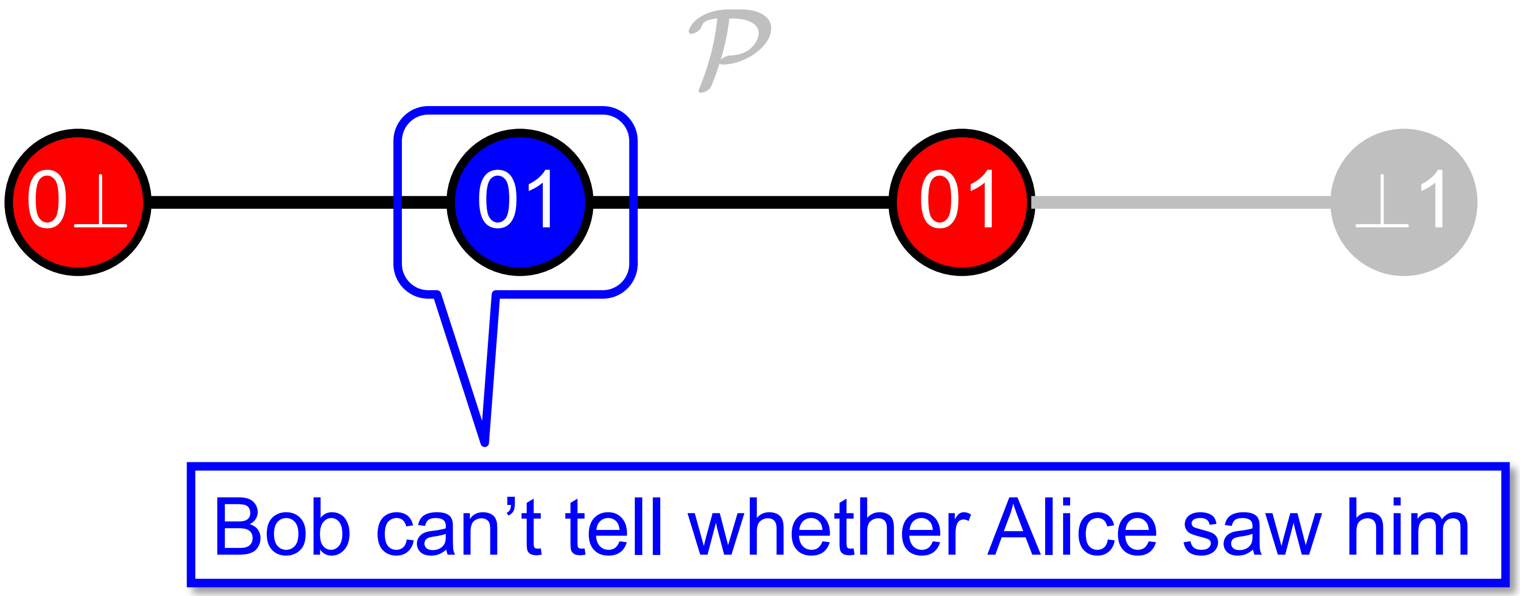


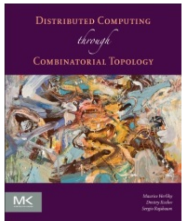
One-Round Protocol Graph



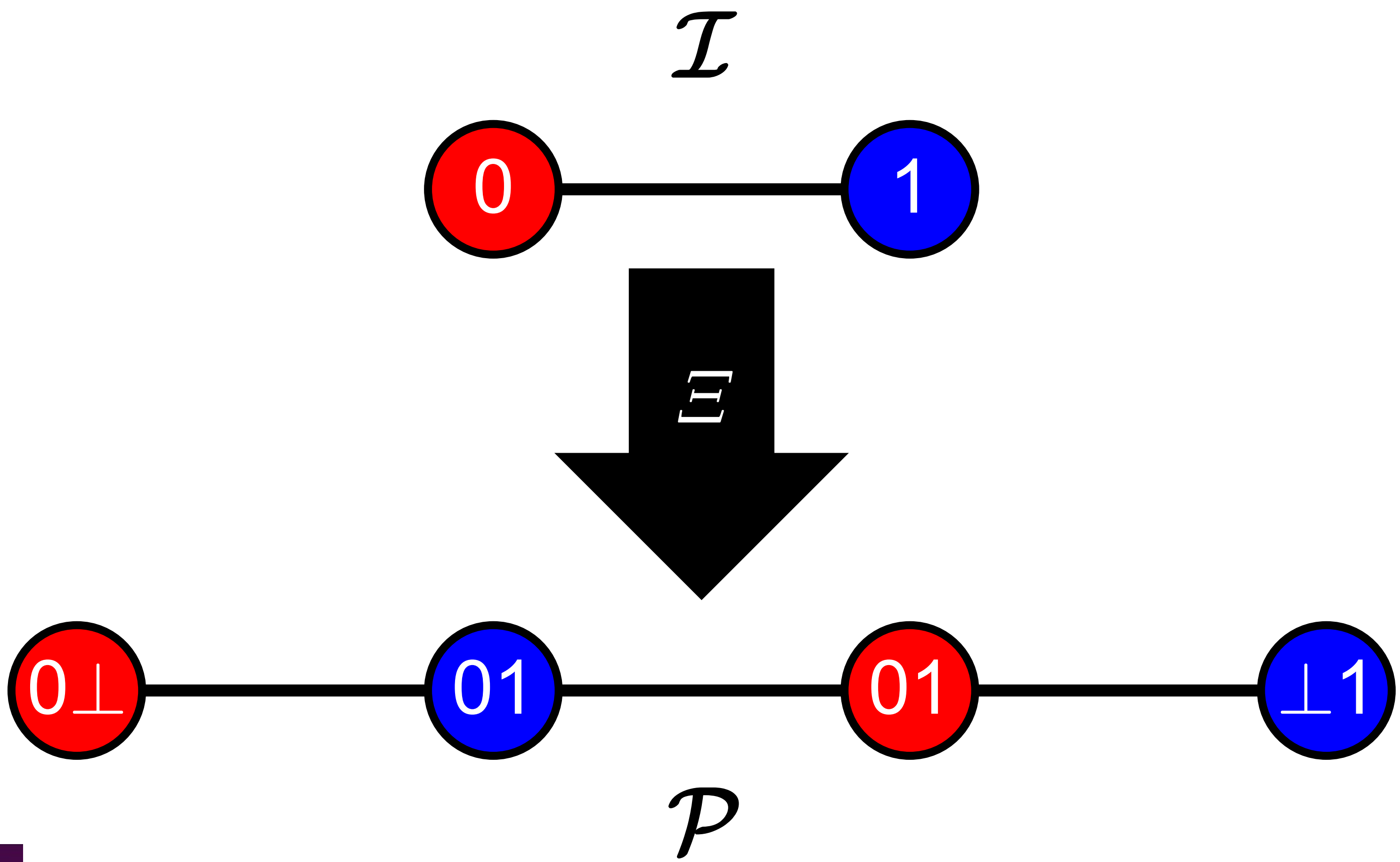


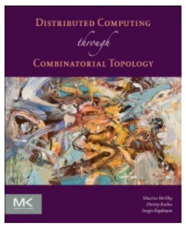
One-Round Protocol Graph



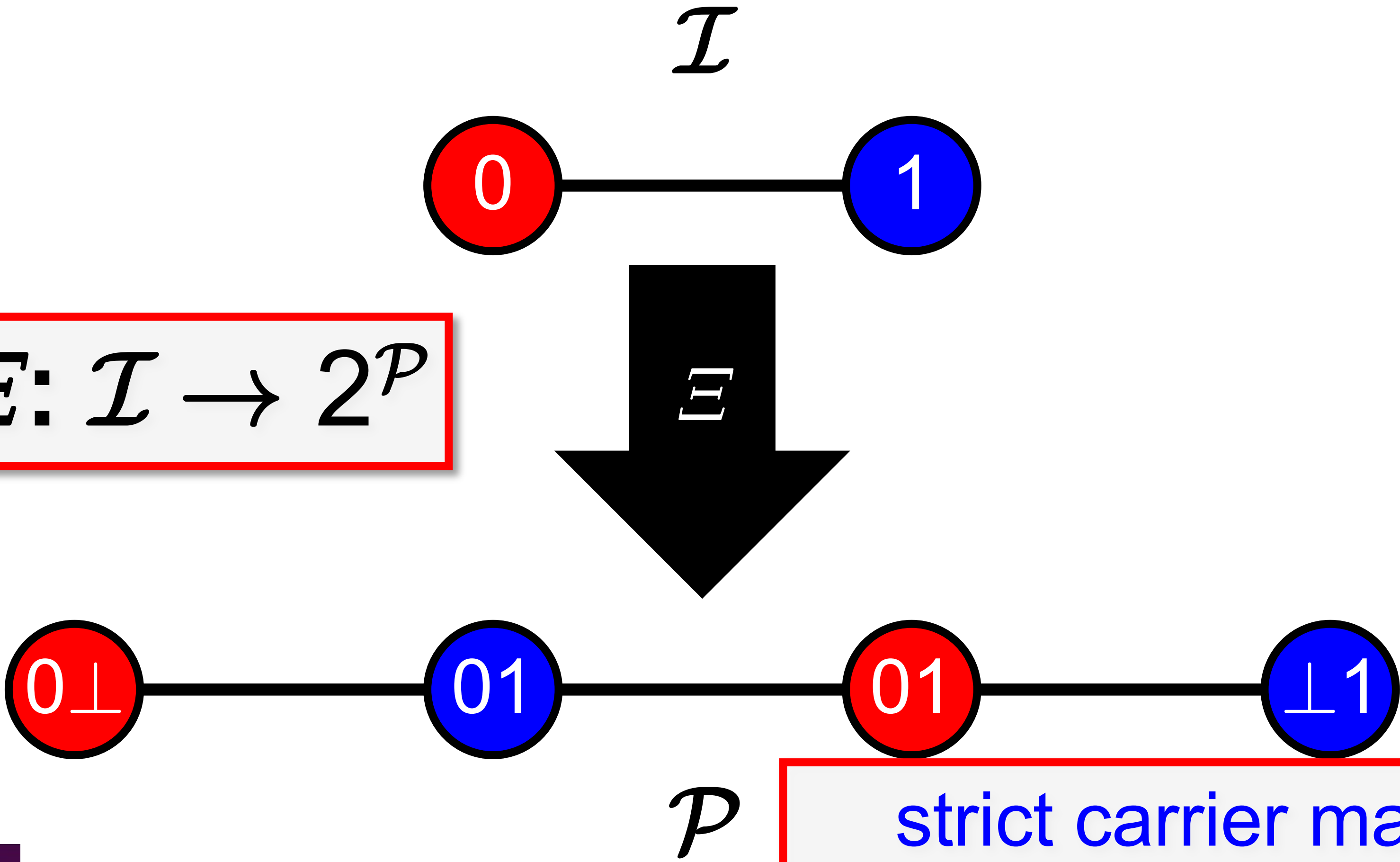


Execution Carrier Map





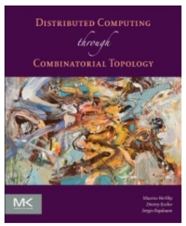
Execution Carrier Map



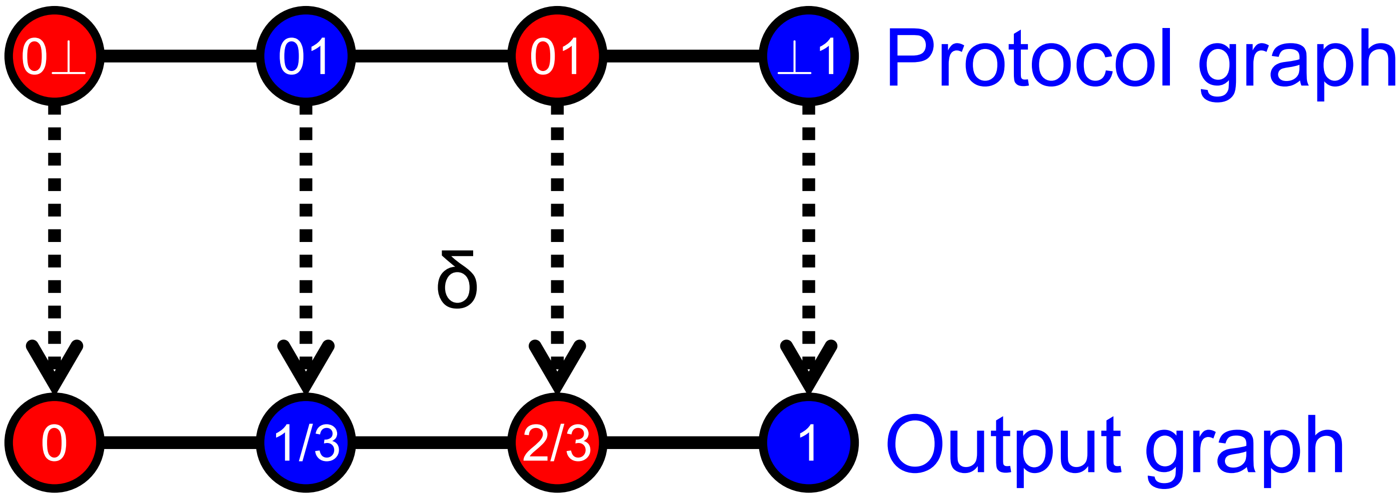
$\Xi \colon \mathcal{I} \to 2^{\mathcal{P}}$

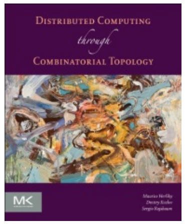
strict carrier map

$\Xi(\sigma) \cap \Xi(\tau) = \Xi(\sigma \cap \tau)$

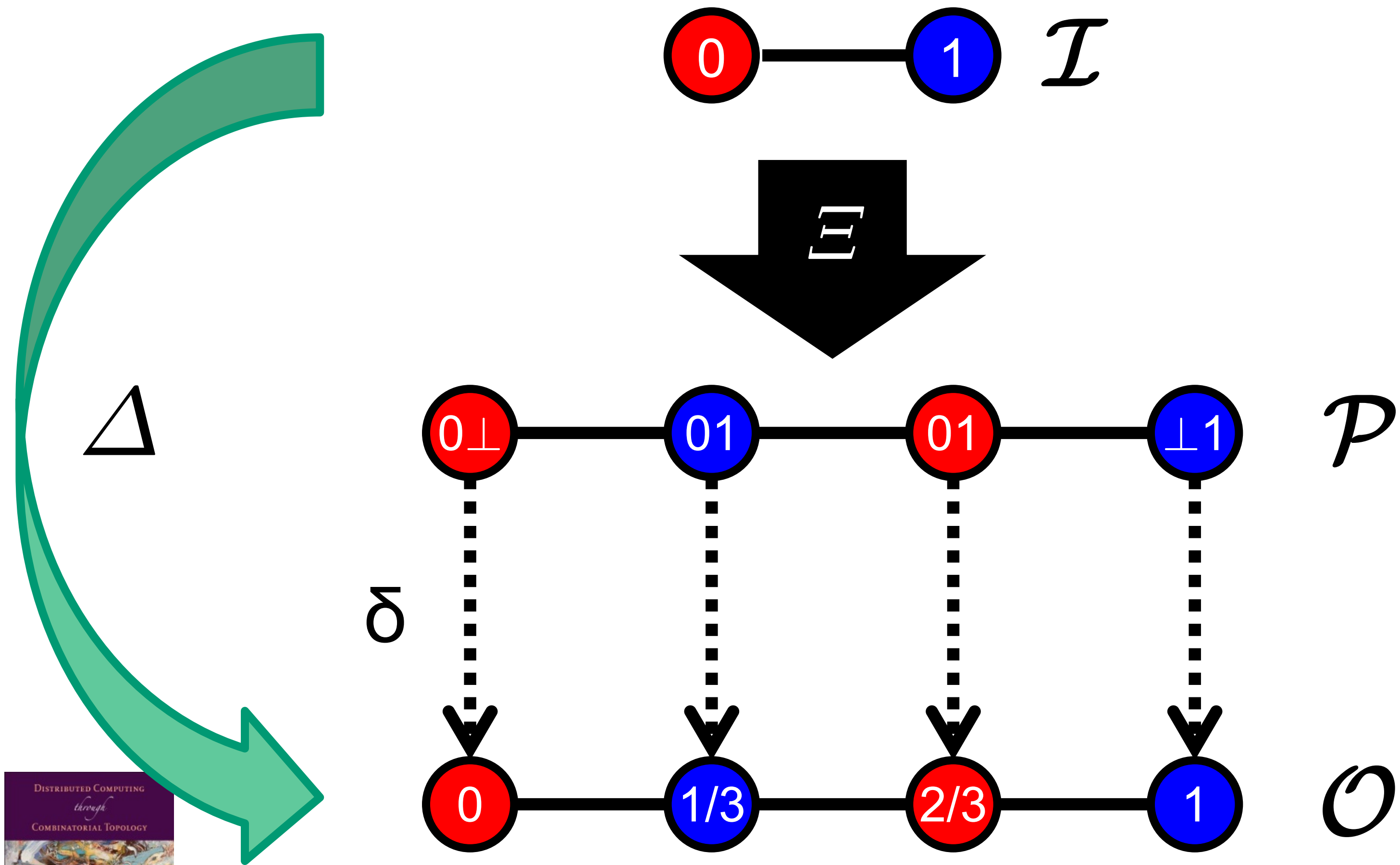


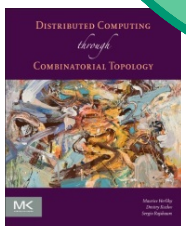
The Decision Map





All Together





Definition

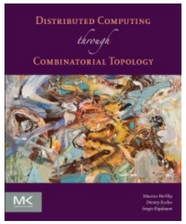
Decision map δ (of protocol Ξ) is *carried by* carrier map Δ if

for each input vertex s,

$\delta(\Xi(s)) \subseteq \Delta(s)$

for each input edge σ,

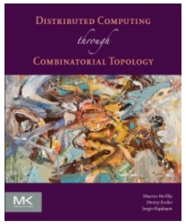
$\delta(\Xi(\sigma)) \subseteq \Delta(\sigma)$.



Meaning

$$\delta(\Xi(s)) \subseteq \Delta(s)$$

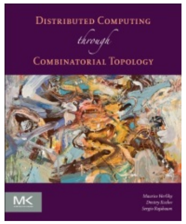
process starts in state s



Meaning

$$\delta(\Xi(s)) \subseteq \Delta(s)$$

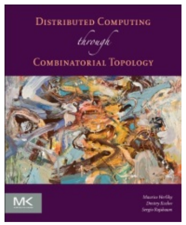
runs the protocol to completion



Meaning

$$\delta(\Xi(s)) \subseteq \Delta(s)$$

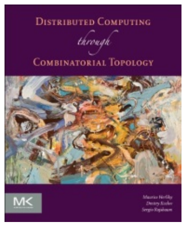
makes a decision …



Meaning

$$\delta(\Xi(s)) \subseteq \Delta(s)$$

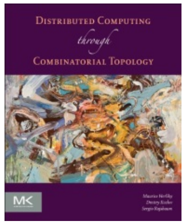
decision is permitted by task carrier map



Solving a Task

Definition

The protocol $(\mathcal{I},\mathcal{P},\Xi)$ *solves* the task $(\mathcal{I}, \mathcal{O}, \Delta)$

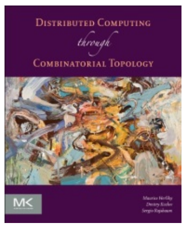


Solving a Task

Definition

The protocol $(\mathcal{I},\mathcal{P},\Xi)$ *solves* the task $(\mathcal{I}, \mathcal{O}, \Delta)$

if there is …



Solving a Task

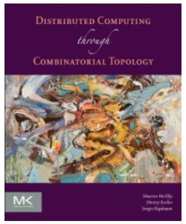
Definition

The protocol $(\mathcal{I},\mathcal{P},\Xi)$ *solves* the task $(\mathcal{I}, \mathcal{O}, \Delta)$

if there is …

a simplicial decision map

$\delta:\mathcal{P} \to \mathcal{O}$



Solving a Task

Definition

The protocol $(\mathcal{I},\mathcal{P},\Xi)$ *solves* the task $(\mathcal{I}, \mathcal{O}, \Delta)$

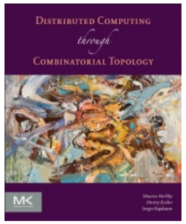
if there is …

a simplicial decision map

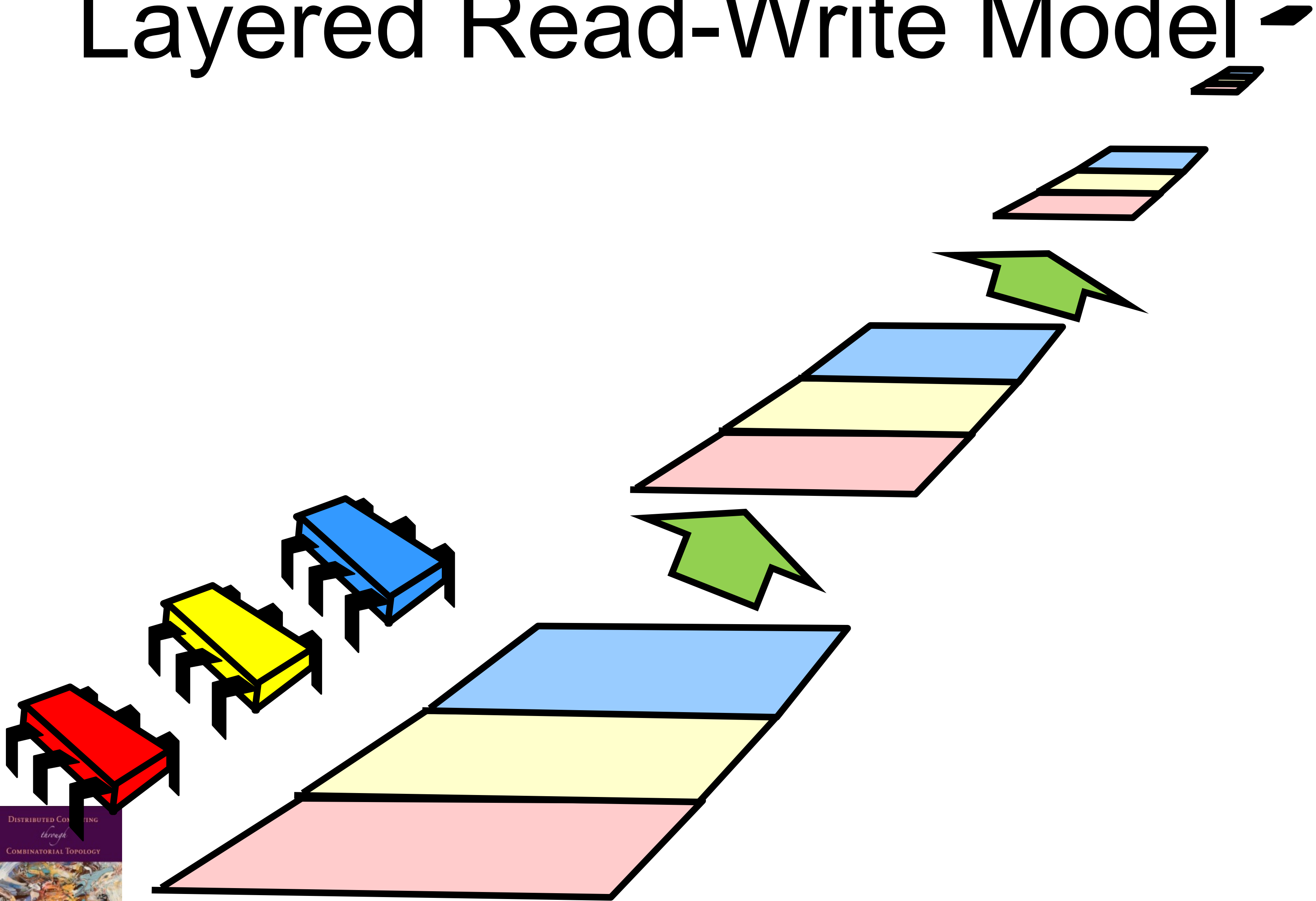
$\delta : \mathcal{P} \to \mathcal{O}$

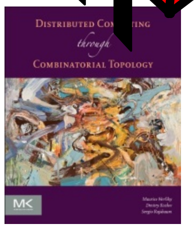
such that δ is carried by Δ.

(δ agrees with Δ)



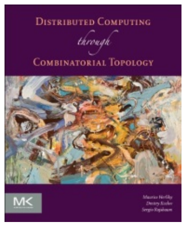
Layered Read-Write Model





Layered Read-Write Protocol (Alice)

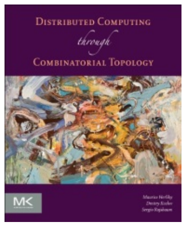
```
shared mem array 0..1,0..L of Value
view: Value := my input value;
for i: int := 0 to L do
   mem[i][A] := view;
   view := view + mem[i][A] + mem[i][B];
return δ(view)
```



Layered Read-Write Protocol (Alice)

```
shared mem array 0..1,0..L of Value
view: Value := my input value;
for i: int := 0 to L do
   mem
   view := view + mem[i][A] + mem[i][B];
return δ(view)
```

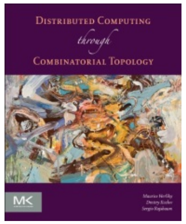
As before, run for `L` *layers*



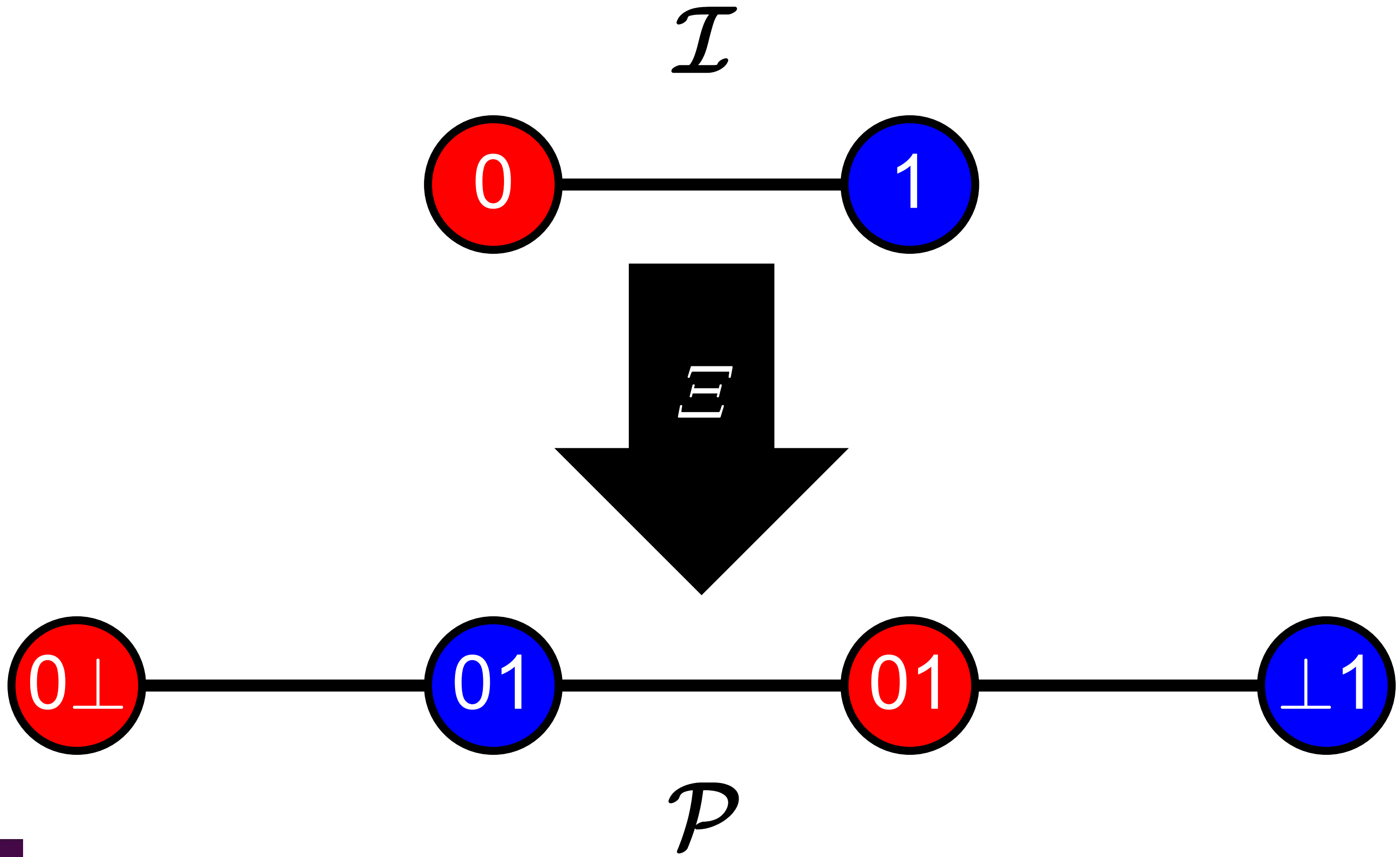
Layered Read-Write Protocol (Alice)

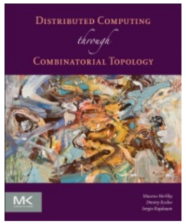
```
shared mem array 0..1,0..L of Value
view: Value := my input value;
for i: int := 0 to L do
   mem[i][A] := view;
   view := view + mem[i][A] + mem[i][B];
return δ(view)
```

Each layer uses a distinct, “clean” memory

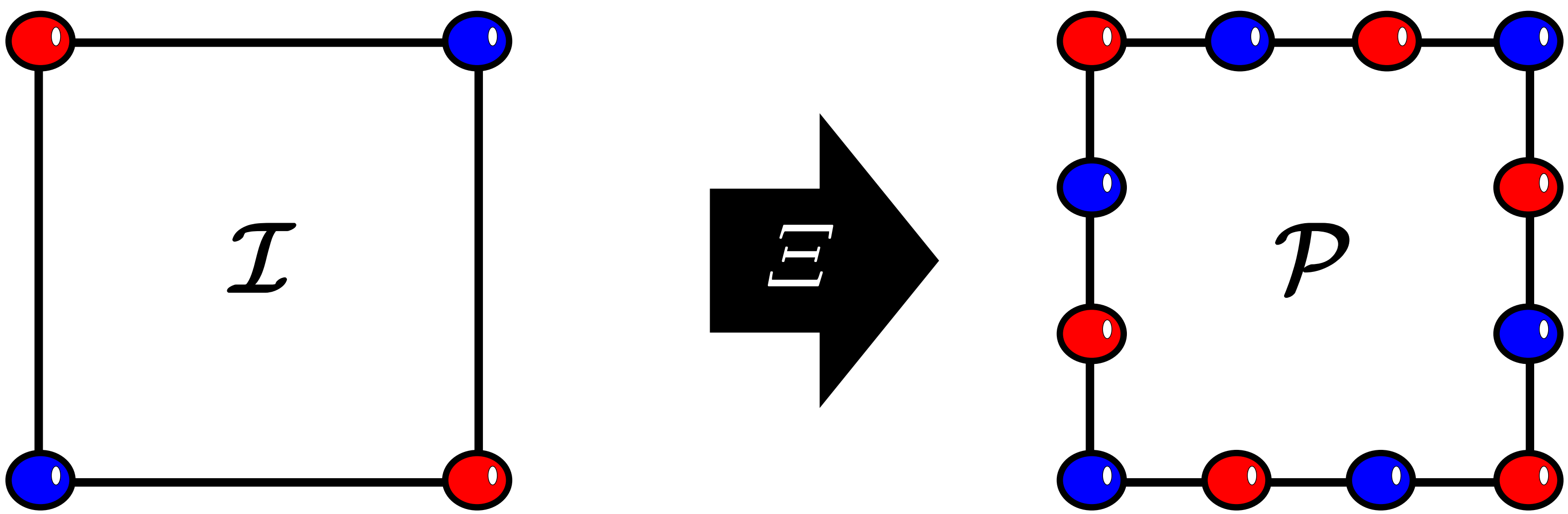


Layered R-W Protocol Graph

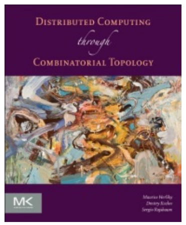




Layered R-W Protocol Graph



$\mathcal{P}$ is always a subdivision of $\mathcal{I}$



Road Map

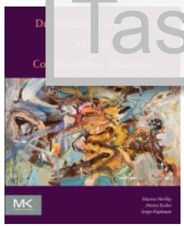
Elementary Graph Theory

Tasks

Models of Computation

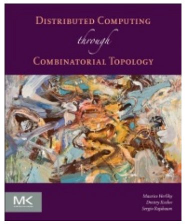
Approximate Agreement

Task Solvability



Alice's 1/3-Agreement Protocol

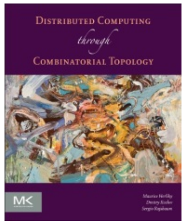
```
mem[A] := 0
other := mem[B]
if other == ⊥ then
  decide 0
else
  decide 2/3
```



Alice's 1/3-Agreement Protocol

```
mem[A] := 0
if mem[B] == ⊥ then
   decide 0
else
   decide 2/3
```

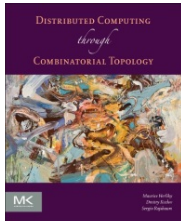
Alice writes her value to memory



Alice’s 1/3-Agreement Protocol

```
mem[A]  := 0
if mem[B] == ⊥ then
   decide 0
else
   decide 2/3
```

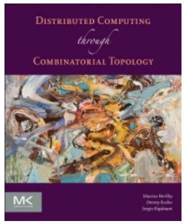
If she doesn’t see Bob’s value, decide her own.

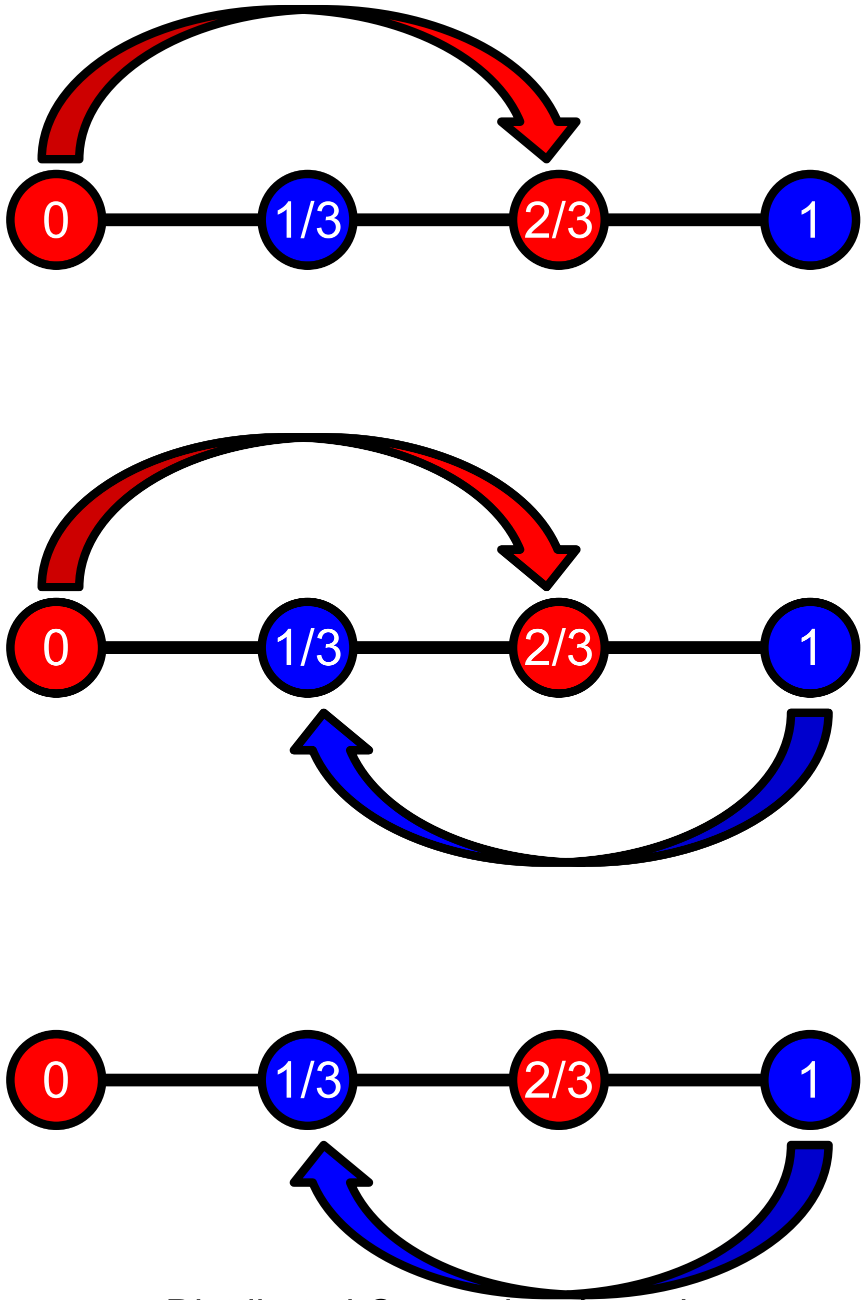


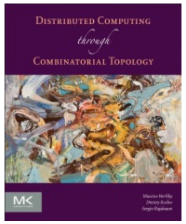
Alice's 1/3-Agreement Protocol

```
mem[A] := 0
if mem[B] == ⊥ then
   decide 0
else
   decide 2/3
```

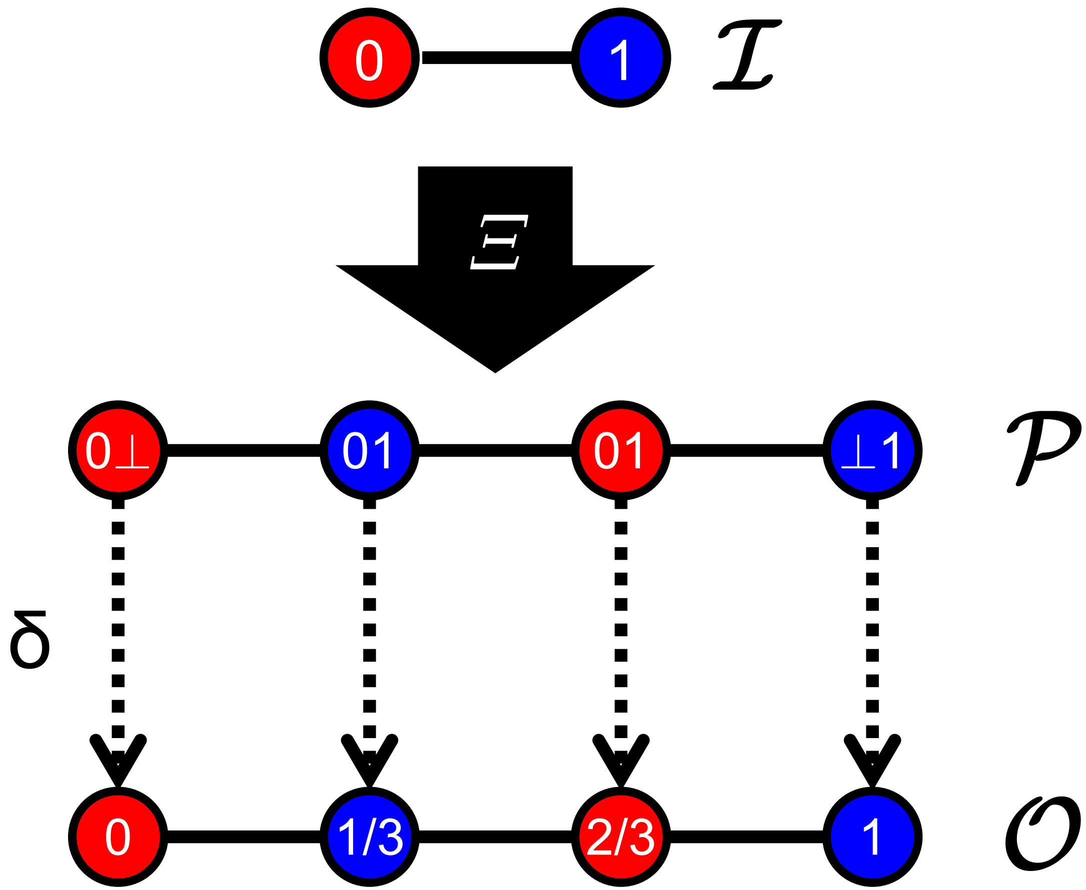
If she see's Bob's value, jump to the middle

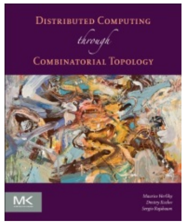



0
1/3
2/3
1
0
1/3
2/3
1
0
1/3
2/3
1

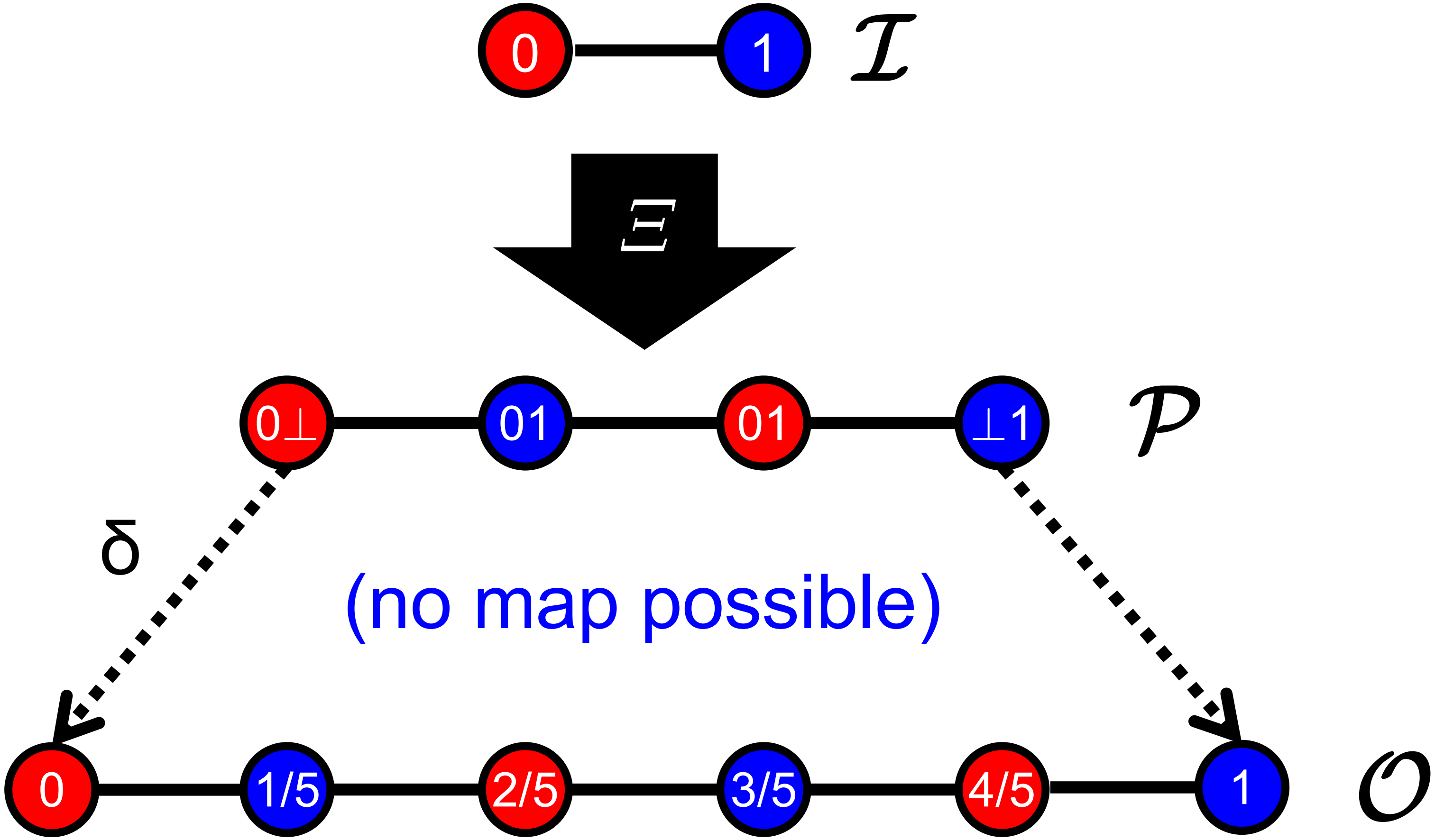

Distributed Computing
through
Combinatorial Topology

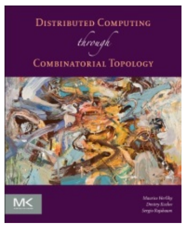
One-Layer 1/3-Agreement Protocol



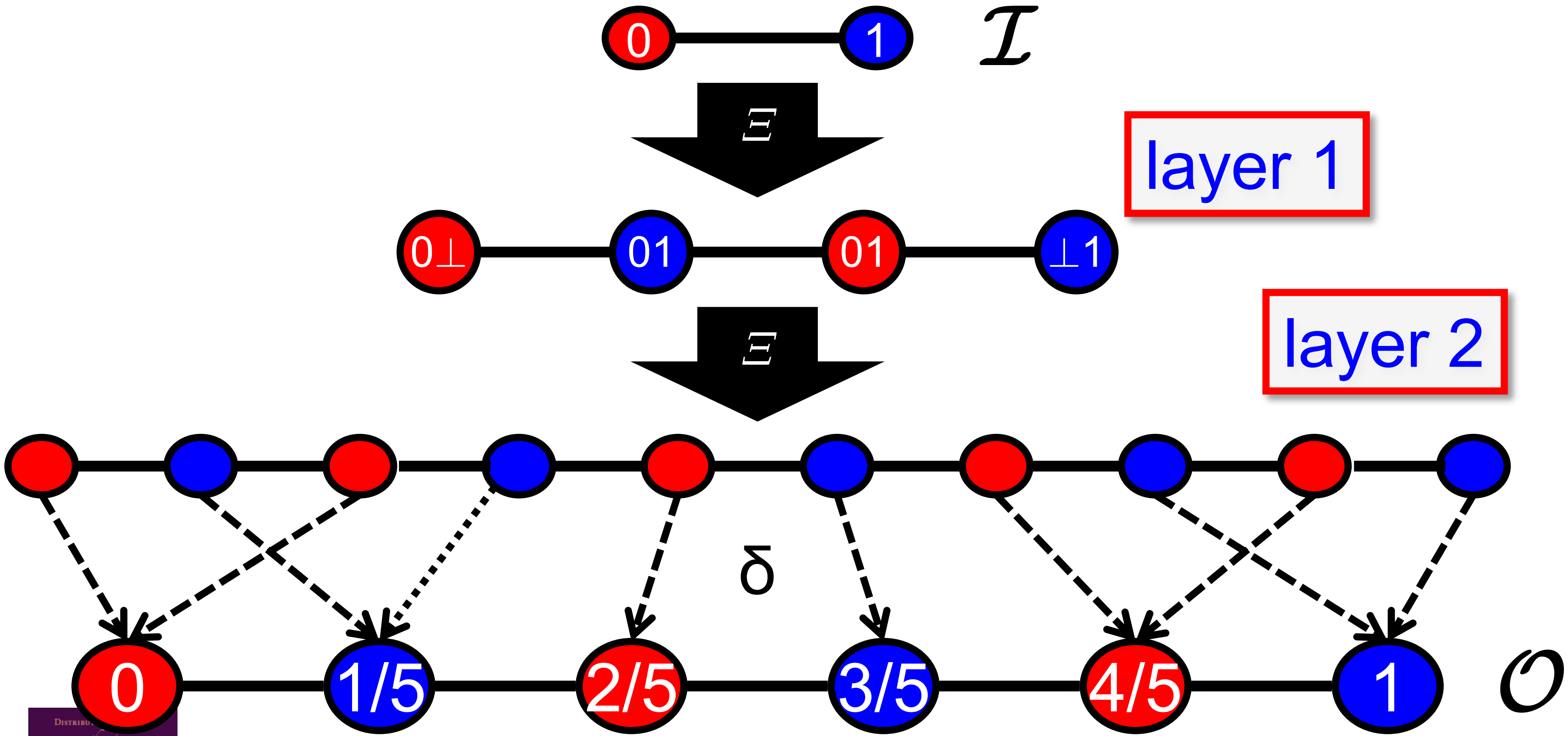


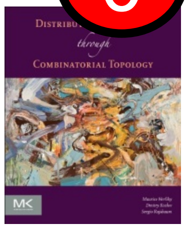
No 1-Layer 1/5-Agreement Protocol





2-Layer 1/5-Agreement



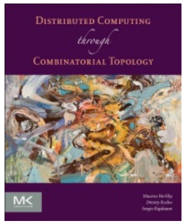


Fact

In the layered read-write model,

The 1/K-Agreement Task

Has a $\lceil \log_3 K \rceil$–layer protocol



Road Map

Elementary Graph Theory

Tasks

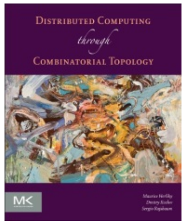
Models of Computation

Approximate Agreement

Task Solvability

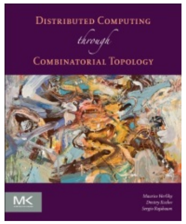
Fact

The protocol graph for any L-layer protocol with input graph $\mathcal{I}$ is a subdivision of $\mathcal{I}$, where each edge is subdivided 3^L times.



Main Theorem

The two-process task $(\mathcal{I}, \mathcal{O}, \Delta)$ is solvable in the layered read-write model if and only if there exists a connected carrier map $\Phi: \mathcal{I} \to 2^{\mathcal{O}}$ carried by Δ.



Proof sketch: the "if" part

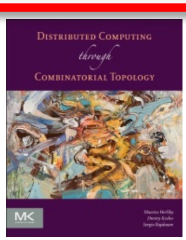
Let $\Phi: \mathcal{I} \to 2^{\mathcal{O}}$ be a connected carrier map carried by Δ.

For each edge $\sigma_i=(s_i,t_i)\in\mathcal{I}$, there is a path π_i in $\Phi(\sigma_i)$ connecting $\Phi(s_i)$ and $\Phi(t_i)$ (choosing just one vertex in each image is enough)

Approximate agreement on the path can be solved using $\lceil \log_3 L \rceil$ layers where L is $\max_{i\in\mathcal{I}} |\pi_i|$

For edges *(s,t)*, *(s,u)*: "glue together" protocols for *(s,t)* and *(s,u)*: they agree on *s*.

The protocol is carried by Δ, so it solves T

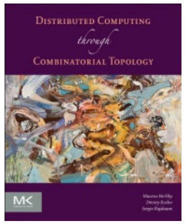


Proof sketch: the “only if” part

Let a layered protocol $\mathcal{P}$ solve T with a decision map δ

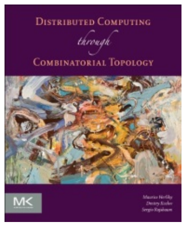
Let $\Xi: \mathcal{I} \to 2^{\mathcal{P}}$ be the protocol carrier map.

Then the composition $\Phi = \delta \circ \Xi$ is a connected carrier map $\mathcal{I} \to 2^{\mathcal{O}}$ carried by Δ (check Problem 3 in Exercise Set 2).



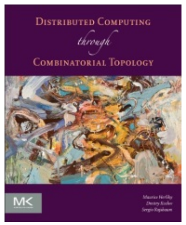
Corollary

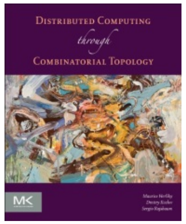
The consensus task has no layered read-write protocol



Corollary

Any ϵ–agreement task has a layered read-write protocol




Distributed Computing
through
Combinatorial Topology
MK