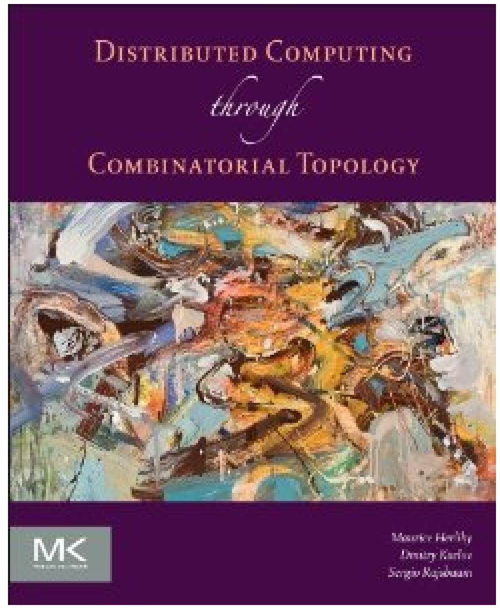
Solvability of Colorless Tasks in Different Models



Companion slides for

Distributed Computing Through Combinatorial Topology

Maurice Herlihy & Dmitry Kozlov & Sergio Rajsbaum

Road Map

Overview of Models

t-resilient layered snapshot models

Layered Snapshots with *k*-set agreement

Adversaries

Message-Passing Systems

Decidability

Road Map

Overview of Models

t-resilient layered snapshot models

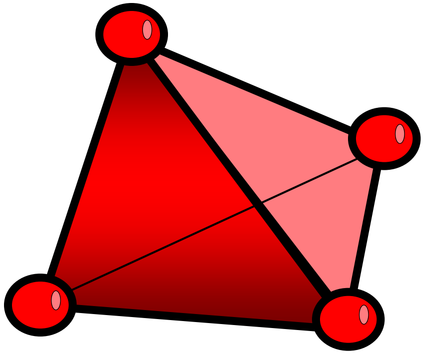
Layered Snapshots with *k*-set agreement

Adversaries

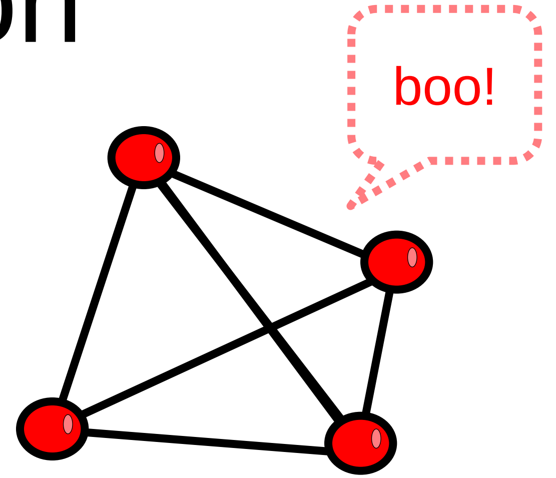
Message-Passing Systems

Decidability

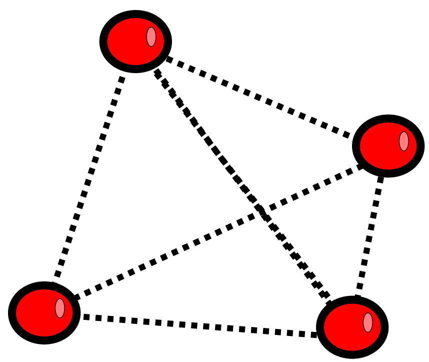
Skeleton



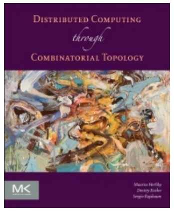
$\mathcal{C}$



skel^1 $\mathcal{C}$



skel^0 $\mathcal{C}$



Parameter *p*

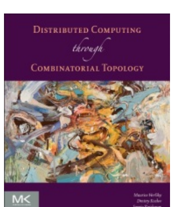
Model characterized by parameter p, $0 \leq p \leq n$

$(\mathcal{I},\mathcal{O},\Delta)$ has a wait-free protocol iff

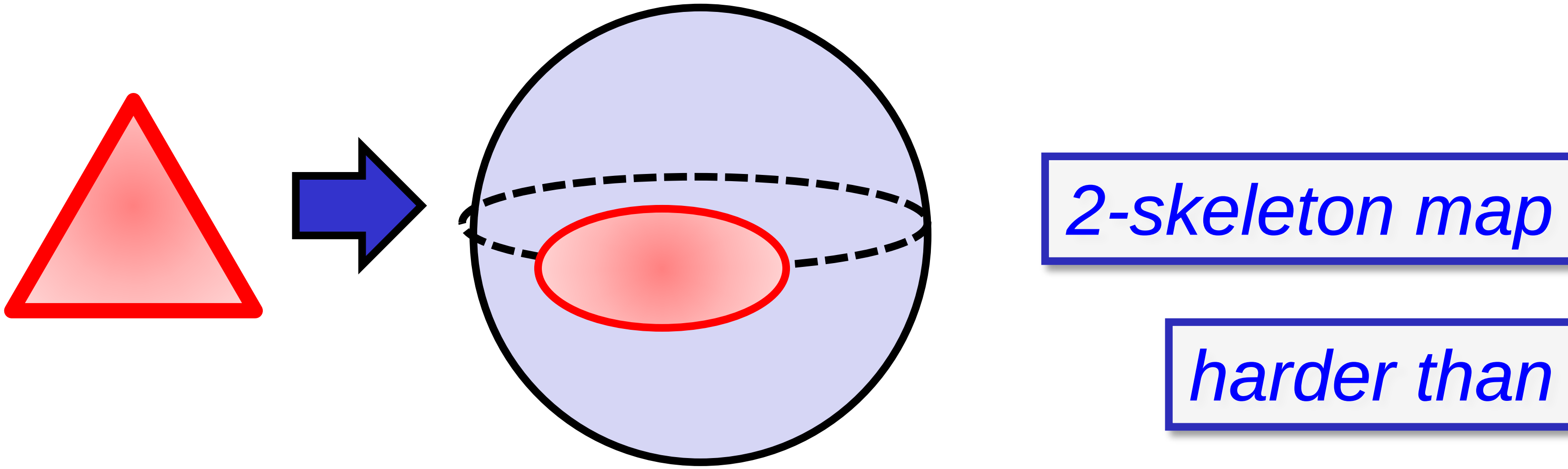
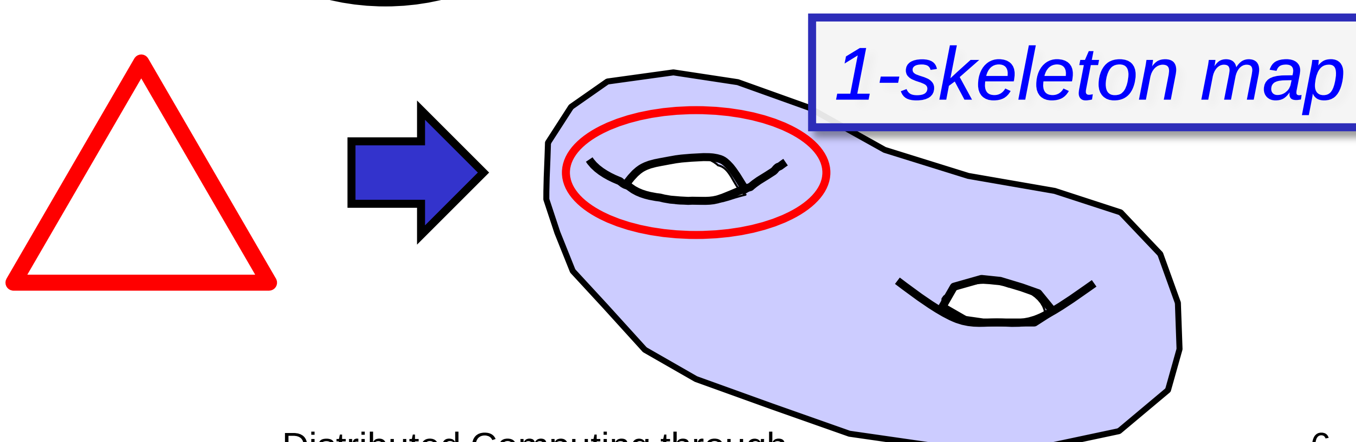
there is a continuous map

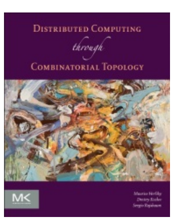
$f: |\text{skel}^p\, \mathcal{I}| \rightarrow |\mathcal{O}|$

carried by Δ.



Dimension of Skeleton map vs Computational Power



Wait-Free Layered Immediate Snapshots

Up to n out of $n+1$ can crash

Just can't wait (to be king)

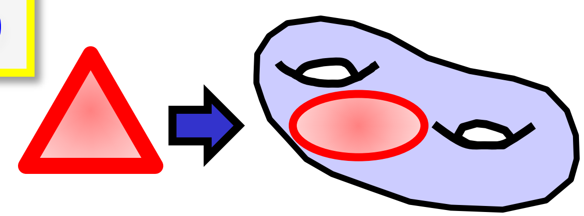
$(\mathcal{I},\mathcal{O},\Delta)$ has a wait-free protocol …

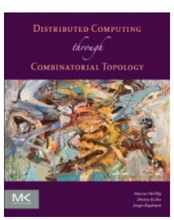
if and only if …

there is a continuous map

$f: |\text{skel}^n\, \mathcal{I}| \rightarrow |\mathcal{O}|$

carried by Δ.





t-resilient Layered Immediate Snapshots

Up to t out of n+1 can crash

OK to wait for n-t+1

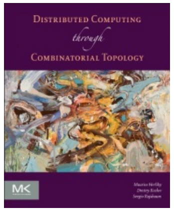
$(\mathcal{I},\mathcal{O},\Delta)$ has a wait-free protocol …

if and only if …

there is a continuous map

$f: |\text{skel}^t\, \mathcal{I}| \to |\mathcal{O}|$

carried by Δ.



Wait-Free Layered Immediate Snapshot with *k*-set Agreement

shared black boxes that solve *k*-set agreement

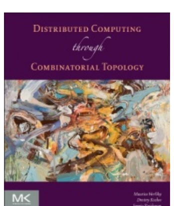
$(\mathcal{I},\mathcal{O},\Delta)$ has a wait-free protocol …

if and only if …

there is a continuous map

f: $|\text{skel}^{k-1}\,\mathcal{I}| \rightarrow |\mathcal{O}|$

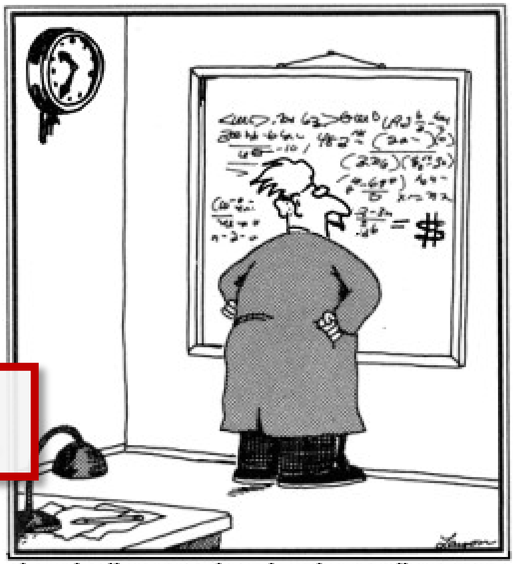
carried by Δ.



Equivalent Models

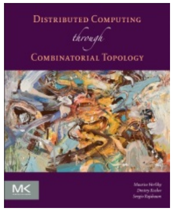
t-resilient model …

wait-free + *t+1*-set agreement …



Einstein discovers that time is actually money.

have *identical* computational power!

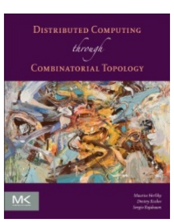


Decidability

Is it *decidable* whether a task has a protocol in a model characterized by:

$f: |\text{skel}^p\,\mathcal{I}| \to |\mathcal{O}|$?

decidable if and only if $p \leq 1$!



Road Map

Overview of Models

t-resilient layered snapshot models

Layered Snapshots with *k*-set agreement

Adversaries

Message-Passing Systems

Decidability

t-Resilient Layered Immediate Snapshot Protocol

```
shared mem array 0..N-1,0..n of Value
view := input
for ℓ:= 0 to N-1 do
    do
        immediate
            mem[ℓ][i] := view;
            snap := snapshot(mem[ℓ][*])
        until |names(snap)| >= n+1-t
    view := values(snap)
return δ(view)
```

t-Resilient Layered Immediate Snapshot Protocol

```
shared mem array 0..N-1,0..n of Value
view := input
for l := 0 to N-1 do
    do
      immediate
        mem[l][i] := view;
        snap := snapshot(mem[l][*])
      until |names(snap)| >= n+1-t
    view := values(snap)
return δ(view)
```

2-dimensional memory array

row is clean *per-layer* memory

column is *per-process* word

t-Resilient Layered Immediate Snapshot Protocol

```
shared mem array 0..N-1,0..n of Value
view := input
for l := 0 to N-1 do
    do

        mem[l][i] := view;
        snap := snapshot(mem[l][*])
      until |names(snap)| >= n+1-t
    view := values(snap)
return δ(view)
```

initial view is input value

t-Resilient Layered Immediate Snapshot Protocol

```
shared mem array 0..N-1,0..n of Value
view := input
for ℓ := 0 to N-1 do
    do
      immediate
        mem[ℓ][i] := view;
        snap := snapshot(mem[ℓ][*])
      until |names(snap)| >= n+1-t
    view := values(snap)
return δ(view)
```

run for *N* layers

t-Resilient Layered Immediate Snapshot Protocol

```
shared mem array 0..N-1,0..n of Value
view := input
for ℓ := 0 to N-1 do
    do
        immediate
            mem[ℓ][i] := view;
            snap := snapshot(mem[ℓ][*])
        until |names(snap)| >= n+1-t
    view := values(snap)
return δ(view)
```

layer ℓ : immediate write & snapshot of row ℓ

t-Resilient Layered Immediate Snapshot Protocol

```
shared mem array 0..N-1,0..n of Value
view := input
for l := 0 to N-1 do
    do
      immediate
        mem[l][i] := view;
        snap := snapshot(mem[l][*])
    until |names(snap)| >= n+1-t
  view := values(snap)
return δ(view)
```

wait to hear from *n*+1-*t* processes

why is this safe?

t-Resilient Layered Immediate Snapshot Protocol

```
shared mem array 0..N-1,0..n of Value
view := input
for l := 0 to N-1 do
    do
      immediate
        mem[l][i] := view;
        snap := snapshot(mem[l][*])
      until |names(snap)| >= n+1-t
    view := values(snap)
return δ(view)
```

new view is set of values seen

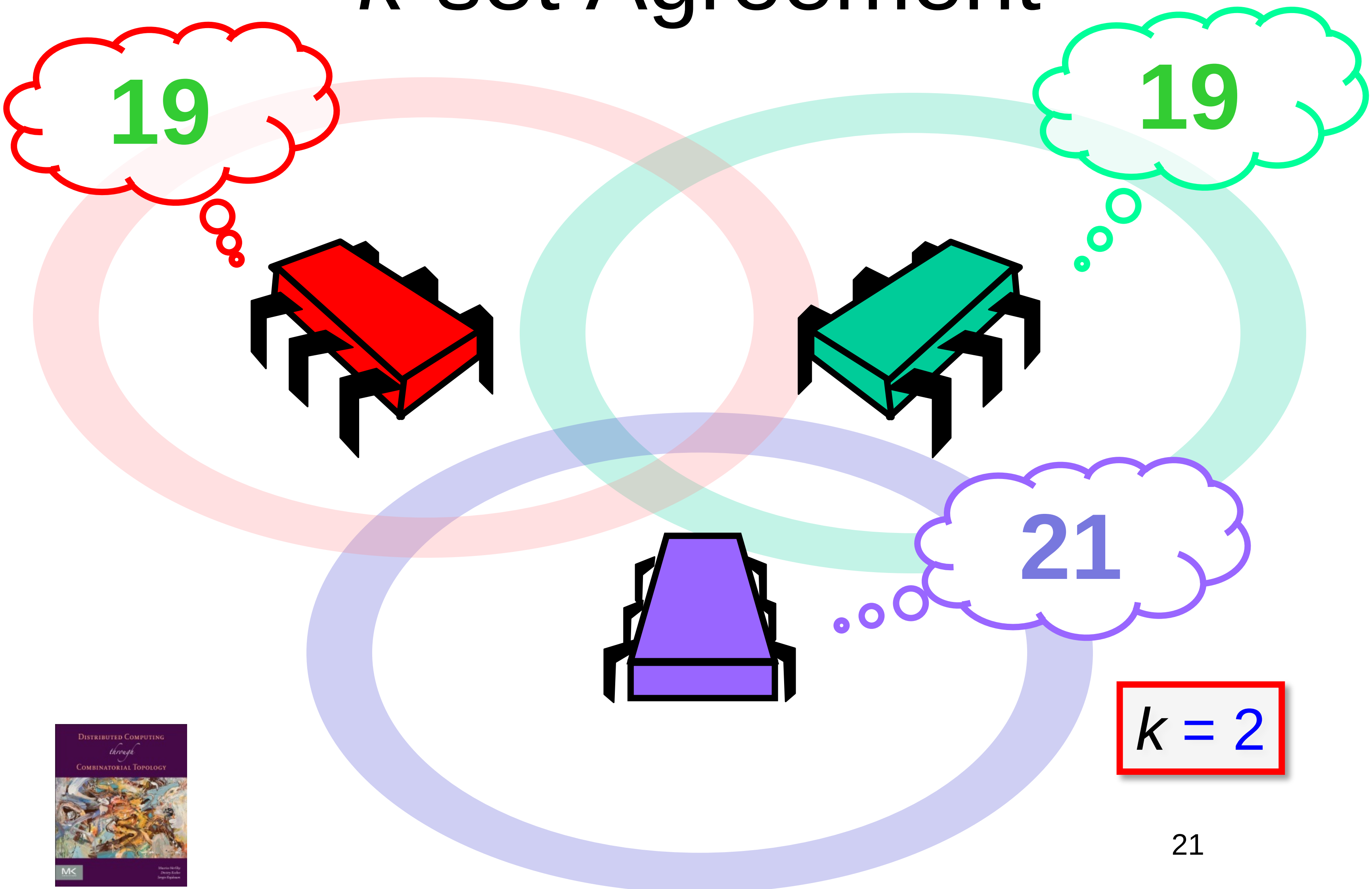
t-Resilient Layered Immediate Snapshot Protocol

```
shared mem array 0..N-1,0..n of Value
view := input
for l := 0 to N-1 do
    do
      immediate
        mem[l][i] := view;
                                          [*])
      until |names(snap)| >= n+1-t
    view = values(snap)
```

finally apply decision map to final view

```
return δ(view)
```

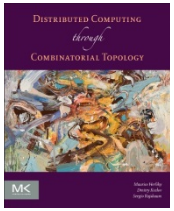
k-set Agreement



$k = 2$

(*t*+1)-Set Agreement

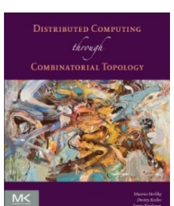
```
view := input
snap: array of Value = ∅
do
  immediate
    mem[0][i] := view;
    snap := snapshot(mem[0][*])
  until |names(snap)| >= n+1-t
return min(values(view))
```



(*t*+1)-Set Agreement

```
view := input
snap: arra
do
  immediate
    mem[0][i] := view;
    snap := snapshot(mem[0][*])
  until |names(snap)| >= n+1-t
return min(values(view))
```

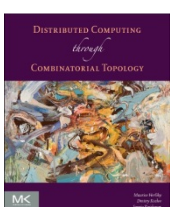
write input and take snapshot



(t+1)-Set Agreement

```
view := input
snap: array of Value = ∅
do
  immediate
    mem[0][i] := view;
    snap := snapshot(mem[0][*])
  until |names(snap)| >= n+1-t
return min(values(view))
```

wait to hear from n+1-t processes



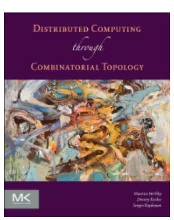
(t+1)-Set Agreement

```
view := input
snap: array of Value = ∅
do
  immediate
    me
    snap := snapshot(mem[0][*])
  until |names(snap)| >= n+1-t
return min(values(view))
```

return least value in view

can miss at most *t* lesser values

most t+1 values returned



Informal Skeleton Lemma

If

We have a protocol for a task ...

And

A protocol for *k*-set agreement …

Then

WLOG, we can "pre-process" with *k*-set agreement.

Skeleton Lemma

If

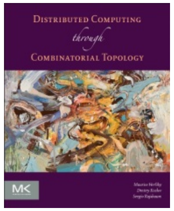
protocol $(\mathcal{I}, \mathcal{P}, \Xi)$ solves task $(\mathcal{I},\mathcal{O},\Delta)$

And

There is a k-set agreement protocol for $\mathcal{I}$

Then

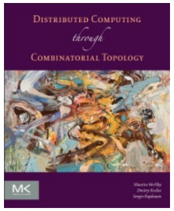
The composition of k-set agreement with $(\mathcal{I},\mathcal{P},\Xi)$ also solves $(\mathcal{I},\mathcal{O},\Delta)$.



Informal Protocol Complex Lemma

WLOG

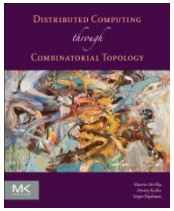
We can assume that any protocol complex is a barycentric subdivision of the input complex.



Informal Protocol Complex Lemma

WLOG

We can assume that any protocol complex is a barycentric subdivision of the input complex.



Protocol Complex Lemma

If

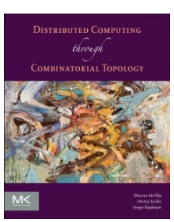
There is a t-resilient layered protocol for $(\mathcal{I},\mathcal{O},\Delta)$ ...

Then

Then there is a protocol $(\mathcal{I},\mathcal{P},\Xi)$ such that …

$\mathcal{P}$ = $\text{Bary}^N(\text{skel}^t\ \mathcal{I})$

$\Xi(\sigma)$ = $\text{Bary}^N \bullet \text{skel}^t(\sigma)$.



Theorem

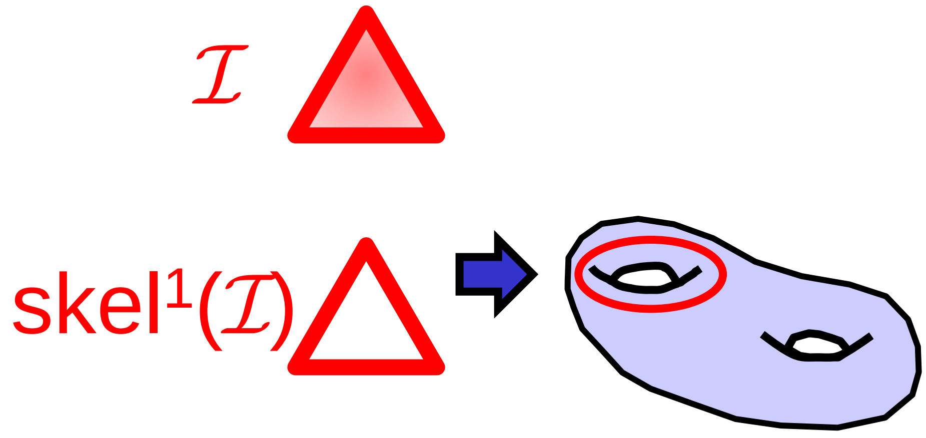
The colorless task $(\mathcal{I},\mathcal{O},\Delta)$ has a t-resilient layered snapshot protocol …

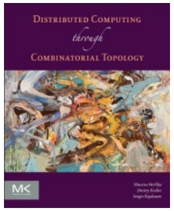
if and only if …

there is a continuous map

$f\colon |\mathrm{skel}^t\, \mathcal{I}| \to |\mathcal{O}|$

carried by Δ.





Protocol Implies Map

May assume protocol complex is $\mathcal{P} = \text{Bary}^{N}\, \text{skel}^{t}\, \mathcal{I}$.

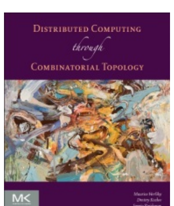
decision map

$\delta: \text{Bary}^{N}\, \text{skel}^{t}\, \mathcal{I} \rightarrow \mathcal{O}$

$|\delta|: |\text{Bary}^{N}\, \text{skel}^{t}\, \mathcal{I}| \rightarrow |\mathcal{O}|$

$|\delta|: |\text{skel}^{t}\, \mathcal{I}| \rightarrow |\mathcal{O}|$

carried by Δ.



Simplicial Approximation Theorem

- Given a continuous map

$$f : |\mathcal{A}| \ \rightarrow \ |\mathcal{B}|$$

- there is an N such that f has a simplicial approximation

$$\phi : \mathrm{Bary}^N \mathcal{A} \rightarrow \mathcal{B}$$

Map Implies Protocol

f: $|\text{skel}^t\, \mathcal{I}| \to |\mathcal{O}|$

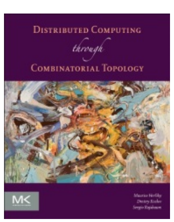
ϕ: $\text{Bary}^N\, \text{skel}^t\, \mathcal{I} \to \mathcal{O}$

carried by Δ.

Solve using …

barycentric agreement

t-set agreement



Road Map

Overview of Models

t-resilient layered snapshot models

Layered Snapshots with *k*-set agreement

Adversaries

Message-Passing Systems

Decidability

Motivation

Today …

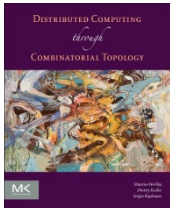
Practically all modern multiprocessors provide synchronization more powerful than read-write …

Like …

test-and-set, compare-and-swap, ….

Here …

we consider protocols constructed by *composing* layered snapshot protocols with *k*-set agreement protocols.



Wait-Free Layered Set Agreement Protocol

```
shared mem array 0..N-1,0..n of Value
shared SA array 0..N-1 of SetAgree
view := input
for ℓ:= 0 to N-1 do
  view: View := SA[ℓ].decide(view)
  immediate
    mem[ℓ][i] := view;
    snap := snapshot(mem[ℓ][*])
  view := values(snap)
return δ(view)
```

Wait-Free Layered Set Agreement Protocol

```
shared mem array 0..N-1,0..n of Value
shared SA array 0..N-1 of SetAgree
view := input
for ℓ := 0 to N-1 do
  view: View := SA[ℓ].decide(view)
  immediate
    mem[ℓ][i] := view;
    snap := snapshot(mem[ℓ][*])
  view := values(snap)
return δ(view)
```

per-level shared memory

Wait-Free Layered Set Agreement Protocol

```
shared mem array 0..N-1,0..n of Value
shared SA array 0..N-1 of k-SetAgree
view := input
for ℓ := 0 to N-1 do
  view: View := SA[ℓ].decide(view)
  immediate
    mem[ℓ][i] := view;
    snap := snapshot(mem[ℓ][*])
  view := values(snap)
return δ(view)
```

per-level *k*-set agreement object

Wait-Free Layered Set Agreement Protocol

```
shared mem array 0..N-1,0..n of Value
shared SA array 0..N-1 of k-SetAgree
view := input
for l := 0 to N-1 do
  view: View := SA[l].decide(view)
  immediate
    mem[l][i] := view;
    snap := snapshot(mem[l][*])
  view := values(snap)
return δ(view)
```

initial view is input value

Wait-Free Layered Set Agreement Protocol

```
shared mem array 0..N-1,0..n of Value
shared SA array 0..N-1 of k-SetAgree
view := input
for ℓ := 0 to N-1 do
  view: View := SA[ℓ].decide(view)
  immediate
    mem[ℓ][i] := view;
    snap := snapshot(mem[ℓ][*])
  view := values(snap)
return δ(view)
```

do *k*-set agreement with others at this level

Wait-Free Layered Set Agreement Protocol

```
shared mem array 0..N-1,0..n of Value
shared SA array 0..N-1 of k SetAgree
view := input
for l := 0 to N-1 do
  view: View := SA[l].decide(view)
  immediate
    mem[ℓ][i] := view;
    snap := snapshot(mem[ℓ][*])
  view := values(snap)
return δ(view)
```

then do immediate snapshot

Wait-Free Layered Set Agreement Protocol

```
shared mem array 0..N-1,0..n of Value
shared SA array 0..N-1 of k-SetAgree
view := input
for ℓ := 0 to N-1 do
  view: View := SA[ℓ].decide(view)
  immediate
    mem[ℓ][i] := view;
    snap := snapshot(mem[ℓ][*])
  view := values(snap)
return δ(view)
```

new view is set of values seen

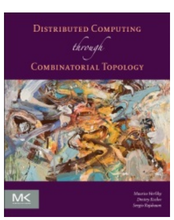
view := values(snap)

Protocol Complex Lemma

If $(\mathcal{I},\mathcal{P},\Xi)$ is a k-set layered snapshot protocol …

then $\mathcal{P}$ is equal to $\mathrm{Bary}^{N}\ \mathrm{skel}^{k\text{-}1}\,\mathcal{I}$, …

for some $N \geq 0$.



Theorem

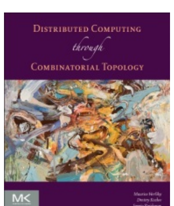
The colorless task $(\mathcal{I},\mathcal{O},\Delta)$ has a wait-free k-set layered snapshot protocol …

if and only if …

there is a continuous map

$f\colon |\text{skel}^{k-1}\,\mathcal{I}| \to |\mathcal{O}|$

carried by Δ.



Theorem

The colorless task $(\mathcal{I},\mathcal{O},\Delta)$ has a wait-free *k*-set layered snapshot protocol …

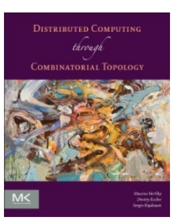
if and only if …

there is a continuous map

$f: |skel^{k-1}\,\mathcal{I}| \rightarrow |\mathcal{O}|$

carried by Δ.

k-1 skeleton, not *t*-skeleton!



Road Map

Overview of Models

t-resilient layered snapshot models

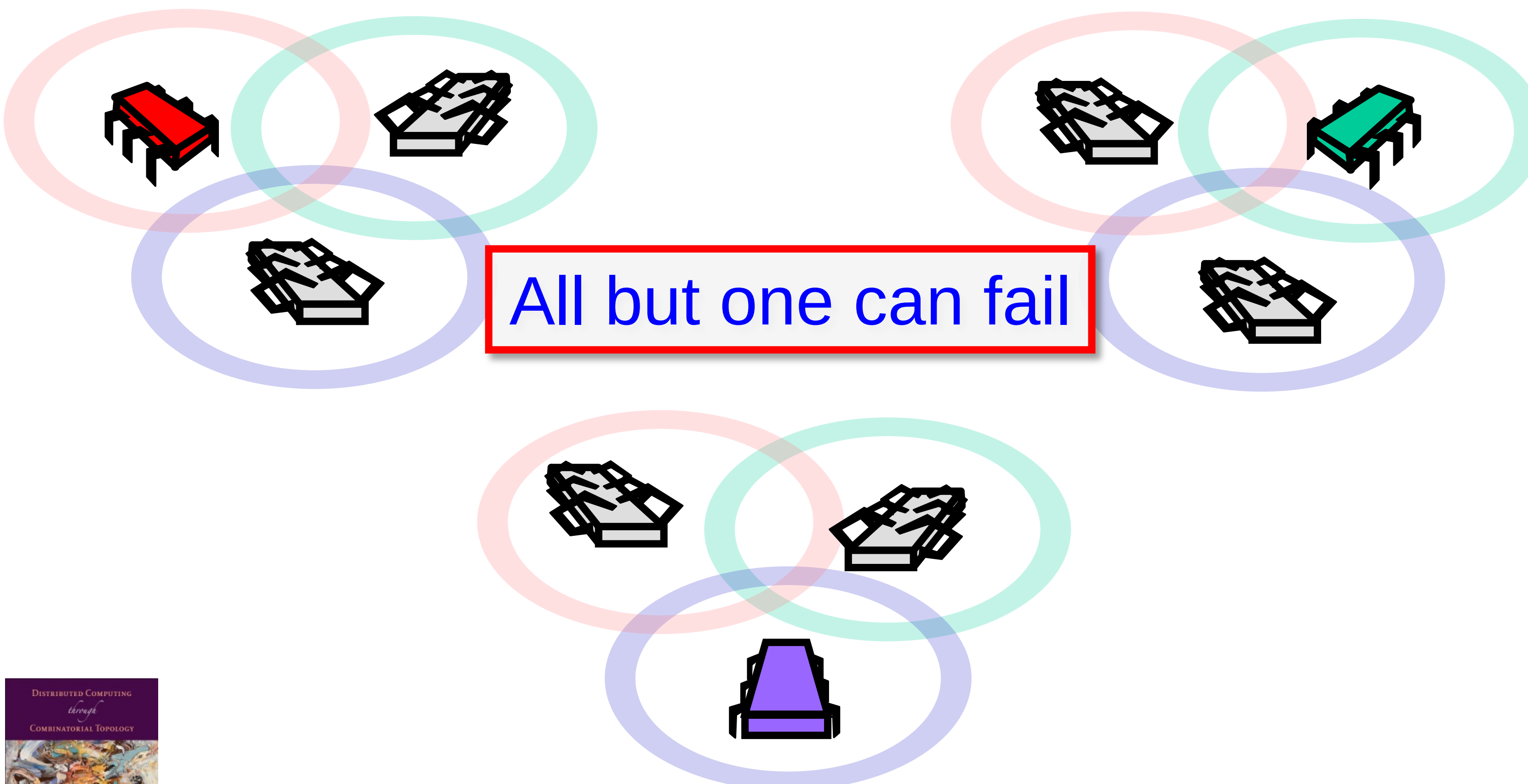
Layered Snapshots with *k*-set agreement

Adversaries

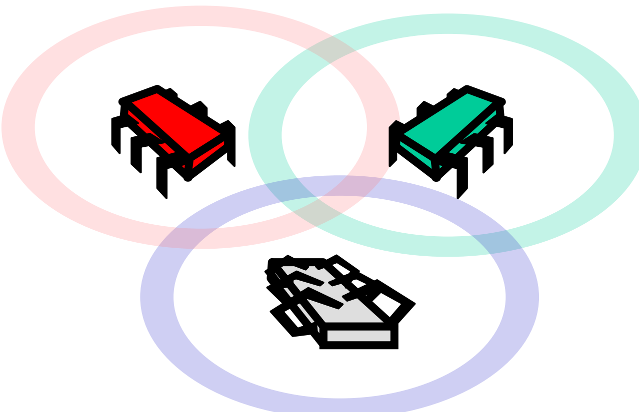
Message-Passing Systems

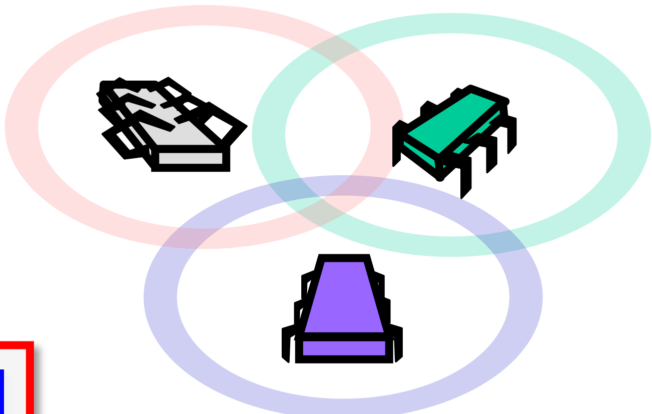
Decidability

Wait-Free

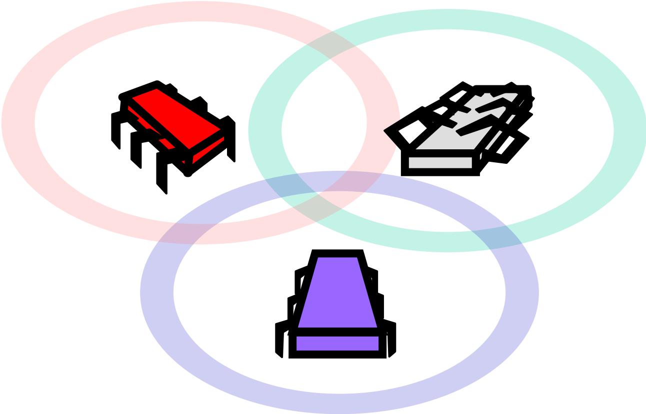


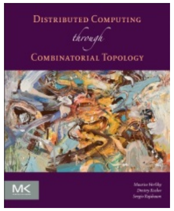
t-resilient



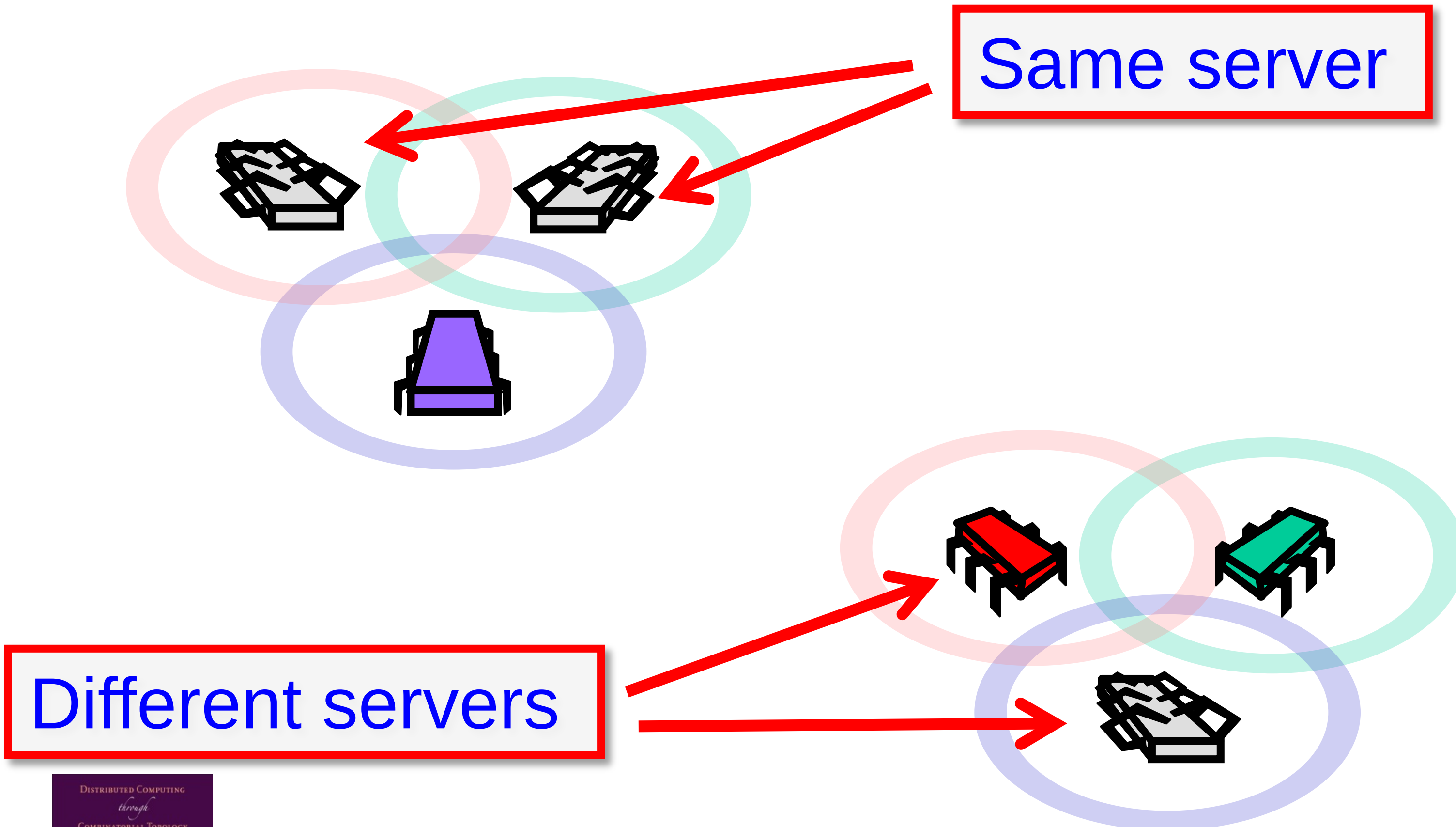


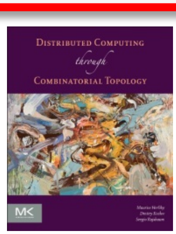
$\leq$ *t* can fail



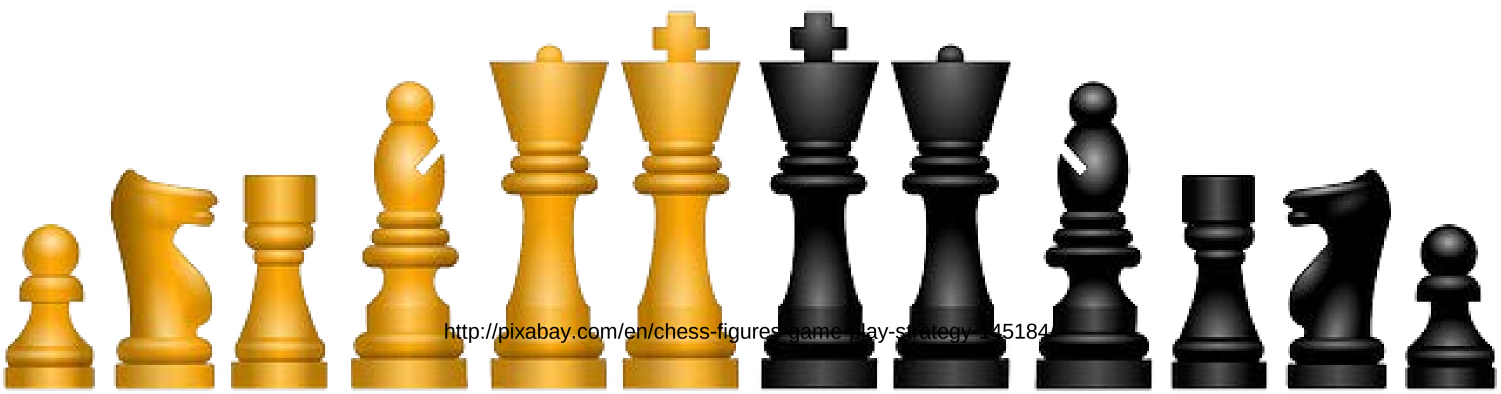


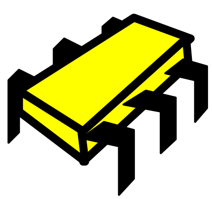
Irregular Failures

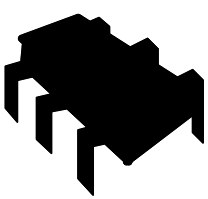


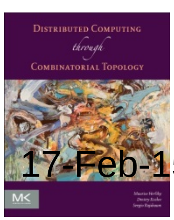


Adversaries

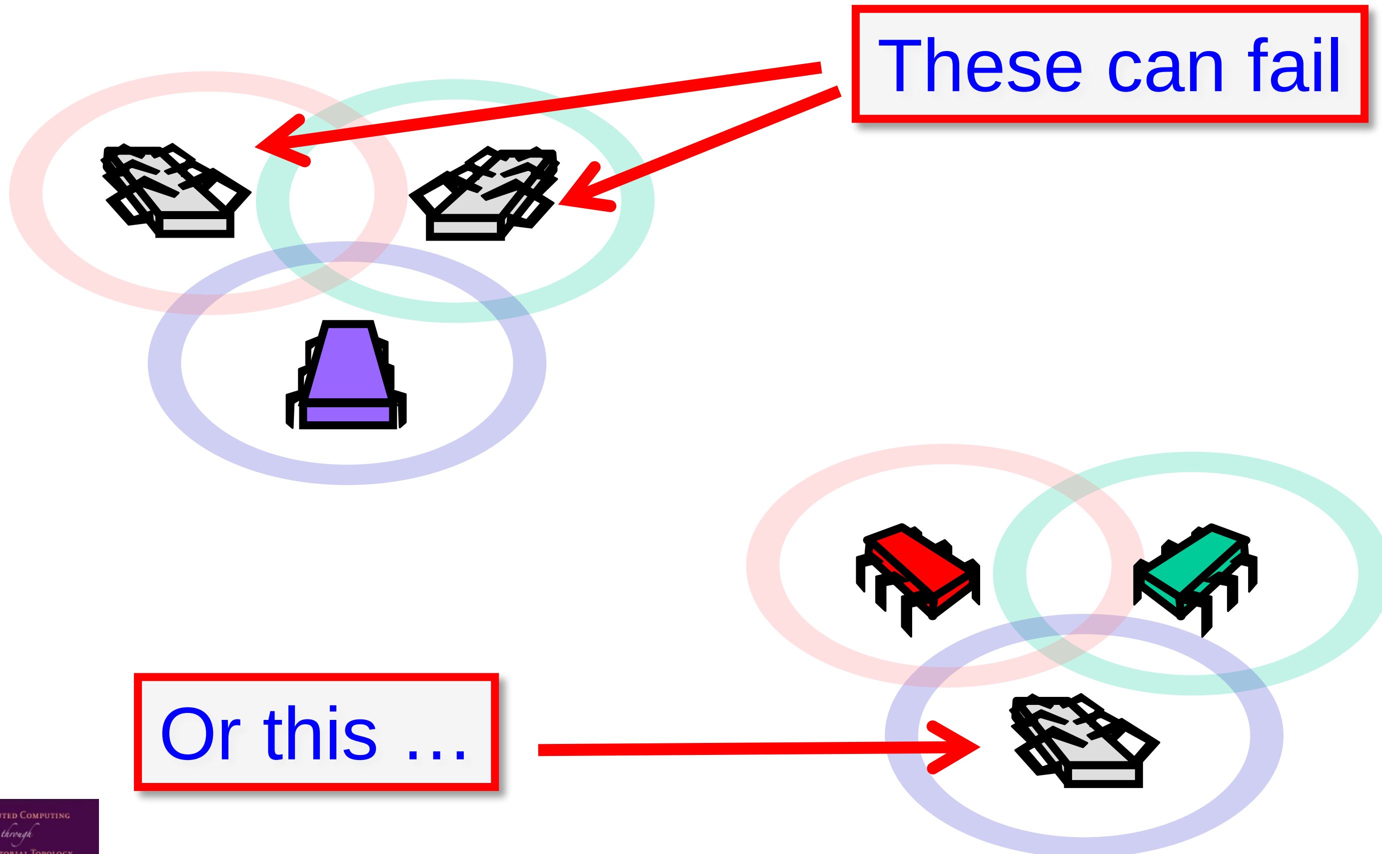


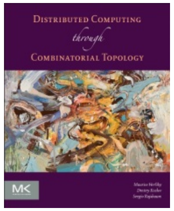




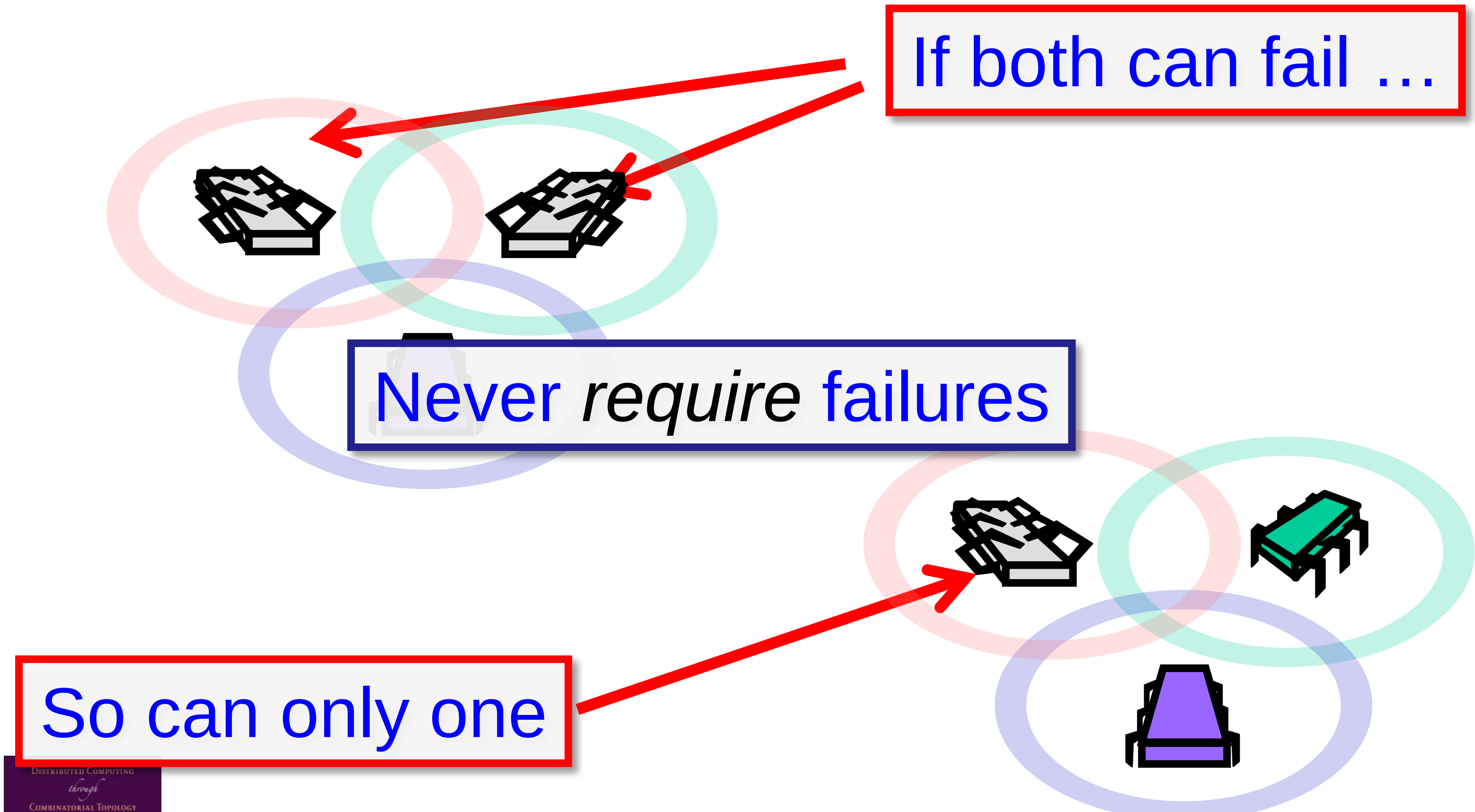


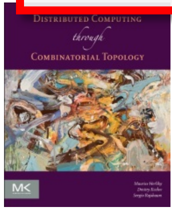
Faulty Sets


These can fail
Or this ...

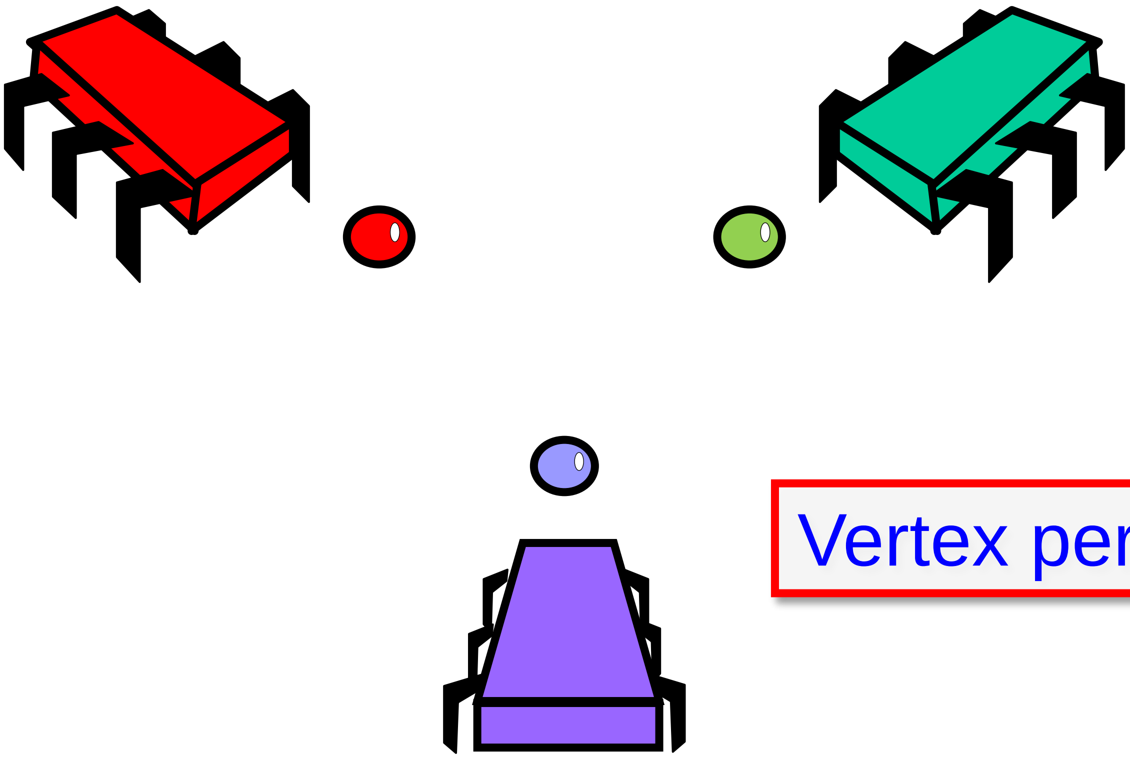

Distributed Computing
through
Combinatorial Topology

Faulty Sets Closed under Containment

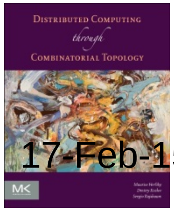




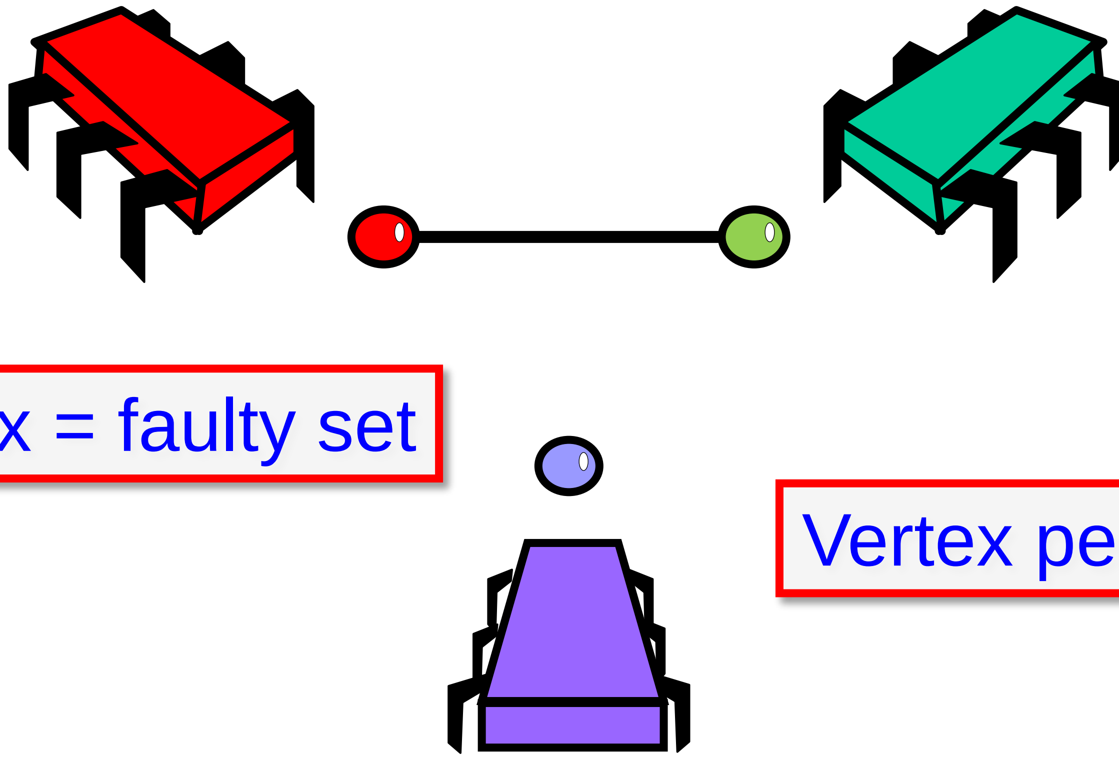
Failure Complex



Vertex per process

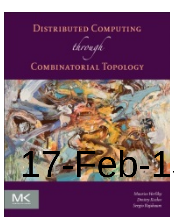


Failure Complex

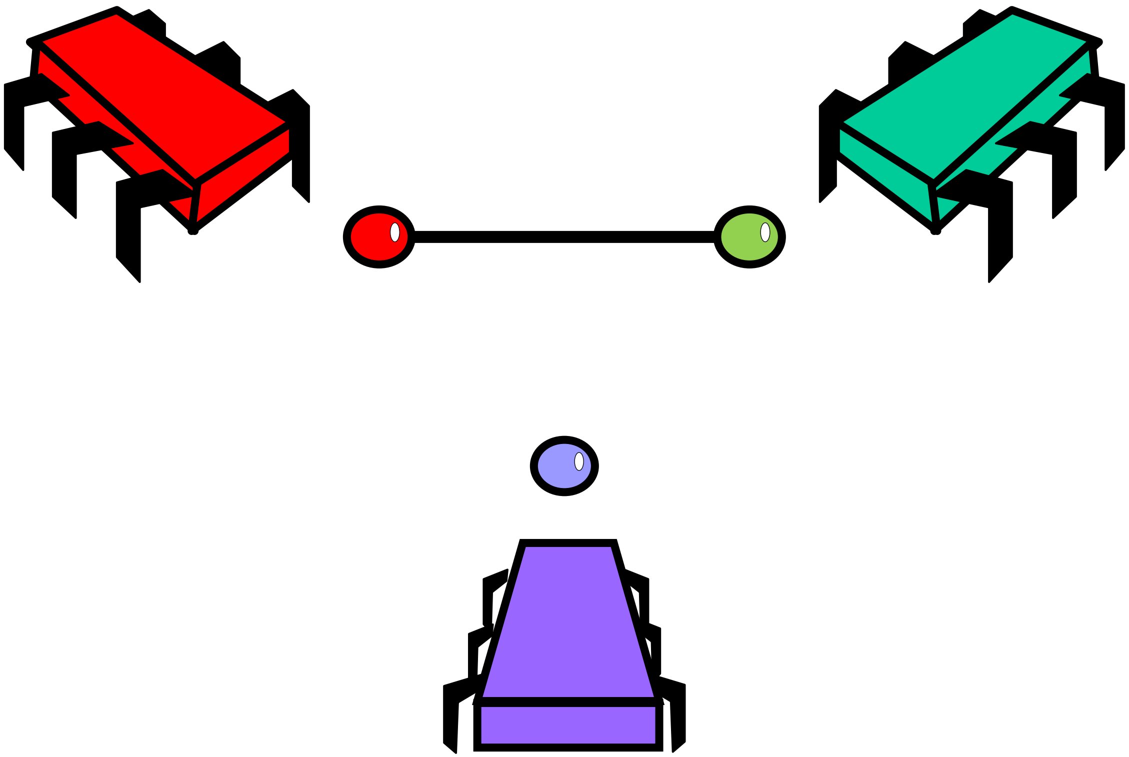


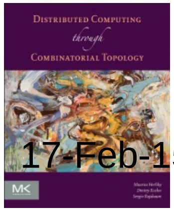
Simplex = faulty set

Vertex per process

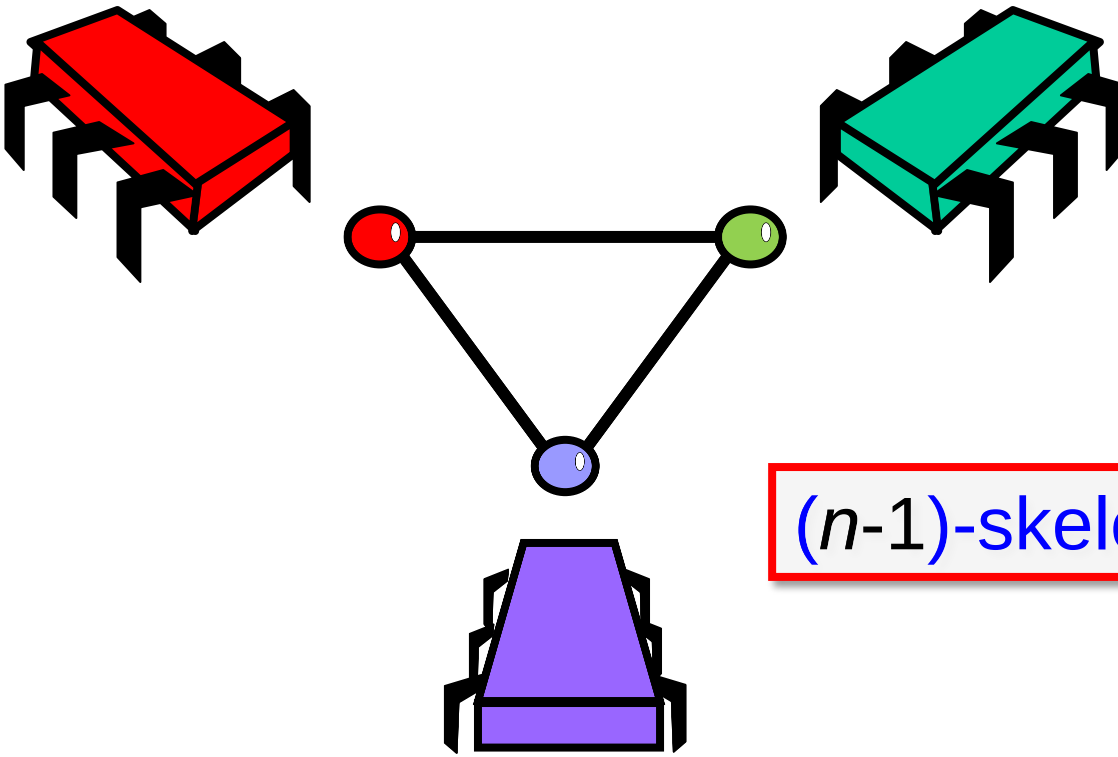


Irregular Failure Complex

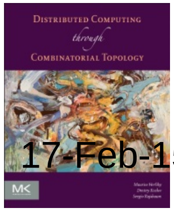




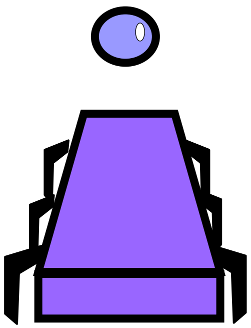
Wait-Free Failure Complex



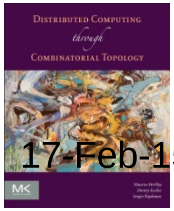
(*n*-1)-skeleton



t-resilient Failure Complex



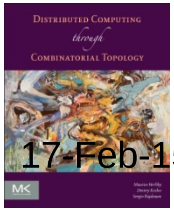
(*t*-1)-skeleton



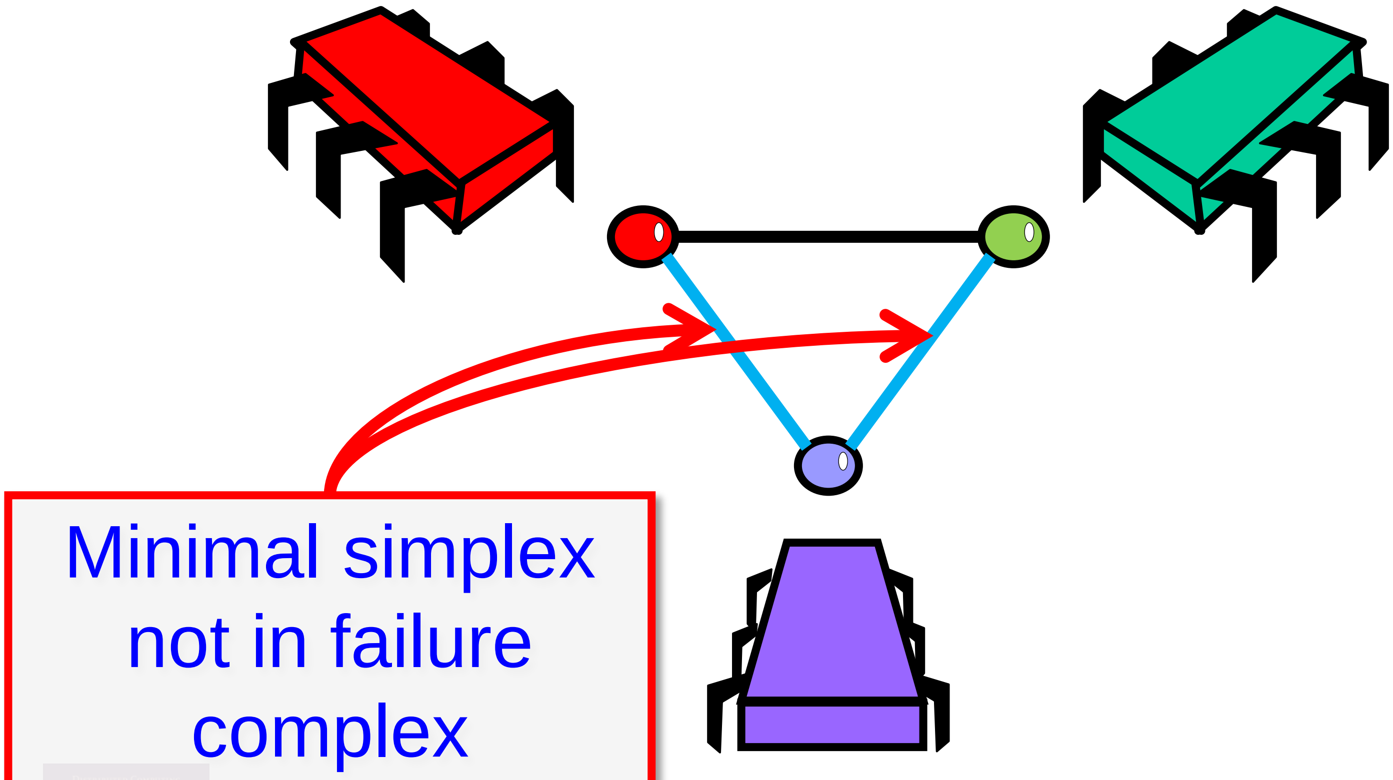
Cores

Minimal set of processes that cannot *all* fail

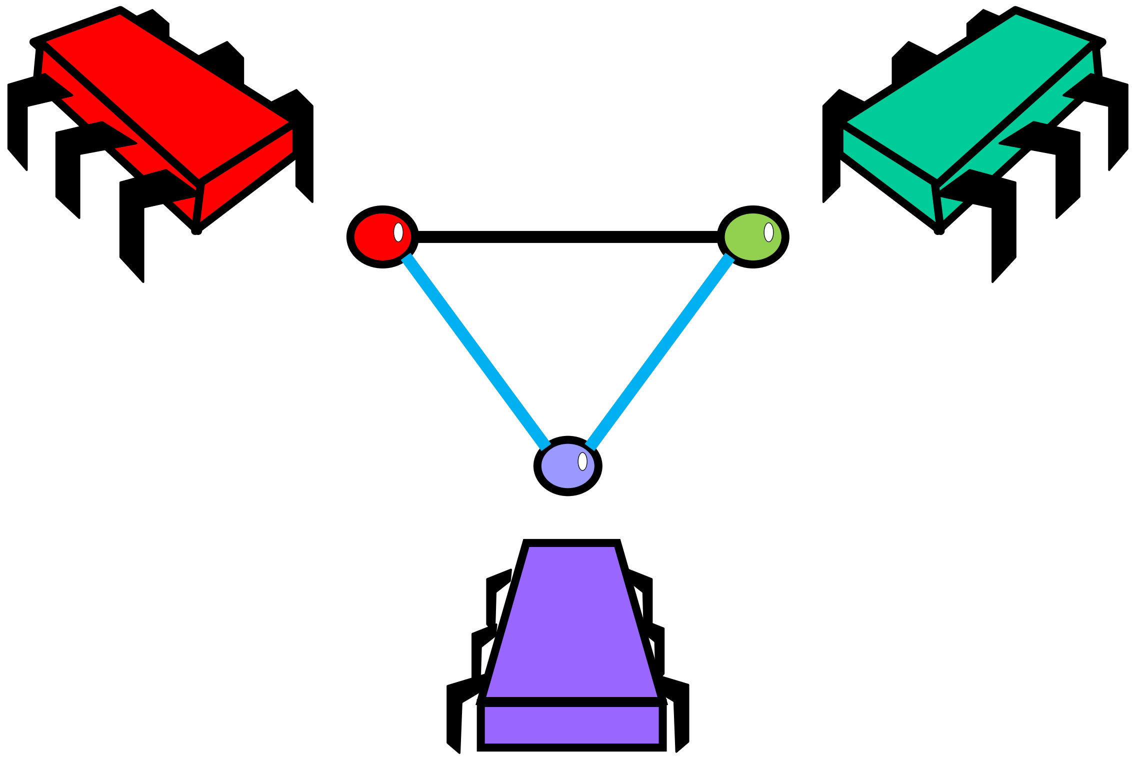
Safe to wait for at least one member of a particular core to show up

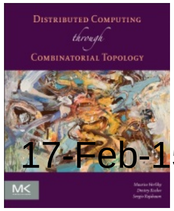


Cores & Failure Complex



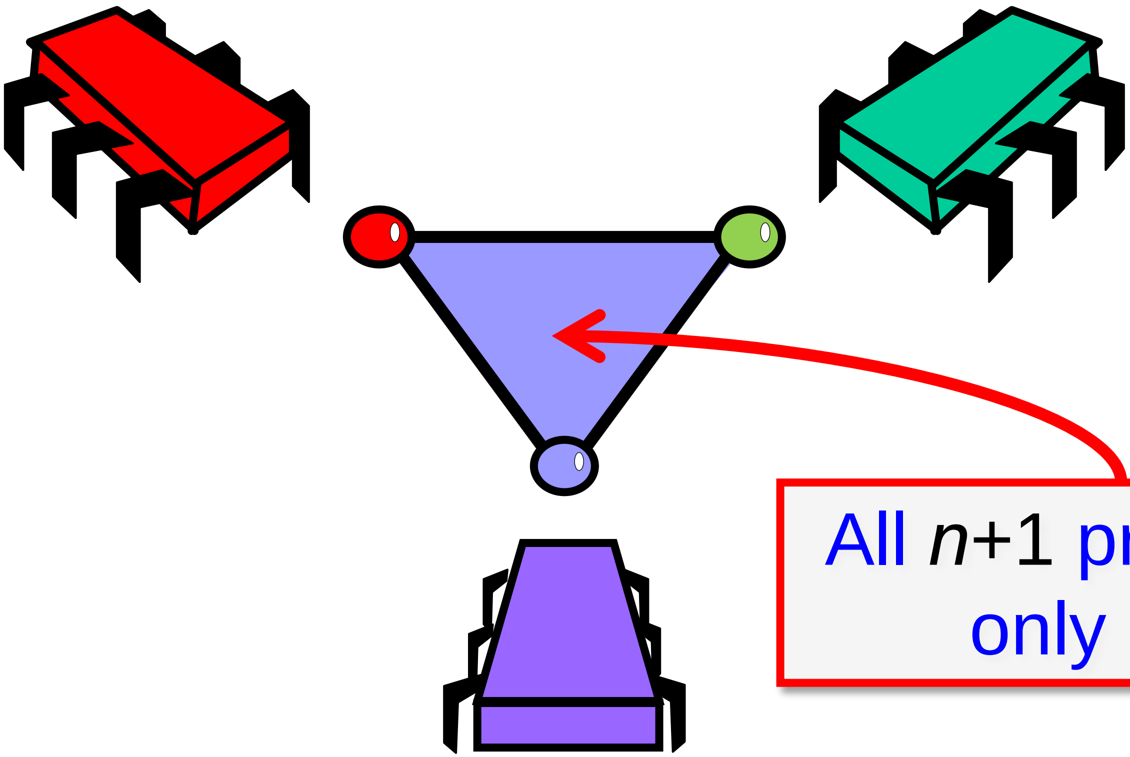
Irregular Failure Complex



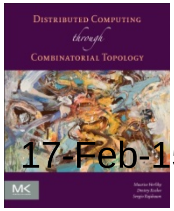


17-Feb-15

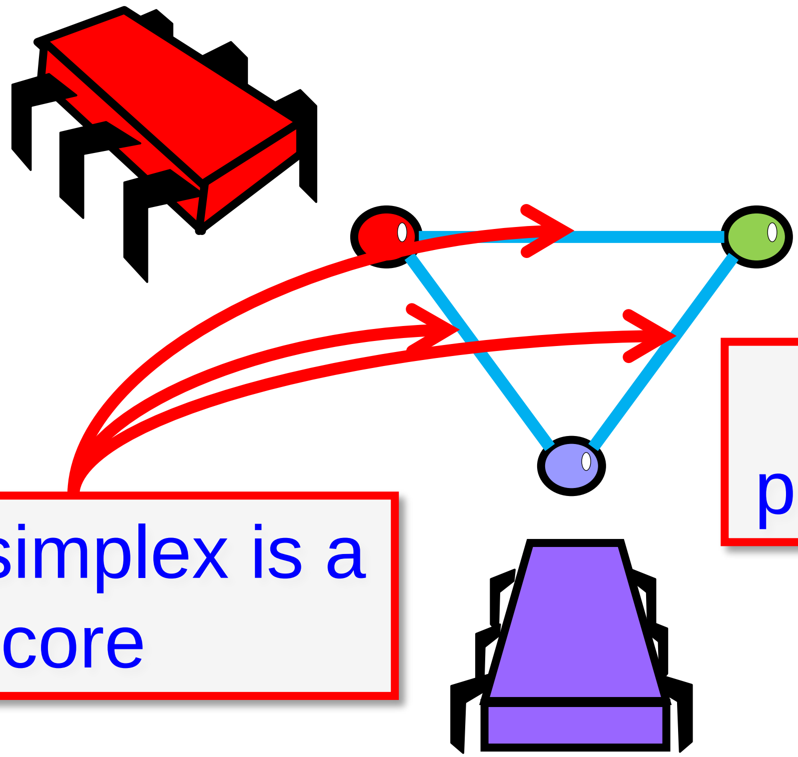
Wait-Free Failure Complex



All *n*+1 processes only core

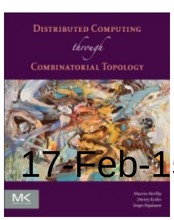


t-resilient Failure Complex



Any *t*-simplex is a core

Any set of *t*+1 processes is a core



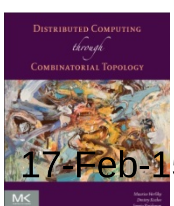
Cores

For many models,

minimum core size…

Completely determines adversary's power to solve *any* colorless task!

So adversaries with same min core size solve the same colorless tasks

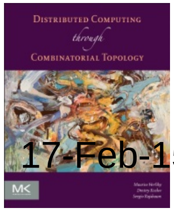


Survivor Sets

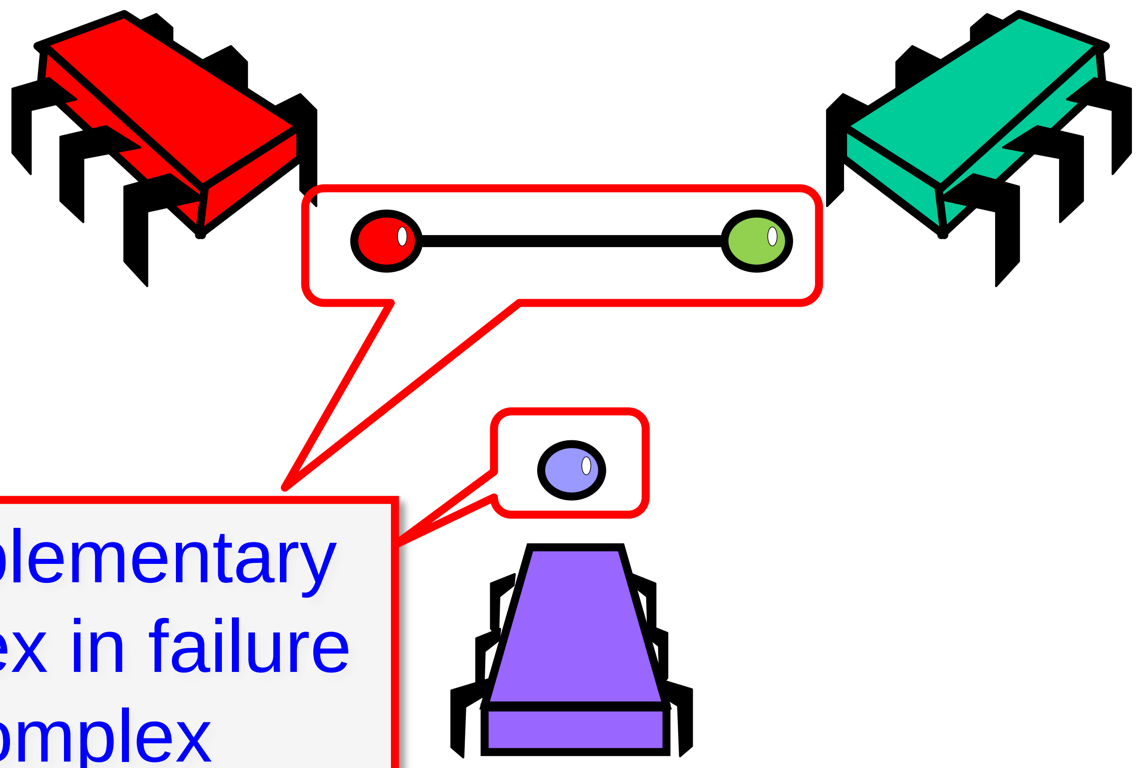
Minimal set of processes that might *all* survive

Safe to wait for all members of some survivor set to show up

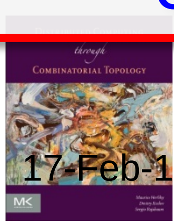
Dual to cores: each one determines the other



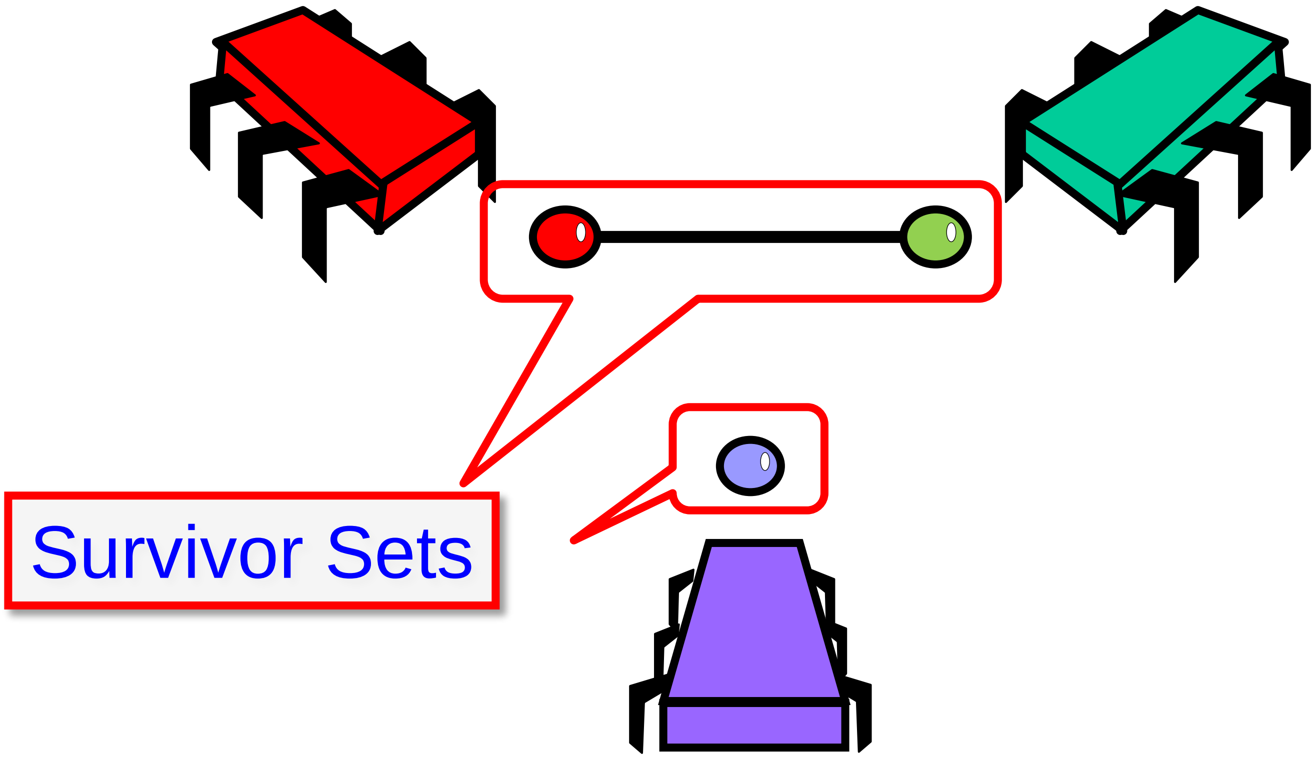
Survivor Sets in Failure Complex

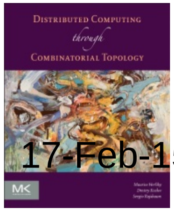


Complementary simplex in failure complex

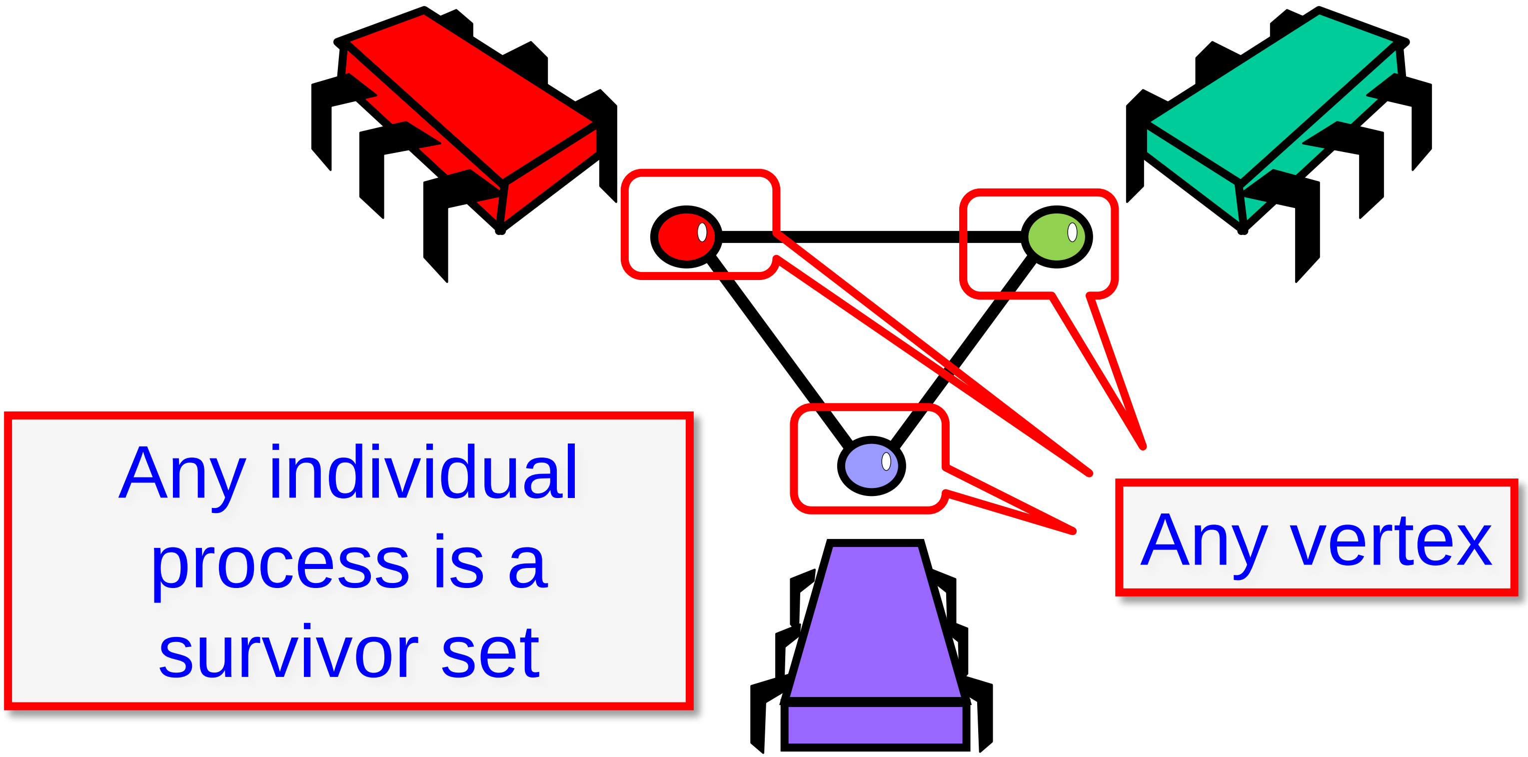


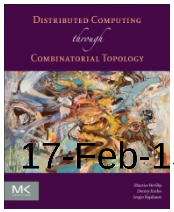
Irregular Failure Complex





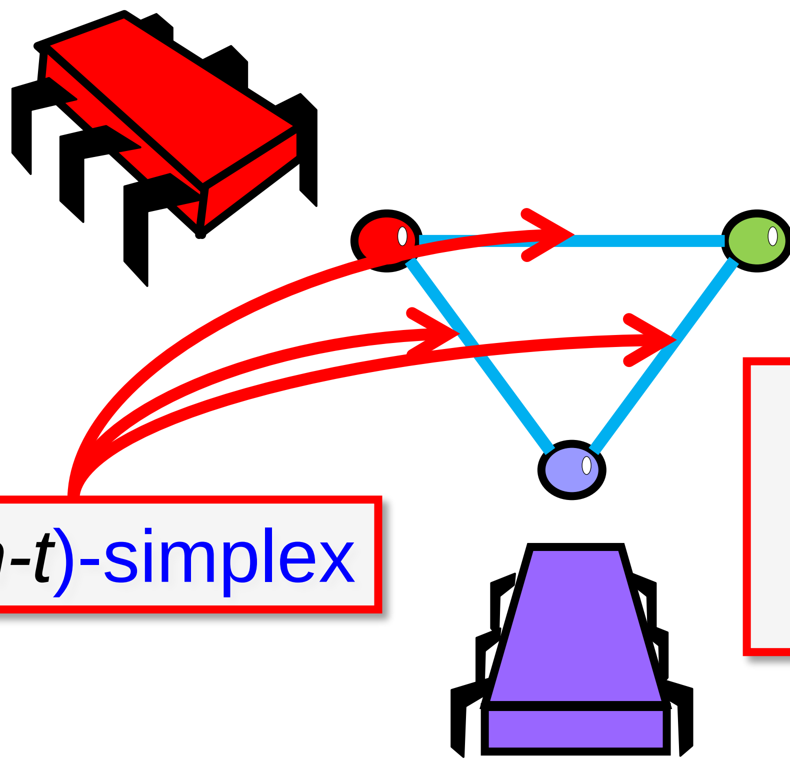
Wait-Free Failure Complex





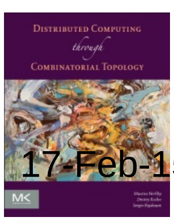
17-Feb-15

t-resilient Failure Complex



Any (n-t)-simplex

Any set of n-t+1 processes is a survivor set



$\mathbb{A}$-Resilient Layered Immediate Snapshot Protocol

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

$\mathbb{A}$-Resilient Layered Immediate Snapshot Protocol

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

wait to hear from a survivor set

why is this safe?

Road Map

Overview of Models

t-resilient layered snapshot models

Layered Snapshots with *k*-set agreement

Adversaries

Message-Passing Systems

Decidability

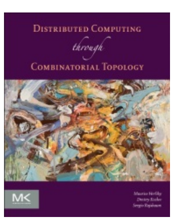
Message Passing

There are *n*+1 asynchronous processes …

that send and receive messages …

via a *fully-connected* communication network.

Message delivery is *reliable* and *FIFO*

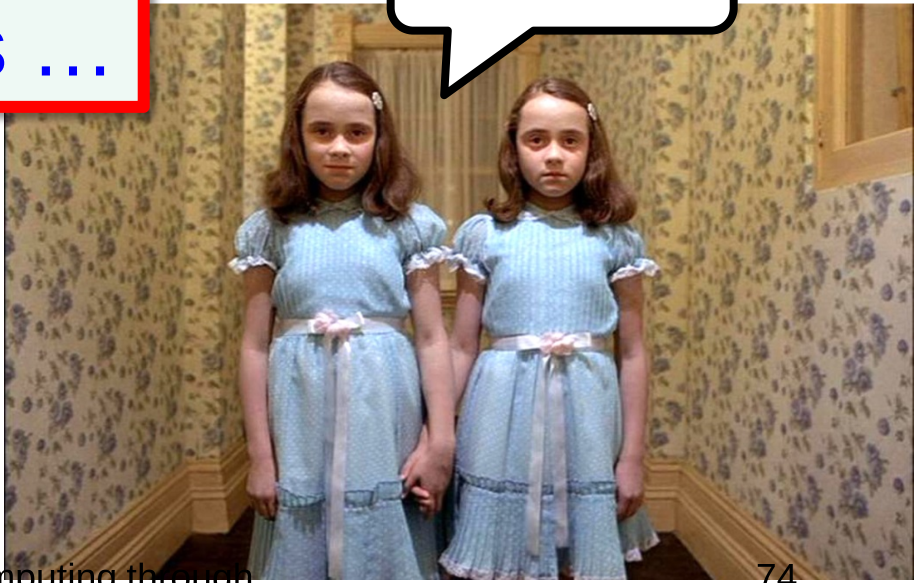


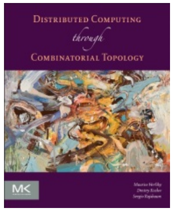
Message-Passing Protocols

decide after finite # steps

but protocol
forwards messages …

forever!



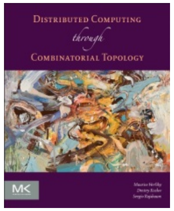


Communication Syntax

send(P, v_0, …, v_ℓ) **to** Q

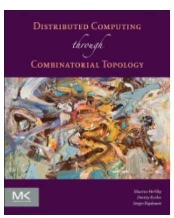
send(P, v_0, …, v_ℓ) **to all**

upon receive(P, v_0, …, v_ℓ) **do**
... // handle message



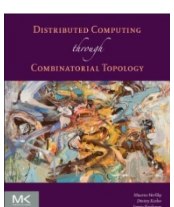
Forwarding

```
background // forward messages forever
    upon receive(Pj,v) do
        send(Pi,v) to all
```



Get Values from *n*+1-*t* Processes

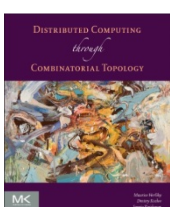
```
getQuorum(): Set of Value
  V: Set of Value := ∅
  q: int := 0
  do
    upon receive(Q,v) do
      V := V ∪ {v}
      q := q + 1
  until q = n+1-t
  return V
```



Get Values from *n*+1-*t* Processes

```
getQuorum(): Set of Value
  V: Set of Value := ∅
  q: int := 0
  do
    upon receive(Q,v) do
      V := V ∪ {v}
      q := q + 1
  until q = n+1-t
  return V
```

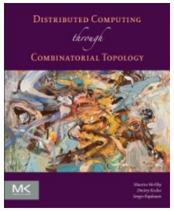
Initially, nothing.



Get Values from n+1-t Processes

```
getQuorum(): Set of Value
   V: Set of Value := ∅
   q: int := 0
   do
     upon receive(Q,v) do
       V := V ∪ {v}
       q := q + 1
   until q = n+1-t
   return V
```

remember values and count

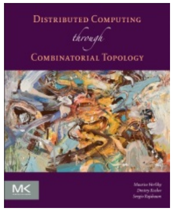


Get Values from *n*+1-*t* Processes

```
getQuorum(): Set of Value
  V: Set of Value := ∅
  q: int := 0
  do
      V := V ∪ {v}
      q := q + 1
  until q = n+1-t
  return V
```

safe to wait for *n*+1-*t* values

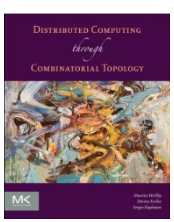
until q = n+1-t



Get Values from *n*+1-*t* Processes

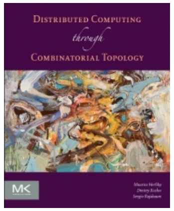
```
getQuorum(): Set of Value
  V: Set of Value := ∅
  q: int := 0
  do
      V := V ∪ {v}
      q := q + 1
  until q = n+1-t
  return V
```

return values when enough received



Protocol for (*t*+1)-Set Agreement

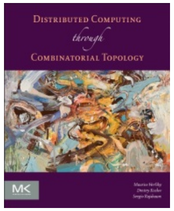
```
SetAgree(vi): value
  send(P, vi) to all
  V: Set of Value := getQuorum()
  return min(V)
```



Protocol for (*t*+1)-Set Agreement

```
SetAgree(v): value
  send(P, v) to all
  V: Set of Value := getQuorum()
  return min(V)
```

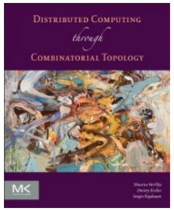
broadcast my value



Protocol for (*t*+1)-Set Agreement

```
SetAgree(v): value
  send(P,v) to all
  V: Set of Value := getQuorum()
  return min(V)
```

get values from all but *t*

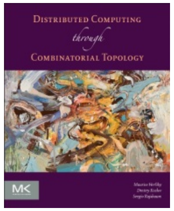


Protocol for (*t*+1)-Set Agreement

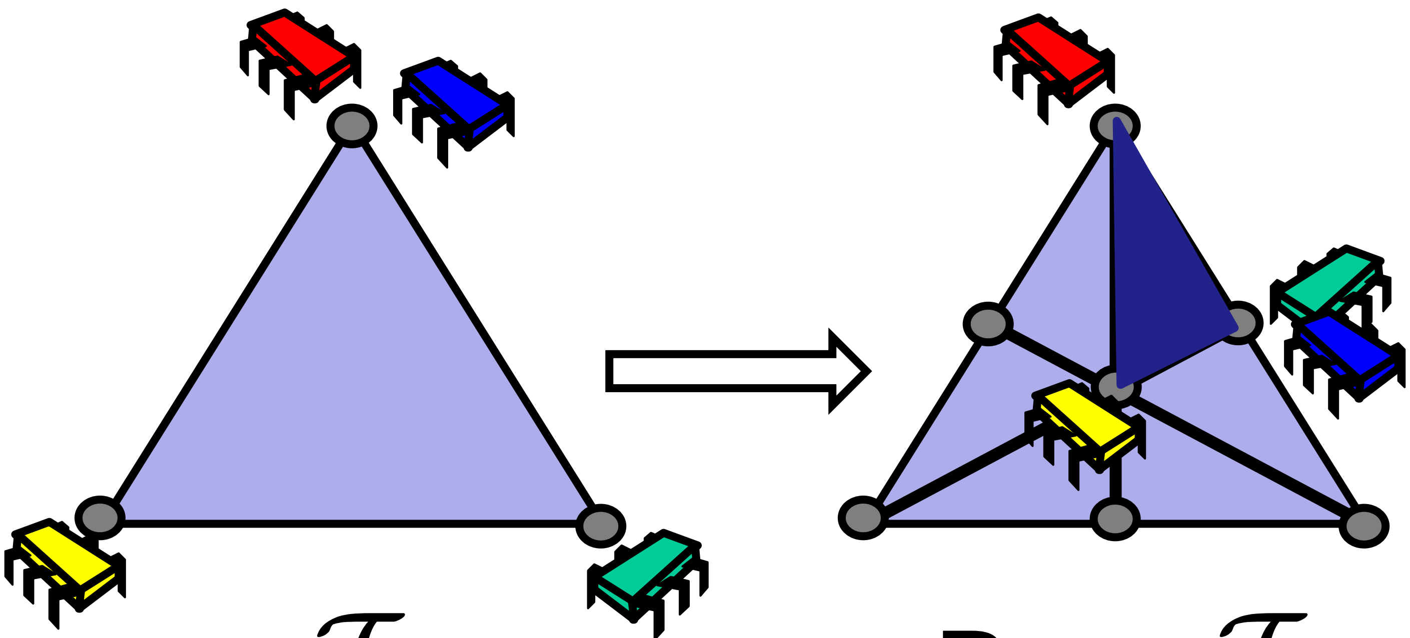
```
SetAgree(v): value
  send(P, v) to all
  V: Set of Value := getQuorum()
  return min(V)
```

return min value received

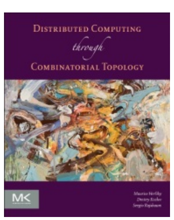
possible to “miss” only *t* lesser values



Barycentric Agreement

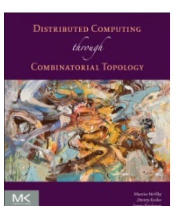


$\mathcal{I}$ Bary $\mathcal{I}$



Barycentric Agreement Protocol

BaryAgree(v_i: Vertex): set of Vertex
  V_i: set of Vertex := $\{v_i\}$
  count: int := 0
  while count < n+1-t do
    send(P_i, V_i) to all
    on receive(P_j, V_j) do
      if $V_i = V_j$ then count := count + 1
        else if $V_j \setminus V_i \neq \emptyset$ then
          $V_i := V_i \cup V_j$
          count := 0
  return V_i



Barycentric Agreement Protocol

BaryAgree(v_i: Vertex): set of Vertex

V_i: set of Vertex := $\{v_i\}$

count: int := 0

while count < n+1-t do

send

Set of messages P_i has received

on receive(P_j, V_j) do

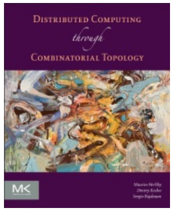
if $V_i = V_j$ then count := count + 1

else if $V_j \setminus V_i \neq \emptyset$ then

$V_i := V_i \cup V_j$

count := 0

return V_i



Barycentric Agreement Protocol

BaryAgree(v_i: Vertex): set of Vertex

V_i: set of Vertex := $\{v_i\}$

count: int := 0

while count < n+1-t do

send(P_i, V_i) to all

on receive(P_j, V_j) do

if $V_i = V_j$ then count := count + 1

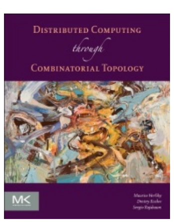
else if $V_j \setminus V_i \neq \emptyset$ then

$V_i := V_i \cup V_j$

count := 0

return V_i

keep track of confirmations received so far



Barycentric Agreement Protocol

BaryAgree(v_i: Vertex): set of Vertex

V_i: set of Vertex := $\{v_i\}$

count: int := 0

while count < n+1-t do

send(P_i, V_i) to all

on receive(P_j, V_j) do

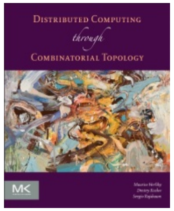
get confirmation from each non-faulty process

else if $V_j \setminus V_i \neq \emptyset$ then

$V_i := V_i \cup V_j$

count := 0

return V_i



Barycentric Agreement Protocol

BaryAgree(v_i: Vertex): set of Vertex

broadcast message set received

count: int := 0

while count < n+1-t do

send(P_i, V_i) to all

on receive(P_j, V_j) do

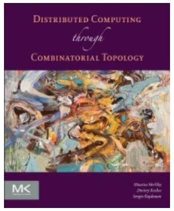
if $V_i = V_j$ then count := count + 1

else if $V_j \setminus V_i \neq \emptyset$ then

$V_i := V_i \cup V_j$

count := 0

return V_i



Barycentric Agreement Protocol

BaryAgree(v_i: Vertex): set of Vertex

V_i: set of Vertex := $\{v_i\}$

count: i

while co < n+1-t do

send(P_i, V_i) to all

collect responses

on receive(P_j, V_j) do

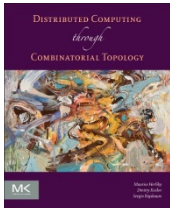
if $V_i = V_j$ then count := count + 1

else if $V_j \setminus V_i \neq \emptyset$ then

$V_i := V_i \cup V_j$

count := 0

return V_i



Barycentric Agreement Protocol

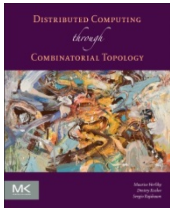
BaryAgree(v_i: Vertex): set of Vertex
  V_i: set of Vertex := {v_i}
  count: int := 0

remember if message confirms my view

    send(P_i, V_i) to all
    on receive(P_j, V_j) do

if $V_i = V_j$ then count := count + 1

      else if $V_j \setminus V_i \neq \emptyset$ then
        $V_i := V_i \cup V_j$
        count := 0
  return V_i



Barycentric Agreement Protocol

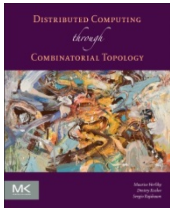
BaryAgree(v_i: Vertex): set of Vertex
  V_i: set of Vertex := $\{v_i\}$
  count: int := 0

otherwise learned something new, start over

    send(P_i, V_i) to all
    on receive(P_j, V_j) do
      if $V_i = V_j$ then count := count + 1

else if $V_j \setminus V_i \neq \emptyset$ then
  $V_i := V_i \cup V_j$
  count := 0

  return V_i

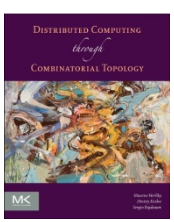


Barycentric Agreement Protocol

BaryAgree(v_i: Vertex): set of Vertex
V_i: set of Vertex := $\{v_i\}$
count: int := 0
while count < n+1-t do
send(P_i, V_i) to all
on receive(P_j, V_j) do
if $V_i = V_j$ then count := count + 1
… then
$V_i := V_i \cup V_j$
count := 0

return when enough agree

return V_i

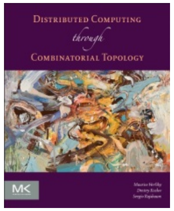


Wait, There's More!

background

upon receive(P_j, V_j) do
$V_i := V_i \cup V_j$
send(P_i, V_i) to all

the operating system runs forever …



Wait, There's More!

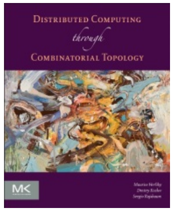
keep forwarding new values

background

upon receive(P_j, V_j) do

$V_i := V_i \cup V_j$

send(P_i, V_i) to all



Lemma: Protocol Terminates

Suppose BWOC P_i runs forever …

Eventually V_i assumes final value V …

while count < n+1-t do

Non-faulty P_j, where $V_j = V'$, receives V

P_j must have sent V_j to P_i

$V_j \subset V$ OR $V_j = V$

P_j will sent V to P_i

P_j has sent V to P_i

return V_i

All V_i, V_j Totally Ordered

BaryAgree(v_i: Vertex): set of Vertex

If P_i broadcasts $V^{(0)}, \ldots, V^{(k)}$, then $V^{(i)} \subset V^{(i+1)}$

count: int := 0

To decide …

P_i received V_i from X, $|X| \geq n+1-t$

P_j received V_j from Y, $|Y| \geq n+1-t$

if $V_i = V_j$ then count := count + 1

some $P_k \in X \cap Y$ sent both V_i, V_j

then

$V_i := V_i \cup V_j$

so V_i, V_j are ordered.

count := 0

return V_i

Theorem

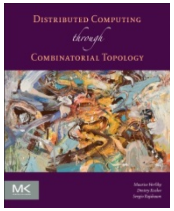
For $2t < n+1$, colorless task $(\mathcal{I},\mathcal{O},\Delta)$ has a t-resilient message-passing protocol …

if and only if …

there is a continuous map

$f: |\text{skel}^t\, \mathcal{I}| \to |\mathcal{O}|$

carried by Δ.



Theorem

For $2t < n+1$, colorless task $(\mathcal{I},\mathcal{O},\Delta)$ has a t-resilient message-passing protocol …

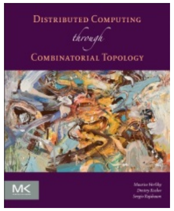
if and only if …

there is a continuous map

$f: |\text{skel}^t\, \mathcal{I}| \rightarrow |\mathcal{O}|$

carried by Δ.

same as snapshot when $2t < n+1$!



Road Map

Overview of Models

t-resilient layered snapshot models

Layered Snapshots with *k*-set agreement

Adversaries

Message-Passing Systems

Decidability

Automatic Proofs?

What if we could program a Turing machine to tell whether a task has a protocol?

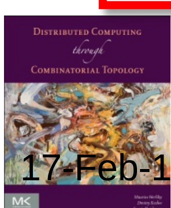
In wait-free read-write memory?

Or other models?

We could …

automatically generate conference papers

No need for grad students



Alas no

Whether a protocol exists for a task in …

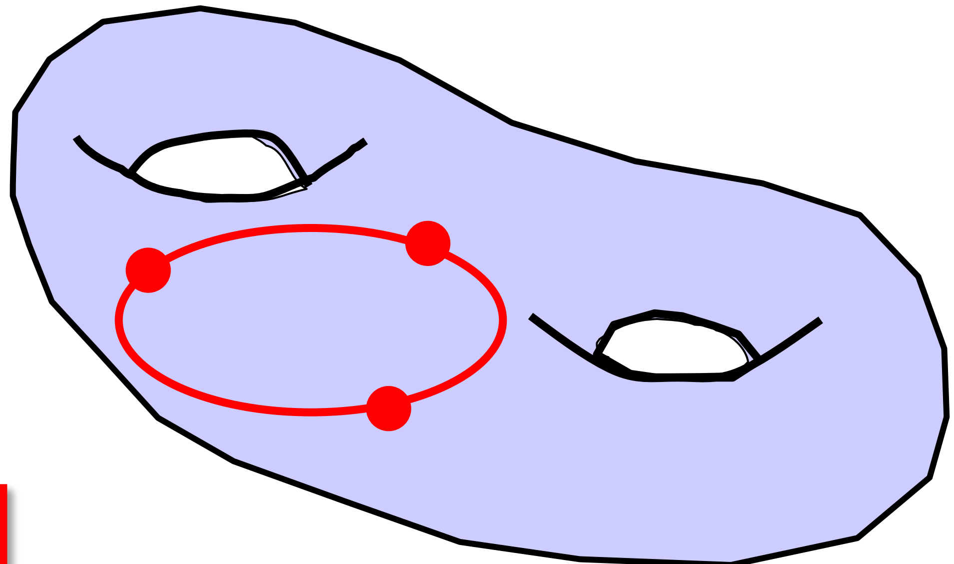
Read-write memory for 3+ processes …

Read-write memory & *k*-set agreement …

for $k > 2$

Is *undecidable*.

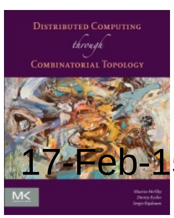
Loop Agreement



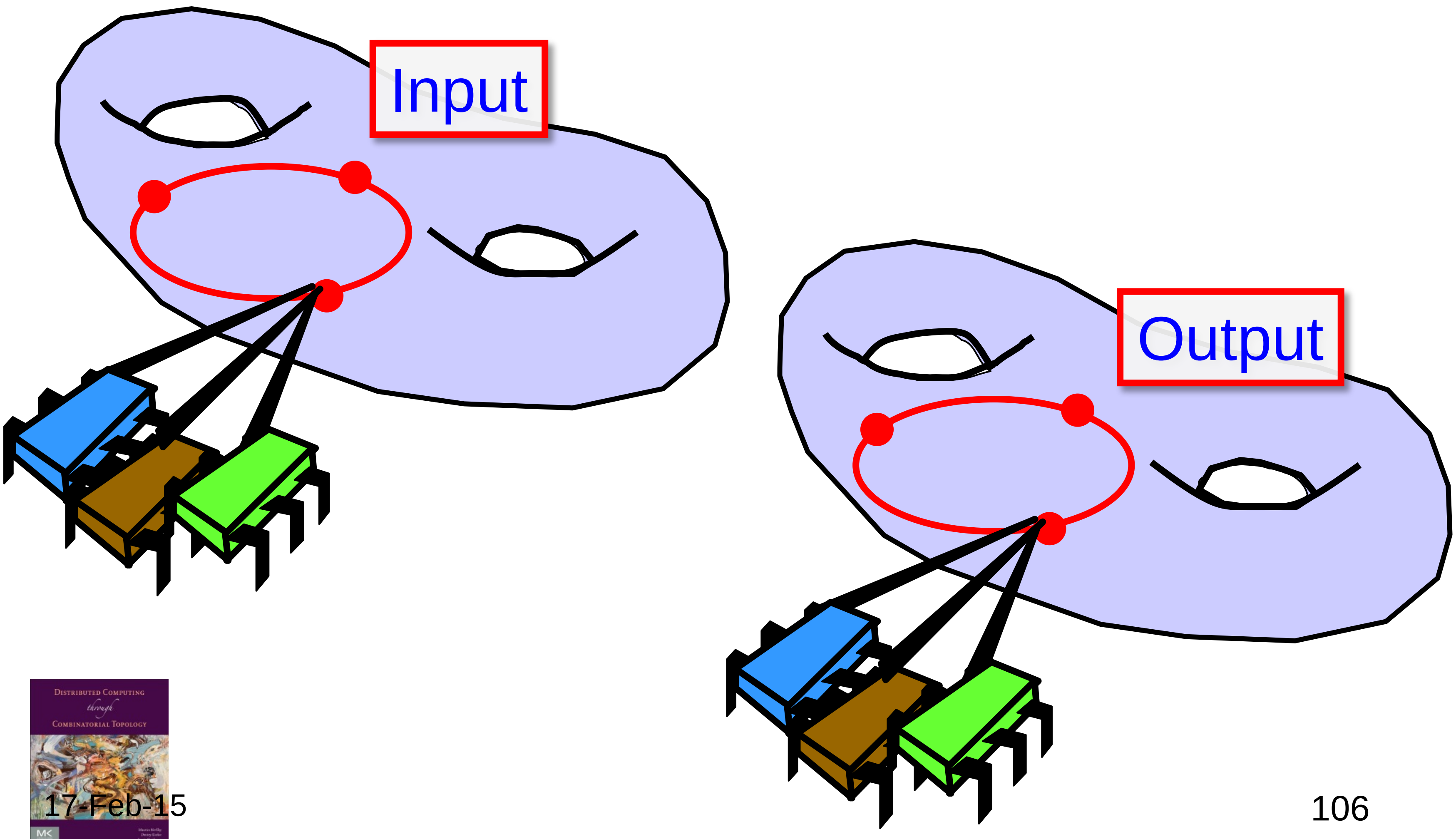
Complex

Loop

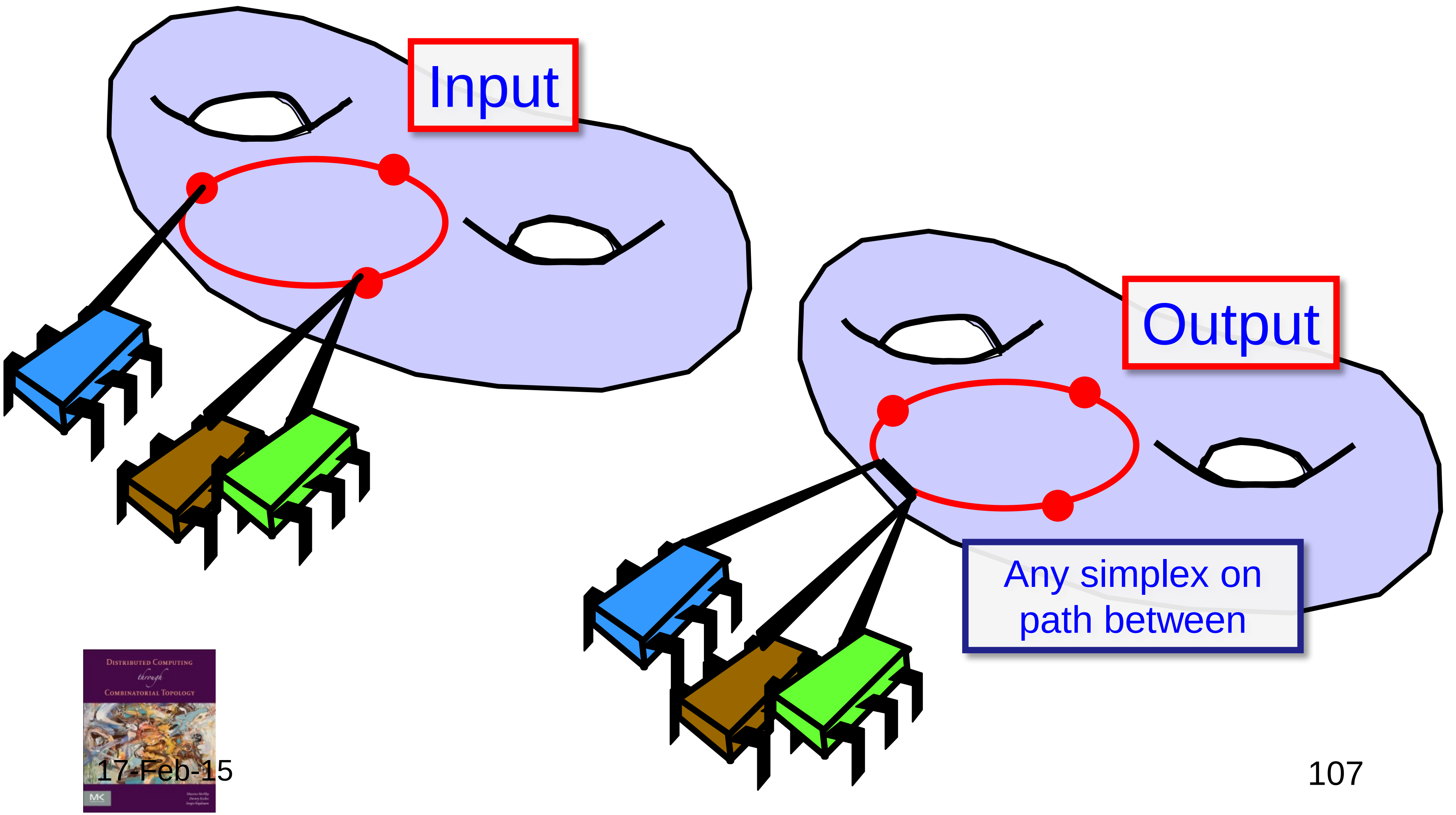
Three *rendez-vous* points



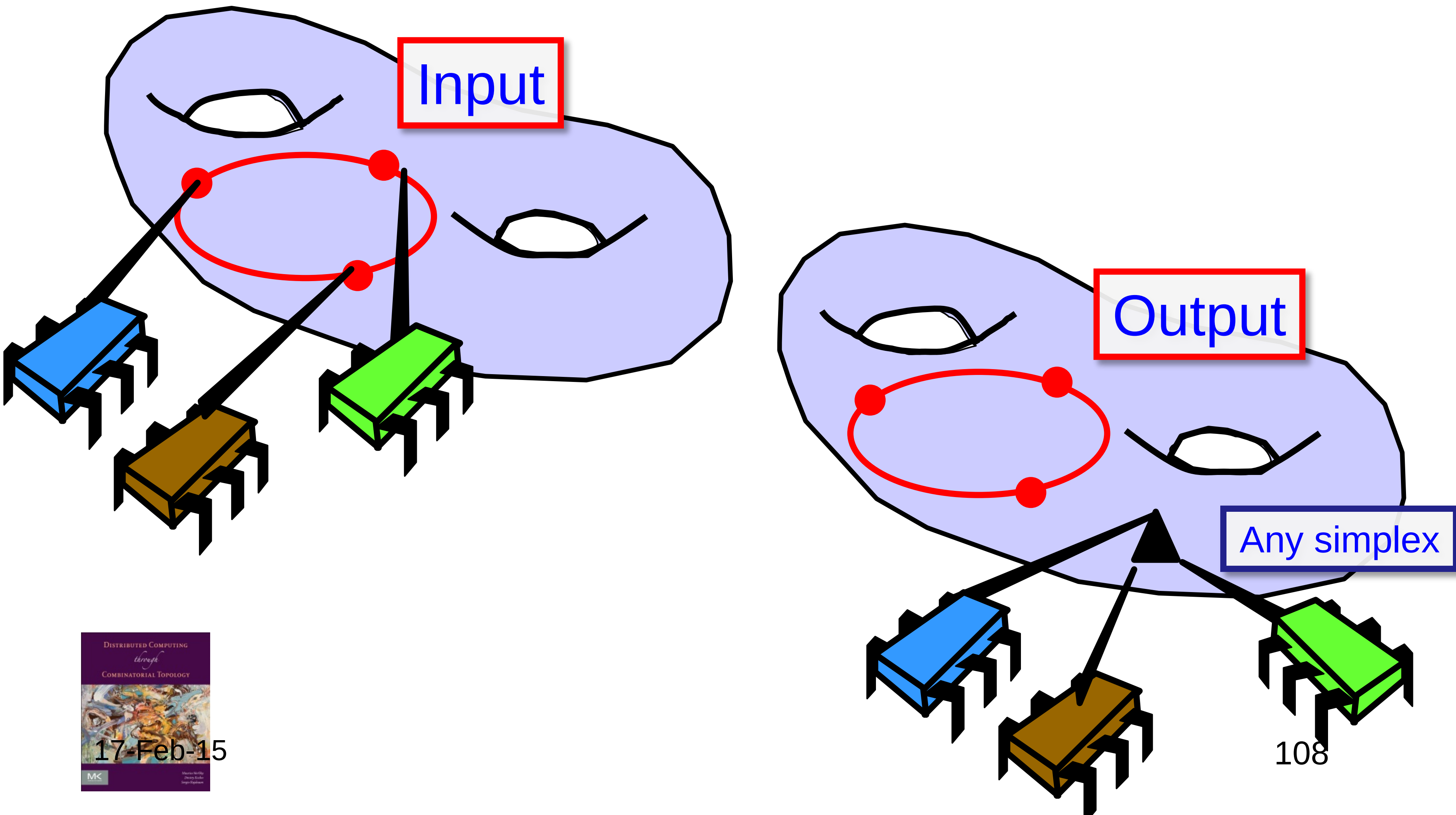
One Rendez-Vous Point



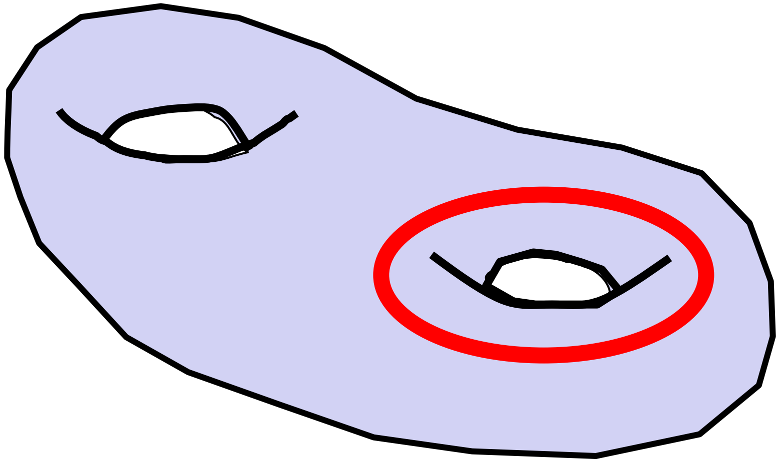
Two Rendez-Vous Points



Three Rendez-Vous Points

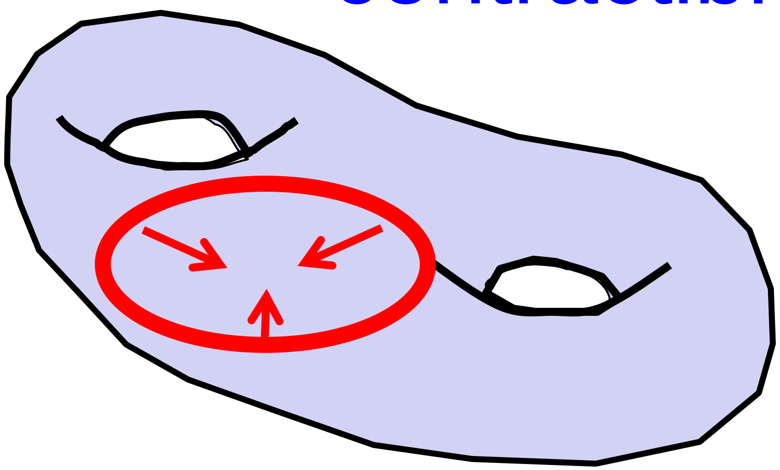


Contractibility

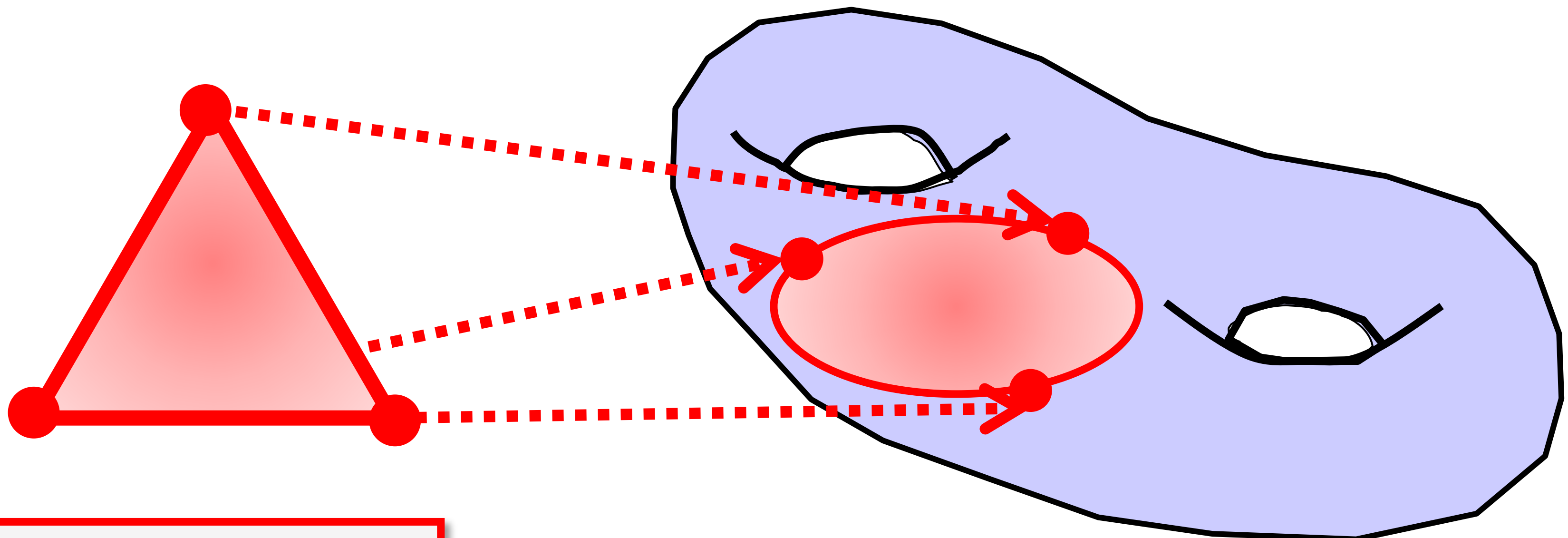


not contractible

contractible



Solvable Iff Loop Contractible



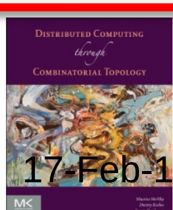
Undecidability

But Contractiblity is *undecidable* …

even for finite complexes!

(reduces to the word problem for finitely-presented groups)

Undecidable whether a task has a protocol in wait-free read-write memory



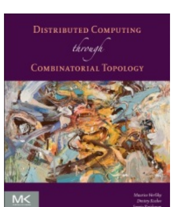
Other Models

Wait-free read-write memory
plus *k*-set agreement , for $k > 2$

Solvable iff f: $\text{skel}^{k-1}\, \mathcal{I}^* \rightarrow \mathcal{O}^*$ exists …

Implies contractible, for $k > 2$

Undecidable whether a task has a
protocol in wait-free read-write memory
plus *k*-set agreement , for $k > 2$


Distributed Computing
through
Combinatorial Topology
MK