The Triumph of Randomization

The Big Picture

- Does randomization make for more powerful algorithms?
  - Does randomization expand the class of problems solvable in polynomial time?
  - Does randomization help compute problems fast in parallel in the PRAM model?

You tell me!

The Triumph of Randomization?

Well, at least for distributed computations!

- no deterministic 1-crash-resilient solution to Consensus
- f resilient randomized solution to consensus $(f < n/2)$ for crash failures
- randomized solution for Consensus exists even for Byzantine failures!

A simple randomized algorithm

M. Ben Or. "Another advantage of free choice: completely asynchronous agreement protocols" (PODC 1983, pp. 27-30)

- exponential number of operations per process
- BUT more practical protocols exist
  - down to $O(n\,log^2 n)$ expected operations/process
  - $n-1$ resilient

The protocol's structure

An infinite repetition of asynchronous rounds

- in round r, p only handles messages with timestamp r
- each round has two phases
- in the first, each p broadcasts an a-value which is a function of the b-values collected in the previous round (the first a-value is the input bit)
- in the second, each p broadcasts a b-value which is a function of the collected a-values
- decide stutters

Ben Or's Algorithm

a_p := input bit; r := 1;
repeat forever
{phase 1}
send (a_p, r) to all
Let A be the multiset of the first $n-f$ a-values with timestamp r received
if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p := v
else b_p := $\perp$
{phase 2}
send (b_p, r) to all
Let B be the multiset of the first $n-f$ b-values with timestamp r received
if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p := v
else if $(\exists b \in B : b \neq \perp)$ then a_p := b
else a_p := $\$$ {$\$$ is chosen uniformly at random to be 0 or 1}
r := $r+1$

Validity

a_p := input bit; r := 1;
repeat forever
{phase 1}
send (a_p, r) to all
Let A be the multiset of the first $n-f$ a values with timestamp r received
if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p := v
else b_p := $\perp$
{phase 2}
send (b_p, r) to all
Let B be the multiset of the first $n-f$ b values with timestamp r received
if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p := v
else if $(\exists b \in B : b \neq \perp)$ then a_p := b
else a_p := $\$$ {$\$$ is chosen uniformly at random to be 0 or 1}
r := $r+1$

Validity

a_p := input bit; r := 1;
repeat forever
{phase 1}
send (a_p, r) to all
Let A be the multiset of the first $n-f$ a values with timestamp r received
if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p := v
else b_p := $\bot$
{phase 2}
send (b_p, r) to all
Let B be the multiset of the first $n-f$ b values with timestamp r received
if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p := v
else if $(\exists b \in B : b \neq \bot)$ then a_p := b
else a_p := \$ {\$ is chosen uniformly at random to be 0 or 1}
r := $r+1$

- All identical inputs (i)
- Each process set a-value := i and broadcasts it to all
- Since at most f faulty, every correct process receives at least $n-f$ identical a-values in round 1
- Every correct process sets b-value := i and broadcasts it to all
- Again, every correct process receives at least $n-f$ identical b-values in round 1 and decides i

A useful observation

a_p := input bit; r := 1;
repeat forever
{phase 1}
send (a_p, r) to all
Let A be the multiset of the first $n-f$ a values with timestamp r received
if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p := v
else b_p := $\bot$
{phase 2}
send (b_p, r) to all
Let B be the multiset of the first $n-f$ b values with timestamp r received
if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p := v
else if $(\exists b \in B : b \neq \bot)$ then a_p := b
else a_p := $\$$ {$\$$ is chosen uniformly at random to be 0 or 1}
r := $r+1$

Lemma For all r, either
$b_{p,r} \in \{1, \bot\}$ for all p or
$b_{p,r} \in \{0, \bot\}$ for all p

A useful observation

```
1:  a_p := input bit; r := 1;
2:  repeat forever
3:  {phase 1}
4:  send (a_p, r) to all
5   Let A be the multiset of the first n−f a values with
        timestamp r received
6:  if (∃v ∈ {0,1} : ∀a ∈ A : a = v) then b_p := v
7:  else b_p := ⊥
8:  {phase 2}
9:  send (b_p, r) to all
10: Let B be the multiset of the first n−f b values with
        timestamp r received
11: if (∃v ∈ {0,1} : ∀b ∈ B : b = v) then decide(v); a_p := v
12: else if (∃b ∈ B : b ≠ ⊥) then a_p := b
13: else a_p := $   {$ is chosen uniformly at random
                     to be 0 or 1}
14: r := r+1
```

Lemma For all r, either

$b_{p,r} \in \{1, \perp\}$ for all p or

$b_{p,r} \in \{0, \perp\}$ for all p

Proof By contradiction.

Suppose p and q at round r such that $b_{p,r} = 0$ and $b_{q,r} = 1$

From lines 6,7 p received $n-f$ distinct 0s, q received $n-f$ distinct 1s

Then, $2(n-f) \leq n$

But this implies $n \leq 2f$ Contradiction

Corollary It is impossible that two processes p and q decide on different values at round r

Agreement

1: a_p := input bit; r := 1;
2: repeat forever
3: {phase 1}
4: send (a_p, r) to all
5 Let A be the multiset of the first $n-f$ a values with timestamp r received
6: if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p := v
7: else b_p := $\perp$
8: {phase 2}
9: send (b_p, r) to all
10: Let B be the multiset of the first $n-f$ b values with timestamp r received
11: if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p := v
12: else if $(\exists b \in B : b \neq \perp)$ then a_p := b
13: else a_p := \$ {\$ is chosen uniformly at random to be 0 or 1}
14: r := r+1

- Let r be the first round in which a decision is made
- Let p be a process that decides in r
- By the Corollary, no other process can decide on a different value in r
- To decide, p must have received $n-f$ "i" from distinct processes
- every other correct process has received "i" from at least $n-2f \geq 1$
- By lines 11 and 12, every correct process sets its new a-value to for round $r+1$ to "i"
- By the same argument used to prove Validity, every correct process that has not decided "i" in round r will do so by the end of round $r+1$

Termination I

```
a_p := input bit; r := 1;
repeat forever
{phase 1}
send (a_p, r) to all
Let A be the multiset of the first n−f  a values with
    timestamp r received
if (∃v ∈ {0,1} : ∀a ∈ A : a = v) then b_p := v
else b_p := ⊥
{phase 2}
send (b_p, r) to all
Let B be the multiset of the first n−f  b values with
    timestamp r received
if (∃v ∈ {0,1} : ∀b ∈ B : b = v)  then decide(v);  a_p := v
else if (∃b ∈ B : b ≠ ⊥)  then a_p := b
else a_p := $    {$ is chosen uniformly at random
                  to be 0 or 1}
r := r+1
```

- Remember that by Validity, if all (correct) processes propose the same value "i" in phase 1 of round r, then every correct process decides "i" in round r.
- The probability of all processes proposing the same input value (a landslide) in round 1 is

  Pr[landslide in round 1] = $1/2^n$

- What can we say about the following rounds?

Termination II

1: a_p := input bit; r := 1;
2: repeat forever
3: {phase 1}
4: send (a_p, r) to all
5 Let A be the multiset of the first $n-f$ a values with timestamp r received
6: if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p := v
7: else b_p := $\perp$
8: {phase 2}
9: send (b_p, r) to all
10: Let B be the multiset of the first $n-f$ b values with timestamp r received
11: if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p := v
12: else if $(\exists b \in B : b \neq \perp)$ then a_p := b
13: else a_p := \$ {\$ is chosen uniformly at random to be 0 or 1}
14: r := $r+1$

- In round r > 1, the a-values are not necessarily chosen at random!
- By line 12, some process may set its a-value to a non-random value v
- By the Lemma, however, all non-random values are identical!
- Therefore, in every r there is a positive probability (at least $1/2^n$) for a landslide
- Hence, for any round r

  Pr[no lanslide at round r] $\leq 1 - 1/2^n$
- Since coin flips are independent:

  Pr[no lanslide for first k rounds] $\leq (1 - 1/2^n)^k$
- When $k = 2^n$, this value is about 1/e; then, if $k = c2^n$

  Pr[landslide within k rounds] ≥

  $1-(1-1/2^n)^k \geq 1-(1-1/e^c)$

  which converges quickly to 0 as c grows

Unreliable Failure Detectors for Reliable Distributed Systems

A different approach

- Augment the asynchronous model with an unreliable failure detector for crash failures
- Define failure detectors in terms of abstract properties, not specific implementations
- Identify classes of failure detectors that allow to solve Consensus

The Model

General

- asynchronous system
- processes fail by crashing
- a failed process does not recover

Failure Detectors

- outputs set of processes that it currently suspects to have crashed
- the set may be different for different processes

Completeness

Strong Completeness   Eventually every process that crashes is permanently suspected by every correct process

Weak Completeness   Eventually every process that crashes is permanently suspected by some correct process

Accuracy

Strong Accuracy

No correct process is ever suspected

Weak Accuracy

Some correct process is never suspected

Accuracy

Strong Accuracy

No correct process is ever suspected

Weak Accuracy

Some correct process is never suspected

Eventual Strong Accuracy

There is a time after which no correct process is ever suspected

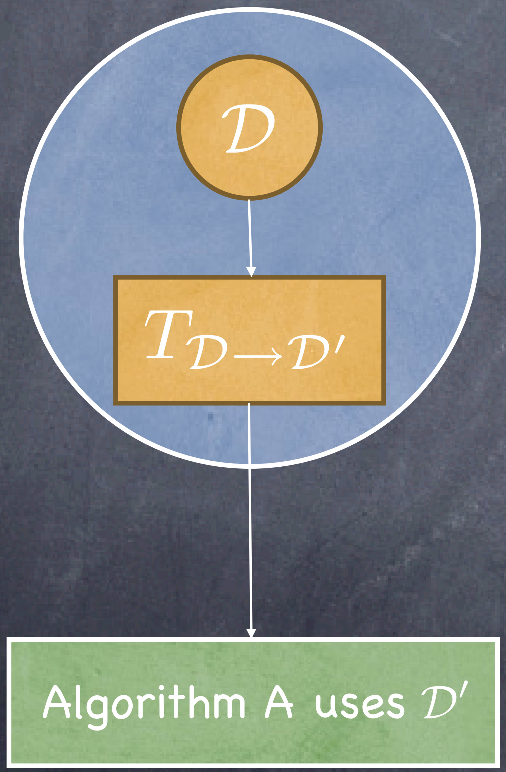
Eventual Weak Accuracy

There is a time after which some correct process is never suspected

Failure detectors

Completeness	Accuracy			
	Strong	Weak	Eventual strong	Eventual weak
Strong	Perfect P	Strong S	$\Diamond P$	$\Diamond S$
Weak	Quasi Q	Weak W	$\Diamond Q$	$\Diamond W$

Reducibility



$T_{\mathcal{D}\rightarrow\mathcal{D}'}$ transforms failure detector $\mathcal{D}$ into failure detector $\mathcal{D}'$

If we can transform $\mathcal{D}$ into $\mathcal{D}'$ then we say that $\mathcal{D}$ is stronger than $\mathcal{D}'$ and that $\mathcal{D}'$ is reducible to $\mathcal{D}$

If $\mathcal{D} \geq \mathcal{D}'$ and $\mathcal{D}' \geq \mathcal{D}$ then we say that $\mathcal{D}$ and $\mathcal{D}'$ are equivalent:

$$\mathcal{D} \equiv \mathcal{D}'$$

Simplify, Simplify!

- All weakly complete failure detectors are reducible to strongly complete failure detectors

$$P \geq Q, \quad S \geq W, \quad \Diamond P \geq \Diamond Q, \quad \Diamond S \geq \Diamond W$$

Simplify, Simplify!

- All weakly complete failure detectors are reducible to strongly complete failure detectors

$$P \geq Q, \quad S \geq W, \quad \Diamond P \geq \Diamond Q, \quad \Diamond S \geq \Diamond W$$

- All strongly complete failure detectors are reducible to weakly complete failure detectors (!)

$$Q \geq P, \quad W \geq S, \quad \Diamond Q \geq \Diamond P, \quad \Diamond W \geq \Diamond S$$

Weakly and strongly complete failure detectors are equivalent!

From Weak Completeness to Strong Completeness

Every process p executes the following:

$output_p$:= 0

cobegin

|| Task 1: repeat forever

{ p queries its local failure detector module $\mathcal{D}_p$ }

$suspects_p$:= $\mathcal{D}_p$

send ($p, suspects_p$) to all

|| Task 2: when receive($q, suspects_q$) from some q

$output_p$:= ($output_p \cup suspects_p$) – $\{q\}$

coend

The Theorems

Theorem 1 In an asynchronous system with W, consensus can be solved as long as $f \leq n-1$

The Theorems

Theorem 1 In an asynchronous system with W, consensus can be solved as long as $f \leq n-1$

Theorem 2 There is no f-resilient consensus protocol using $\Diamond P$ for $f \geq n/2$

The Theorems

Theorem 1 In an asynchronous system with W, consensus can be solved as long as $f \leq n-1$

Theorem 2 There is no f-resilient consensus protocol using $\Diamond P$ for $f \geq n/2$

Theorem 3 In asynchronous systems in which processes can use $\Diamond W$, consensus can be solved as long as $f < n/2$

The Theorems

Theorem 1 In an asynchronous system with W, consensus can be solved as long as $f \leq n-1$

Theorem 2 There is no f-resilient consensus protocol using $\Diamond P$ for $f \geq n/2$

Theorem 3 In asynchronous systems in which processes can use $\Diamond W$, consensus can be solved as long as $f < n/2$

Theorem 4 A failure detector can solve consensus only if it satisfies weak completeness and eventual weak accuracy–i.e. $\Diamond W$ is the weakest failure detector that can solve consensus.

Solving consensus using S

S: Strong Completeness, Weak Accuracy

- at least some correct process c is never suspected

- Each process p has its own failure detector
- Input values are chosen from the set {0,1}

Notation

We introduce the operators $\oplus, \star, \otimes$

They operate element-wise on vectors whose entries have values from the set $\{0, 1, \perp\}$

$$v \star \perp = v \quad \perp \star v = v$$

$$v \star v = \perp \quad \perp \star \perp = \perp$$

$$v \oplus \perp = v \quad \perp \oplus v = v$$

$$v \oplus v = v \quad \perp \oplus \perp = \perp$$

$$v \otimes \perp = \perp \quad \perp \otimes v = \perp$$

$$v \otimes v = v \quad \perp \otimes \perp = \perp$$

Given two vectors A and B, we write $A \leq B$ if

$A[i] \neq \perp$ implies $B[i] \neq \perp$

Solving Consensus using any $\mathcal{D} \in \mathcal{S}$

V_p := $(\bot, \ldots, \bot, v_p, \bot, \ldots, \bot)$ {p's estimate of the proposed values}

Δ_p := $(\bot, \ldots, \bot, v_p, \bot, \ldots, \bot)$

{phase 1} {asynchronous rounds r_p, $1 \leq r_p \leq n-1$}

for r_p := 1 to $n-1$

    send (r_p, Δ_p, p) to all

    wait until [$\forall q$: received (r_p, Δ_q, q) or $q \in \mathcal{D}_p$] {query the failure detector}

    O_p := V_p

    V_p := $V_p \oplus (\oplus_{q\ received}\ \Delta_q)$

    Δ_p := $V_p \star O_p$ {value is only echoed the first time it is seen}

{phase 2}

send (r_p, V_p, p) to all

wait until [$\forall q$: received (r_p, V_q, q) or $q \in \mathcal{D}_p$]

V_p := $\otimes_{q\ received}\ V_q$ {computes the "intersection", including V_p}

{phase 3}

decide on leftmost non-$\bot$ coordinate of V_p

A useful Lemma

```
V_p := (⊥, ..., ⊥, v_p, ⊥, ..., ⊥)
    {p's estimate of the proposed values}
Δ_p := (⊥, ..., ⊥, v_p, ⊥, ..., ⊥)
{phase 1}
{asynchronous rounds r_p, 1≤ r_p ≤ n − 1}
    for r_p := 1 to n-1
        send (r_p, Δ_p ,p) to all
        wait until [∀q: received (r_p, Δ_q ,q) or q ∈ D_p]
        O_p := V_p
        V_p := V_p ⊕ (⊕_q received Δ_q)
        Δ_p := V_p ⋆ O_p        {value is only echoed first time it is seen}
{phase 2}
    send (r_p, V_p ,p) to all
    wait until [∀q: received (r_p, V_q ,q) or q ∈ D_p]
    V_p := ⊗_q received V_q     {computes the "intersection", including V_p}
{phase 3}
    decide on leftmost non- ⊥ coordinate of V_p
```

Lemma 1 After phase 1 is complete, $V_c \leq V_p$ for all processes p that complete phase 1

A useful Lemma

```
Vp := (⊥, …, ⊥, vp, ⊥, …, ⊥)
    {p's estimate of the proposed values}
Δp := (⊥, …, ⊥, vp, ⊥, …, ⊥)
{phase 1}
{asynchronous rounds rp, 1≤ rp ≤ n − 1}
    for rp := 1 to n-1
        send (rp, Δp ,p) to all
        wait until [∀q: received (rp, Δq ,q) or q ∈ 𝒟p]
        Op := Vp
        Vp := Vp ⊕ (⊕q received Δq)
        Δp := Vp ★ Op        {value is only echoed first time it is seen}
{phase 2}
    send (rp, Vp ,p) to all
    wait until [∀q: received (rp, Vq ,q) or q ∈ 𝒟p]
    Vp := ⊗q received Vq    {computes the "intersection", including Vp}
{phase 3}
    decide on leftmost non- ⊥ coordinate of Vp
```

Lemma 1 After phase 1 is complete, $V_c \leq V_p$ for all processes p that complete phase 1

Proof We show that

$$V_c[i] = v_i \wedge v_i \neq \perp \Rightarrow \forall p : V_p[i] = v_i$$

Let r be the first round when c sees v_i

- $r \leq n-2$
  - will send to all Δ_c with v_i in round r
  - By weak accuracy, all correct processes receive v_i by next round
- $r = n-1$
  - v_i has been forwarded $n-1$ times: every other process has seen v_i

Two additional cool lemmas

```
Vp := (⊥, ..., ⊥, vp, ⊥, ..., ⊥)
   {p's estimate of the proposed values}
Δp := (⊥, ..., ⊥, vp, ⊥, ..., ⊥)
{phase 1}
{asynchronous rounds rp, 1≤ rp ≤ n − 1}
    for rp := 1 to n-1
        send (rp, Δp ,p) to all
        wait until [∀q: received (rp, Δq ,q) or q ∈ 𝒟p]
        Op := Vp
        Vp := Vp ⊕ (⊕q received Δq)
        Δp := Vp ★ Op        {value is only echoed first time it is seen}
{phase 2}
    send (rp, Vp ,p) to all
    wait until [∀q: received (rp, Vq ,q) or q ∈ 𝒟p]
    Vp := ⊗q received Vq    {computes the "intersection", including Vp}
{phase 3}
    decide on leftmost non- ⊥ coordinate of Vp
```

Lemma 2 After phase 2 is complete, $V_c = V_p$ for each p that completes phase 1

Proof

- All processes that completed phase 2 have received V_c. Since V_c is the smallest V vector,

$$V_c[i] \neq \bot \Rightarrow V_p[i] \neq \bot \quad \forall p$$

- By the definition of $\otimes$

$$V_c[i] = \bot \Rightarrow V_p[i] = \bot \quad \forall p$$

after phase 2

Lemma 3 $V_c \neq (\bot, \bot, \bot, \ldots, \bot)$

Solving consensus

```
Vp := (⊥, …, ⊥, vp, ⊥, …, ⊥)
    {p's estimate of the proposed values}
Δp := (⊥, …, ⊥, vp, ⊥, …, ⊥)
{phase 1}
{asynchronous rounds rp, 1≤ rp ≤ n − 1}
    for rp := 1 to n-1
        send (rp, Δp ,p) to all
        wait until [∀q: received (rp, Δq ,q) or q ∈ 𝒟p]
        Op := Vp
        Vp := Vp ⊕ (⊕q received Δq)
        Δp := Vp ★ Op        {value is only echoed first time it
is seen}
{phase 2}
    send (rp, Vp ,p) to all
    wait until [∀q: received (rp, Vq ,q) or q ∈ 𝒟p]
    Vp := ⊗q received Vq    {computes the "intersection",
including Vp}
{phase 3}
    decide on leftmost non- ⊥ coordinate of Vp
```

Theorem The protocol to the left satisfies Validity, Agreement, and Termination

Proof

Left as an exercise

A lower bound - I

Theorem Consensus with $\Diamond P$ requires $f < n/2$

A lower bound - I

Theorem Consensus with $\Diamond P$ requires $f < n/2$

Proof

- Suppose n is even, and a protocol exists that solves consensus when $f = n/2$
- Divide the set of processes in two sets of size $n/2$, P_1 and P_2

A lower bound - II

Consider three executions:

$P_1 \leftarrow 0; P_2 \leftarrow 0$

All processes in P_2 crash before they can propose

Detectors work perfectly

A lower bound - II

Consider three executions:

$P_1 \leftarrow 0; P_2 \leftarrow 0$

All processes in P_2 crash before they can propose

Detectors work perfectly

P_1 decides 0

after t_1

A lower bound - II

Consider three executions:

$P_1 \leftarrow 0; P_2 \leftarrow 0$

All processes in P_2 crash before they can propose

Detectors work perfectly

$P_1 \leftarrow 1; P_2 \leftarrow 1$

All processes in P_1 crash before they can propose

Detectors work perfectly

P_1 decides 0

after t_1

A lower bound - II

Consider three executions:

$P_1 \leftarrow 0; P_2 \leftarrow 0$

All processes in P_2 crash before they can propose

Detectors work perfectly

P_1 decides 0

after t_1

$P_1 \leftarrow 1; P_2 \leftarrow 1$

All processes in P_1 crash before they can propose

Detectors work perfectly

P_2 decides 1

after t_2

A lower bound - II

Consider three executions:

$P_1 \leftarrow 0; P_2 \leftarrow 0$ All processes in P_2 crash before they can propose Detectors work perfectly	$P_1 \leftarrow 0; P_2 \leftarrow 1$ No process crashes Detectors make mistakes: until $max(t_1, t_2)$, P_1 believes P_2 crashed, and vice versa	$P_1 \leftarrow 1; P_2 \leftarrow 1$ All processes in P_1 crash before they can propose Detectors work perfectly
P_1 decides 0		P_2 decides 1
after t_1		after t_2

A lower bound - II

Consider three executions:

$P_1 \leftarrow 0; P_2 \leftarrow 0$ All processes in P_2 crash before they can propose Detectors work perfectly	$P_1 \leftarrow 0; P_2 \leftarrow 1$ No process crashes Detectors make mistakes: until $max(t_1, t_2)$, P_1 believes P_2 crashed, and vice versa	$P_1 \leftarrow 1; P_2 \leftarrow 1$ All processes in P_1 crash before they can propose Detectors work perfectly
P_1 decides 0 after t_1	P_1 decides 0 P_2 decides 1	P_2 decides 1 after t_2

The case of the Rotating Coordinator

Solving consensus with $\Diamond W$ (actually, $\Diamond S$)

- Asynchronous rounds
- In round r, only messages timestamped r are sent and processed (except for DECIDE messages)
- Each process p has an opinion $v_p \in \{0,1\}$
- Each opinion has a time of adoption t_p (initially, $t_p = 0$
- Each round has a coordinator c such that $c_{id} = (r \ mod \ n)+1$

One round, four phases

Phase 1

Each process, including c, sends its opinion timestamped r

One round, four phases

Phase 1

Each process, including c, sends its opinion timestamped r

Phase 2

c waits for first $|n/2+1|$ opinions with timestamp r

c selects v, one of the most recently adopted opinions

v becomes c's suggestion for round r

c sends its suggestion to all

One round, four phases

Phase 1

Each process, including c, sends its opinion timestamped r

Phase 2

c waits for first $\lfloor n/2+1 \rfloor$ opinions with timestamp r

c selects v, one of the most recently adopted opinions

v becomes c's suggestion for round r

c sends its suggestion to all

Phase 3

Each p waits for a suggestion, or for failure detector to signal c is faulty

If a suggestion is received, it is adopted: $v_p := v$; $t_p := r$; ACK to c

Otherwise, NACK to c

One round, four phases

Phase 1

Each process, including c, sends its opinion timestamped r

Phase 2

c waits for first $\lfloor n/2+1 \rfloor$ opinions with timestamp r

c selects v, one of the most recently adopted opinions

v becomes c's suggestion for round r

c sends its suggestion to all

Phase 3

Each p waits for a suggestion, or for failure detector to signal c is faulty

If a suggestion is received, it is adopted: $v_p := v$; $t_p := r$; ACK to c

Otherwise, NACK to c

Phase 4

c waits for first $\lfloor n/2+1 \rfloor$ responses

if all ACKs, then c decides on v and sends DECIDE to all

if p receives DECIDE, then p decides on v

Consensus using $\Diamond S$

v_p := input bit; r := 0; t_p := 0; $state_p$:= undecided

while p undecided do

r := $r+1$

c := $(r\ mod\ n)$ + 1

{phase 1: all processes send opinion to current coordinator}

p sends (p, r, v_p, t_p) to c

{phase 2: current coordinator gather a majority of opinions}

c waits for first $\lceil n/2{+}1 \rceil$ opinions (q, r, v_q, t_q)

c selects among them the value v_q with the largest t_q

c sends (c, r, v_q) to all

{phase 3: all processes wait for new suggestions from the current coordinator}

p waits until suggestion (c, r, v) arrives or $c \in \Diamond S_p$

if suggestion is received then {v_p := v; t_p := r; p sends (r, ACK) to c}

else p sends (r, NACK) to c

{phase 4: coordinator waits for majority of replies. If majority adopted the coordinator's suggestion, then coordinator sends request to decide}

c waits for first $\lceil n/2{+}1 \rceil$ (r, ACK) or (r, NACK)

if c receives $\lceil n/2{+}1 \rceil$ ACKS, then c sends (r, DECIDE, v) to all

when p delivers (r, DECIDE, v) then {p decides v ; $state_p$:= decided}

Validity

v_p := input bit; r := 0; t_p := 0; $state_p$:= undecided
while p undecided do
 r := r+1
 c := (r mod n) + 1
 {phase 1: all processes send their opinion to current coordinator}
  p sends (p, r, v_p, t_p) to c
  {phase 2: current coordinator gather a majority of opinions}
 c waits for first $\lceil n/2+1 \rceil$ opinions (q, r, v_q, t_q)
  c selects among them the value v_q with largest t_q
  c sends (c, r, v_q) to all
 {phase 3: all processes wait for new suggestions from the current coordinator}
  p waits until suggestion (c, r, v) arrives or c ∈ ◊S_p
  if the suggestion is received then
   {v_p := v; t_p := r; p sends (r, ACK) to c }
  else p sends (r, NACK) to c
 {phase 4: coordinator waits for majority of replies. If majority adopted the coordinator's suggestion, then coordinator sends request to decide}
  c waits for first $\lceil n/2+1 \rceil$ (r, ACK) or (r, NACK)
  if c receives $\lceil n/2+1 \rceil$ ACKs, then
   c sends (r, DECIDE, v) to all
when p delivers (r, DECIDE, v) then
  {p decides v ; $state_p$:= decided}

- The value decided upon must have been suggested by the coordinator in some round
- A coordinator suggests a value only by selecting it among the participants' opinions
- From the algorithm, it is clear that each opinion correspond to a value proposed by some process

Agreement

```
vp := input bit; r := 0; tp := 0; statep:= undecided
while p undecided do
 r := r+1
 c := (r mod n) + 1
 {phase 1: all processes send their opinion to current coordinator}
   p sends (p, r, vp, tp) to c
   {phase 2: current coordinator gather a majority of opinions}
 c waits for first ⌈n/2+1⌉ opinions (q, r, vq, tq)
   c selects among them the value vq with largest tq
   c sends (c, r, vq) to all
 {phase 3: all processes wait for new suggestions from the current
 coordinator}
   p waits until suggestion (c, r, v) arrives or c ∈ ◊Sp
   if the suggestion is received then
      {vp := v; tp := r; p  sends (r, ACK) to c }
   else p  sends (r, NACK) to c
 {phase 4: coordinator waits for majority of replies. If majority adopted
the coordinator's suggestion, then coordinator sends request to decide}
   c waits for first ⌈n/2+1⌉ (r, ACK) or (r, NACK)
   if c receives ⌈n/2+1⌉ ACKs, then
      c sends (r, DECIDE, v) to all
when p delivers (r, DECIDE, v) then
   {p decides v ; statep := decided}
```

Strong Agreement All processes that decide, decide the same value

Proof

- Trivially true if no process decides
- If some process decides, it has delivered (-, DECIDE, -) from a coordinator
- The coordinator has received a majority of (-, ACK)
- Let r be the earliest round in which a majority of (-, ACK) have been **sent** to the coordinator c of r
- Let v_c be the value suggested by c in Phase 2 of round r
- Enter the Locking Lemma!

The Locking Lemma - I

```
vp := input bit; r := 0; tp := 0; statep:= undecided
while p undecided do
 r := r+1
 c := (r mod n) + 1
 {phase 1: all processes send their opinion to current coordinator}
   p sends (p, r, vp, tp) to c
   {phase 2: current coordinator gather a majority of opinions}
 c waits for first ⌈n/2+1⌉ opinions (q, r, vq, tq)
   c selects among them the value vq with largest tq
   c sends (c, r, vq) to all
 {phase 3: all processes wait for new suggestions from the current
 coordinator}
   p waits until suggestion (c, r, v) arrives or c ∈ ◊Sp
   if the suggestion is received then
     {vp := v; tp := r; p  sends (r, ACK) to c }
   else p  sends (r, NACK) to c
 {phase 4: coordinator waits for majority of replies. If majority adopted
the coordinator's suggestion, then coordinator sends request to decide}
   c waits for first ⌈n/2+1⌉ (r, ACK) or (r, NACK)
   if c receives ⌈n/2+1⌉ ACKs, then
     c sends (r, DECIDE, v) to all
when p delivers (r, DECIDE, v) then
   {p decides v ; statep := decided}
```

Locking Lemma For all rounds r': $r' \geq r$ if a coordinator c' sends $v_{c'}$, then $v_{c'} = v_c$

Proof

- Trivially holds for $r' = r$
- Assume it holds for all $r' : r \leq r' < k$
- Let c_k be the coordinator for round k
- If c_k suggests v_{c_k}, it must have received opinions from a majority of processes
- There exists some p that sent an ACK in Phase 3 of round r and whose opinion has been received by c_k
- Consider the time of adoption t_p
- In Phase 3 of round r, $t_p = r$
- In Phase 2 of round k, $t_p \geq r$
- For any t_q collected in round k, $t_q < k$

The Locking Lemma - II

```
vp := input bit; r := 0; tp := 0; statep:= undecided
while p undecided do
 r := r+1
 c := (r mod n) + 1
 {phase 1: all processes send their opinion to current coordinator}
   p sends (p, r, vp, tp) to c
   {phase 2: current coordinator gather a majority of opinions}
 c waits for first ⌈n/2+1⌉ opinions (q, r, vq, tq)
   c selects among them the value vq with largest tq
   c sends (c, r, vq) to all
 {phase 3: all processes wait for new suggestions from the current
 coordinator}
   p waits until suggestion (c, r, v) arrives or c ∈ ◊Sp
   if the suggestion is received then
     {vp := v; tp := r; p  sends (r, ACK) to c }
   else p  sends (r, NACK) to c
 {phase 4: coordinator waits for majority of replies. If majority adopted
the coordinator's suggestion, then coordinator sends request to decide}
   c waits for first ⌈n/2+1⌉ (r, ACK) or (r, NACK)
   if c receives ⌈n/2+1⌉ ACKs, then
     c sends (r, DECIDE, v) to all
when p delivers (r, DECIDE, v) then
   {p decides v ; statep := decided}
```

- Consider t, the largest time of adoption collected by c_k. Clearly, $r \leq t < k$
- c_k adopted its suggestion from q, where q is the process that sent (q, k, v_q, t)
- The coordinator of round t sent its suggestion in Phase 2 of round t, where $r \leq t < k$
- By the Induction Hypothesis, that coordinator sent v_c!
- Then, c_k sets v_{c_k} to v_c

Been there, done that?

Agreement

```
vp := input bit; r := 0; tp := 0; statep:= undecided
while p undecided do
 r := r+1
 c := (r mod n) + 1
 {phase 1: all processes send their opinion to current coordinator}
    p sends (p, r, vp, tp) to c
    {phase 2: current coordinator gather a majority of opinions}
 c waits for first ⌈n/2+1⌉ opinions (q, r, vq, tq)
    c selects among them the value vq with largest tq
    c sends (c, r, vq) to all
 {phase 3: all processes wait for new suggestions from the current
 coordinator}
    p waits until suggestion (c, r, v) arrives or c ∈ ◊Sp
    if the suggestion is received then
       {vp := v; tp := r; p  sends (r, ACK) to c }
    else p  sends (r, NACK) to c
 {phase 4: coordinator waits for majority of replies. If majority adopted
the coordinator's suggestion, then coordinator sends request to decide}
    c waits for first ⌈n/2+1⌉ (r, ACK) or (r, NACK)
    if c receives ⌈n/2+1⌉ ACKs, then
       c sends (r, DECIDE, v) to all
 when p delivers (r, DECIDE, v) then
    {p decides v ; statep := decided}
```

All processes that decide, decide v_c

Proof

- Suppose p delivers (r^*, DECIDE, v_{c^*})
- The coordinator c^* for round r^* has sent (r^*, DECIDE, v_{c^*}) in Phase 4 of round r^*
- To do so c^* must have received a majority of (r^*,ACK) in Phase 4 of r^*
- r is the earliest round in which a majority of (r, ACK) have been sent to a round's coordinator
- Clearly, $r \leq r^*$
- By the locking Lemma, c' must have suggested the locked value: $v_{c^*} = v_c$

Termination

v_p := input bit; r := 0; t_p := 0; $state_p$:= undecided
while p undecided do
 r := r+1
 c := (r mod n) + 1
 {phase 1: all processes send their opinion to current coordinator}
  p sends (p, r, v_p, t_p) to c
  {phase 2: current coordinator gather a majority of opinions}
 c waits for first ⌈n/2+1⌉ opinions (q, r, v_q, t_q)
  c selects among them the value v_q with largest t_q
  c sends (c, r, v_q) to all
 {phase 3: all processes wait for new suggestions from the current coordinator}
  p waits until suggestion (c, r, v) arrives or c ∈ $\diamond S_p$
  if the suggestion is received then
   {v_p := v; t_p := r; p sends (r, ACK) to c }
  else p sends (r, NACK) to c
 {phase 4: coordinator waits for majority of replies. If majority adopted the coordinator's suggestion, then coordinator sends request to decide}
  c waits for first ⌈n/2+1⌉ (r, ACK) or (r, NACK)
  if c receives ⌈n/2+1⌉ ACKs, then
   c sends (r, DECIDE, v) to all
when p delivers (r, DECIDE, v) then
  {p decides v ; $state_p$:= decided}

- No correct process is blocked forever at a wait statement
- By eventual weak accuracy, there is a correct process c and a time t such that no process suspects c after t
- There is a round r such that:
  - all correct processes reach r after time t (no one suspects c)
  - c is the coordinator for round r
- If some correct process decides, eventually all do on the same value by Agreement