Bonus Lecture #1: The FLP Impossibility Theorem

COMS 4995-001:
The Science of Blockchains
URL: https://timroughgarden.org/s25/

Tim Roughgarden

Goals for Bonus Lecture #1

1. Understanding the asynchronous model.
   - what does "no assumptions on message delays" mean?

2. Proof of the FLP Theorem.
   - state machine replication (SMR) is "unsolvable" in asynchrony
   - need to compromise to make further progress
     - pull back to "partial synchrony" (see next lecture)
     - relax consistency guarantees (could be a good project)
     - randomized protocols that succeed with high probability
       - could also be a good project

SMR: Synchrony vs. Asynchrony

Lecture #3: in the synchronous model, can solve the SMR problem (i.e., via a consistent and live protocol), even with an arbitrary number of crash faults.

- uncrashed validators remain consistent, guarantee liveness

SMR: Synchrony vs. Asynchrony

Lecture #3: in the synchronous model, can solve the SMR problem (i.e., via a consistent and live protocol), even with an arbitrary number of crash faults.

- uncrashed validators remain consistent, guarantee liveness

FLP Theorem: in the asynchronous model, even with the threat of just one crash fault, can't solve SMR via any protocol.

  - ouch!

SMR: Synchrony vs. Asynchrony

Lecture #3: in the synchronous model, can solve the SMR problem (i.e., via a consistent and live protocol), even with an arbitrary number of crash faults.

- uncrashed validators remain consistent, guarantee liveness

FLP Theorem: in the asynchronous model, even with the threat of just one crash fault, can't solve SMR via any protocol.

  – ouch!

Question: what's the "asynchronous model"?

The Setup (≈ Asynchronous Model)

- shared global clock, timesteps 0,1,2,…
  - traditional asynchronous model does not have this (only makes today's impossibility result stronger)
- pool M of outstanding messages (sent but not yet received)

The Setup (≈ Asynchronous Model)

- shared global clock, timesteps 0,1,2,…
- pool M of outstanding messages (sent but not yet received)
- at each timestep t=0,1,2,…

The Setup (≈ Asynchronous Model)

- shared global clock, timesteps 0,1,2,…
- pool M of outstanding messages (sent but not yet received)
- at each timestep t=0,1,2,…
  1. adversary decides:
     - which messages of M to deliver to their recipients (if any)
     - which validators to crash (if any)

The Setup (≈ Asynchronous Model)

- shared global clock, timesteps 0,1,2,…
- pool M of outstanding messages (sent but not yet received)
- at each timestep t=0,1,2,…
  1. adversary decides:
     - which messages of M to deliver to their recipients (if any)
     - which validators to crash (if any)
  2. non-crashed validators decide which txs to finalize, messages to send
     - as instructed by whatever protocol they're running
     - messages sent injected directly into M

The Setup (≈ Asynchronous Model)

- shared global clock, timesteps 0,1,2,…
- pool M of outstanding messages (sent but not yet received)
- at each timestep t=0,1,2,…
  1. adversary decides:
     - which messages of M to deliver to their recipients (if any)
     - which validators to crash (if any)
  2. non-crashed validators decide which txs to finalize, messages to send
- constraints on adversary:
  - only allowed to crash (at most) one validator
  - every message sent must eventually get delivered

The Setup (≈ Asynchronous Model)

- shared global clock, timesteps 0,1,2,…
- pool M of outstanding messages (sent but not yet received)
- at each timestep t=0,1,2,…
  1. adversary decides which messages of M to deliver to their recipients (if any) and which validators to crash (if any)
  2. non-crashed validators decide which txs to finalize, messages to send
  – constraints on adversary: only allowed to crash (at most) one validator, and every message sent must eventually get delivered
- at most two transactions exist, *a* & *b*

The Setup (≈ Asynchronous Model)

- shared global clock, timesteps 0,1,2,…
- pool M of outstanding messages (sent but not yet received)
- at each timestep t=0,1,2,…
    1. adversary decides which messages of M to deliver to their recipients (if any) and which validators to crash (if any)
    2. non-crashed validators decide which txs to finalize, messages to send

    – constraints on adversary: only allowed to crash (at most) one validator, and every message sent must eventually get delivered
- at most two transactions exist, *a* & *b*
- each validator receives either *a* or *b* at the start of the protocol

The FLP Impossibility Theorem

- shared global clock, timesteps 0,1,2,…
- pool M of outstanding messages (sent but not yet received)
- at each timestep t=0,1,2,…
    1. adversary decides which messages of M to deliver to their recipients (if any) and which validators to crash (if any)
    2. non-crashed validators decide which txs to finalize, messages to send
- constraints on adversary: only allowed to crash (at most) one validator, and every message sent must eventually get delivered
- at most two transactions exist, *a* & *b*
- each validator receives either *a* or *b* at the start of the protocol

Theorem: [FLP85] no SMR protocol guarantees consistency and liveness in the setup above.

Preliminaries

- “input 0” = tx *a*, “input 1” = tx *b* [each validator gets input 0 or 1]

Preliminaries

- "input 0" = tx *a*, "input 1" = tx *b* [each validator gets input 0 or 1]
- validator i "outputs 0" (respectively, "outputs 1") if tx *a* (respectively, tx *b*) is the first tx it finalizes

Preliminaries

- "input 0" = tx *a*, "input 1" = tx *b* [each validator gets input 0 or 1]
- validator i "outputs 0" (respectively, "outputs 1") if tx *a* (respectively, tx *b*) is the first tx it finalizes

Assume [for contradiction]: protocol Π guarantees consistency and liveness in the preceding setup.

Preliminaries

- “input 0” = tx *a*, “input 1” = tx *b* [each validator gets input 0 or 1]
- validator i “outputs 0” (respectively, “outputs 1”) if tx *a* (respectively, tx *b*) is the first tx it finalizes

Assume [for contradiction]: protocol Π guarantees consistency and liveness in the preceding setup.

- “protocol” = specifies what validators should do in each timestep
  - as a function of their input, the timestep, and messages received
- think of Π as deterministic (or with adversary-controlled randomness)

Preliminaries (con'd)

Assume [for contradiction]: protocol Π guarantees consistency and liveness in the preceding setup.

Consequences:

Preliminaries (con'd)

Assume [for contradiction]: protocol Π guarantees consistency and liveness in the preceding setup.

Consequences:

1. liveness of Π ➔ every non-faulty validator eventually outputs 0 or 1

Preliminaries (con'd)

Assume [for contradiction]: protocol Π guarantees consistency and liveness in the preceding setup.

Consequences:

1. liveness of Π ➔ every non-faulty validator eventually outputs 0 or 1
2. consistency of Π ➔ all non-faulty validators eventually output the same thing

Preliminaries (con’d)

Assume [for contradiction]: protocol Π guarantees consistency and liveness in the preceding setup.

Consequences:

1. liveness of Π ➔ every non-faulty validator eventually outputs 0 or 1
2. consistency of Π ➔ all non-faulty validators eventually output the same thing
3. if all inputs are 0 (respectively, 1) ➔ all outputs are 0 (respectively, 1)

Configurations

Definition: a *configuration* C := the state of all validators and the message pool M at the beginning of a timestep.

- "state" of validator = input and messages received (and when)
  - snapshot of an execution at the beginning of some timestep

Configurations

Definition: a *configuration* C := the state of all validators and the message pool M at the beginning of a timestep.

- "state" of validator = input and messages received (and when)
  - snapshot of an execution at the beginning of some timestep

Note: strategy of adversary in a timestep (which messages to deliver, validator to crash) induces a transition C → C'.

Configurations

Definition: a *configuration* C := the state of all validators and the message pool M at the beginning of a timestep.

- "state" of validator = input and messages received (and when)
  - snapshot of an execution at the beginning of some timestep

Note: strategy of adversary in a timestep (which messages to deliver, validator to crash) induces a transition C → C'.

Proof plan: devise strategy of adversary resulting in an infinite sequence $C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow$ …. of configurations such that no validator outputs in any C_t. [note: would contradict liveness]

The Value of a Configuration

Definition: a *benign adversary* always delivers all messages in the pool M and never crashes any validators.

The Value of a Configuration

Definition: a *benign adversary* always delivers all messages in the pool M and never crashes any validators.

Definition: for a configuration C, *val(C)* := the output of the protocol ∏ with an adversary that is benign from C onward.

The Value of a Configuration

Definition: a *benign adversary* always delivers all messages in the pool M and never crashes any validators.

Definition: for a configuration C, *val(C)* := the output of the protocol ∏ with an adversary that is benign from C onward.

- i.e., val(C)=0 if all validators eventually output 0
- i.e., val(C)=1 if all validators eventually output 1

The Value of a Configuration

Definition: a *benign adversary* always delivers all messages in the pool M and never crashes any validators.

Definition: for a configuration C, *val(C)* := the output of the protocol ∏ with an adversary that is benign from C onward.

- i.e., val(C)=0 if all validators eventually output 0
- i.e., val(C)=1 if all validators eventually output 1
- note: by consequences (1)-(3) above, no other possibilities
  - (technically, defined only for configurations C with at most one crash)

The Value of a Configuration

Definition: a *benign adversary* always delivers all messages in the pool M and never crashes any validators.

Definition: for a configuration C, *val(C)* := the output of the protocol Π with an adversary that is benign from C onward.

- i.e., val(C)=0 if all validators eventually output 0
- i.e., val(C)=1 if all validators eventually output 1
- note: by consequences (1)-(3) above, no other possibilities

Next: define a "pivotal" configuration as (roughly) one in which crashing a validator flips the output of the protocol.

Pivotal Configurations

Definition: for a validator i, a configuration C is *i-restricted* if:

Pivotal Configurations

Definition: for a validator i, a configuration C is *i-restricted* if:

- no validators have crashed yet

Pivotal Configurations

Definition: for a validator i, a configuration C is *i-restricted* if:

- no validators have crashed yet
- messages in M from i = all messages sent by i in some interval {t',t'+1,…,t-1}   [where t = current timestep in configuration C]
  - all messages i sent before t' already delivered
  - nobody has heard anything from i from t' onward

Pivotal Configurations

Definition: for a validator i, a configuration C is *i-restricted* if:

- no validators have crashed yet
- messages in M from i = all messages sent by i in some interval {t',t'+1,…,t-1}   [where t = current timestep in configuration C]
  - all messages i sent before t' already delivered
  - nobody has heard anything from i from t' onward
  - as far as other validators j≠i can tell, i crashed at time t'
    - only difference is the state of M, which validators do not observe

Pivotal Configurations

Definition: for a validator i, a configuration C is *i-restricted* if:

- no validators have crashed yet
- messages in M from i = all messages sent by i in some interval {t',t'+1,…,t-1} [where t = current timestep in configuration C]

Definition: for an i-restricted configuration C, *val(C \ i)* := output of the protocol Π with an adversary that:

- at timesteps < t: behaves identically to the adversary in C (delivers same msgs each timestep) except it crashes i at t'
- at timesteps ≥ t: is benign

Pivotal Configurations

Definition: for a validator i, a configuration C is *i-restricted* if:

- no validators have crashed yet
- messages in M from i = all messages sent by i in some interval {t',t'+1,…,t-1} [where t = current timestep in configuration C]

Definition: for an i-restricted configuration C, *val(C \ i)* := output of the protocol ∏ with an adversary that:

- at timesteps < t: behaves identically to the adversary in C (delivers same msgs each timestep) except it crashes i at t'
- at timesteps ≥ t: is benign

Definition: an i-restricted C is *i-pivotal* if val(C) ≠ val(C \ i).

- key point: C pivotal ➔ no validators have output yet (why?)

An Infinite Sequence of Pivotal Configurations

Definition: an i-restricted C is *i-pivotal* if val(C) ≠ val(C \ i).

- key point: C pivotal ➔ no validators have output yet (why?)

An Infinite Sequence of Pivotal Configurations

Definition: an i-restricted C is *i-pivotal* if val(C) ≠ val(C \ i).

- key point: C pivotal ➔ no validators have output yet (why?)

Recall proof plan: devise strategy of adversary resulting in an infinite sequence $C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow$ …. of configurations such that no validator outputs in any C_t. [contradicts liveness]

- suffices to use only pivotal configurations
- we will exhibit such a sequence, inductively

Base Case: A Pivotal Initial Configuration

Base Case: A Pivotal Initial Configuration

- let X_i = initial configuration in which validators 1,2,…,i have input 1 and validators i+1,i+2,…,n have input 0
  – note: all X_i's j-restricted for all j   [no crashes, M is empty]

Base Case: A Pivotal Initial Configuration

- let X_i = initial configuration in which validators 1,2,…,i have input 1 and validators i+1,i+2,…,n have input 0
  - note: all X_i's j-restricted for all j   [no crashes, M is empty]
- note: for some i ≥1, $val(X_{i-1})=0$ and $val(X_i)=1$
  - follows from fact that $val(X_0)=0$ and $val(X_n)=1$

Base Case: A Pivotal Initial Configuration

- let X_i = initial configuration in which validators 1,2,…,i have input 1 and validators i+1,i+2,…,n have input 0
  - note: all X_i's j-restricted for all j [no crashes, M is empty]
- note: for some i ≥1, $val(X_{i-1})=0$ and $val(X_i)=1$
  - follows from fact that $val(X_0)=0$ and $val(X_n)=1$
- on the other hand: $val(X_{i-1} \setminus i) = val(X_i \setminus i)$
  - if i crashes immediately, doesn't matter whether its input was 0 or 1

Base Case: A Pivotal Initial Configuration

- let X_i = initial configuration in which validators 1,2,…,i have input 1 and validators i+1,i+2,…,n have input 0
  - note: all X_i's j-restricted for all j   [no crashes, M is empty]
- note: for some i ≥1, val(X_{i-1})=0 and val(X_i)=1
  - follows from fact that val(X_0)=0 and val(X_n)=1
- on the other hand: val(X_{i-1} \ i) = val(X_i \ i)
  - if i crashes immediately, doesn't matter whether its input was 0 or 1
  - in general: if validator sees identical messages at every timestep in two different executions, will behave identically (including the same output)

Base Case: A Pivotal Initial Configuration

- let X_i = initial configuration in which validators 1,2,…,i have input 1 and validators i+1,i+2,…,n have input 0
  - note: all X_i's j-restricted for all j   [no crashes, M is empty]
- note: for some $i \geq 1$, $val(X_{i-1})=0$ and $val(X_i)=1$
  - follows from fact that $val(X_0)=0$ and $val(X_n)=1$
- on the other hand: $val(X_{i-1} \setminus i) = val(X_i \setminus i)$
  - if i crashes immediately, doesn't matter whether its input was 0 or 1
- so: either (i) $val(X_{i-1} \setminus i) = 1$ (in which case X_{i-1} is i-pivotal) or (ii) $val(X_i \setminus i) = 0$ (in which case X_i is i-pivotal)
  - either way, we have our initial pivotal configuration C_0

Inductive Step: Extending the Sequence

- let C_t be an i-pivotal configuration (need to exhibit pivotal C_{t+1})

Inductive Step: Extending the Sequence

- let C_t be an i-pivotal configuration (need to exhibit pivotal C_{t+1})
- consider transition $C_t \rightarrow Y$ if adversary delivers all messages of M except those sent by i (and doesn't crash anybody)
  - note: because C_t is i-restricted, so is Y [with the same value of t']

Inductive Step: Extending the Sequence

- let C_t be an i-pivotal configuration (need to exhibit pivotal C_{t+1})
- consider transition $C_t \rightarrow Y$ if adversary delivers all messages of M except those sent by i (and doesn't crash anybody)
  - note: because C_t is i-restricted, so is Y [with the same value of t']
- easy case: Y is also i-pivotal [done, just take C_{t+1} = Y]

Inductive Step: Extending the Sequence

- let C_t be an i-pivotal configuration (need to exhibit pivotal C_{t+1})
- consider transition $C_t \rightarrow Y$ if adversary delivers all messages of M except those sent by i (and doesn't crash anybody)
  - note: because C_t is i-restricted, so is Y [with the same value of t']
- easy case: Y is also i-pivotal [done, just take C_{t+1} = Y]
- harder case: Y not i-pivotal. picture then is:

Inductive Step: Extending the Sequence

- let C_t be an i-pivotal configuration (need to exhibit pivotal C_{t+1})
- consider transition $C_t \rightarrow Y$ if adversary delivers all messages of M except those sent by i (and doesn't crash anybody)
  - note: because C_t is i-restricted, so is Y [with the same value of t']
- easy case: Y is also i-pivotal [done, just take C_{t+1} = Y]
- harder case: Y not i-pivotal. picture then is:

in harder case, Y is not i-pivotal

val(Y) = val(Y \ i)

Inductive Step: Extending the Sequence

- let C_t be an i-pivotal configuration (need to exhibit pivotal C_{t+1})
- consider transition $C_t \rightarrow Y$ if adversary delivers all messages of M except those sent by i (and doesn't crash anybody)
  - note: because C_t is i-restricted, so is Y [with the same value of t']
- easy case: Y is also i-pivotal [done, just take C_{t+1} = Y]
- harder case: Y not i-pivotal. picture then is:

because C_t is i-pivotal → $val(C_t) \neq val(C_t \setminus i)$

in harder case, Y is not i-pivotal → $val(Y) = val(Y \setminus i)$

Inductive Step: Extending the Sequence

- let C_t be an i-pivotal configuration (need to exhibit pivotal C_{t+1})
- consider transition $C_t \rightarrow Y$ if adversary delivers all messages of M except those sent by i (and doesn't crash anybody)
  - note: because C_t is i-restricted, so is Y [with the same value of t']
- easy case: Y is also i-pivotal [done, just take $C_{t+1} = Y$]
- harder case: Y not i-pivotal. picture then is:

because C_t is i-pivotal

$$\mathrm{val}(C_t) \neq \mathrm{val}(C_t \setminus i)$$

$$\shortparallel$$

$$\mathrm{val}(Y) = \mathrm{val}(Y \setminus i)$$

in harder case, Y is not i-pivotal

you check:
$\mathrm{val}(C_t \setminus i)$, $\mathrm{val}(Y \setminus i)$ defined by the exact same execution
[with adversary crashing i at t']

Inductive Step: Extending the Sequence

- let C_t be an i-pivotal configuration (need to exhibit pivotal C_{t+1})
- consider transition $C_t \rightarrow Y$ if adversary delivers all messages of M except those sent by i (and doesn't crash anybody)
  - note: because C_t is i-restricted, so is Y [with the same value of t']
- easy case: Y is also i-pivotal [done, just take C_{t+1} = Y]
- harder case: Y not i-pivotal. picture then is:

because C_t is i-pivotal

$$\mathrm{val}(C_t) \neq \mathrm{val}(C_t \setminus i)$$

$$\Vert$$

$$\mathrm{val}(Y) = \mathrm{val}(Y \setminus i)$$

in harder case, Y is not i-pivotal

you check:
$\mathrm{val}(C_t \setminus i)$, $\mathrm{val}(Y \setminus i)$ defined by the exact same execution [with adversary crashing i at t']

- upshot: $\mathrm{val}(C_t) \neq \mathrm{val}(Y)$

Inductive Step: Extending the Sequence

- let C_t be an i-pivotal configuration (need to exhibit pivotal C_{t+1})
- consider transition $C_t \rightarrow Y$ if adversary delivers all messages of M except those sent by i (and doesn't crash anybody)
  - note: because C_t is i-restricted, so is Y [with the same value of t']
- easy case: Y is also i-pivotal [done, just take C_{t+1} = Y]
- harder case: Y not i-pivotal. picture then is:

because C_t is i-pivotal

$$\mathrm{val}(C_t) \neq \mathrm{val}(C_t \setminus i)$$

$$\|$$

$$\mathrm{val}(Y) = \mathrm{val}(Y \setminus i)$$

in harder case, Y is not i-pivotal

you check:
$\mathrm{val}(C_t \setminus i)$, $\mathrm{val}(Y \setminus i)$ defined by the exact same execution [with adversary crashing i at t']

- upshot: $\mathrm{val}(C_t) \neq \mathrm{val}(Y)$, say $\mathrm{val}(C_t) = 0$ and $\mathrm{val}(Y) = 1$

Extending the Sequence (con'd)

- the story so far: val(C_t) = 0 and val(Y) = 1
  - in transition $C_t \rightarrow Y$, adversary delays i's messages, is otherwise benign

Extending the Sequence (con’d)

- the story so far: val(C_t) = 0 and val(Y) = 1
  - in transition $C_t \rightarrow Y$, adversary delays i’s messages, is otherwise benign
- in C_t, M must contain p≥1 messages sent by i

Extending the Sequence (con'd)

- the story so far: val(C_t) = 0 and val(Y) = 1
  - in transition $C_t \rightarrow Y$, adversary delays i's messages, is otherwise benign
- in C_t, M must contain $p \geq 1$ messages sent by i
- define transition $C_t \rightarrow Y_r$ by adversary delivering all messages of M except the last r of i's messages (and doesn't crash anybody)

Extending the Sequence (con'd)

- the story so far: val(C_t) = 0 and val(Y) = 1
  - in transition $C_t \rightarrow Y$, adversary delays i's messages, is otherwise benign
- in C_t, M must contain $p \geq 1$ messages sent by i
- define transition $C_t \rightarrow Y_r$ by adversary delivering all messages of M except the last r of i's messages (and doesn't crash anybody)
  - val(Y_0) = val(C_t) = 0; since Y_p = Y, val(Y_p) = val(Y) = 1

Extending the Sequence (con'd)

- the story so far: val(C_t) = 0 and val(Y) = 1
  - in transition $C_t \rightarrow Y$, adversary delays i's messages, is otherwise benign
- in C_t, M must contain $p \geq 1$ messages sent by i
- define transition $C_t \rightarrow Y_r$ by adversary delivering all messages of M except the last r of i's messages (and doesn't crash anybody)
  - val(Y_0) = val(C_t) = 0; since Y_p = Y, val(Y_p) = val(Y) = 1
  - so there exists $r \geq 1$ such that val(Y_{r-1}) = 0 and val(Y_r) = 1

Extending the Sequence (con'd)

- the story so far: val(C_t) = 0 and val(Y) = 1
  - in transition $C_t \rightarrow Y$, adversary delays i's messages, is otherwise benign
- in C_t, M must contain $p \geq 1$ messages sent by i
- define transition $C_t \rightarrow Y_r$ by adversary delivering all messages of M except the last r of i's messages (and doesn't crash anybody)
  - val(Y_0) = val(C_t) = 0; since Y_p = Y, val(Y_p) = val(Y) = 1
  - so there exists $r \geq 1$ such that val(Y_{r-1}) = 0 and val(Y_r) = 1
  - let j = recipient of r^{th} message ($j \neq i$)

Extending the Sequence (con'd)

- the story so far: val(C_t) = 0 and val(Y) = 1
  - in transition $C_t \rightarrow Y$, adversary delays i's messages, is otherwise benign
- in C_t, M must contain $p \geq 1$ messages sent by i
- define transition $C_t \rightarrow Y_r$ by adversary delivering all messages of M except the last r of i's messages (and doesn't crash anybody)
  - val(Y_0) = val(C_t) = 0; since $Y_p = Y$, val(Y_p) = val(Y) = 1
  - so there exists $r \geq 1$ such that val(Y_{r-1}) = 0 and val(Y_r) = 1
  - let j = recipient of r^{th} message ($j \neq i$)
- you check: Y_{r-1}, Y_r both j-restricted [with t'=t]

Extending the Sequence (con’d)

- define transition $C_t \rightarrow Y_r$ by adversary delivering all messages of M except the last r of i’s messages (and doesn’t crash anybody)
- there exists $r \geq 1$ such that $val(Y_{r-1}) = 0$ and $val(Y_r) = 1$
  - let j = recipient of r^{th} message ($j \neq i$)
  - you check: Y_{r-1}, Y_r both j-restricted [with t’=t]

Extending the Sequence (con'd)

- define transition $C_t \rightarrow Y_r$ by adversary delivering all messages of M except the last r of i's messages (and doesn't crash anybody)
- there exists $r \geq 1$ such that $val(Y_{r-1}) = 0$ and $val(Y_r) = 1$
  - let j = recipient of r^{th} message ($j \neq i$)
  - you check: Y_{r-1}, Y_r both j-restricted [with t'=t]
- on the other hand: $val(Y_{r-1} \setminus j) = val(Y_r \setminus j)$
  - if j crashes at start of timestep t, doesn't matter whether it was going to receive the r^{th} message at the timestep [no one will ever know]

Extending the Sequence (con'd)

- define transition $C_t \rightarrow Y_r$ by adversary delivering all messages of M except the last r of i's messages (and doesn't crash anybody)
- there exists $r \geq 1$ such that $val(Y_{r-1}) = 0$ and $val(Y_r) = 1$
  - let j = recipient of r^{th} message ($j \neq i$)
  - you check: Y_{r-1}, Y_r both j-restricted [with t'=t]
- on the other hand: $val(Y_{r-1} \setminus j) = val(Y_r \setminus j)$
  - if j crashes at start of timestep t, doesn't matter whether it was going to receive the r^{th} message at the timestep [no one will ever know]
- so: either (i) $val(Y_{r-1} \setminus j) = 1$ (in which case Y_{r-1} is j-pivotal) or (ii) $val(Y_r \setminus j) = 0$ (in which case Y_r is j-pivotal)
  - either way, we have our next pivotal configuration C_{t+1}

Completing the Proof

Upshot: there is an infinite sequence $C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow$ …. of pivotal configurations (➔ no validator outputs in any C_t).

»

Completing the Proof

Upshot: there is an infinite sequence $C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow$ …. of pivotal configurations (➔ no validator outputs in any C_t).

Question: are we done? [contradicts liveness?]

Completing the Proof

Upshot: there is an infinite sequence $C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow$ …. of pivotal configurations ($\Rightarrow$ no validator outputs in any C_t).

»

Question: are we done? [contradicts liveness?]

Issue: in adversary strategy above, are its constraints respected?

Completing the Proof

Upshot: there is an infinite sequence $C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow$ …. of pivotal configurations (➔ no validator outputs in any C_t).

»

Question: are we done? [contradicts liveness?]

Issue: in adversary strategy above, are its constraints respected?

- good news: never uses a crash fault (only the *threat* of a fault)

Completing the Proof

Upshot: there is an infinite sequence $C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow$ …. of pivotal configurations (➔ no validator outputs in any C_t).

»

Question: are we done? [contradicts liveness?]

Issue: in adversary strategy above, are its constraints respected?

- good news: never uses a crash fault (only the *threat* of a fault)
- bad news: not guaranteed to eventually deliver every message

Completing the Proof

Upshot: there is an infinite sequence $C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow$ …. of pivotal configurations ($\Rightarrow$ no validator outputs in any C_t).

Question: are we done? [contradicts liveness?]

Issue: in adversary strategy above, are its constraints respected?

- good news: never uses a crash fault (only the *threat* of a fault)
- bad news: not guaranteed to eventually deliver every message
  - problem: if for some t, $C_t \rightarrow C_{t+1} \rightarrow C_{t+2} \rightarrow$… are all i-pivotal configurations generated using the "easy case," i's messages never get delivered

Completing the Proof

Upshot: there is an infinite sequence $C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow$ …. of pivotal configurations (➔ no validator outputs in any C_t).

»

Issue: in adversary strategy above, are its constraints respected?

- good news: never uses a crash fault (only the *threat* of a fault)
- bad news: not guaranteed to eventually deliver every message
  - problem: if for some t, $C_t \rightarrow C_{t+1} \rightarrow C_{t+2} \rightarrow$… are all i-pivotal configurations generated using the “easy case,” i’s messages never get delivered
  - fix:

Completing the Proof

Upshot: there is an infinite sequence $C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow$ …. of pivotal configurations (➔ no validator outputs in any C_t).

»

Issue: in adversary strategy above, are its constraints respected?

- good news: never uses a crash fault (only the *threat* of a fault)
- bad news: not guaranteed to eventually deliver every message
  - problem: if for some t, $C_t \rightarrow C_{t+1} \rightarrow C_{t+2} \rightarrow$… are all i-pivotal configurations generated using the "easy case," i's messages never get delivered
  - fix: modify adversary strategy to crash i at timestep t, act benign thereafter
    - other validators can't tell the difference, protocol behavior unchanged
    - now a valid adversary strategy ➔ contradicts liveness of ∏!

Getting Around the FLP Theorem

Perspective: impossibility results like the FLP Theorem give guidance on how to compromise to make progress.

Getting Around the FLP Theorem

Perspective: impossibility results like the FLP Theorem give guidance on how to compromise to make progress.

Possible compromises:

1. Pull back from asynchrony to "partial synchrony" (next lecture).
   - "sweet spot" hybrid of the synchronous, asynchronous models

Getting Around the FLP Theorem

Perspective: impossibility results like the FLP Theorem give guidance on how to compromise to make progress.

Possible compromises:

1. Pull back from asynchrony to "partial synchrony" (next lecture).
   - "sweet spot" hybrid of the synchronous, asynchronous models
2. Solve a problem easier than SMR (e.g., with relaxed consistency requirements).
   - agreement on total ordering of txs is overkill in some applications

Getting Around the FLP Theorem

Perspective: impossibility results like the FLP Theorem give guidance on how to compromise to make progress.

Possible compromises:

1. Pull back from asynchrony to "partial synchrony" (next lecture).
   – "sweet spot" hybrid of the synchronous, asynchronous models
2. Solve a problem easier than SMR (e.g., with relaxed consistency requirements).
   – agreement on total ordering of txs is overkill in some applications
3. Use randomized protocols, solve SMR with high probability.
   – rich academic literature on this topic