Fault-Tolerant Distributed Algorithms

(Part 1b: Fault-Tolerant Consensus)

Ulrich Schmid

s@ecs.tuwien.ac.at

Technische Universität Wien
Embedded Computing Systems Group (E182/2)
Processor Failures

“Deterministic” Failure Models:

- An unknown set $F$ of processors may be(come) faulty
- We do not know when a processor becomes faulty

We just know

- how many processors $0 \leq f \leq n$ may at most be faulty during the entire execution ($|F| \leq f$)
- which kind of failures are allowed:
  - Crash failures: A processor simply stops executing events (also in the middle of a broadcast)
  - Byzantine failures: A processor can do whatever it wants (including sending arbitrary messages)
- Communication is still reliable [could be dropped]
The Consensus Problem

Every processor $p_i$ has

- an input value $x_i$ from some finite set (often binary)
- an output value $y_i$, initially undefined
- a consensus algorithm that computes a value for $y_i$

Required properties in every admissible execution:

- **Termination:** $y_i$ is irrevocably assigned a value at every non-faulty processor $p_i$ eventually
- **Agreement:** $y_i = y_j$ for all terminated non-faulty processors $p_i$ and $p_j$
- **Validity:** If $x_k = v$ for all processors $p_k$, then $y_i = v$ for every terminated non-faulty processor $p_i$
Overview of Consensus Results

Synchronous message passing case:

<table>
<thead>
<tr>
<th>$f$-resilient Algorithm</th>
<th>Crash</th>
<th>Byzantine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numer of rounds:</td>
<td>$f + 1$</td>
<td>$f + 1$</td>
</tr>
<tr>
<td>Number or procs:</td>
<td>$n \geq f + 1$</td>
<td>$n \geq 3f + 1$</td>
</tr>
<tr>
<td>Message size:</td>
<td>$\text{poly}$</td>
<td>$\text{poly}$</td>
</tr>
</tbody>
</table>

Asynchronous case:

- **Impossible** in both message passing and SHM systems even for $f = 1$ crash failures
- Reason: Correct processors never know whether a still missing message from some process will ever arrive
Synchronous Consensus
A Simple Consensus Algorithm 15

Every processor $p_i$ maintains a set $V$ of values seen so far, initially $V = \{x_i\}$

- add received new values to $V$ and forward them
- proceed for $f + 1$ rounds

Pseudo-code Algorithm 15 for $p_i$, $0 \leq i \leq n - 1$:

1. Initially $V_i = \{x_i\}$

2. for $k = 1$ to $f + 1$ do // for $f + 1$ rounds

3. send $\{v \in V_i : p_i$ has not already sent $v\}$ to all

4. receive $V_j$ from $p_j$ for all $j$ (including $i$)

5. $V_i := \bigcup_{j=0}^{n-1} V_j$

6. $y_i := \min(V_i)$ // decide at end of round $f + 1$
Theorem 7. Algorithm 15 is a $f$-crash resilient synchronous consensus algorithm.
Correctness Algorithm 15

**Theorem 7.** *Algorithm 15 is a $f$-crash resilient synchronous consensus algorithm.*

**Proof.** Termination is trivial, we thus have to show:

- **Validity:** Obvious, since every $\min(V_i)$ must be some $p_j$’s $x_j$.
- **Agreement:** It suffices to show that if $x \in V_i$ at the end of round $f + 1 \Rightarrow x \in V_j$, for any non-faulty $p_i$ and $p_j$.

Let $r$ be first round where $x$ is added to any non-faulty $p_i$’s set $V_i$.

- If $r \leq f$, then $p_i$ sends $x$ in round $r + 1 \leq f + 1$ to $p_j$, which causes $p_j$ to add $x$ to $V_j$ and we are done.
- If $r = f + 1$, there must be a chain of $f + 1$ different processors $p_{i_1}, \ldots, p_{i_{f+1}}$ along which $p_{i_1}$’s initial value $x$ was forwarded to $p_i$. Still, we have at most $f$ faulty processors, so at least one must be correct, which contradicts minimality of $r = f + 1$. 
Synchronous Byzantine Consensus (I)

We now increase the adverse capabilities of faulty processors:
- They need not adhere to the algorithm at all
- They can send any message, even inconsistently, to any receiver
- They can collude in an attempt to maximize their adverse power.

Also need to adapt validity condition:
- We cannot assume anything about the initial value $x_k$ of a Byzantine processor $p_k$
- **Byzantine validity**: If $x_k = v$ for all correct processors $p_k$, then $y_i = v$ for every terminated correct processor $p_i$
Synchronous Byzantine Consensus (II)

Some interesting results:

- \[ n \geq 3f + 1 \text{ lower bound for required number of processors} \]
- \[ f + 1 \text{ lower bound for number of rounds (already for crash failures!)} \]
- Exponential Information Gathering (EIG) algorithm
- Phase King algorithm

Naive Approach

Recall Algorithm 15:
- Just forward all values received in round $k - 1$ in round $k$
- Decide on minimum value at the end of round $f + 1$

Problem: Byzantine faulty processor can
- inconsistently send different values, in any round
- “drive” any number of correct processors towards some value

⇒ easily violate agreement

Just replacing minimum by majority does not help ⇒ need additional ideas
EIG Algorithm (I)

Requirements and properties:
- $n \geq 3f + 1$
- $f + 1$ rounds

Principle of operation: Trace sources of information
- Every processor $p_i$ sends its $x_i$ to all in the first round
- Forwarding stage: $f$ additional rounds where every $p_j$ forwards the information obtained in the previous round ("$p_j$ says that $p_k$ says that ... that $p_i$ sent value $x_i$")
- Decision stage: At the end of round $f + 1$, compute decision based on the values received in forwarding stage
EIG Algorithm (II)

Every node maintains a labeled tree data structure with $f + 2$ levels (height $f + 1$):

- The level-0 root has the empty label $\varepsilon$.
- A level-$k$ node, $1 \leq k \leq f + 1$, is labeled with a unique variation (without replacement) $\pi = i_1 i_2 \cdots i_k$ of processor indices $\in \{0, \ldots, n - 1\}$.
- The leafs are at level $f + 1$.
- Every node at level $k < f + 1$ has degree $n - k$.
- $\text{tree}_i(\pi)$ denotes the value stored in $p_i$’s tree node with label $\pi$.
- A node with label $\pi = \pi' i_k$ (and the edge leading to it) corresponds to processor $p_{ik}$ as it gets its data from $p_{ik}$.
EIG Algorithm (III)

Forwarding stage:

- Every $p_i$ stores $x_i$ into the root of its tree
- In round $k$, $1 \leq k \leq f + 1$, processor $p_i$
  - sends level $k - 1$ of its tree to all
  - stores in its node with label $\pi'_i k$ the value $v$ received from $p_{i_k}$ from its level-$k - 1$ node with label $\pi'$ (or $v_\bot$ in case of no or an erroneous message)
  - means “$p_{i_k}$ says that $p_{i_{k-1}}$ says that ... that $p_{i_2}$ says that $p_{i_1}$ sent $v$”

Decision stage:

- At the end of round $f + 1$, processor $p_i$ decides
  $y_i = \text{resolve}_i(\varepsilon)$
The recursive majority vote $\text{resolve}_i$ is defined as

- $\text{resolve}_i(\pi) = \text{tree}_i(\pi)$ if $\pi$ is a leaf
- $\text{resolve}_i(\pi)$ is the majority of $\text{resolve}_i(\pi'')$ for all children $\pi'' = \pi k$ of $\pi$ (or $v_\perp$ if no majority exists)

$\Rightarrow$ Corresponds to building up a $\text{resolve tree}$ that has the same leafs as the forwarding tree
EIG Algorithm (IV)

The recursive majority vote \( \text{resolve}_i \) is defined as

- \( \text{resolve}_i(\pi) = \text{tree}_i(\pi) \) if \( \pi \) is a leaf
- \( \text{resolve}_i(\pi) \) is the majority of \( \text{resolve}_i(\pi'') \) for all children \( \pi'' = \pi_k \) of \( \pi \) (or \( v_\bot \) if no majority exists)

\( \Rightarrow \) Corresponds to building up a resolve tree that has the same leaves as the forwarding tree

A few additional definitions for our analysis:

- A node \( \pi \) is common if \( \text{resolve}_i(\pi) = \text{resolve}_j(\pi) \) for all non-faulty \( p_i \) and \( p_j \)
- A subtree has a common frontier if there is a common node on every path from the root to its leaves
Lemma 15. If the subtree rooted at node $\pi$ has a common frontier, then $\pi$ is common.
Lemma 15. If the subtree rooted at node $\pi$ has a common frontier, then $\pi$ is common.

Proof. By induction on the level of $\pi$. If $\pi$ is a leaf, the statement follows directly from the definition of a common frontier.

Induction step: Assume $\pi$ is a node at level $\ell$, and that the lemma holds for nodes at level $\ell + 1$. If $\pi$ was not common,

- every subtree rooted at a child $\pi_k$ of $\pi$ must have a common frontier
- since every child $\pi_k$ has level $\ell + 1$, the induction hypothesis reveals that they must all be common
- All non-faulty processors resolve the same value for all children and hence for $\pi$, i.e., $\pi$ must be common.
Analysis EIG Algorithm (II)

Lemma 16. For all tree node labels $\pi$ and correct processors $p_i, p_j, p_k$, we have $\text{tree}_i(\pi_j) = \text{tree}_j(\pi)$, and hence $\text{tree}_i(\pi_j) = \text{tree}_k(\pi_j)$. 
Lemma 16. For all tree node labels $\pi$ and correct processors $p_i, p_j, p_k$, we have $\text{tree}_i(\pi j) = \text{tree}_j(\pi)$, and hence $\text{tree}_i(\pi j) = \text{tree}_k(\pi j)$.

Proof. Since $p_j$ is correct, it faithfully sends its value $\text{tree}_j(\pi)$ to $p_i$. Since the latter is also correct, it stores this value in $\text{tree}_i(\pi j)$. \qed
Analysis EIG Algorithm (II)

Lemma 16. For all tree node labels $\pi$ and correct processors $p_i, p_j, p_k$, we have $\text{tree}_i(\pi_j) = \text{tree}_j(\pi)$, and hence $\text{tree}_i(\pi_j) = \text{tree}_k(\pi_j)$.

Proof. Since $p_j$ is correct, it faithfully sends its value $\text{tree}_j(\pi)$ to $p_i$. Since the latter is also correct, it stores this value in $\text{tree}_i(\pi_j)$. □

Lemma 16. For every tree node label $\pi = \pi'j$ and correct processors $p_j, p_i$, it holds that $\text{resolve}_i(\pi) = \text{tree}_i(\pi)$ at every non-faulty $p_i$. 
Lemma 16. For all tree node labels $\pi$ and correct processors $p_i, p_j, p_k$, we have $\text{tree}_i(\pi_j) = \text{tree}_j(\pi)$, and hence $\text{tree}_i(\pi_j) = \text{tree}_k(\pi_j)$.

Proof. Since $p_j$ is correct, it faithfully sends its value $\text{tree}_j(\pi)$ to $p_i$. Since the latter is also correct, it stores this value in $\text{tree}_i(\pi_j)$.

Lemma 16. For every tree node label $\pi = \pi'j$ and correct processors $p_j, p_i$, it holds that $\text{resolve}_i(\pi) = \text{tree}_i(\pi)$ at every non-faulty $p_i$.

Proof. By induction on the level of $\pi$, starting from the leaves:

- Induction basis: If $\pi$ is a leaf, the lemma holds by definition of recursive majority.

- Induction step: If $\pi = \pi'j$ ending in correct $p_j$ is a non-leaf, it has at most level $f$ and hence at least degree $n - f$. 

 occasions
Proof. (cont.)

Induction step: If $\pi = \pi'j$ ending in correct $p_j$ is a non-leaf, it has at most level $f$ and hence at least degree $n - f$.

- Since $n \geq 3f + 1$, $\pi$ has a majority of children $\pi_k$ corresponding to a correct $p_k$.

- Applying the induction hypothesis reveals, at any correct $p_i$,
  \[ resolve_i(\pi_k) = tree_i(\pi_k) \]

- Since, by the previous lemma,
  \[ tree_i(\pi_k) = tree_k(\pi) = tree_i(\pi) \]
  this implies $resolve_i(\pi_k) = tree_i(\pi)$

- Hence, all of $\pi$’s non-faulty children and thus $\pi$ resolve to $tree_i(\pi)$ as asserted.
**Theorem 18.** Every $\pi = \pi' j$ ending in a correct processor $p_j$ is common.

**Proof.** For correct processors $p_i, p_k$, our previous results establish:

- By the last but one lemma, $\text{tree}_i(\pi) = \text{tree}_k(\pi)$
- By the previous lemma, $\text{resolve}_i(\pi) = \text{tree}_i(\pi) = \text{tree}_k(\pi) = \text{resolve}_k(\pi)$. 

\[\square\]
Theorem 19. For $n \geq 3f + 1$, EIG solves consensus in presence of up to $f$ Byzantine failures.
Theorem 19. For $n \geq 3f + 1$, EIG solves consensus in presence of up to $f$ Byzantine failures.

Proof. Validity: If all non-faulty processors start with the same input $v$,
- a majority of children $j$ of the root at any non-faulty $p_i$ satisfy $\text{resolve}_i(j) = \text{tree}_i(j) = \text{tree}_j(\varepsilon) = v$ by our lemmas
- Hence, $\text{resolve}_i(\varepsilon) = v$ as well.

Agreement: Each path from a child of the root to a leaf involves $f + 1$ nodes that correspond to different processors. Hence,
- at least one processor on every path from the root to the leaves is correct $\Rightarrow$ the corresponding node is common by Theorem 18
- the root has a common frontier

Hence, the root must be common, which completes our proof. \qed
Less Costly Alternative to EIG?

Recall: The EIG algorithm has
- optimal time complexity \( (f + 1) \) rounds
- optimal resilience \( n \geq 3f + 1 \)
- exponential message complexity

Alternative idea: Don’t trace sources of information but
- just disseminate values as in the crash-tolerant Algorithm 15
- decide on majority value if “overwhelming majority” exists
- rely on a single correct processor’s value otherwise
Phase King Algorithm (I)

Operates in $f + 1$ phases of 2 rounds each

- First round: Disseminate current preference values system-wide
- Second round: Use the rotating coordinator principle to select single correct processor (the "king") if no "overwhelming majority" exists

The Phase King algorithm

- solves consensus with polynomial message complexity
- with sub-optimal round complexity $(2(f + 1))$
- and sub-optimal resilience $(n \geq 4f + 1)$
Phase King Algorithm (II)

Pseudo-code Algorithm 16 for $p_i$, $0 \leq i \leq n - 1$:

1. $v := x$  // Init preference to own proposed value

2. for $k = 1$ to $f + 1$ do  // for $f + 1$ phases (2 rounds each)
3.  /* round 2k-1 */
4.     send $\langle v \rangle$ to all processors
5.     receive $\langle v_j \rangle$ from all $p_j$
6.     $maj :=$ majority among $v_j$ ($v_\perp$ if none)
7.     $mult :=$ multiplicity of $maj$ among $v_j$
8.  /* round 2k */
9.     if $i = k$ then send $\langle maj \rangle$ to all processors
10.    receive $\langle king-maj \rangle$ from $p_k$ ($v_\perp$ if none)
11. if $mult > n/2 + f$ then $v := maj$ else $v := king-maj$

12. $y := v$  // decide at end of phase $f + 1$
We say

- $p_i$ prefers value $v$ at the beginning of phase $k$ [= the end of phase $k - 1$, with phase 0 representing the initial configuration] if

- $v_{i^{2k-2}} = v$ at the end of round $2k - 2$
Analysis of Phase King Algorithm (I)

We say

- $p_i$ prefers value $v$ at the beginning of phase $k$ [= the end of phase $k - 1$, with phase 0 representing the initial configuration] if

- $v_{2k-2}^i = v$ at the end of round $2k - 2$

**Lemma 23** (Persistence of agreement). *If all correct processors prefer $v$ at the beginning of phase $1 \leq k \leq f + 1$, then they all prefer $v$ at the end of phase $k$.***
We say

- $p_i$ prefers value $v$ at the beginning of phase $k$ [= the end of phase $k - 1$, with phase 0 representing the initial configuration] if

- $v_{2k-2}^i = v$ at the end of round $2k - 2$

**Lemma 23** (Persistence of agreement). *If all correct processors prefer $v$ at the beginning of phase $1 \leq k \leq f + 1$, then they all prefer $v$ at the end of phase $k$.*

**Proof.** By the code,

- every processor receives at least $n - f$ copies of $v$ in the first round of phase $k$
- $n - f > n/2 + f$ since $n > 4f$, so all processors prefer $v$ at the end of phase $k$
Persistence of agreement already implies

- Validity
- Termination
Persistence of agreement already implies

- Validity
- Termination

For agreement: Since there are $f + 1$ phases

- Every phase has a different king

$\Rightarrow$ There is at least one phase $g$ with a correct king

It only remains to be shown that all correct processors prefer same value at the end of phase $g$ . . .
Lemma 25. Let $g$ be a phase with a correct king $p_g$. Then all correct processors finish phase $g$ with the same preference value.
Lemma 25. Let $g$ be a phase with a correct king $p_g$. Then all correct processors finish phase $g$ with the same preference value.

Proof. 2 exhaustive cases:

- All correct processors $p_j$ use $\text{king-maj}_j$ for their preference. Since $p_g$ is correct, $\text{king-maj}_j$ must be the same at all $p_j$.
- Suppose some $p_i$ uses $\text{maj}_i$ for its preference, then
  - $p_i$ must have received $> n/2 + f$ messages containing $\text{maj}_i$ in the first round
  - every other processor $p_j$, including $p_g$, must have received $> n/2$ of those messages as well, and thus set $\text{maj}_j = \text{maj}_i$
  $\Rightarrow$ every processor $p_j$ sets $\text{king-maj}_j = \text{maj}_i$ as well
  $\Rightarrow$ every processor $p_j$ assigns $v_j = \text{maj}_i$. 
Asynchronous Consensus
Impossibility in Asynchronous Systems

Wait-free case (up to $f = n - 1$ crash failures)

- Wait-free $\simeq$ algorithms must not wait for messages since they could block
- Impossibility easier to show since many faulty processes

General case $f = 1$

- Same as wait-free case for $n = 2$
- [Show impossibility for arbitrary $n$ by clever reduction]

Above results proved for (single-writer) SHM systems.

- Impossibility for MP systems by simple reduction
- [Well-known direct proof by Fischer, Lynch & Paterson]
Asynchronous Bivalence Proofs: Definitions

A configuration $C$ in an admissible execution is called

- **0-decided** if some (correct or faulty) $p_i$ has already decided 0
- **1-decided** if some $p_i$ has already decided 1
- **0-valent** if all decided configurations $C'$ reachable from $C$ are 0-decided
- **1-valent** if all decided configurations $C'$ reachable from $C$ are 1-decided

Classify configurations $C$ as

- **univalent** if $C$ is either 1-valent or 0-valent
- **bivalent** if both a 0-decided and a 1-decided configuration can be reached from $C$
Asynchronous Configuration Trees (SHM)

Consider all admissible executions \( \text{exec}(C^0, \sigma) \) of an asynchronous wait-free SHM algorithm

- starting from some fixed initial state \( C^0 \)
- with arbitrary infinite schedule \( \sigma \) (no restriction)

All reachable configurations can be arranged in a configuration tree, with

- vertices representing (unique) configurations [encode number of steps taken by every processor in configuration]
- edges represent steps
- every vertex \( C \) has exactly \( n \) successors \( C_i = i(C) \), \( 0 \leq i \leq n - 1 \), corresponding to \( p_i \) taking the next step
Lemma 30. Let $C_1$ and $C_2$ be two univalent configurations of a wait-free binary consensus algorithm. If $C_1 \overset{p_i}{\sim} C_2$ for some correct $p_i$, then both configurations have the same valence.
Lemma 30. Let $C_1$ and $C_2$ be two univalent configurations of a wait-free binary consensus algorithm. If $C_1 \sim C_2$ for some correct $p_i$, then both configurations have the same valence.

Proof. Consider an infinite $p_i$-only schedule $\sigma$ starting from $C_1$:
- $p_i$ must decide in $\text{exec}(C_1, \sigma)$ because algorithm is wait-free
- Since $C_1$ is $v$-valent for some $v \in \{0, 1\}$, the decision must be $v$

Now apply $\sigma$ to $C_2$:
- Yields a feasible execution since $p_i$ starts from same configuration
- $p_i$ must also decide in $\text{exec}(C_2, \sigma)$ and its decision must also be $v$. $\square$
Lemma 31. Every wait-free binary consensus algorithm has a bivalent initial configuration.
Lemma 31. Every wait-free binary consensus algorithm has a bivalent initial configuration.

Proof. Consider the following initial configurations:

- $I_0$ where all processors $p_i$ start with input value $x_i = 0$ ⇒ must be 0-valent by validity
- $I_1$ where all processors $p_i$ start with input value $x_i = 1$ ⇒ must be 1-valent by validity

Now consider initial configuration $I_{01}$ where $x_0 = 0$ and $x_i = 1$ for $1 \leq i \leq n - 1$. Assume, by way of contradiction, that is is univalent:

- $I_{01} \xrightarrow{p_0} I_0$ ⇒ $I_{01}$ must be 0-valent by preparation lemma
- $I_{01} \xrightarrow{p_1} I_1$ ⇒ $I_{01}$ must be 1-valent by preparation lemma

⇒ Contradiction; so $I_{01}$ must be bivalent.
Lemma 32. Every bivalent configuration of a wait-free binary consensus algorithm has at least one bivalent successor configuration.
**Lemma 32.** Every bivalent configuration of a wait-free binary consensus algorithm has at least one bivalent successor configuration.

**Proof.** Every configuration $C$ has exactly $n$ possible successor configurations $C_k$, depending on which of $p_0, \ldots, p_{n-1}$ takes the next step.

- Assume, by way of contradiction, that all $C_k$ are univalent.
- Since $C$ is bivalent, there must be $i$ and $j$ such that $C_i = i(C)$ and $C_j = j(C)$ are 0-valent and 1-valent, respectively.

Distinguish 2 possible cases . . .
Bivalent Successor Configuration (II)

Proof. (cont.) Distinguish 2 possible cases:

- If the steps $i$ and $j$ commute (read/write different registers or read the same one), $i(j(C)) = j(i(C)) \Rightarrow i(C)$ and $j(C)$ cannot have different valences.

- If $i$ writes some register and $j$ reads it (both $i$ and $j$ writing same register disallowed since single-writer), consider $i(C)$ and $i(j(C))$:
  - $i(j(C))$ is 1-valent since $j(C)$ is 1-valent
  - $i(C)$ is 0-valent
  - $i(C) \overset{p_i}{\sim} i(j(C)) \Rightarrow$ should have same valence by preparation lemma.
Theorem 34. There is no single-writer SHM wait-free binary consensus algorithm for $n$ processors.
**Theorem 34.** *There is no single-writer SHM wait-free binary consensus algorithm for $n$ processors.*

*Proof.* We know from earlier lemmas:

- There is a bivalent initial configuration
- Every bivalent configuration has at least one bivalent successor configuration

Hence there is at least one non-terminating execution. ☐
The above impossibility proof was easy. Why?

- Single-writer SHM
- Wait-free property \((f = n - 1)\) gives adversary much power
- Configuration tree has simple structure
How to make things more complicated?

- Multiple-writer SHM can be simulated atop of single-writer SHM $\Rightarrow$ above impossibility actually holds for any SHM system

- Non-trivial admissibility conditions make configuration tree complex (not “closed”)
  - Could adapt our SHM bivalence proof for $f = 1$ (using schedules incorporating round robin execution)
  - MP systems further complicated by message delivery requirement: Fischer, Lynch and Patterson’s famous proof even more complex
Impossibility 1-resilient Consensus? (III)

Alternative solution: Use (clever) reduction:

- Assume that there is a $n$-processor consensus algorithm $A$ that can cope with $f = 1$ crashes
- Use $A$ to construct a 2-processor consensus algorithm that can cope with a single crash, by letting
  - simulating processors $p_0, p_1$ simulate the exec of
  - simulated processors $q_0, \ldots, q_{n-1}$

- Naive solution: $p_0, p_1$ simulate $n/2$ simulated processors each $\Rightarrow$ does not work since crash of simulating processor would result in $f = n/2$

- Borowsky-Gafni-Simulation: Both of $p_0, p_1$ asynchronously execute code for all $q_0, \ldots, q_{n-1}$ in round-robin order, and agree on every steps’ result
Theorem 38. There is no $n$ processor consensus algorithm for asynchronous message passing systems that can tolerate even a single crash failure.
Theorem 38. There is no $n$ processor consensus algorithm for asynchronous message passing systems that can tolerate even a single crash failure.

Proof. We again use reduction, by simulating an MP system atop of a SHM system:

- For every ordered pair of processors, there is a single-writer single-reader R/W link register (unbounded range)
- Sender appends new message to prior content of all outbound link registers
- Receiver polls all inbound link registers in round-robin fashion to get new messages
- Additional receive delay does not matter since we are dealing with asynchronous system
Proof. (cont.)

If there was a MP consensus algorithm $A$ that tolerates a single crash,

- this simulation in conjunction with $A$ would yield a SHM consensus algorithm that tolerates a single crash

- such a SHM algorithm does not exist $\Rightarrow$ contradiction.