The Impossibility of Asynchronous Consensus

An overview of the proof of Fischer, Lynch, and Paterson

Journal of the ACM, April 1985

The Consensus Problem

- n processes, some of which may be faulty
- each correct process “proposes” a value
- in finite time, we want
  - *termination:* each correct process decides on a value
  - *agreement:* all correct processes decide on the same value
  - *validity:* the agreed-on value was proposed by one of the correct processes

The FLP Result

- Michael Fischer, Nancy Lynch, and Michael Paterson, “Impossibility of Consensus with One Faulty Process,” JACM, Apr. 1985
  - Assuming asynchronous (arbitrary delay) messages, we can’t achieve consensus even if all initial votes are either ‘0’ or ‘1’, all messages are eventually delivered, and faulty processes are fail-stop.
  - More complex proposals, unreliable messages, or Byzantine behavior would only make things worse.

Aside: Byzantine Generals [Lamport, 1982]

- Related problem, but with a notion of "leader," and all correct processes have to decide on the leader's proposal.
- "Standard" consensus and Byzantine Generals are equally difficult (reductions in both directions—not shown here :-)

But we achieve consensus every day!

- Only by compromising on the goals
  - arrange for crashed processes to recover and continue
  - assume a perfect failure detector (declare a process dead if you don't hear from it for too long)
  - randomize (make the possibility of indefinite indecision arbitrarily low)

- Credit: presentation here borrows heavily from Henry Robinson

http://the-paper-trail.org/blog/a-brief-tour-of-flp-impossibility/

System Model

- A *configuration* captures the current state of every process and the set of in-flight messages.
- Initially all processes are in the same state except for their initial proposal and their id, and they run the same code.
- In each time step, some process receives a message (or starts from initial state), optionally sends a message, and updates its internal state.
- At most one process may fail, by stopping.

Proof Structure

- Lemma 1: commutativity of schedules
- Lemma 2: order of message receipt matters
- Lemma 3: *pumping* (bulk of the proof)
- Main result follows

Lemma 1: commutativity of schedules

- If we're in configuration C and two messages are receivable, one by process p and another by process q ≠ p, then receiving the messages in either order leads to the same configuration.
- Proof: straightforward. Every process is deterministic based on local state and content of incoming message. Configuration is just the union of local states & in-flight messages.

Lemma 2: receipt order matters

- I.e., execution isn't determined solely by initial conditions. Proof:
  - Suppose the contrary: everything is predetermined.
  - Consider all possible initial configurations. List these in Grey-code order of set of initial proposals.
  - Each configuration differs from neighbors in the list in only one process.
  - 0...0 must decide 0. 1...1 must decide 1. Somewhere in the list there are neighbors with different decisions.
  - But the one process that differs in these neighbors can fail!

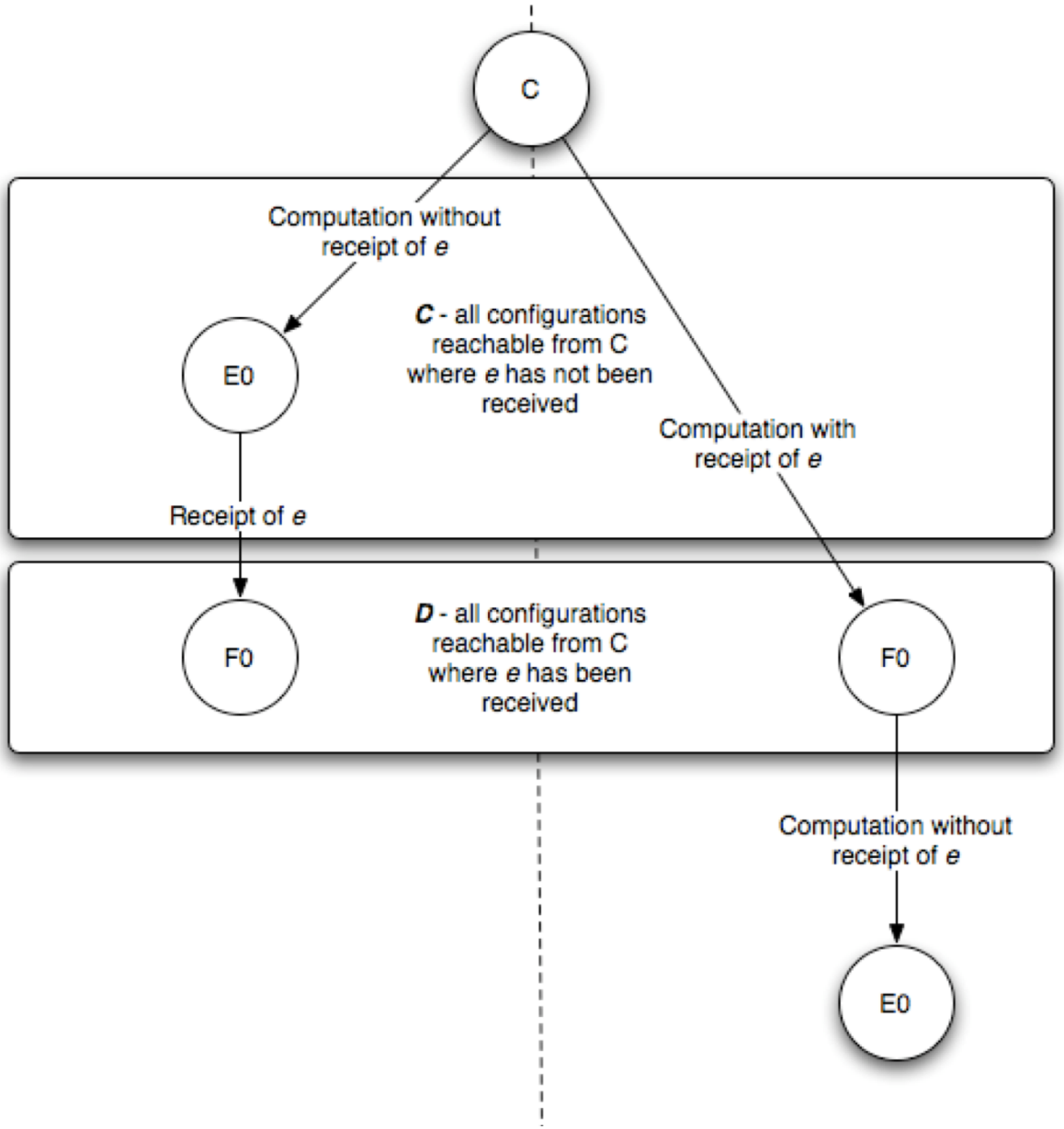
Lemma 3: pumping

- Call a configuration *0-valent* if it must decide 0; *1-valent* if it must decide 1; *bivalent* if it could go either way.
- Suppose we start in a bivalent configuration C, in which e might be sent. Consider all chains of configurations starting in C and ending w/ receipt of e. Let $\mathcal{D}$ be the set of ending configurations of those chains. Claim: $\mathcal{D}$ contains a bivalent config.
- That is: if we can delay e arbitrarily long, we can guarantee the existence of an execution in which e is received in a bivalent config → indefinite delay.

Proof

- Suppose the contrary: no bivalent configurations in $\mathcal{D}$. Let $\mathcal{C}$ be the set of configurations reachable from C w/out receiving e. (Note that every config in $\mathcal{D}$ is reached by receiving e when in some config in $\mathcal{C}$.)
- Claim 3a: there must be both 0-valent and 1-valent configurations in $\mathcal{D}$.
  - Consider 0-valency first. Clearly there is a 0-valent configuration E0 reachable from C, since C is bivalent. Reaching it might or might not entail receiving e.

- E0 is univalent, and must exist, since C is bivalent
- Fix F0 in $\mathcal{D}$: must also be univalent, by assumption
- F0 and E0 must have same valency (0); whichever comes first, the other exists
- Same argument works for 1-valent case.



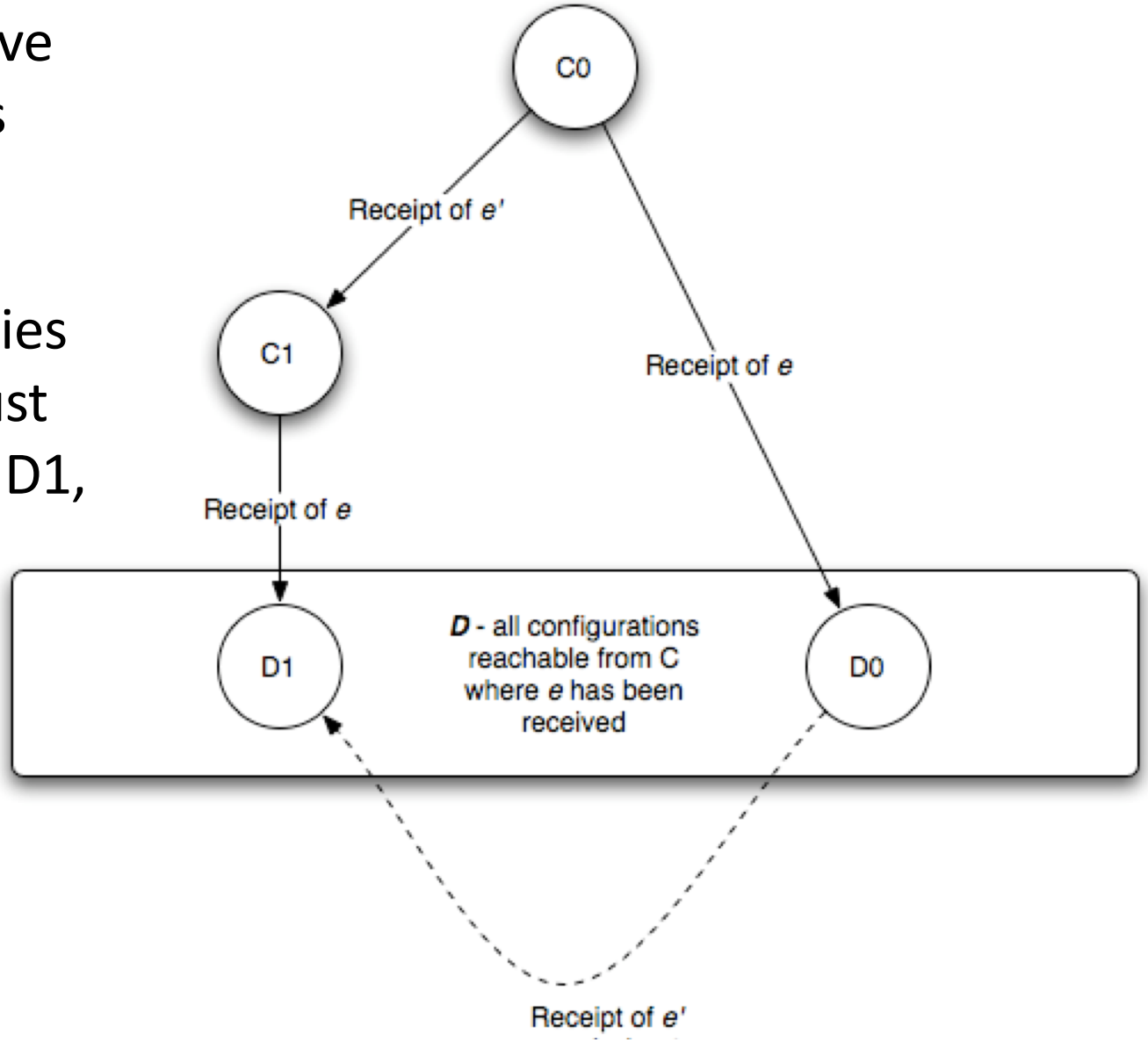
- Claim 3b: there is a pair of configs C0, C1 in $\mathcal{C}$ s.t.
  - receiving e in C0 takes you to a 0-valent config D0 in $\mathcal{D}$
  - receiving e in C1 takes you to a 1-valent config D1 in $\mathcal{D}$
  - C0 and C1 are neighbors: you get to one of them by receiving some message e' when in the other. WLOG, say C0 → e' → C1 (can easily enumerate the other case).
- To see this, assume the contrary. Then (by logic similar to the proof of Lemma 2) on every chain from the initial configuration C, receiving e in any config takes you to uniformly 0-valent or 1-valent configs. But the root is in all chains, at it's supposed to be bivalent!

Taking Stock

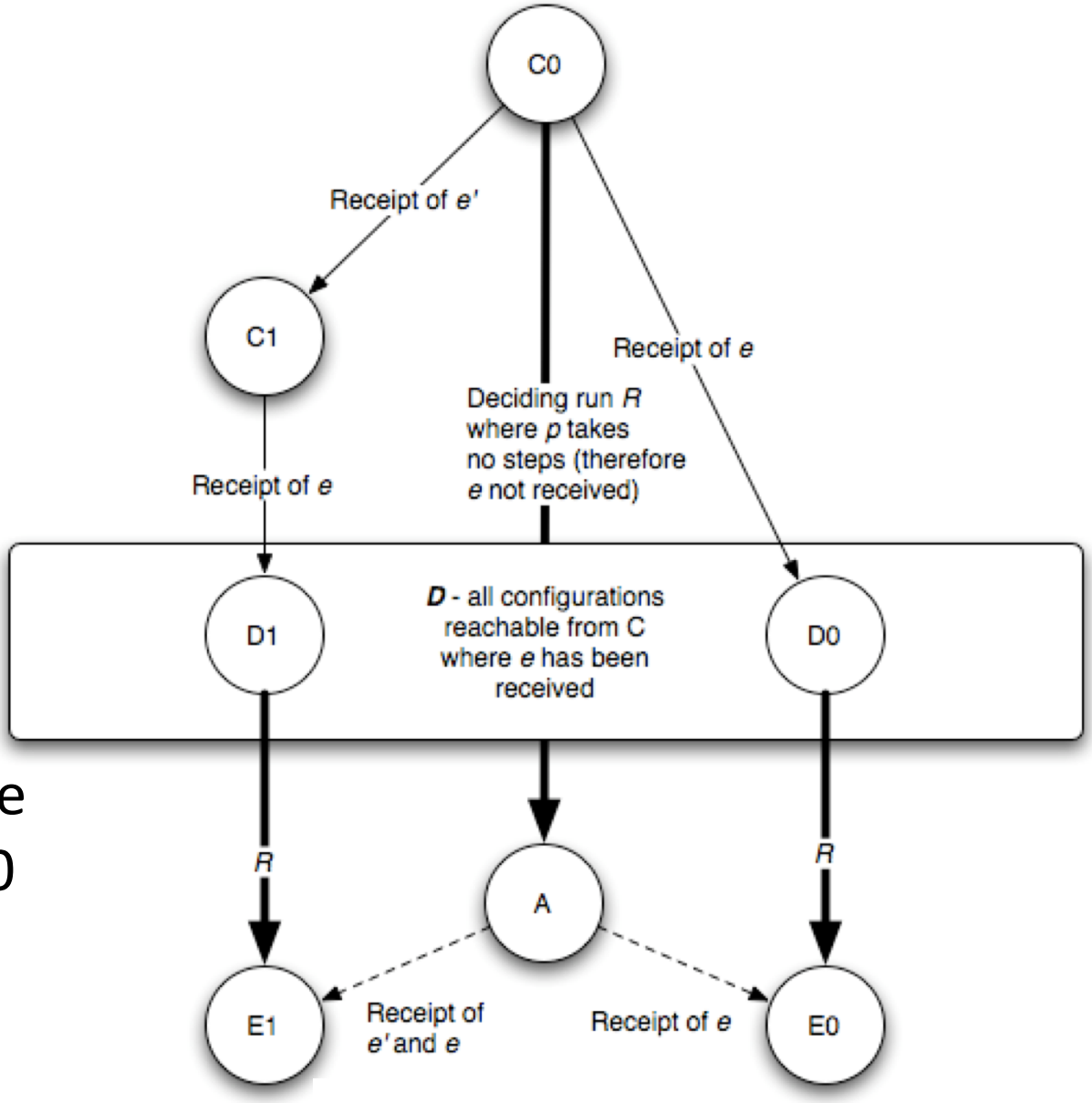
- Assuming that there are no bivalent configs in $\mathcal{D}$, know that
  - There are both 0-valent & 1-valent configs in $\mathcal{D}$
  - There are neighbors C0 and C1 that go to $\mathcal{D}$ configs D0 and D1, of different valency, on e
- Note that in D1 we have received e' but in D0 we haven't.
- Want a contradiction with no further assumptions.
- Two cases: e and e' are received by different or same processes.

- Case 1: e and e’ have different recipients

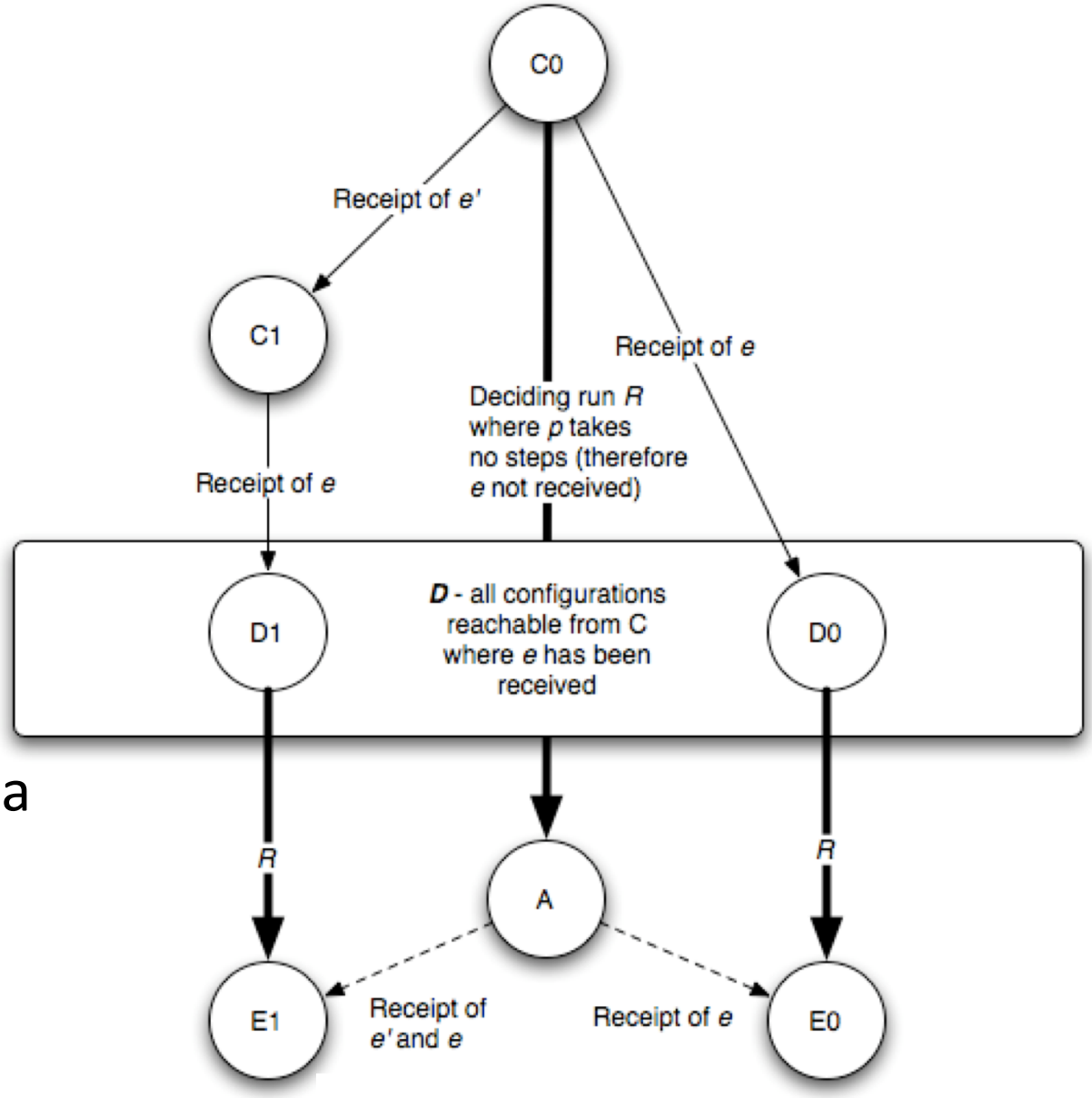
- D0 and D1 have different univalencies
- By Lemma 2, e’ must take us from D0 to D1, a contradiction



- Case 2: e and e’ have same recipient, p
- Consider finite deciding run from C0, in which p takes no steps (has to exist, because p might fail); say this ends in A
- Take same sequence of message receipts and run from D0 & D1 (has to make sense, since D0 and D1 differ from C0 only in the state of p)



- (continued) config E0, reached from D0, must be 0-valent; config E1, reached from D1, must be 1-valent
- But by Lemma 2,
  A → e → E0 and
  A → e' → e → E1
- This means A is bivalent, contradicting assumption that it ends a deciding run



Back to Main Theorem

- Lemma 2 says there's a bivalent starting config.
- Lemma 3 says we can receive a nonzero number of messages from that config and end up in another bivalent config. We can repeat this inductively and get a non-deciding chain of arbitrary length.
- Note that we used the possibility of a (single) failure twice in the proof.