5.5.1 Randomized Agreement with Crash Failures

We will first present a randomized distributed algorithm for crash failures, which requires a strict majority of non-faulty processes. This algorithm works in both synchronous and asynchronous systems. The termination condition now specifies that the *expected* number of rounds, which are imposed by the algorithm, is finite.

Algorithm 5.10 *Ben-Or's randomized algorithm for consensus in synchronous and asynchronous systems with crash failures* [5].

Idea: Every process starts with a binary input value `v`. We assume that the number of traitors f satisfies $n > f/2$. The algorithm proceeds in rounds (indicated by `r`) consisting of three phases: a *notification* phase (messages contain the messagwe type `N`), a *proposal* phase (messages contain the message type `P`), and a *decision* phase. When a process expects messages from all processors, it is no use waiting for more than $n - f$ messages, because the f faulty processors may not send the required message. When not enough processors support a possible decision, a process starts the next round with a new, random value `v`. Even though every process goes through a series of rounds, the algorithm is suitable for asynchronous systems—the rounds do impose a level of synchrony.

Implementation:

```
r ← 1
decided ← false
    do forever
       broadcast (N;r,v)
       await n − f messages of the form (N;r,*)
       if (> n/2 messages (N;r,w) received with w=0 or 1) then
          broadcast (P;r,w)
       else broadcast (P;r,?)
       if decided then STOP
       else await n − f messages of the form (P;r,*)
       if (> 1 messages (P;r,w) received with w=0 or 1) then
          v ← w
          if (> f messages (P;r,w)) then
             decide w
             decided ← true
       else v ← random(0,1)
       r ← r + 1
```

5.5.2 Randomized Byzantine Agreement

In this section we present a randomized algorithm for Byzantine agreement which is valid for both synchronous and asynchronous systems, and which is in fact very similar to Algorithm 5.10.

Algorithm 5.11 *Ben-Or's randomized algorithm for consensus in synchronous and asynchronous systems with Byzantine failures* [5].

Idea: Compared to Algorithm 5.10, this algorithm has the following differences. First, the requirement on the number of traitors f is now $n > 5f$. Secondly, the decision procedure is more complicated.

Implementation:

```
r ← 1
decided ← false
    do forever
        broadcast (N;r,v)
        await n − f messages of the form (N;r,*)
        if (> (n+f)/2 messages (N;r,w) received with w=0 or 1) then
            broadcast (P;r,w)
        else broadcast (P;r,?)
        if decided then STOP
        else await n − f messages of the form (P,r,*)
        if (> f messages (P;r,w) received with w=0 or 1) then
            v ← w
            if (> 3f messages (P;r,w)) then
                decide w
                decided ← true
        else v ← random(0,1)
        r ← r + 1
```

Correctness: We prove the correctness of this algorithm with the following lemmas. The correctness hinges on the three conditions for certain numbers of messages received for taking some action in the algorithm. Note that by Lemma 5.13, the algorithm achieves validity in that if all correct processes start with the same value, they all decide on this value. Also, by Lemma 5.14 agreement is achieved in that all correct processes decide on the same value.

Lemma 5.12 *If a correct process proposes* `v` *in round* `r`*, then no other correct process proposes* `1-v` *in round* `r`.

PROOF. For a process to propose the value `v`, the process must have received more than $(n+f)/2$ messages of the form `(N,r,v)`. Of these, more than $(n - f)/2$ are from correct processes, which is a majority of the correct processes. □

Lemma 5.13 *If at the beginning of round* `r` *all correct processes have the same value* `v`*, then they all decide* `v` *in round* `r`.

PROOF. Each correct process will receive at least $n - f$ messages, at least $n - 2f$ of which are from correct processes, and so of the form `(N,r,v)`. Because $n > 5f$, we have $n - 2f = n/2+n/2 - 2f > (n+f)/2$, and so, each correct process proposes `v`. So, each correct process receives at least $n - 2f$ messages of the form `(P,r,v)`, and so, because $n > 5f$, we have $n - 2f > 3f$, and so each correct process decides `v`. □

Lemma 5.14 *If a correct process decides* v *in round* r*, then all correct processes decide* v *in round* r+1.

PROOF. It is enough to show that all correct processes propose v in round r+1. If a correct process decides v in round r, it must have received more than $3f$ proposals for v, m of which are from correct processes for some $m > 2f$. So every other correct processor receives at least $m \quad f > f$ proposals for v, so it starts the next round with this value. Now use Lemma 5.13. □

Theorem 5.15 *If $n > 5f$, Algorithm 5.11 guarantees Agreement and Validity, and terminates with probability 1.*

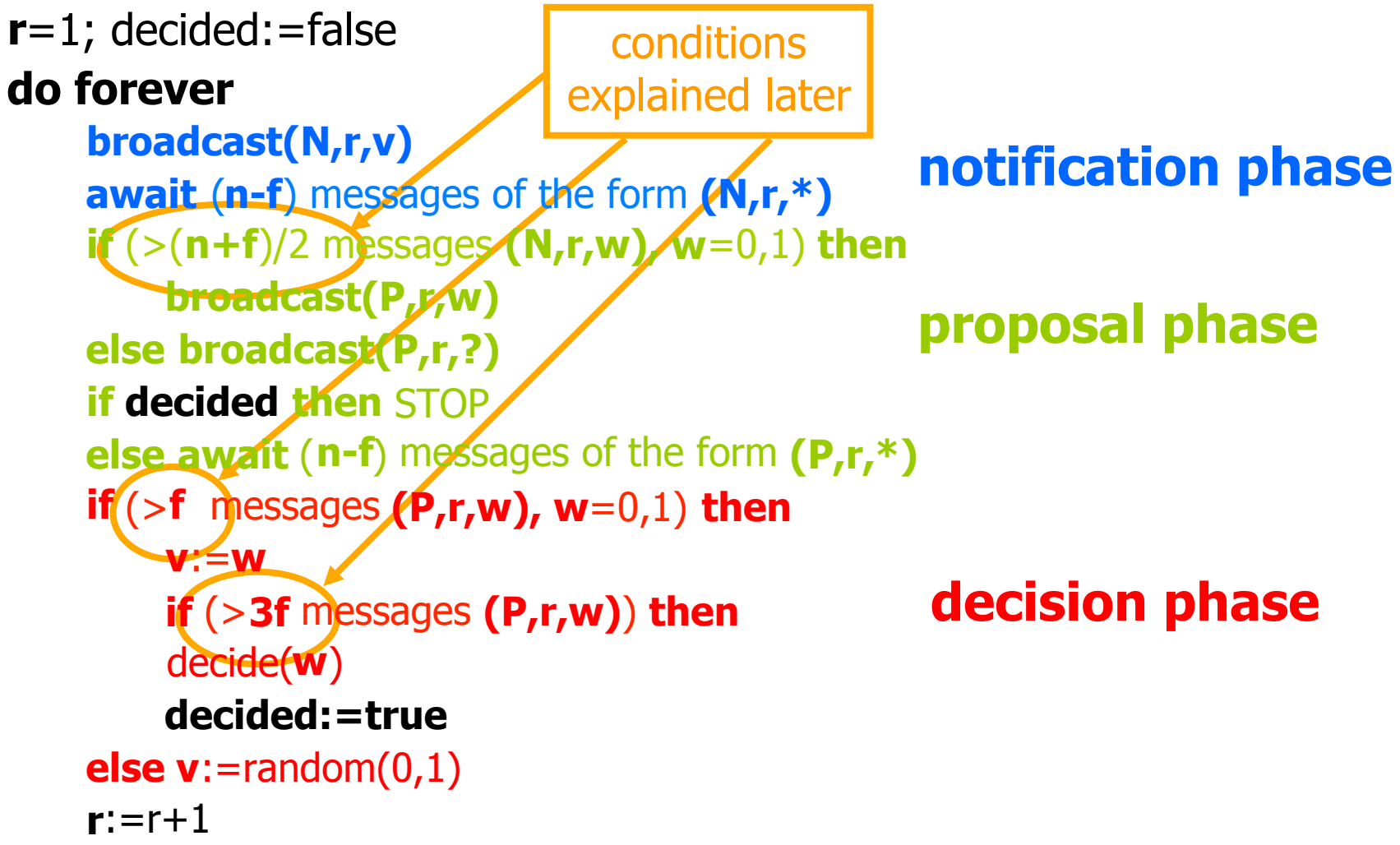
PROOF. With probability 1, enough correct processes will pick a common value v to have at least one correct process decide. □

The expected number of rounds of this algorithm is exponential in the number of processes: When it is needed that the (correct) processes pick a random number, the probability that they all draw the same number (bit) is equal to $2 \cdot 2^{\ n}$. Because it does not matter what value the failing processes pick, and because not all values have to be equal, the time complexity is actually better, but still exponential.

Randomized Byzantine agreement (1/6)

- Solution for **synchronous and asynchronous systems!!**
- **n** processes, of which at most **f** fail, **n>5f**
- Every process has an initial value **v**
- The algorithm **proceeds in rounds consisting of three phases**:
  1. a **notification** phase (messages have message type **N**)
  2. a **proposal** phase (messages have message type **P**)
  3. a **decision** phase
- When a process expects messages **from all other processes**, it is no use waiting for more than **n-f** messages
- When not enough processors **support a possible decision**, a process starts the next round with a new random input value **v**

Randomized Byzantine agreement (2/6)



Randomized Byzantine agreement (3/6)

- "No simultaneous contradicting proposals by correct processes"
- **Lemma 1**:
  - If a **correct process proposes v** in round **r**, then
    no other correct process proposes 1-v in round **r**
- **Proof**:
  - a process that does a proposal has received more than **(n+f)/2** messages **(N,r,v)**
  - of these, more than **(n-f)/2** are from correct processes, which is a **majority of the correct processes**
  - so there can only be one proposed value

Randomized Byzantine agreement (4/6)

- "When all correct processes have the same value, immediate decision"
- **Lemma 2**:
  - If at the start of round **r all correct processes have the same value v**, then **they all decide v** in round **r**
- **Proof**:
  - each correct process receives at least **n-f notification** messages, at least **n-2f** of which are from correct processes, and so of the form **(N,r,v)**
  - because **n>5f**, we have **n-2f = n/2+n/2-2f > n/2+5f/2-2f = (n+f)/2**
  - this is exactly the condition for every correct process to propose **v**
  - so, each correct process receives at least **n-2f** messages of the form **(P,r,v)**
  - because **n>5f**, we have **n-2f>3f**, which is exactly the condition for every correct process to decide **v**

Randomized Byzantine agreement (5/6)

- "Decision of any correct process immediately followed by others"
- **Lemma 3**:
  - If a **correct process decides v** in round **r**,
    all correct processes decide v in round **r+1**
- **Proof**:
  - **enough**: all correct processes propose **v** in round **r+1**
  - if a process decides **v** in round **r**, it must have received more than **3f** proposals for **v**, **m** of which are from correct processes for some **m>2f**
  - so every other correct processor receives at least **m-f>f** proposals for **v**, so it starts the next round with this value
  - now use Lemma 2

Randomized Byzantine agreement (6/6)

- **Theorem**:
  - If **n>5f**, the algorithm guarantees **agreement, validity, and terminates with probability 1**
- **Proof**:
  - with probability 1, enough processors will pick the same value **v** to have at least one correct process decide
- **Expected number of rounds** is of order $\mathbf{2^n}$ (in fact, slightly better)
- **Remark**: randomization is used only if there is not enough initial support for any decision anyway