Dependable and Distributed Systems

Professor Matthew Leeke
School of Computer Science
University of Birmingham

Topic 9 - Reliable Links and Failure Detectors

Asynchronous Fault Tolerant Algorithms

How to study asynchronous algorithms?

Specifications - safety and liveness

Assumptions about safety and liveness

We can see these approaches represented when we try to build perfect communication links between processes

Assumptions On Channels

A channel should ideally transport any message from its source to its destination

Real channels are unreliable

In what way way are they unreliable?

Fair loss is usually a realistic assumption on channels

What Is Fair Loss?

Fair loss is a combination of assumptions that we make about the properties of communication links to provide an accurate modelling mechanism

If you try often enough, eventually the message will get through

Message don’t appear out of nowhere

More formally…

Fair Loss Link Properties

Fair Loss Property 1 (Fair loss) - If message m is sent infinitely often by p_i to p_j and neither p_i or p_j crash then m is delivered infinitely often to p_j

Fair Loss Property 2 (Finite Duplication) - If message m is sent finitely often by p_i to p_j then m is delivered a finite number of times to p_j

Fair Loss Property 3 (No Creation) - No message is delivered unless it was sent

Fair Loss Links

Fair loss links reflect the possibility for explicit retransmission

Well defined interface:

flp2pSend (fair loss peer-to-peer send)

flp2pDeliver (fair loss peer-to-peer deliver)

Which properties are safety and which are liveness?

Can we build reliable links from fair loss links?

Stubborn Links

Stubborn Link Property 1 (Stubborn Delivery) - If a process p_i sends a message m to a correct process p_j and p_i does not crash then p_j takes delivery of m an infinite number of times

Stubborn Link Property 2 (No Creation) - No message is delivered unless it was sent

Look familiar? It's helpful…

From Fair Loss To Stubborn Links

Well defined interface for stubborn channels:

sp2pSend (stubborn peer-to-peer deliver)

sp2pDeliver (stubborn loss peer-to-peer deliver)

We can implement stubborn links using fair loss links

The main idea is to perform retransmission within the algorithm

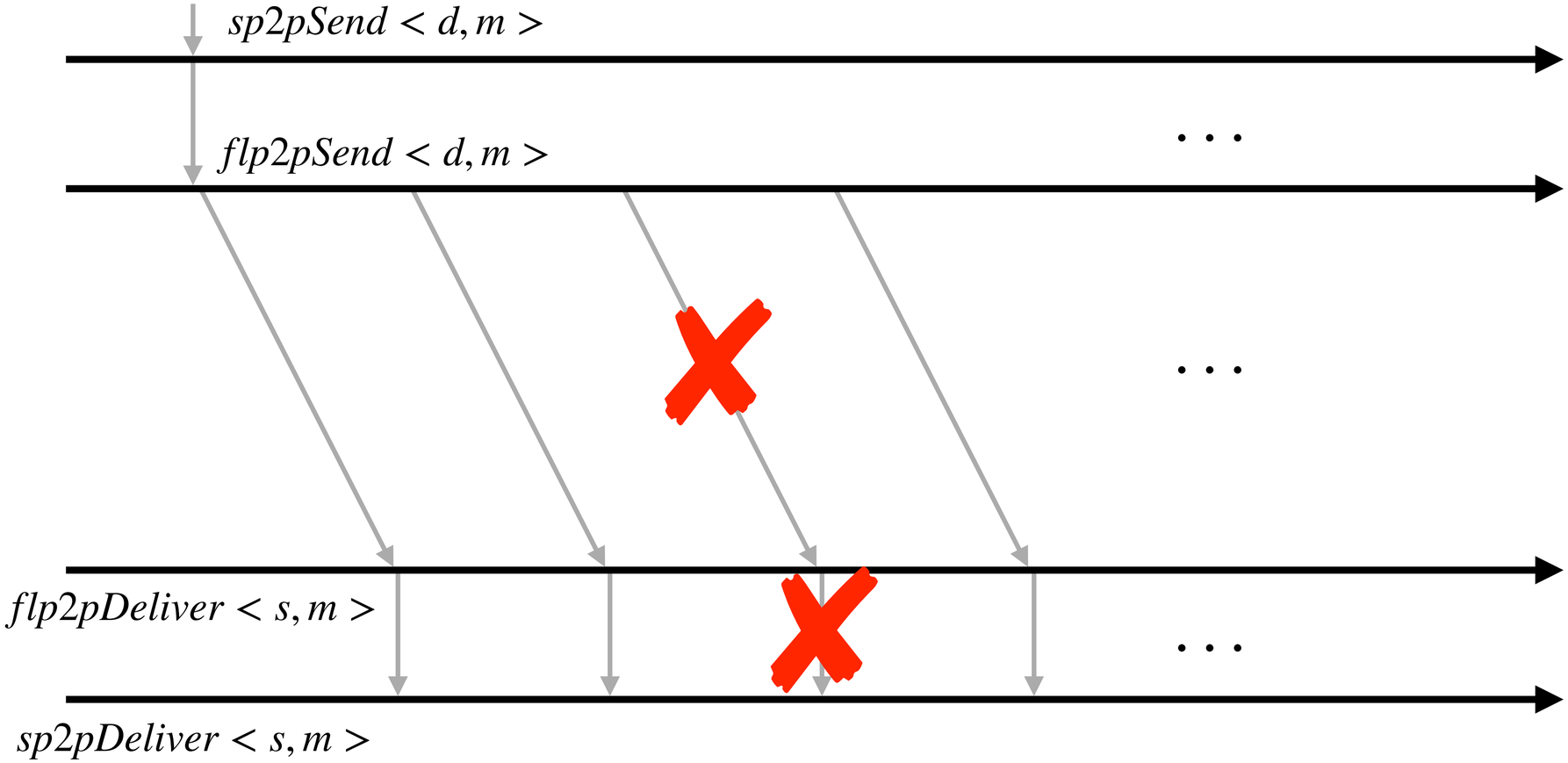
Stubborn Links Algorithm

Implements : *StubbornLinks* (*sp2p*)
Uses : *FairLossLinks* (*flp2p*)

upon event $< sp2pSend, dest, m >$ *do*
    while (*true*) *do*
        trigger $< flp2pSend, dest, m >$

upon event $< flp2pDeliver, src, m >$ *do*
    trigger $< sp2pDeliver, src, m >$

Example Execution



From Stubborn Links to Reliable Links

Stubborn links still deliver a message infinitely often

Have the receiver check whether the message has been delivered before

Reliable links are also known as perfect links

Deliver at most once

Implemented on top of stubborn links

Reliable Link Properties

Reliable Link Property 1 (Validity) - If p_i and p_j are correct then every message sent by p_i to p_j is eventually delivered to p_j

Reliable Link Property 2 (No Duplication) - No message is delivered (to a process) more than once

Reliable Link Property 2 (No Creation) - No message is delivered unless it was sent

Reliable Link Notes

Reliable Link Property 1 is a liveness property

Reliable Link Property 2 and Reliable Link Property 3 are safety properties

We talk about the non-duplication of the delivery and not of the message, hence there is a distinction between Reliable Link Property 2 and Reliable Link Property 3

Reliable Links Algorithm

$$
\begin{aligned}
&\textit{Implements} : \textit{ReliableLinks}\ (rp2p)\\
&\textit{Uses} : \textit{StubbornLinks}\ (sp2p)\\
&\\
&\textit{upon event}\ < \textit{Init} > \textit{do}\\
&\qquad \textit{delivered} = \{\}\\
&\\
&\textit{upon event}\ < rp2pSend, dest, m > \textit{do}\\
&\qquad \textit{trigger}\ < sp2pSend, dest, m >\\
&\\
&\textit{upon event}\ < sp2pDeliver, src, m > \textit{do}\\
&\qquad \textit{if}\ m \notin \textit{delivered}\\
&\qquad\qquad \textit{trigger}\ < rp2pDeliver, src, m >\\
&\qquad\qquad \textit{delivered} = \textit{delivered} \cup \{m\}
\end{aligned}
$$

Reliable Link Assumption

We will assume reliable links for the development of future algorithms (unless otherwise specified)

Messages are uniquely identifies and the message identify included the sender's identifier

Roughly speaking, reliable links ensure that messages exchanges between correct processes are not lost

Failure Detection

Timing Assumptions

Timing assumptions relate to different:

- Processing speeds of processes (process asynchrony)
- Transmission speeds of messages (channel asynchrony)

Three basic types of system model

- Synchronous, partially synchronous and asynchronous

Timing Assumptions

Synchronous

Processing - The time it takes for a process to execute a step is bounded and know

Delays - There is a known upper bound on the time it takes for a message to be received

Clocks - The drift between local clock and the global real time clock is bounded and known

Timing Assumptions

Asynchronous

No assumptions

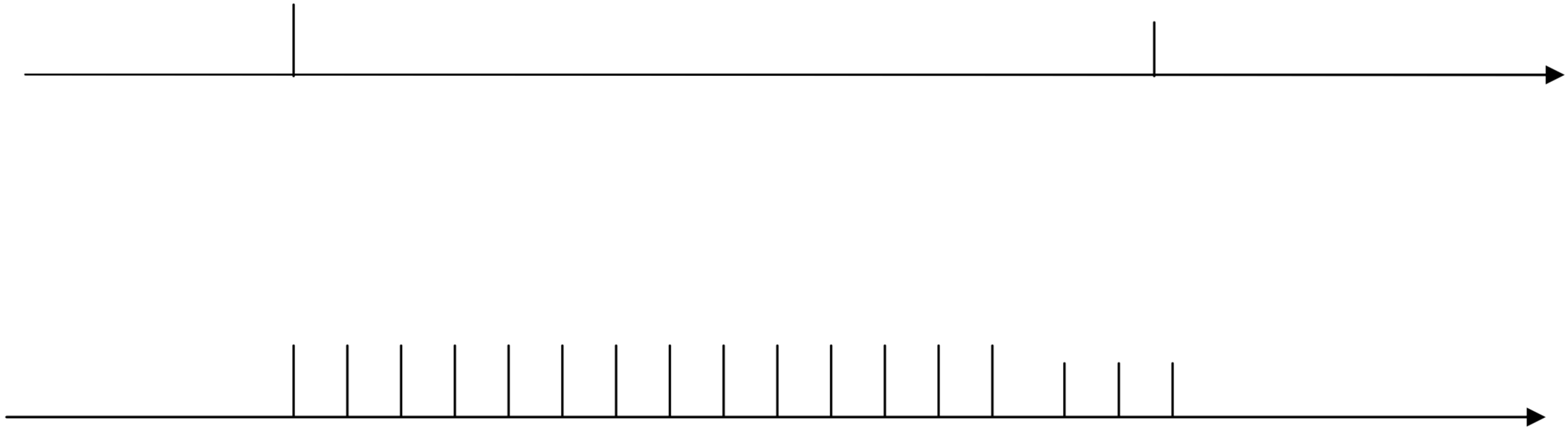
Challenging to design fault tolerant solutions, even for basic problems

Partially Synchronous (often taken to be “Eventually Synchronous”)

Timing bounds eventually hold (but you never know when)

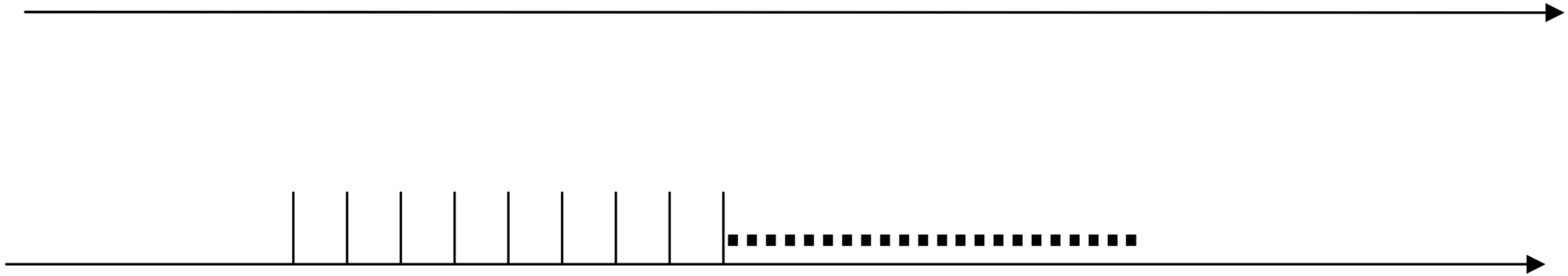
Synchronous Systems and Failure Detection

While one process taken one step, another process can take at most a bounded number of step



Asynchronous Systems and Failure Detection

While one process takes one step, another process can take any unbounded (but still finite) number of steps



What's The Connection?

Explicit assumptions are cumbersome when you are trying to apply them in the development of fault tolerant algorithms

A failure detector is an abstraction that provided us with a convenient way of encapsulating timing assumptions for distributed systems

Timeouts are the most obvious example of a failure detection mechanism connecting timing assumptions and failure detection

Failure Detectors

A failure detector is a distributed oracle that provides processes with suspicions about crashed processes

It is implement using timing assumptions (inherently encapsulates them)

According to the timing assumptions encapsulated, the suspicions that the failure detector provides may nor may not be accurate

Perfect Failure Detectors

Indication event $< crash, p >$ is used to notify that process p has crashed

Perfect failure detector properties

Perfect Failure Detector Property 1 (Strong Completeness) - Eventually every process that crashes is permanently detected by every correct process

Perfect Failure Detector Property 1 (Strong Accuracy) - No process is detected by any process before it crashes

Failure Detection - Algorithm

Implementation:

1. Processes periodically exchange heartbeat messages

2. A process sets a timeout bases on the worst case round trip of a message exchange

3. A process suspects another process if a timeout expires relating to that process

4. A process that delivers a message. From a suspect's process revises its suspicion and increases the associated timeout

Failure Detection - Algorithm

$$
\begin{array}{l}
\textit{upon} \ < \textit{Init} > \textit{do} \\
\quad \textit{timeout}[1,...,n] = d \\
\quad \textit{Initialise timer for every process} \\
\quad \textit{suspected} = \{\} \\
\\
\textit{periodically do} \\
\quad \textit{for every process } q \textit{ do} \\
\quad \textit{send} \ < \textit{heartbeat}, \textit{self} > \textit{to } q \\
\\
\textit{upon} \ < q \textit{ timer expires} > \ \textit{do} \\
\quad \textit{suspected} = \textit{suspected} \cup \{q\} \\
\quad \textit{Initialise timeout}[q]
\end{array}
$$

$$
\begin{array}{l}
\textit{upon} \ < \textit{heartbeat}, q > \textit{do} \\
\quad \textit{if } q \textit{ is suspected then} \\
\quad\quad \textit{suspected} = \textit{suspected} \setminus \{q\} \\
\quad\quad \textit{timeout}[q] = \textit{timeout}[q] + 1 \\
\quad \textit{Initialise timeout}[q]
\end{array}
$$

Failure Detection - Correctness

Look at different cases (for two processes):

1. Synchronous, where the initial timeout is accurate
2. Synchronous, where the initial timeout is too small
3. Partially synchronous, where the timeout is accurate for the synchronous phase
4. Partially synchronous, where the timeout is too small for the synchronous phase
5. Asynchronous

Aside - Rules of Time-Space Diagrams

Process execution proceeds from left to right

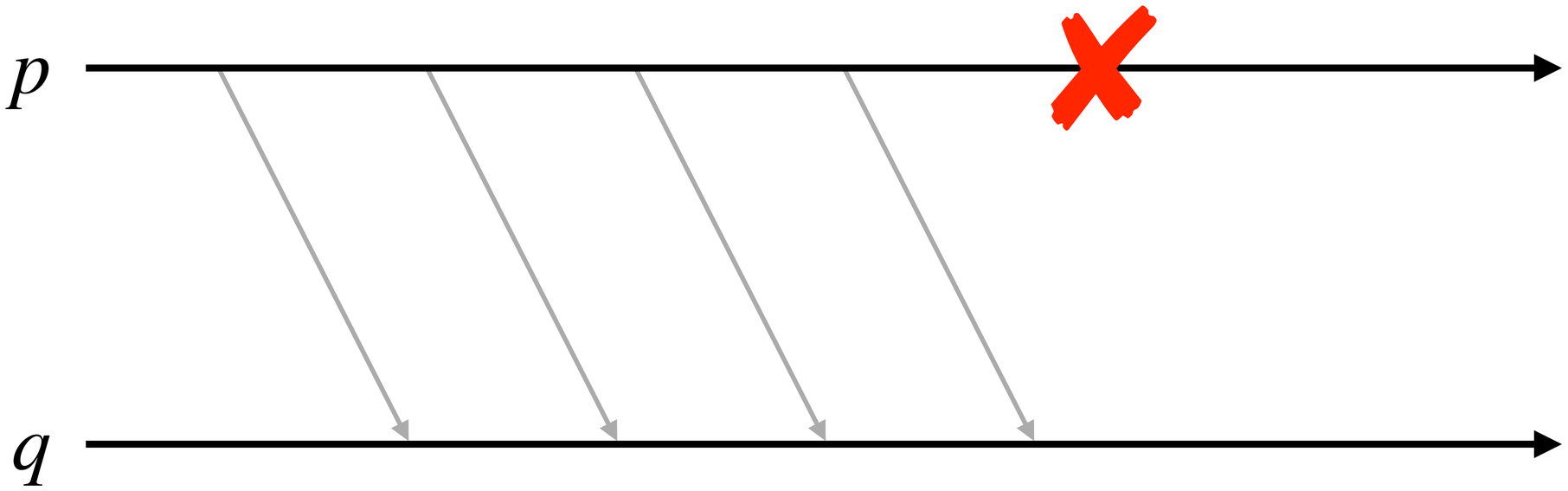
Message arrows connects send and receive events at processes

Message arrows must point to the right

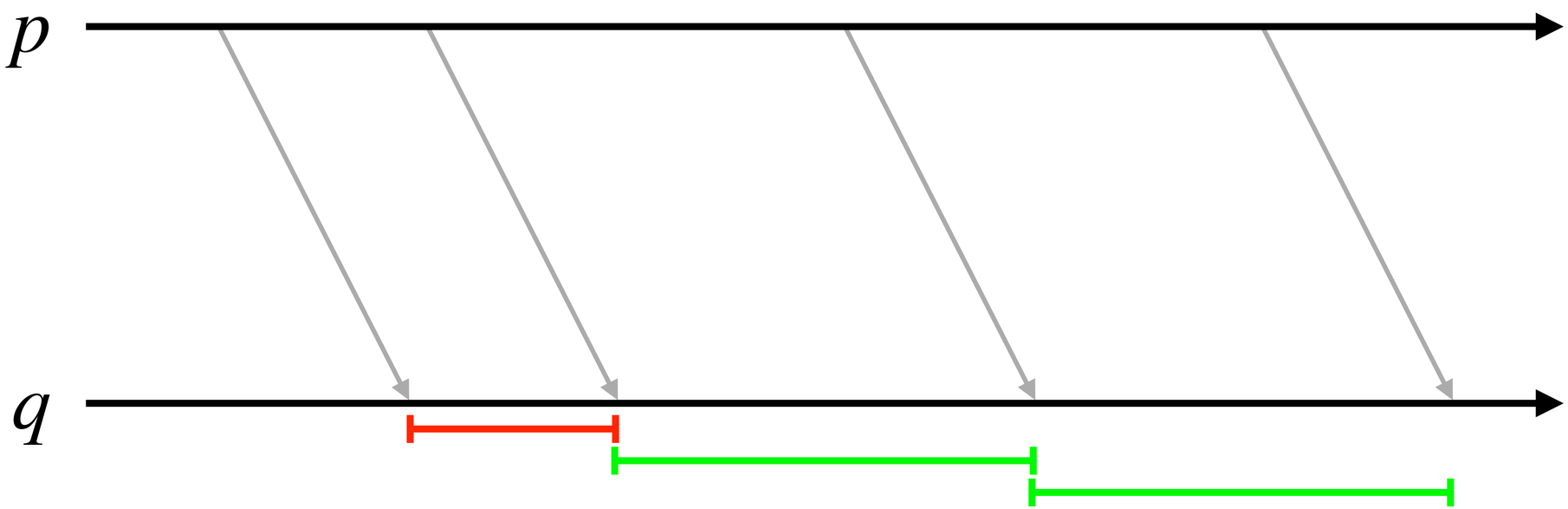
For perfect failure detectors, crash and suspicion can be interpreted as send and receive of a virtual message

Rubber band transformations - Transformation can be applied, e.g., stretching or squashing, can be applied as long as rules of time and space are not violated

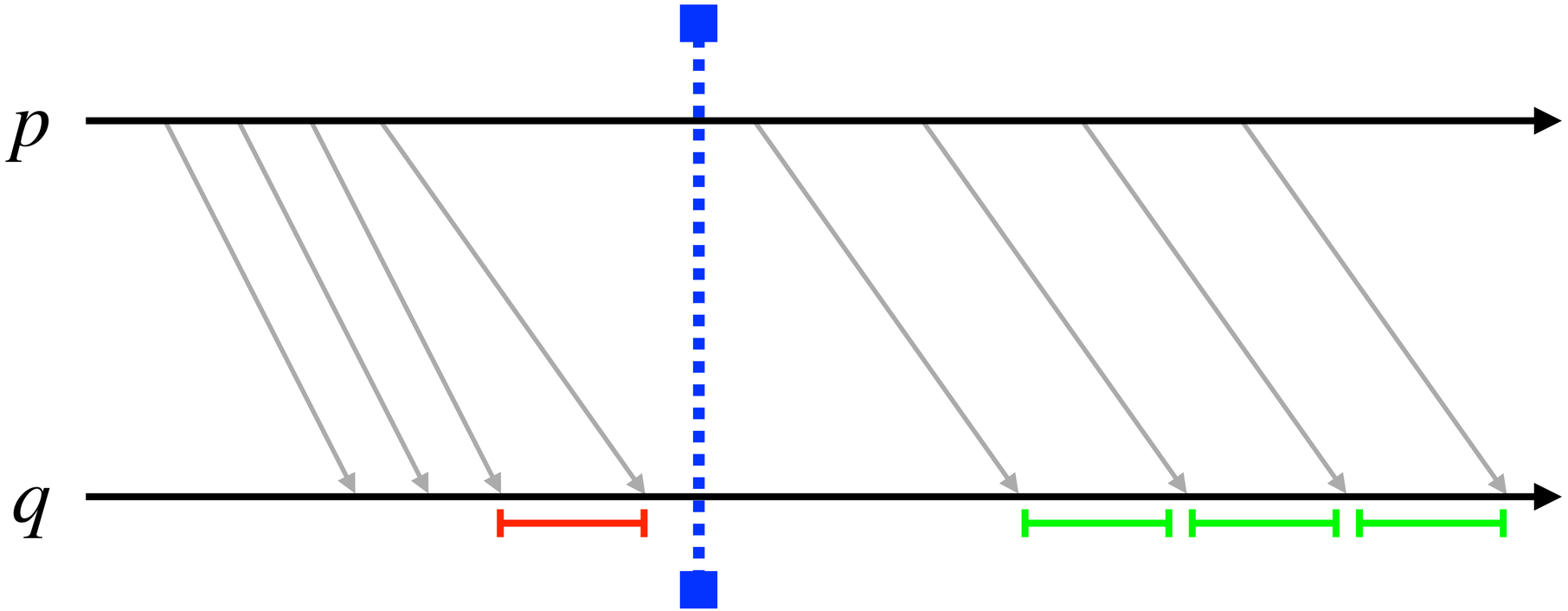
1. Synchronous, Good Timeout



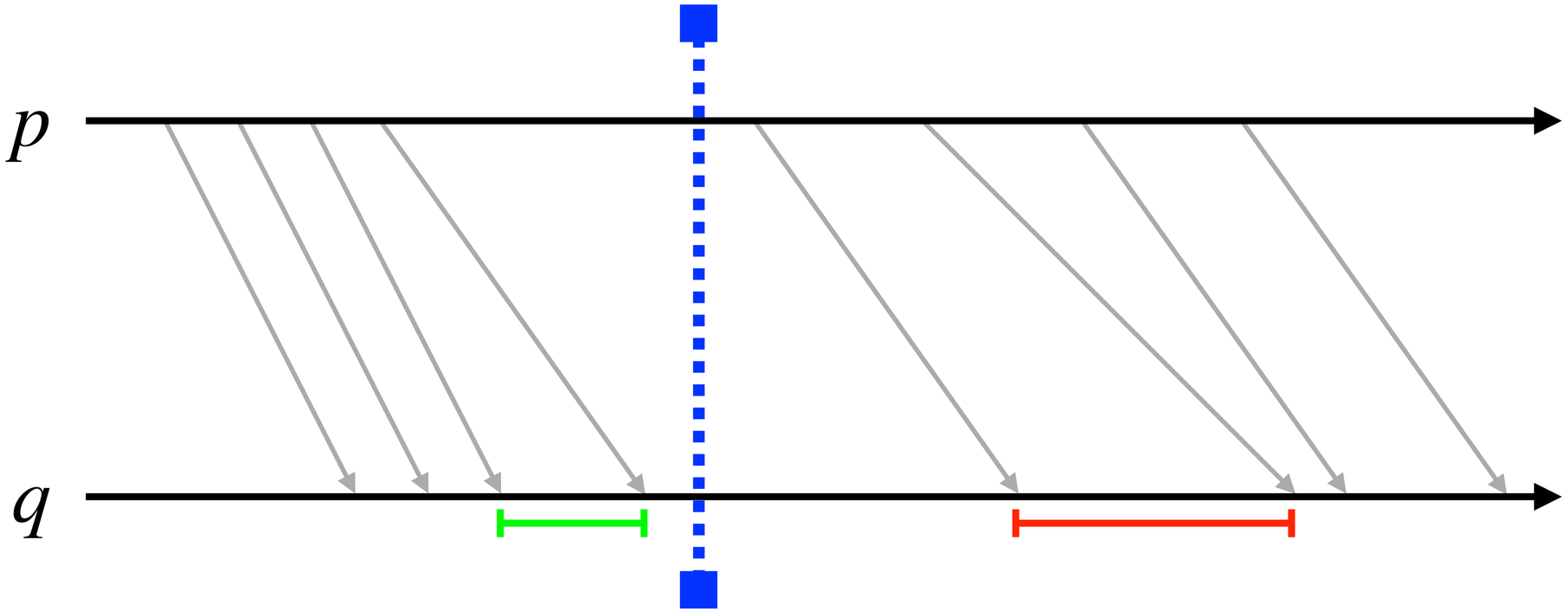
2. Synchronous, Bad Timeout



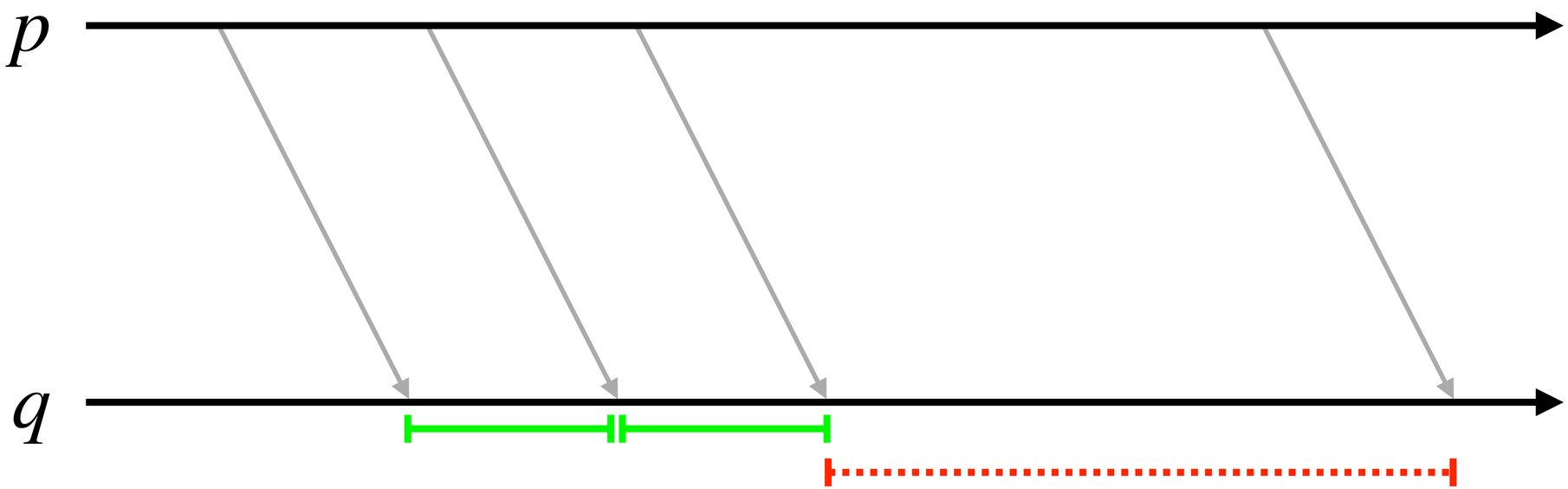
3. Partially Synchronous, Good Timeout



4. Partially Synchronous, Bad Timeout



5. Asynchronous



Failure Detectors

Perfect Failure Detectors

Perfect Failure Detector Property 1 (Strong Completeness) - Eventually every process that crashes is permanently suspected by every correct process

Perfect Failure Detector Property 1 (Strong Accuracy) - No process is suspected before it crashes

Eventually Perfect Failure Detectors

Eventually Perfect Failure Detectors

Eventually Perfect Failure Detector Property 1 (Strong Completeness) - Eventually every process that crashes is permanently suspected by every correct process

Eventually Perfect Failure Detector Property 1 (Eventually Strong Accuracy) - Eventually, no correct process is ever suspected