Fault Tolerance (part 1)

ECE 454 / 751: Distributed Computing

Instructor: Dr. Wojciech Golab

wgolab@uwaterloo.ca

Slides are derived from A. S. Tanenbaum and M. Van Steen,
Distributed Systems: Principles and Paradigms, 2nd Edition, Pearson-Prentice Hall, 2006.

Learning objectives

To develop a working knowledge of:

- failures, errors, and faults
- redundancy in hardware vs. in software
- the consensus problem
- failures in RPC protocols

Basic concepts

Fault tolerance is closely related to **dependability**, which implies the following requirements:

1. **Availability:** The system should operate correctly at any given instant in time. E.g., a real system may be 99% available.
2. **Reliability:** The system should run continuously without interruption. E.g., a real system may have a mean time between failures (MTBF) of one month.
3. **Safety:** Failure of the system should not have catastrophic consequences. E.g., your car can still come to a complete stop if the ABS fails.
4. **Maintainability:** A failed system should be easy to repair. E.g., disks can be replaced easily in a RAID.

Definitions: failure

Failure: when a system cannot fulfill its promises. Examples from http://www.infoworld.com/article/2606209/cloud-computing/162288-The-worst-cloud-outages-of-2014-so-far.html#slide1:

- Dropbox unavailable for two days around Jan 10, 2014. (Cause: scripting glitch.)
- Google services stutter for about one hour on Jan 24, 2014. (Cause: software bug leading to misconfiguration.)
- Samsung's Smart TV service unavailable for hours on April 20, 2014. (Cause: fire at Samsung facility in South Korea.)
- Apple iCloud unavailable for a few hours on June 12, 2014. (Cause: not reported.)
- …

Definitions: error and fault

Error: part of a system's state that may lead to a failure.

- Example: dropped or damaged network packet.
- Example: corrupted data retrieved from hard disk or SSD.

Fault: the cause of an error.

- Example: bad transmission medium, such as when a person talking on a cell phone descends into a subway station, or when a bird flies in front of a microwave receiver.
- Example: hard disk head crashes, or flash cells in an SSD wear out.

A fault may lead to an error, which may lead to a failure.

Three types of faults

Transient: occur once and disappear.

- Example: bird flies in front of microwave receiver.
- Example: electromagnetic interference from solar flares.

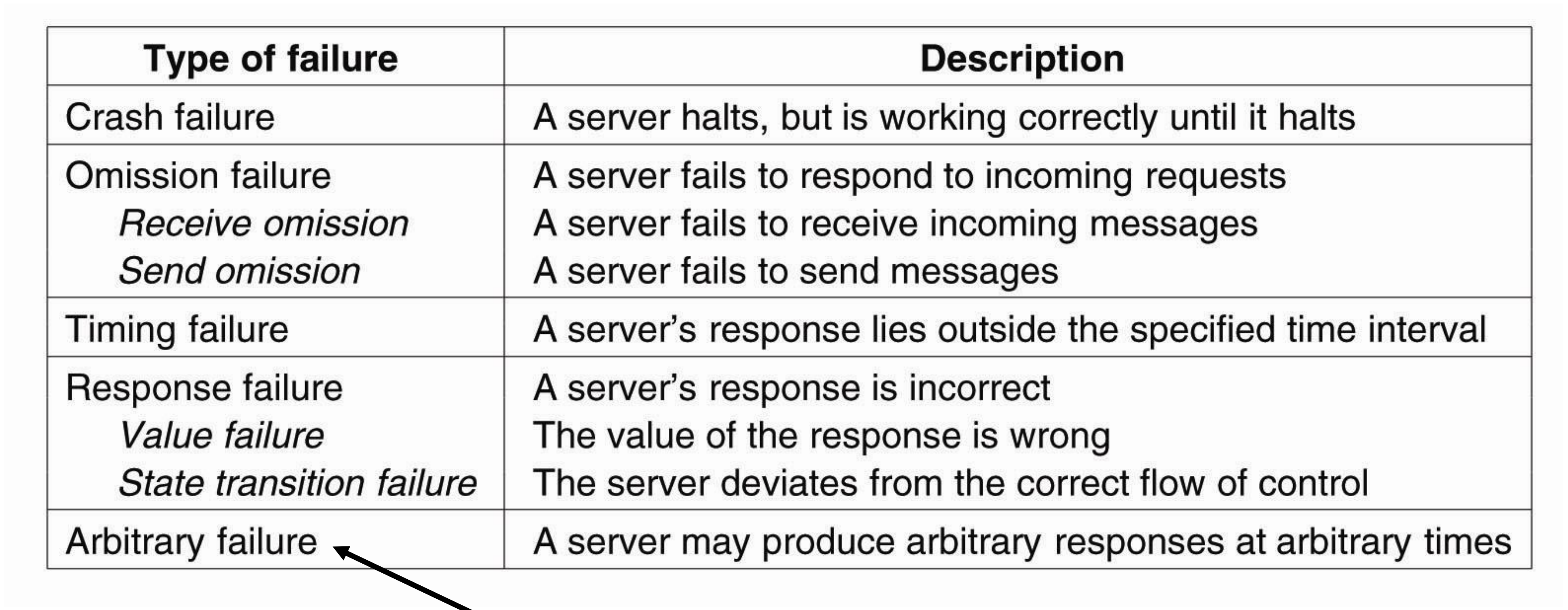
Intermittent: occur and vanish, reappear, etc.

- Example: loose contact on electrical connector, such as when a memory chip or expansion board is not properly seated.

Permanent: continues to exist until faulty component is replaced.

- Example: burnt out power supply in a server.
- Example: hard disk destroyed by power surge.

Five types of failures

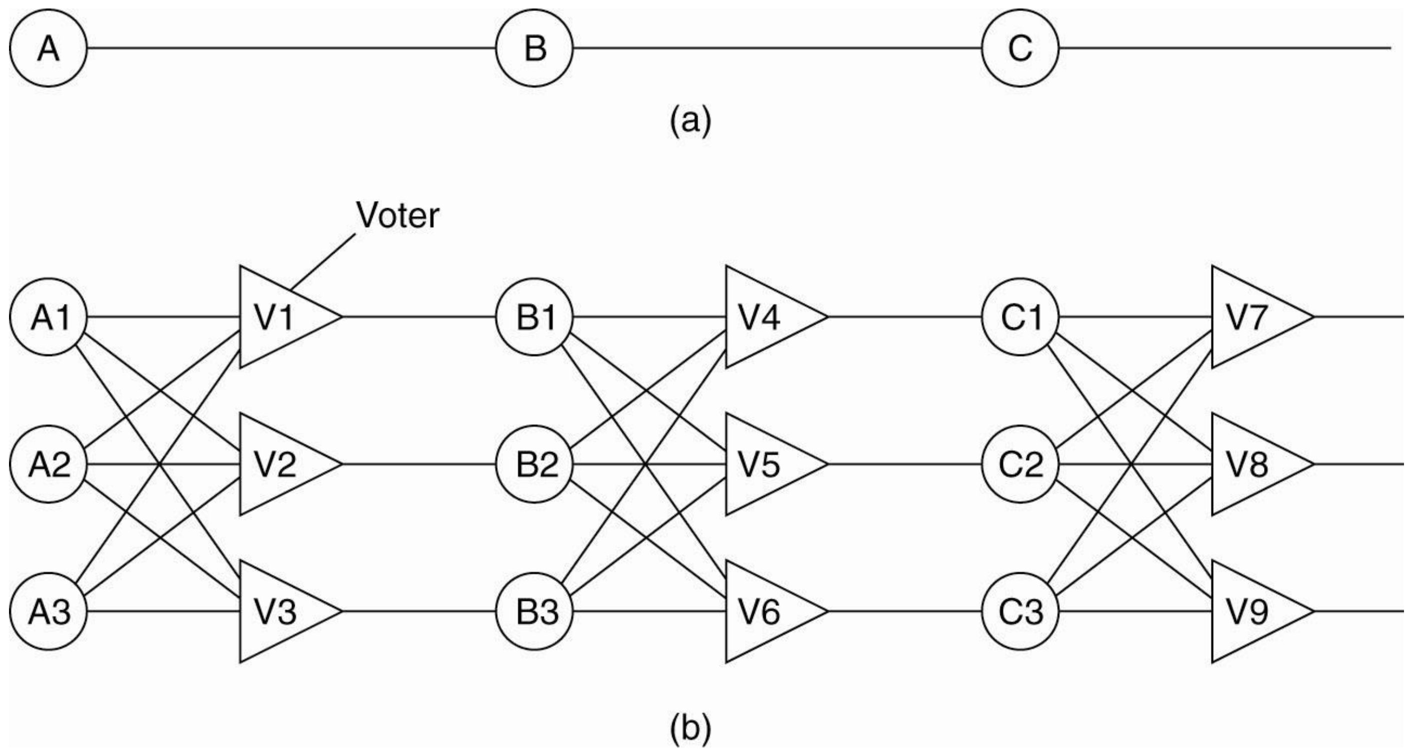


Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure	A server fails to respond to incoming requests
Receive omission	A server fails to receive incoming messages
Send omission	A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure	A server's response is incorrect
Value failure	The value of the response is wrong
State transition failure	The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

Byzantine failure, beyond the scope of this course

Masking failure by redundancy

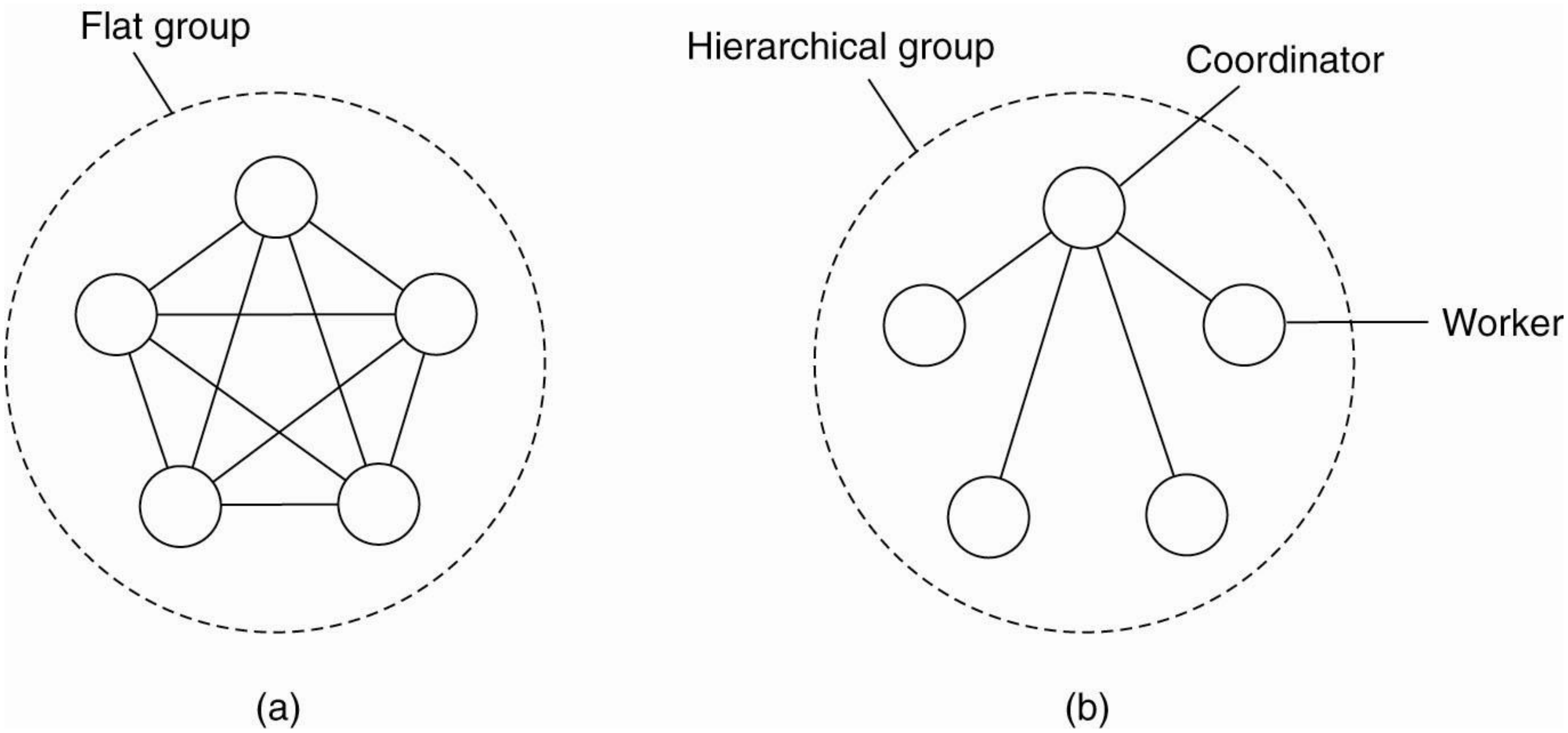
Hardware example: **triple modular redundancy (TMR)**. Diagram of three components (A/B/C) shows before (a) and after picture (b).



Masking failure by redundancy

Software example: **identical processes** organized into a flat group or a hierarchical group.

Food for thought: how can we ensure identical process state?



Agreement: consensus problem

Consensus problem definition:

- Each process has a procedure propose(val) and a procedure decide().
- Each process first proposes a value by calling propose(val) once, with some argument val determined by the initial state of the process.
- Each process then learns the value agreed upon by calling decide().

Safety property 1 (agreement): two calls to decide() never return different values.

Safety property 2 (validity): if a process calls decide() with response v then some process invoked a call to propose(v).

Liveness property: if a process calls propose(v) or decide() and does not fail then this call eventually terminates.

Note: Process crashes and network failures only.

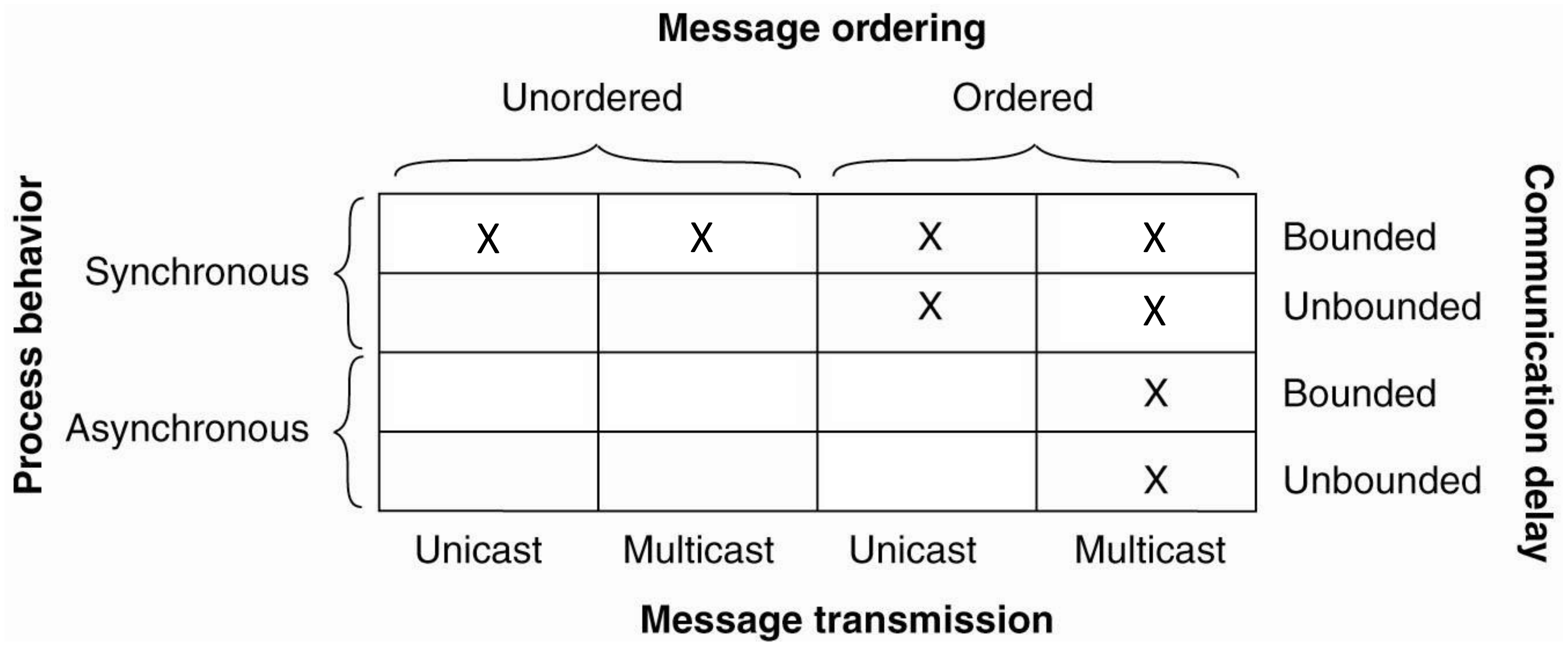
Variations

The possibility of solving consensus in a failure-prone distributed environment depends on several factors.

1. **Synchronous vs. asynchronous processes.** Is there a bound on the amount of time it takes for a process to take its next step? Is the bound known by all processes?
2. **Communication delays.** Is there a bound on the length of time it takes for a sent message to be delivered? Is the bound known by all processes?
3. **Message delivery order**. How does the order in which messages are sent affect the order in which they are delivered to the recipients?
4. **Unicast vs. multicast** messaging.

Possibilities and impossibilities

Very informal summary of possibility and impossibility results for distributed consensus. (X indicates possibility.)



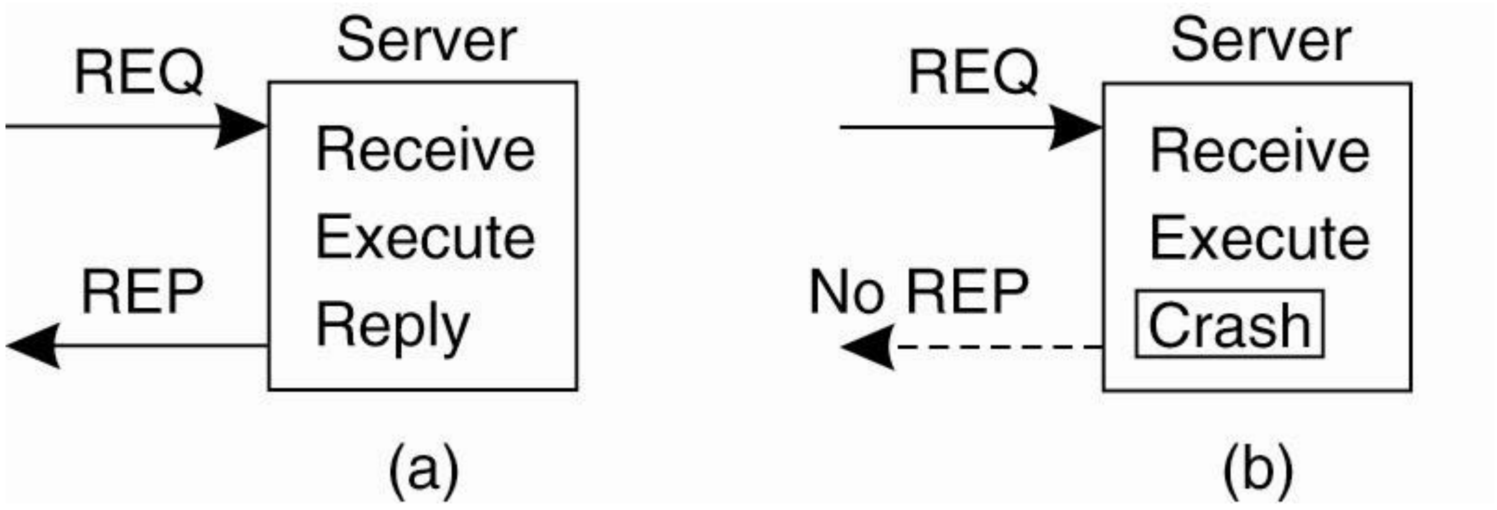
		Message ordering: Unordered		Ordered	
Process behavior	**Communication delay**	Unicast	Multicast	Unicast	Multicast
Synchronous	Bounded	X	X	X	X
	Unbounded			X	X
Asynchronous	Bounded				X
	Unbounded				X
		Message transmission			

RPC semantics under failures

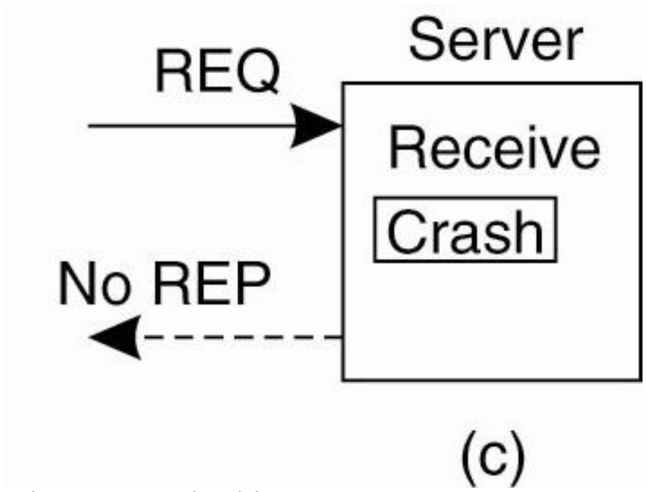
RPC systems may exhibit five classes of failure scenarios:

1. The client is unable to locate the server.
2. The request message from the client to the server is lost.
3. The server crashes after receiving a request.
4. The reply message from the server to the client is lost.
5. The client crashes after sending a request.

RPC server crashes



In the normal case (a) the client knows that the server executed the request. In contrast, in the failure cases (b) and (c) the client cannot tell whether the server crashed before or after executing the request.



Dealing with RPC server crashes

There are several ways to deal with the failure scenario shown in the previous slide:

1. **Reissue** the request, leading to **at-least-once semantics**. As a side-effect, the request may be processed multiple times by the service handler, which is safe as long as the request is **idempotent** (i.e., repeated executions have the same effect as one execution).
2. **Give up** and report a failure, leading to **at-most-once semantics**. There is no guarantee that the request has been processed.
3. **Determine whether the request was processed and reissue if needed**, leading to **exactly once semantics**. This scheme is difficult to implement as the server may have no way of knowing whether it performed a particular action.
4. **Make no guarantees** at all, leading to confusion.

Actions and acknowledgments

Consider an RPC service that prints a message to the screen upon receiving a request from the client.

In theory the service handler has a choice to acknowledge the request to the client either before or after printing the message, leading to different orderings of three events:

- **M**: server replies to client with acknowledgment message
- **P**: server prints the text
- **C**: server crashes

Actions and acknowledgments

M →P →C: A crash occurs after sending the completion message and printing the text.

M →C (→P): A crash happens after sending the completion message, but before the text could be printed.

C (→M →P): A crash happens before the server could do anything.

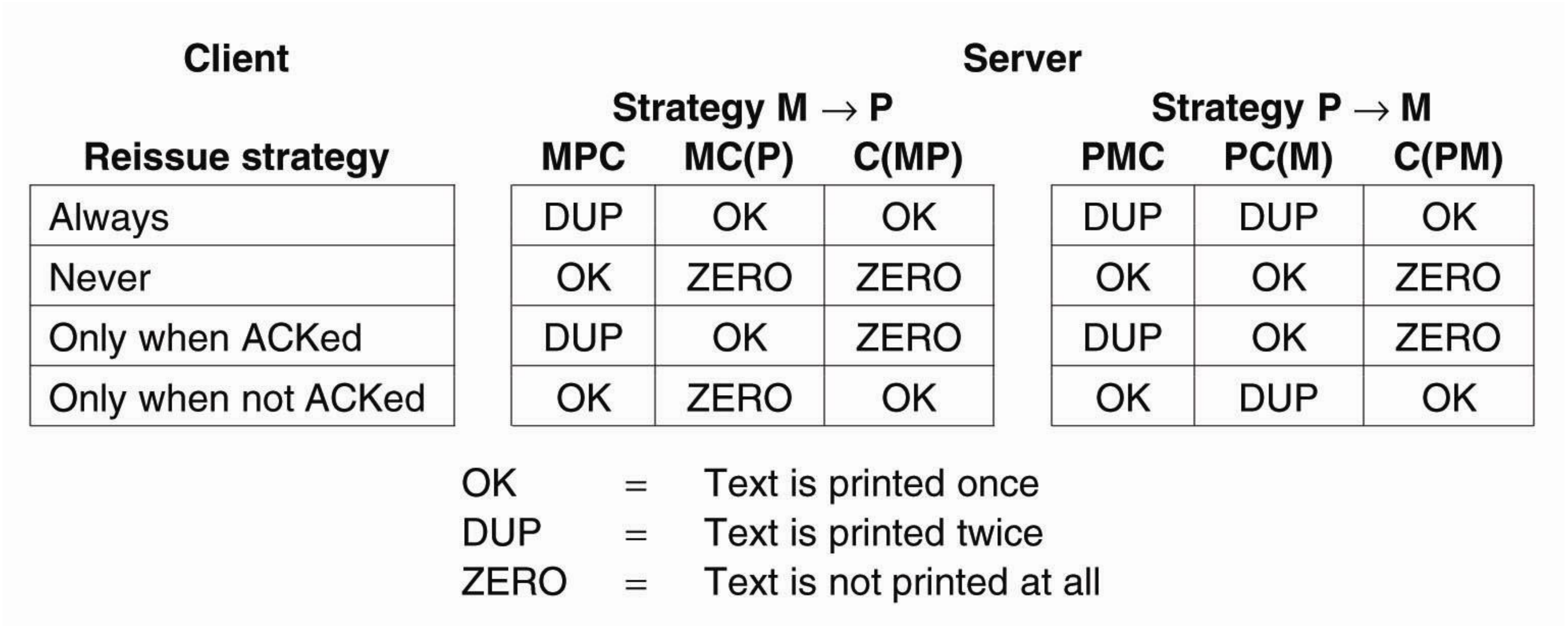
P →M →C: A crash occurs after sending the completion message and printing the text.

P →C (→M): The text is printed, after which a crash occurs before the completion message could be sent.

C (→P →M): A crash happens before the server could do anything.

Actions and acknowledgments

Behaviors under M → P strategy and P → M strategy:



Client	Server					
	Strategy M → P			Strategy P → M		
Reissue strategy	MPC	MC(P)	C(MP)	PMC	PC(M)	C(PM)
Always	DUP	OK	OK	DUP	DUP	OK
Never	OK	ZERO	ZERO	OK	OK	ZERO
Only when ACKed	DUP	OK	ZERO	DUP	OK	ZERO
Only when not ACKed	OK	ZERO	OK	OK	DUP	OK

OK	=	Text is printed once
DUP	=	Text is printed twice
ZERO	=	Text is not printed at all

Assumption: exactly one crash failure, at most one retry.