12th International Parallel Processing Symposium &
9th Symposium on Parallel and Distributed Processing

PROTOCOLS FOR NON-DETERMINISTIC COMMUNICATION OVER SYNCHRONOUS CHANNELS

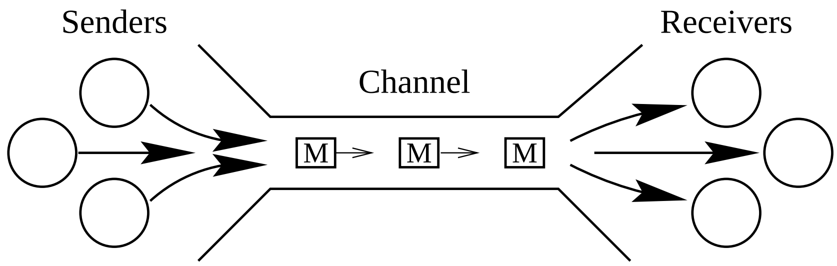
Erik Demaine

Department of Computer Science
University of Waterloo, Waterloo, Ontario, Canada
eddemaine@uwaterloo.ca

March 31, 1998

CSP (COMMUNICATING SEQUENTIAL PROCESSES)

- Concurrent-programming model proposed by C. A. R. Hoare in 1978
  - Basis for several concurrent-programming languages
  - Examples: Concurrent ML, Facile, Fortran-M, occam
- Communicate over synchronous channels



  - **send** (chan, x) waits for matching **receive**
  - **receive** (chan, x) waits for matching **send**
- Non-deterministic choice

```
choose:
    receive (chan1, x1)
    receive (chan2, x2)
    send    (chan3, x3)
```

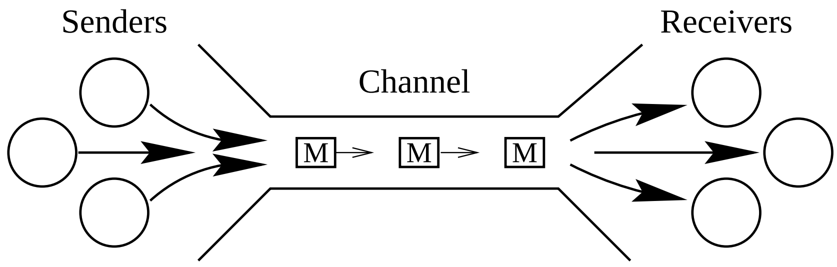
  - Pick exactly one of the listed operations
  - Whichever can complete first, breaking ties arbitrarily
  - **Generalized alternative**: both sends and receives allowed

MOTIVATION

- How to execute non-deterministic choice?
  - Processes change their mind
- How to negotiate with potential partners to pick an operation such that
  - Partner chooses the complement of the operation
  - No deadlock
  - Fast
- Main difficulty: generalized alternative (sends and receives)
- Problem has fairly long history
  - Deadlock-free algorithms: Bornat (1986) and Knabe (1992)

CHANNELS

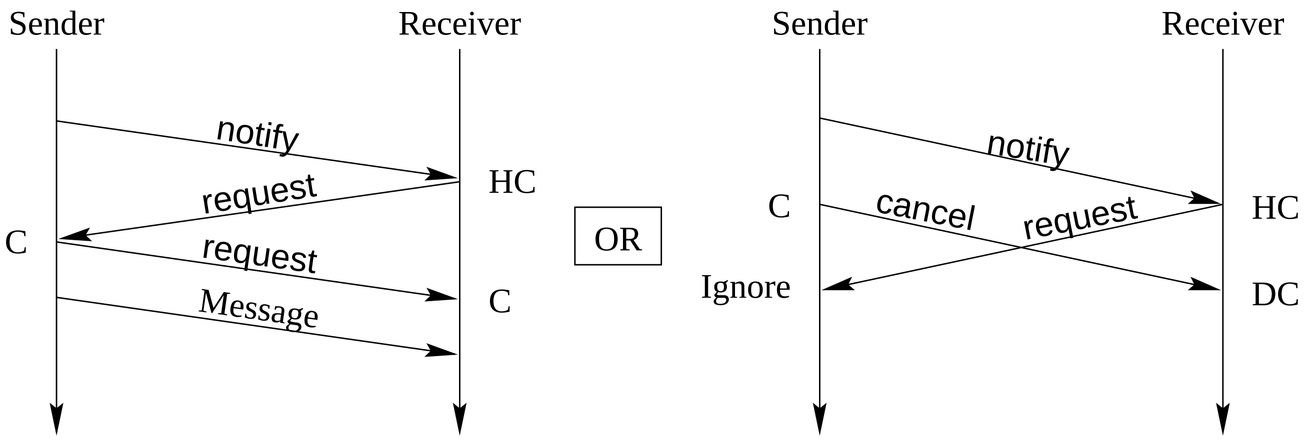
- In CSP, channels are many-to-many



- Difficult to maintain group membership in distributed system

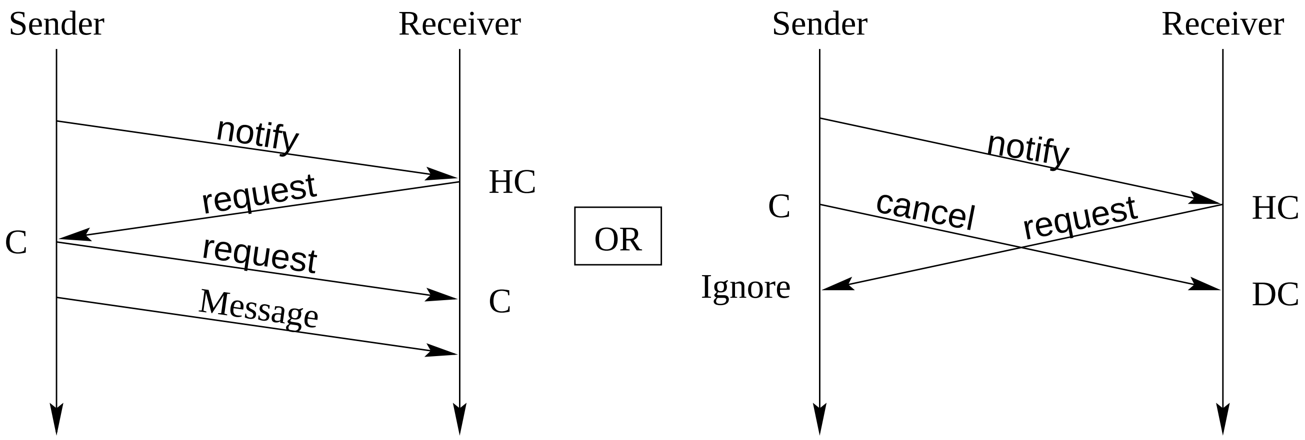
- Many just consider one-to-one channels
- Intermediate: fan channels
  = many-to-one and one-to-many channels
  - **Owner**: the single receiver/sender
  - **Members**: other processes connected to channel

GENERAL FRAMEWORK



- Process p willing to communicate on channel may take **initiative**
  - Send **notify** message to processes on other side
- On receipt at process q, half-commit (HC) if not already
  - Send **request** message to p
- Reply from p either
  - Still willing: **request** message — full commit (C)
  - Chose another option: **cancel** message — decommit (DC)

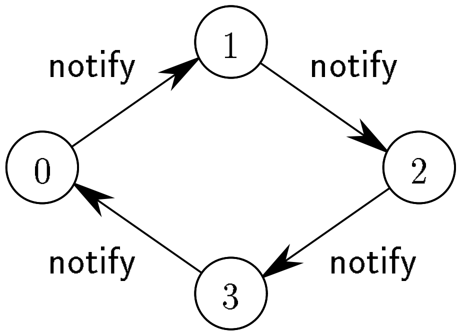
SIMPLE PROTOCOL

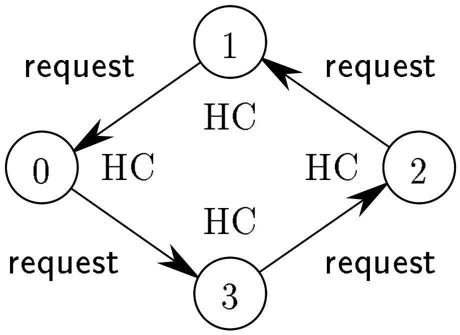


- Take initiative if member of channel
  - Send one message (to owner)
- Performance
  - 4 messages
  - 3 messages: Can piggyback user-level message on **request**
  - Risky to put on first **request** message (one-to-many channels)

DEADLOCK-FREE SIMPLE PROTOCOL

- Get potential deadlock if a cycle of **notify** messages



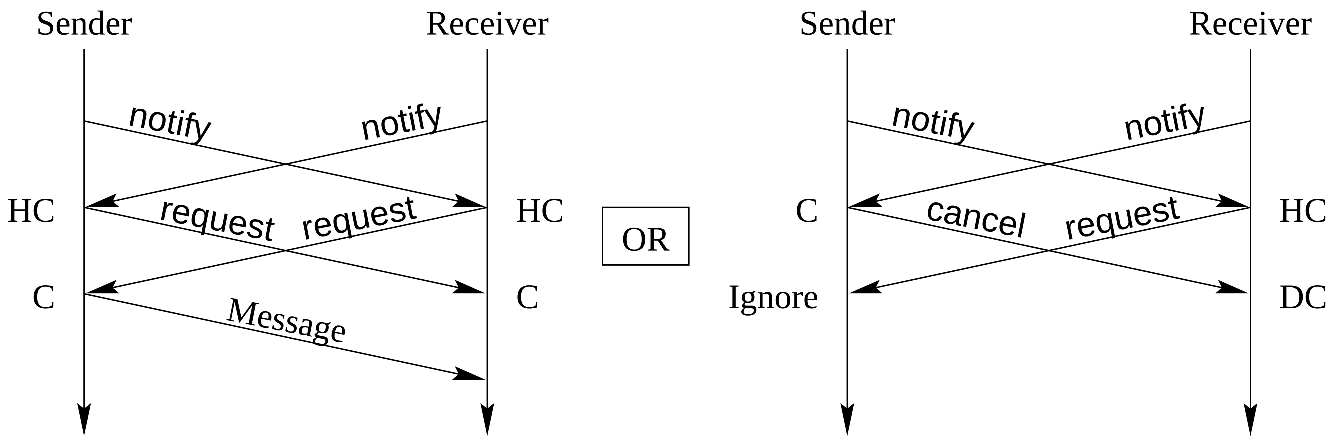


- Use unique id of processes (totally ordered)
- Send **notify** messages to processes with lower id only

  $\Rightarrow$ Cycles of notifies are broken
- Prevents deadlock but requires multicasting

FAST PROTOCOL (with deadlock)

- Takes full initiative



- Performance
  - 3 message cycles
  - Just 2 cycles if we piggyback onto sender's **request** message
  - Optimal for synchronous communication

DEADLOCK-FREE FAST PROTOCOL

- Wait for deadlock to occur (with high probability)
- On timeout, if half-committed,
  - Suspect there is a deadlock
  - Send **relief** message to lesser processes half-committed to me
  - That is, force them to decommit
- Ensures that some process in every cycle will be relieved, but at least one will not

  $\Rightarrow$ Removes (potential) deadlock

INTEROPERABILITY OF PROTOCOLS

- All protocols based on general framework
  - Different initiative
  - Different timeout
- Can use deadlock-avoidance technique for simple protocol

  $\Rightarrow$ Don't need multicasts
- "Partial fast" protocol
  - Owner sends **notify** messages to some of the members
  - None $\Rightarrow$ simple $\longrightarrow$ all $\Rightarrow$ fast
- Possible to adapt protocol to determine
  - Initiative
  - Whether should piggyback on first **request** message
  - Deadlock-timeout value

CONCLUSION

- Summary of protocols and performance:

Protocol	Successful		Unsuccessful	
	Cycles	Messages	Cycles	Messages
Bornat/Knabe	6	7	3–6	4–7
Simple	4	4	2	3
Simple + piggybacking	3	3	2	3
Fast	3	$2m+3$	2	$2m+2$
Fast + piggybacking	2	$2m+2$	2	$2m+2$

- Fan channels almost as efficient as one-to-one channels
- Fast protocol achieves optimal number of message cycles for synchronous communication
- Can avoid deadlock with little overhead
- Future work:
  - Compare protocols for various applications and networks
  - Evaluate effectiveness of adaptive extensions