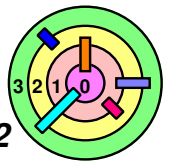
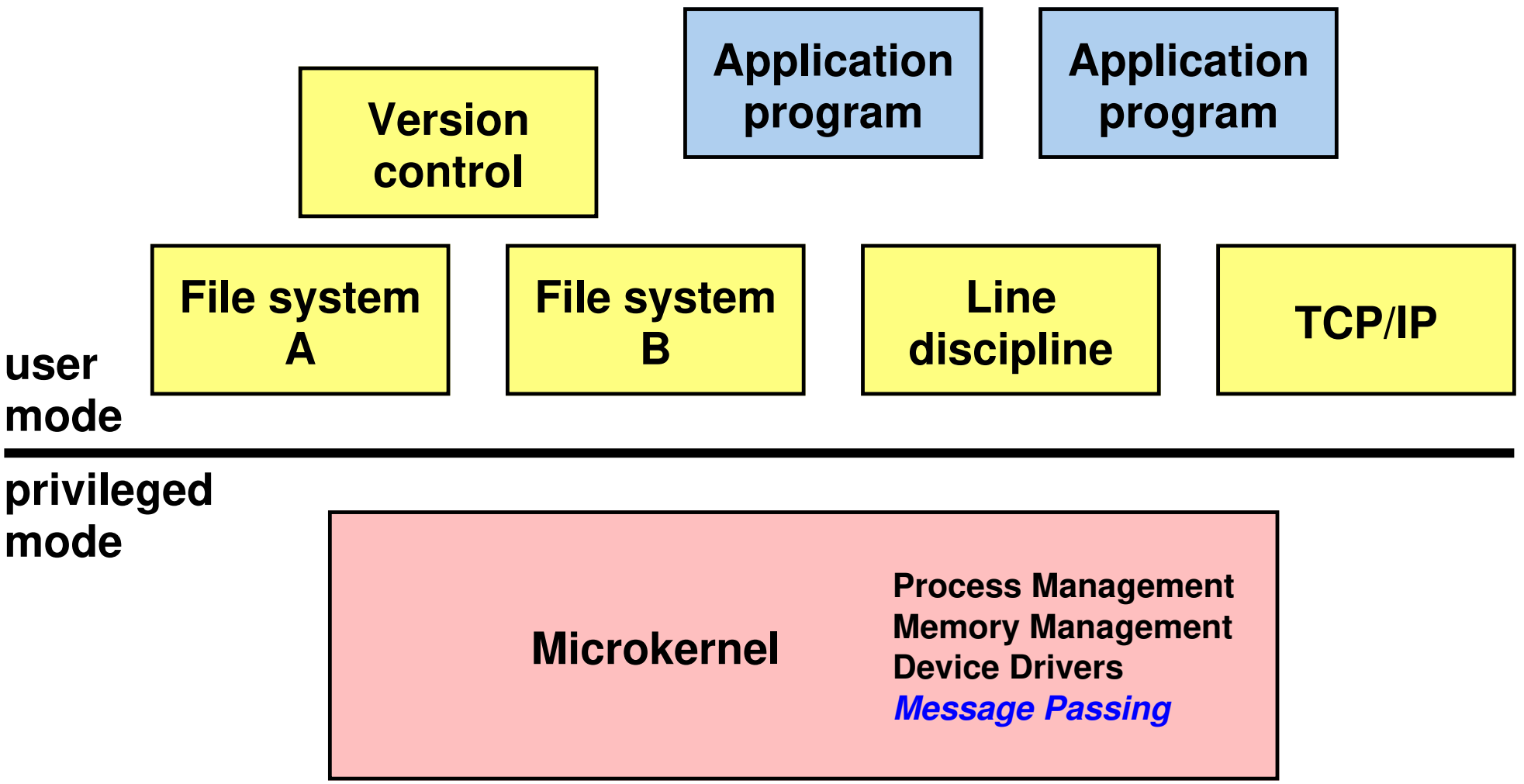
Microkernels
(Back to Section 4.2)

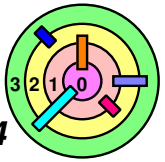


OS Services as User Apps



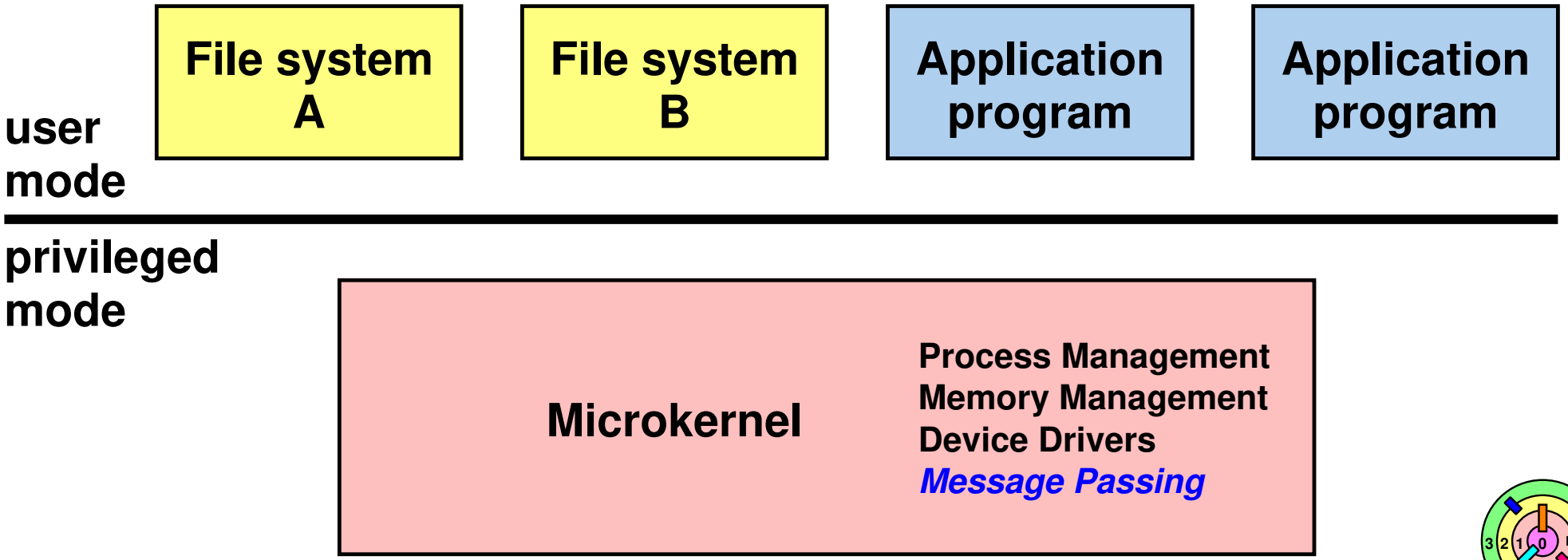
Why?

- It's cool ...
- Assume that OS coders are incompetent, malicious, or both ...
  - OS components run as protected user-level applications
- Extensibility
  - easier to add, modify, and extend user-level components than kernel components



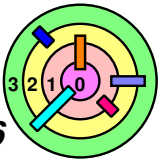
Implementation Issues

- How are modules linked together?
  - e.g., how would you implement `read()/write()`?
    - can't use system calls any more!
  - e.g., which file system supports `read()/write()`?
- How is data moved around efficiently?



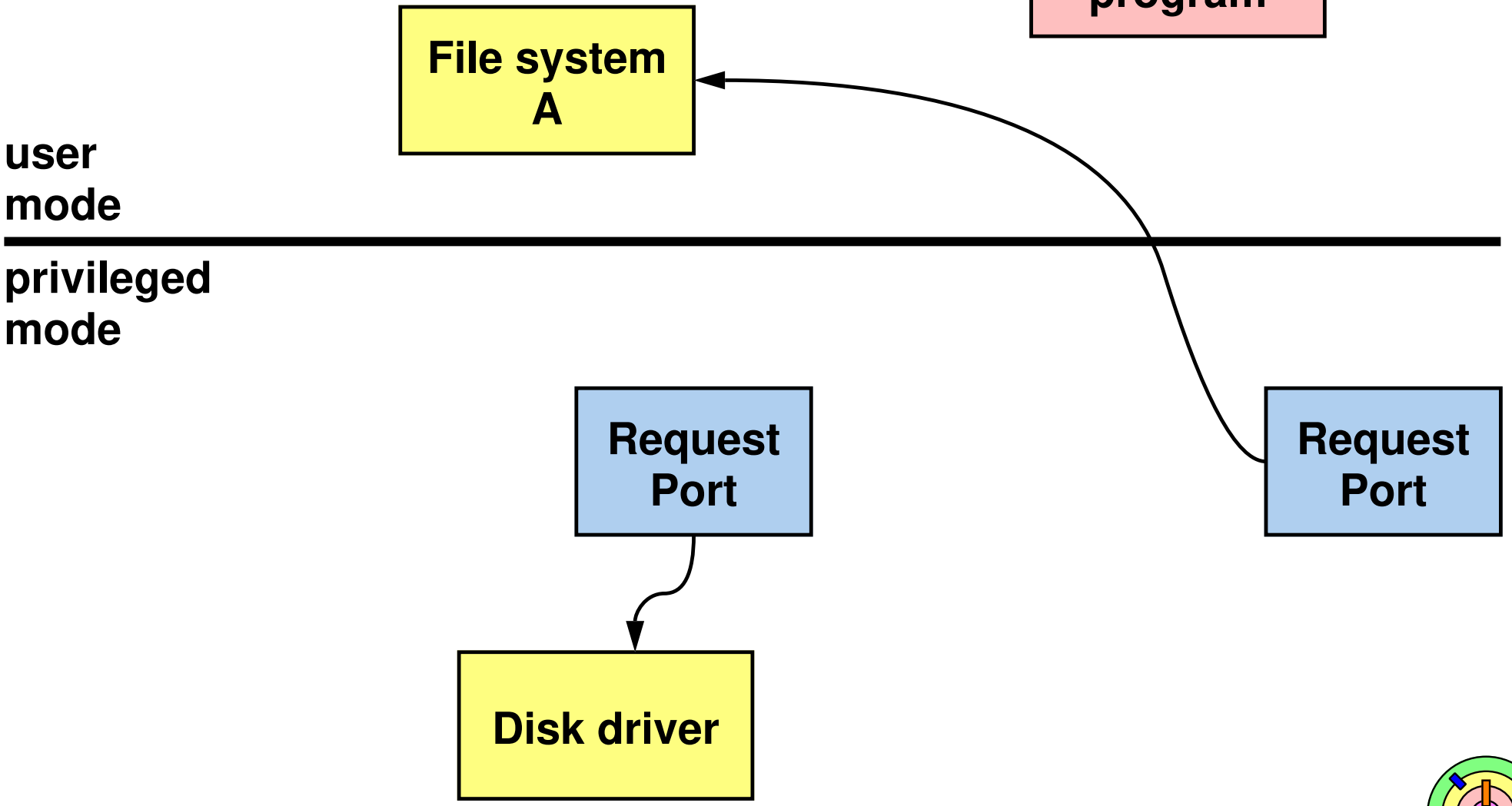
Mach

- Developed at CMU, then Utah
- Early versions shared kernel with Unix
  - basis of NeXT OS
- Later versions still shared kernel with Unix
  - basis of OSF/1
    - basis of Mac OS X
- Even later versions actually functioned as working microkernel
  - basis of GNU/HURD project
    - HURD: HIRD of Unix-replacing daemons
    - HIRD: HURD of interfaces representing depth



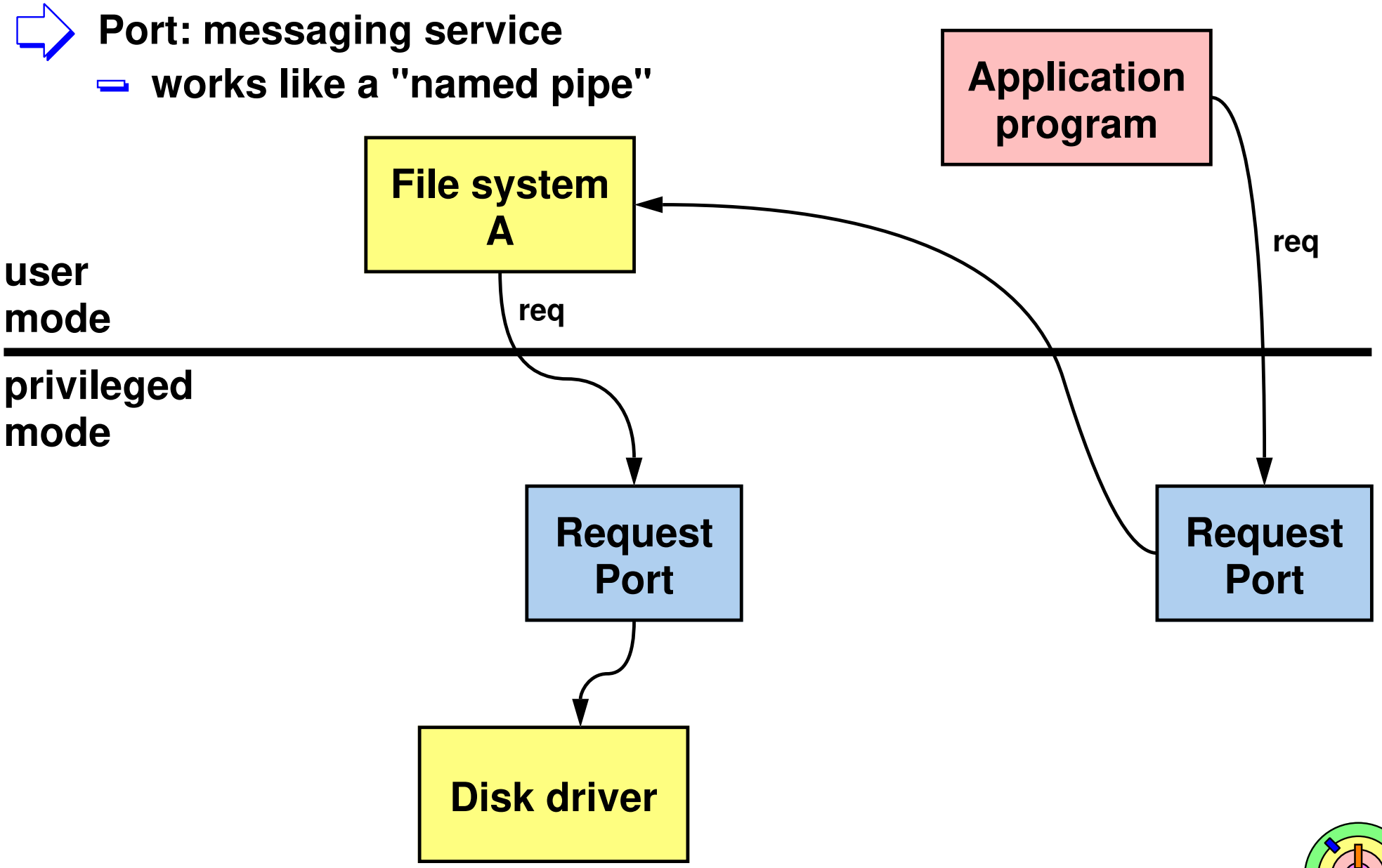
Example

- Port: messaging service
  - works like a "named pipe"



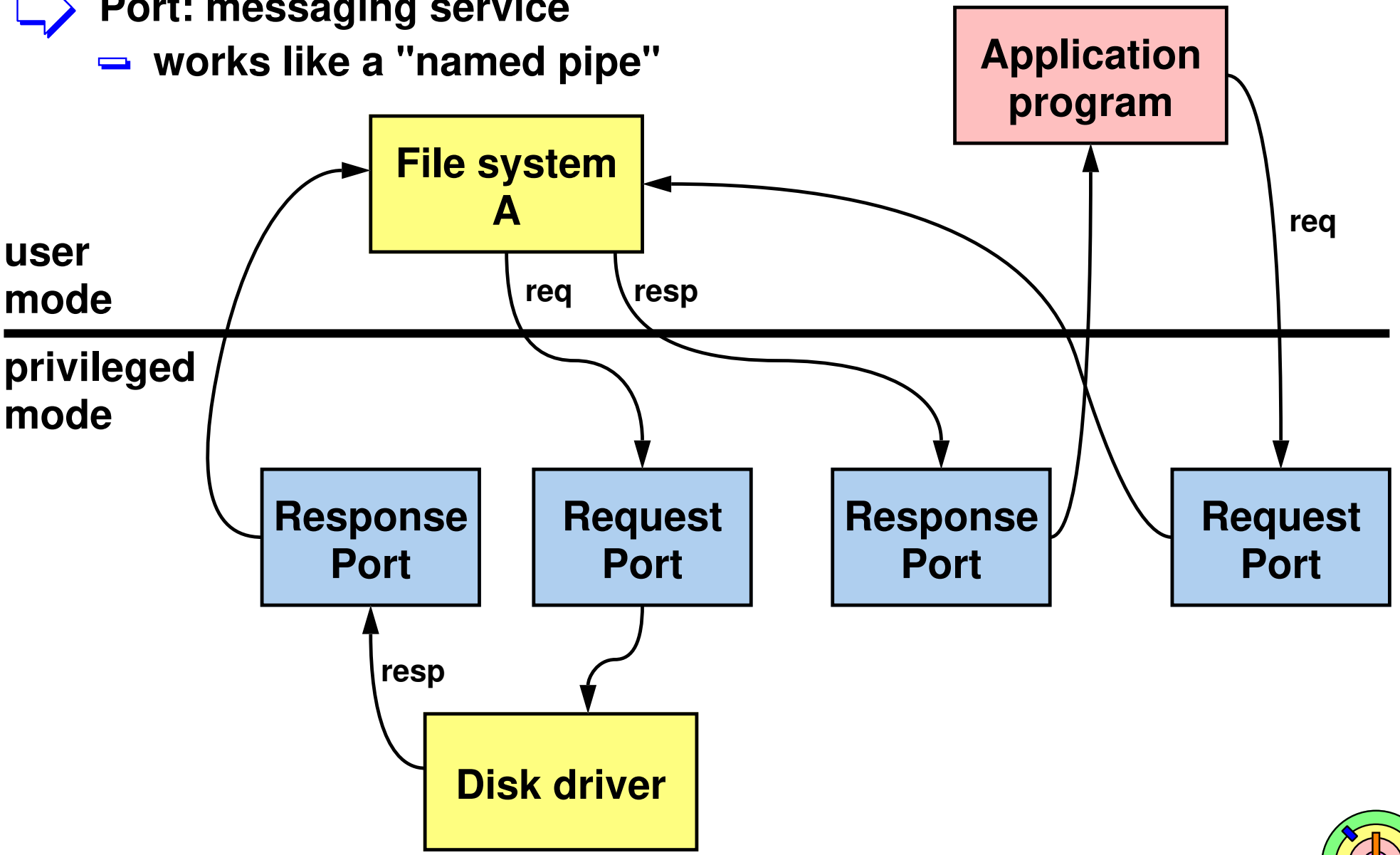
Example

- Port: messaging service
  - works like a "named pipe"



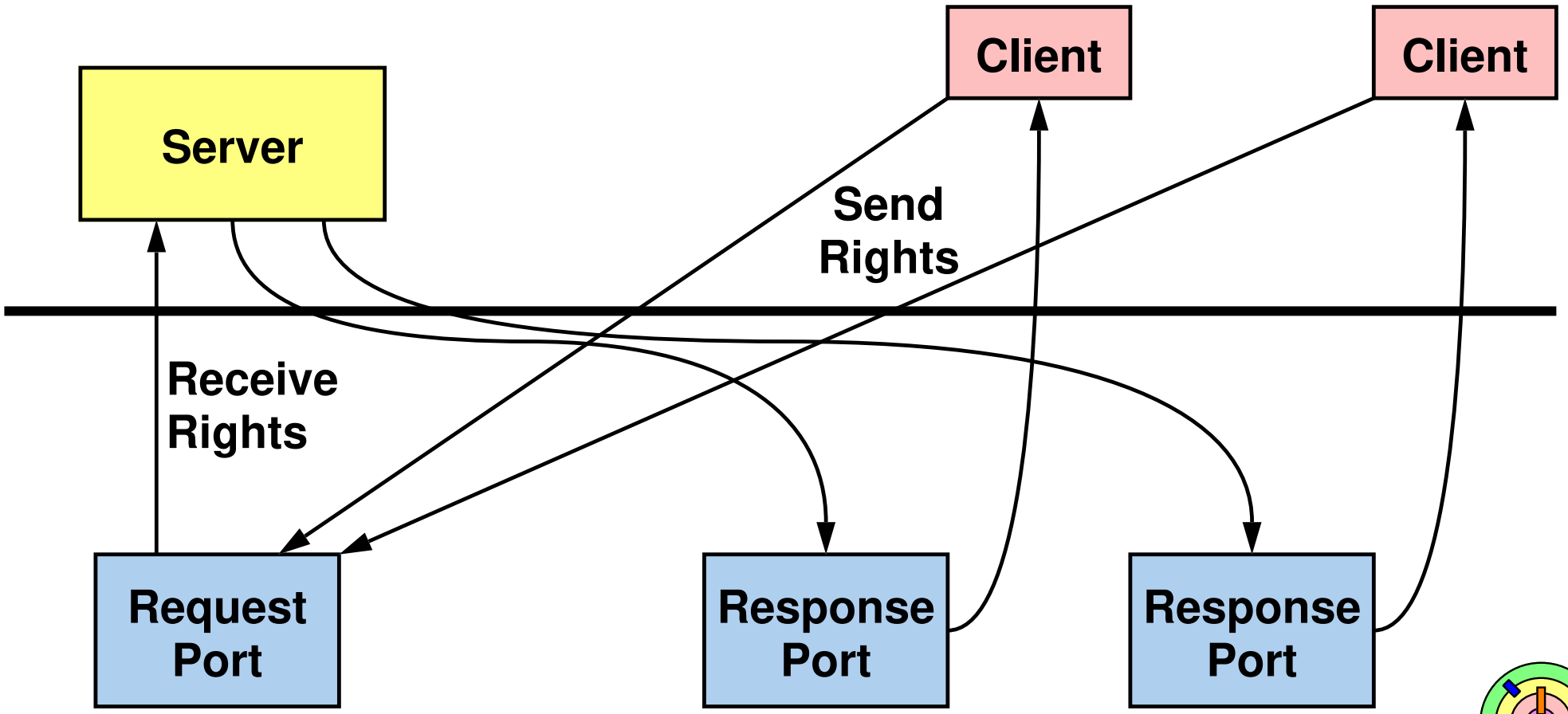
Example

- Port: messaging service
  - works like a "named pipe"



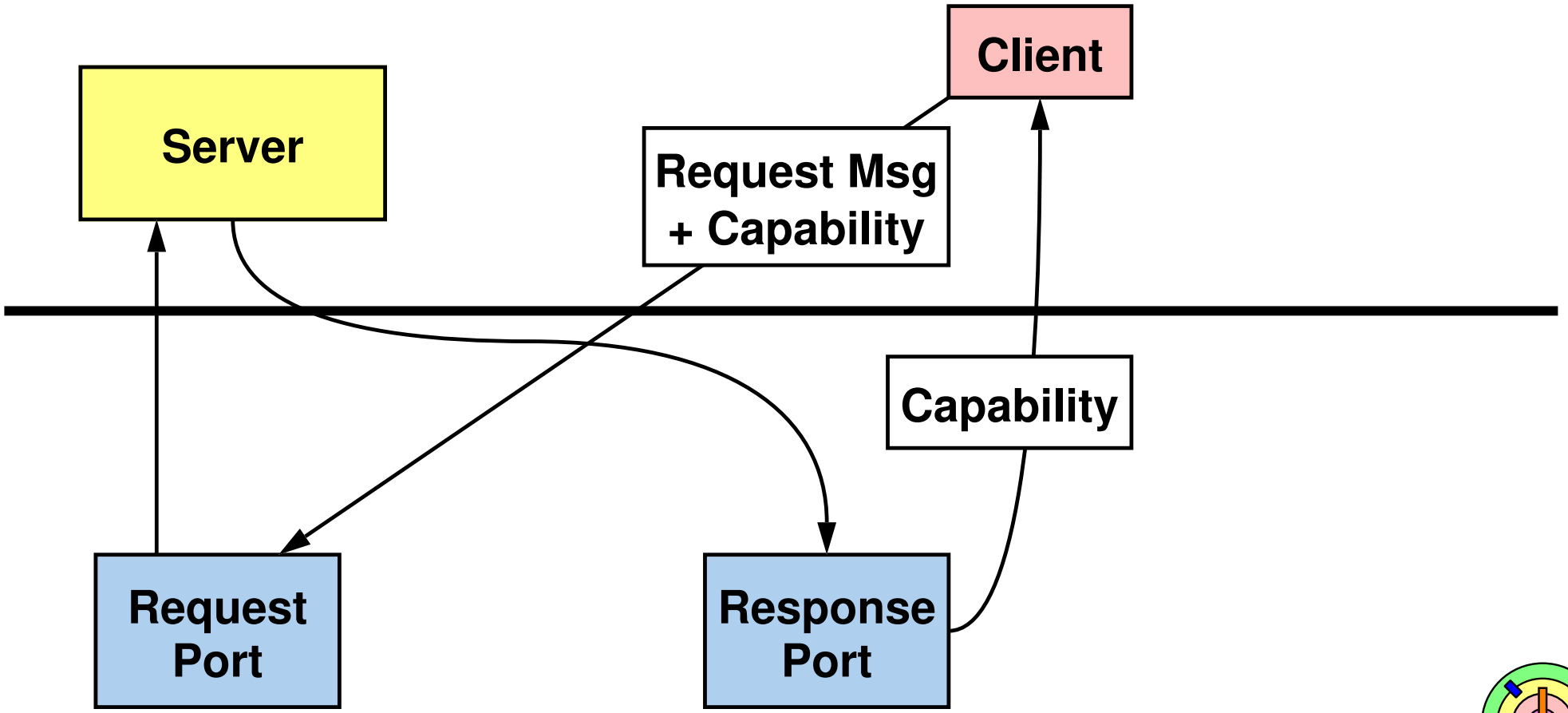
Mach Ports Permissions

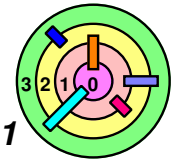
- Linkage construct



Mach Ports Permissions

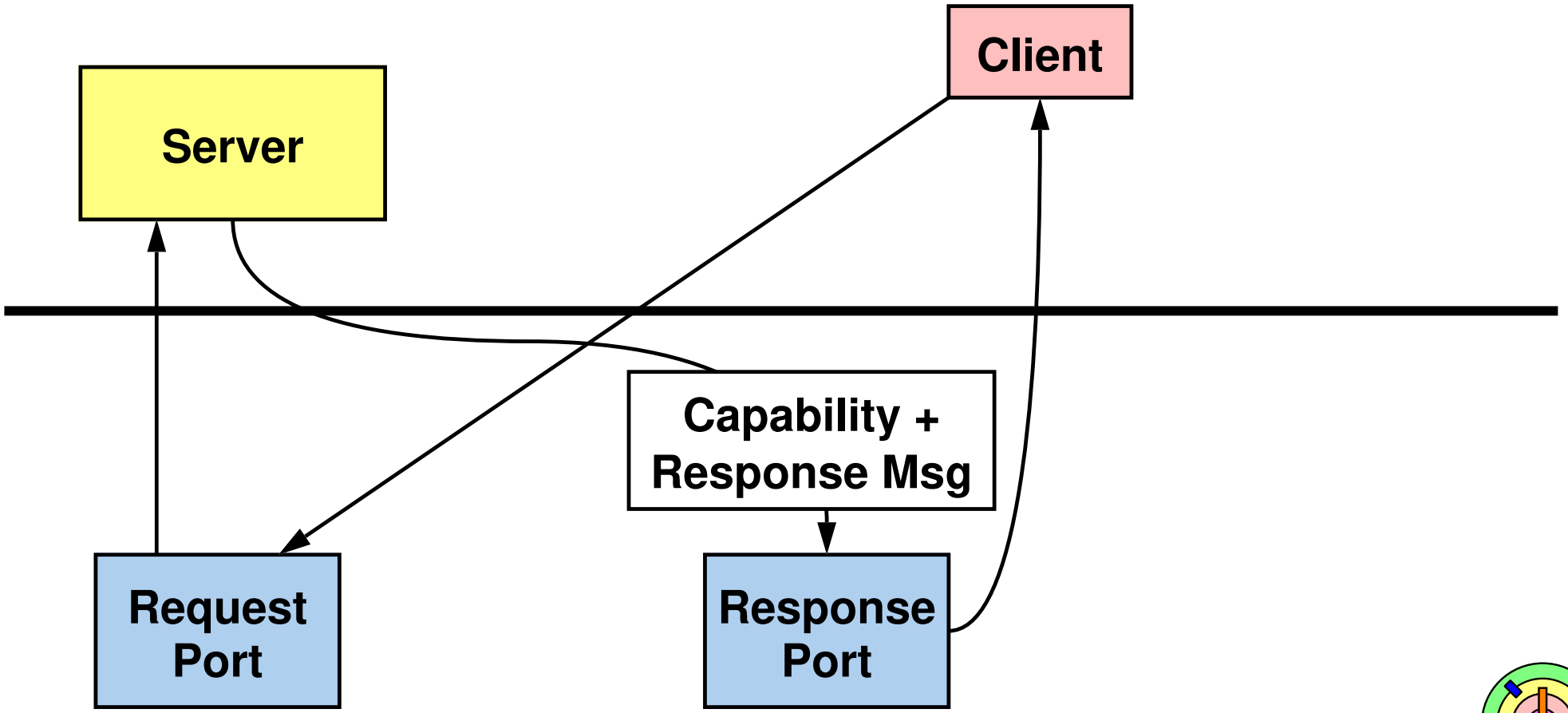
- Communication construct
  - client create *response port* and *capability* (like a key) to send data through it
    - include capability in the request message to server





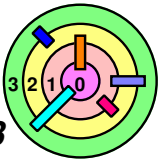
Mach Ports Permissions

- Communication construct
  - client create *response port* and *capability* (like a key) to send data through it
    - include capability in the request message to server



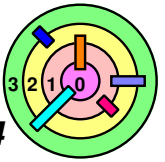
RPC

- Ports used to implement *remote procedure calls*
  - communication across process boundaries
  - if procedures are on same machine ...
    - local RPC



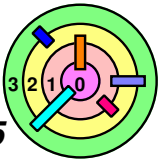
Successful Microkernel Systems

- ?
- ?
- ...



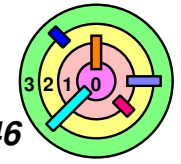
Attempts

- Windows NT 3.1
  - graphics subsystem ran as user-level process
  - moved to kernel in 4.0 for performance reasons
- Mac OS X
  - based on Mach
  - all services in kernel for performance reasons
- HURD
  - based on Mach
  - services implemented as user processes
  - no one uses it, for performance reasons ...



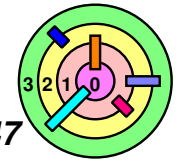
5.3 Scheduling

- *Goals*
- Scheduling Algorithms
- Implementation Issues
- Case Studies



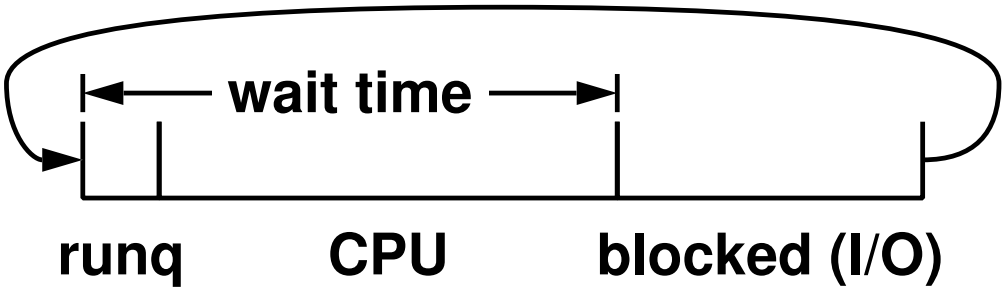
Sample Sorts of Systems

- Simple batch
- Multiprogrammed batch
- Time sharing (i.e., interactive)
- Partitioned servers
- General purpose
- Real time *(i.e., thread must run before a deadline)*
  - *hard real time* (control) vs. *soft real time* (audio/video)
    - for hard real time system, missing deadline means *disaster*
      - e.g., controlling a nuclear power plant, landing (softly) on Mars
      - usually need specialized OS
    - for soft real time system, missing deadline *degrades quality* and user experience
      - e.g., playing streaming audio or video
      - can be supported by general purpose OS



Scheduling

- Goals (some are in conflict with one another)
  - *maximize CPU utilization*
  - *maximize throughput (jobs/sec)*



    - a thread sometimes can be (incorrectly) classified as either *CPU-bound* or *I/O-bound*
  - *minimize wait time* (time till job *voluntarily* gives up the CPU)
  - *minimize response time* (for interactive and real time applications) - can be difficult to define
  - *fairness* - can be difficult to define, but we will try
    - is it fair to approximately assign equal proportion of CPU to threads

- Scheduling is very important because its effects and how well it achieves the above goals can be *felt* by the user