*13.02.2014*

# User-level scheduling

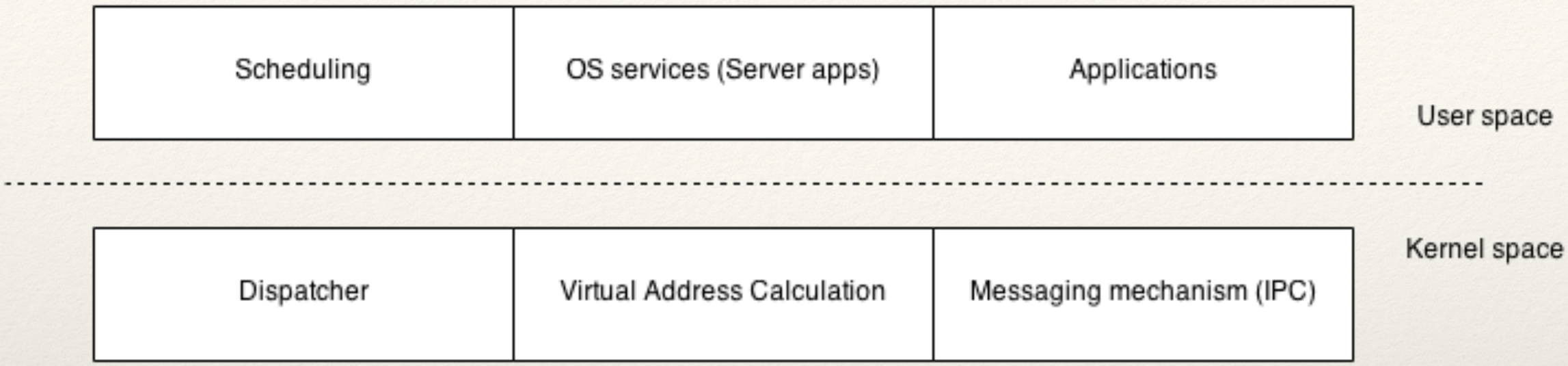
Andreas Zoor &
Nikolai Nagibin

# What's User-level scheduling

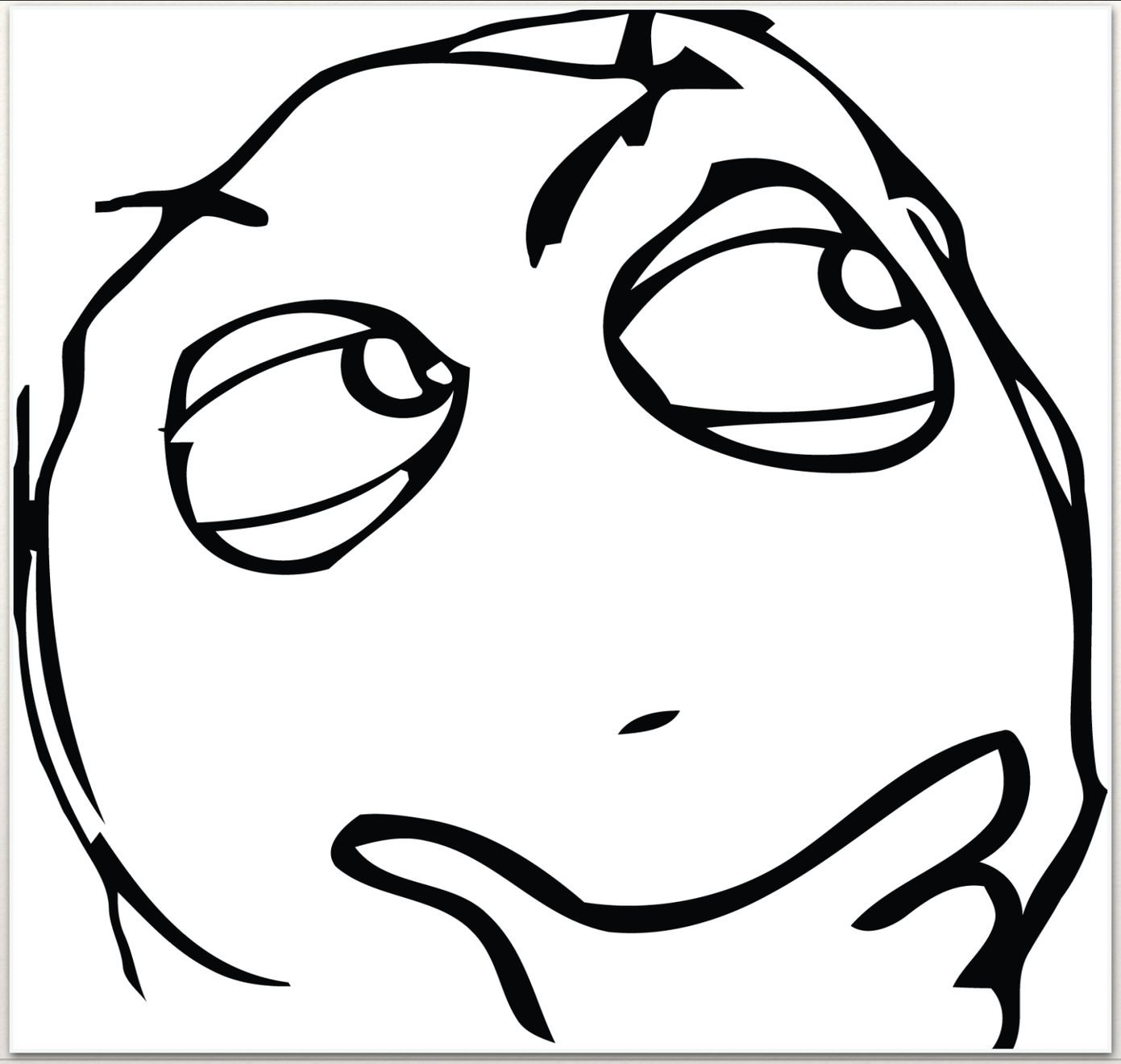
- Export scheduling policies form the kernel in the User-level
- Based on the idea of microkernel based operating systems

# Microkernel based operating systems

- Minimize the kernel part of the operating system
- For more modularity, flexibility and small ”Trusted Computing Base“
- Just include scheduling mechanism, address calculation and a messaging service (IPC)
- All resource-management policies have to be implemented at user-level as server applications



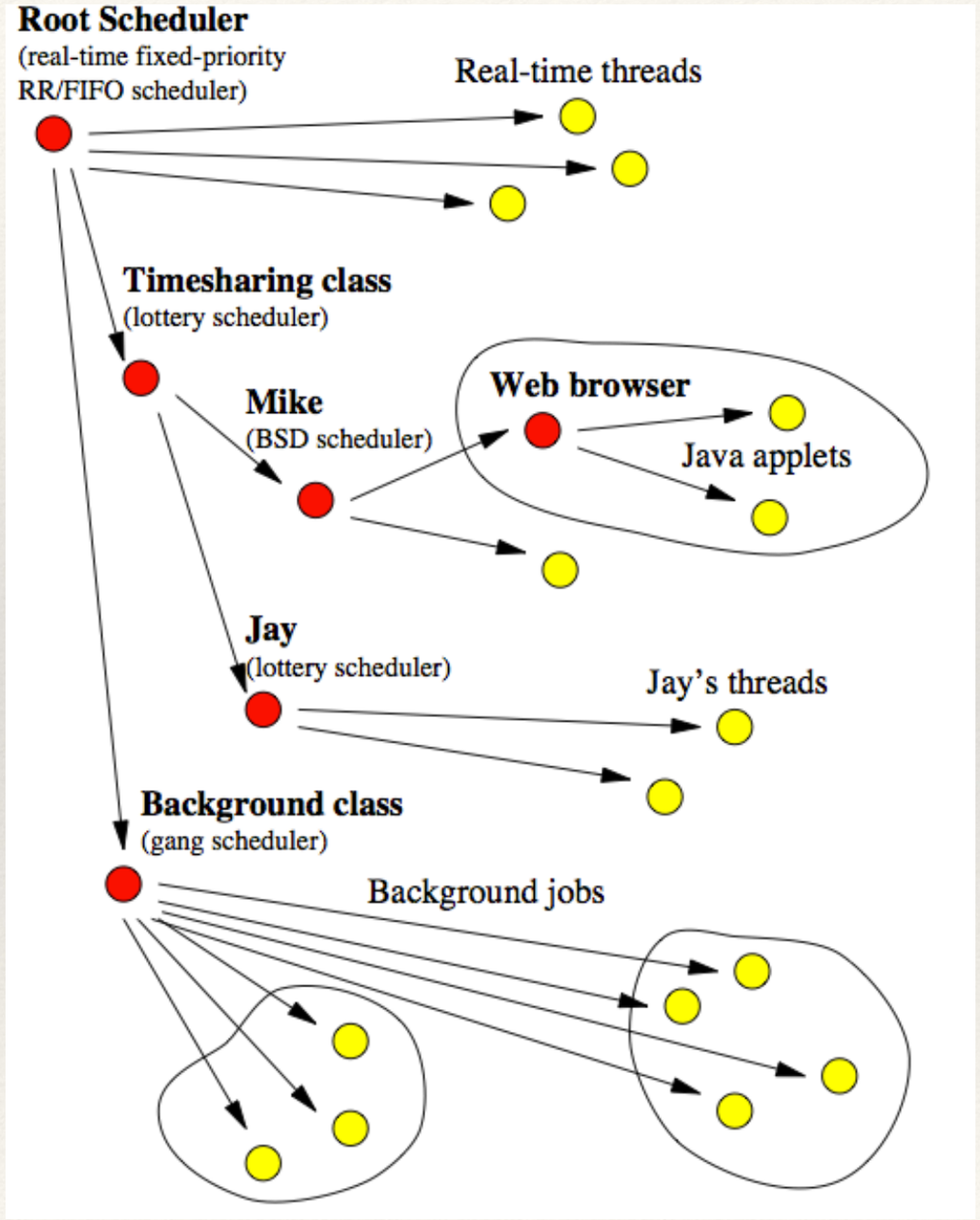

# Idea of user-level scheduling



# Why using user-level scheduling ?

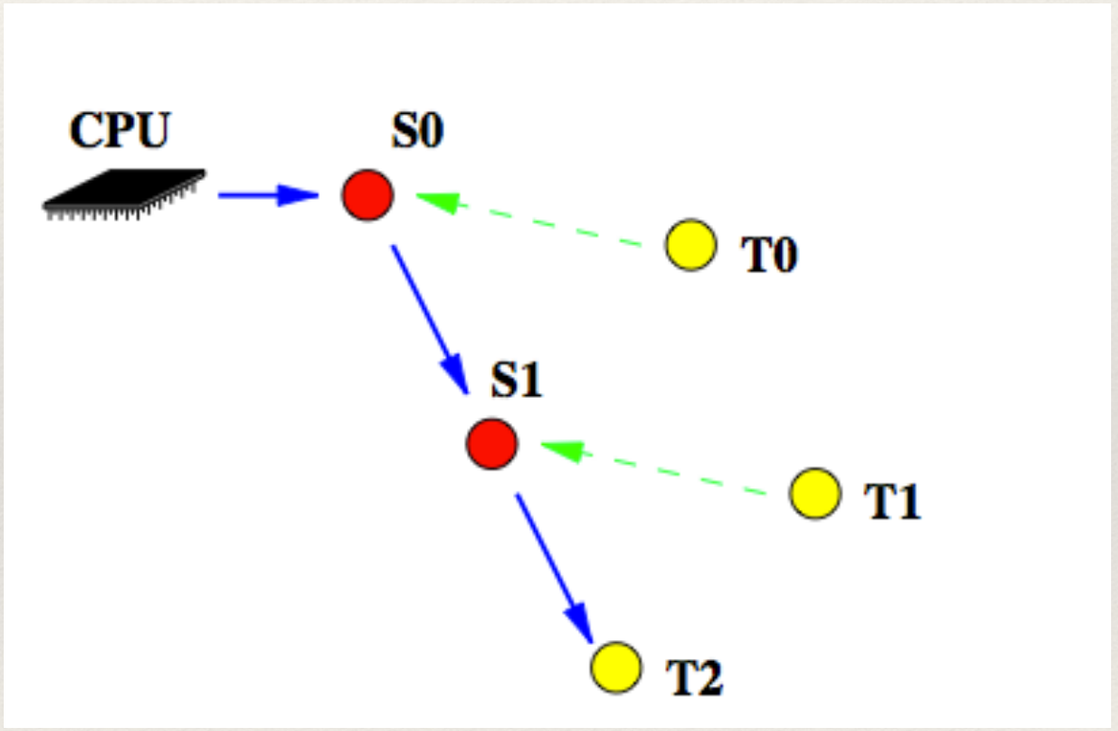
# CPU Inheritance Scheduling

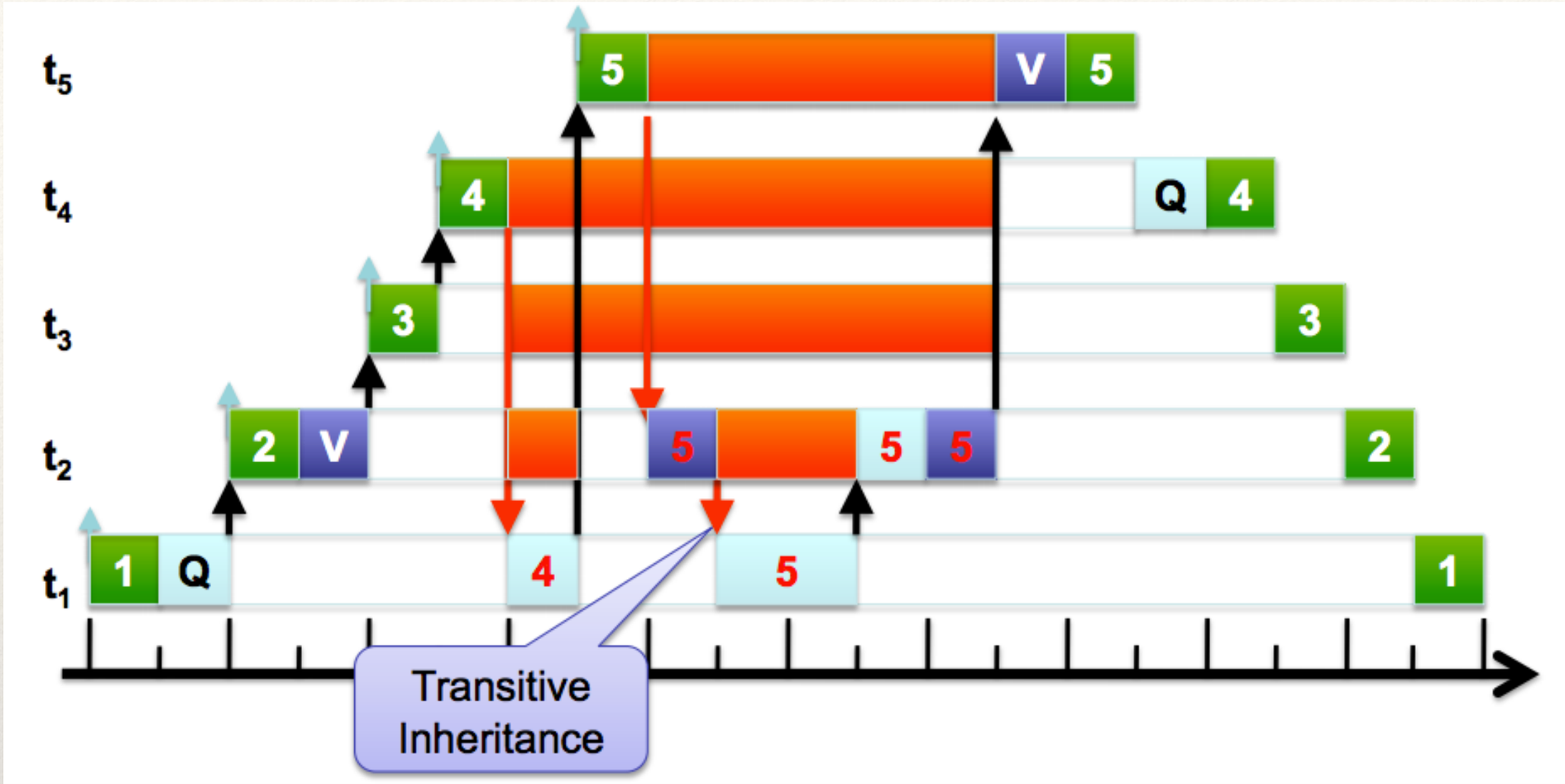
- Traditionally scheduling is on low level: by kernel scheduler or by user-level thread packages
- New approach: higher-level threads donate CPU+resources to others
- "Inheritance": ability to donate and request (virtual) CPU time between threads
- Client threads can act as scheduler threads for others
- Root scheduler owns real CPU time



# CPU Usage Accounting

- Statistical accounting
- Time stamp-based accounting
- Directly implement by root schedulers
- Virtual time information for clients of other schedulers

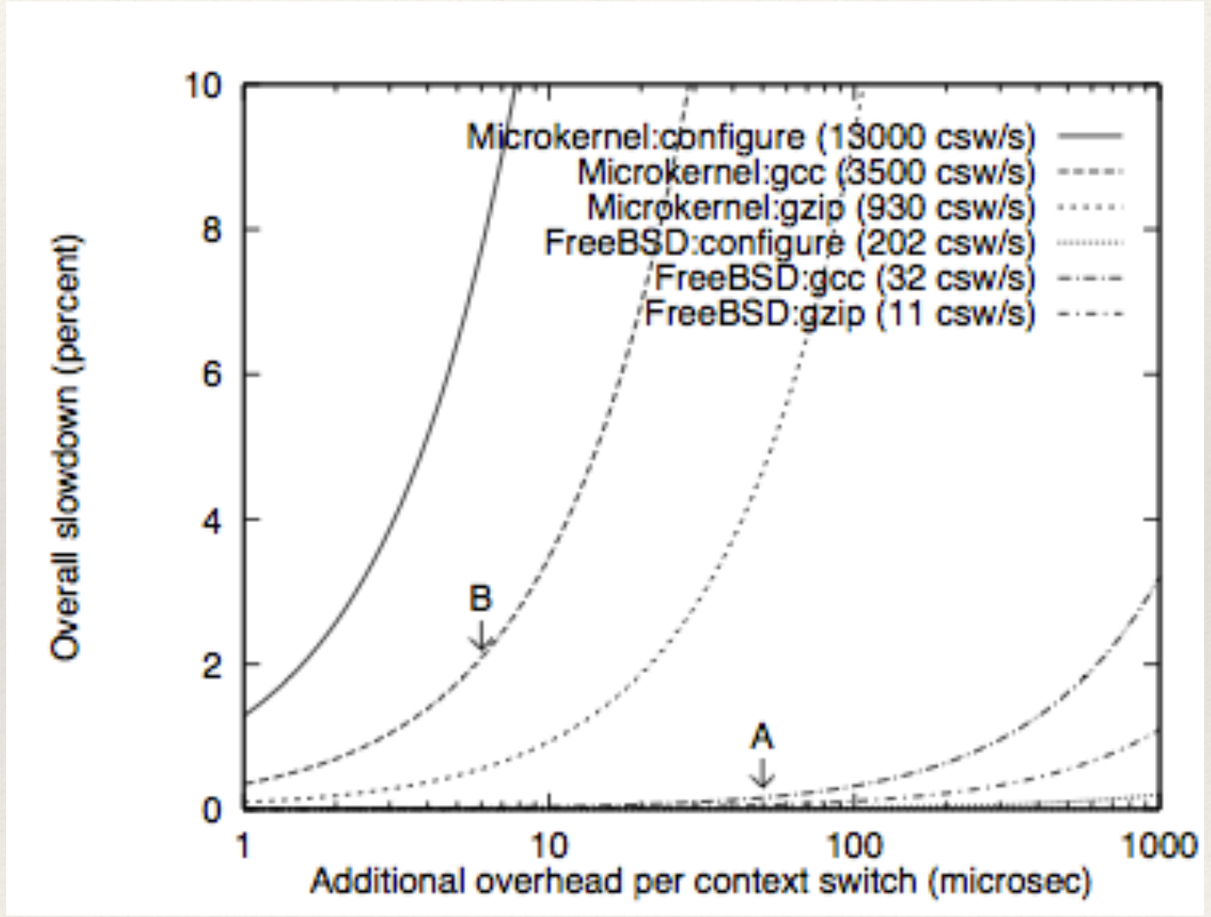


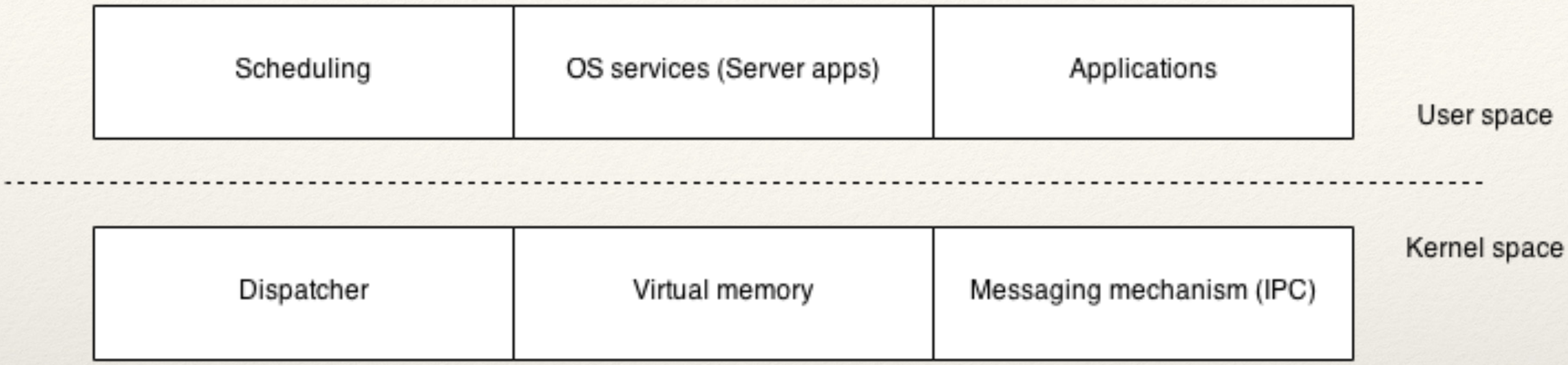



# Priority inversion

# Overhead

- Is it efficient enough in practice?
- Two additional sources of overhead
  - Caused by dispatcher
  - and add. context switches






# View: Exokernel

# Conclusion

- CPU inheritance scheduling has low overhead
  - All threads in one address space
  - A threads has access to memory of all applications
- Is it possible to use the concept of CPU inheritance scheduling for more address spaces?

# That's all Folks

- Thanks for your Attention!