CSE 451 Winter 2013

Section 4

Anton Osobov
aosobov@cs

Some material adapted from previous offerings of CSE 451

Reminders

- Quiz tomorrow (2/1)
- Project 3 is up
  - Due Wednesday, 2/20
  - Group project

Topics for Today

- Project 3
- Processes and Threads

Project 3

- Project groups assigned
  - Each group identified by a letter
  - Project directory at
    - /projects/instr/13wi/cse451/<group letter>
  - You can use this space to set up an SVN repository
  - Can also use online version control as long as it is private
    - GitHub, Bitbucket, etc.

Project 3

- Implement a file-copy utility
  - This is done entirely in user space!
- Three parts
  - Multithreaded + synchronous I/O
  - Single threaded + asynchronous I/O
  - Performance analysis of these two implementations

I/O in Windows

- Advantages of sync I/O?

I/O in Windows

- Advantages of sync I/O?
  - Easier to program
    - Don't have to explicitly synchronize with I/O driver
- Advantages of async I/O?

I/O in Windows

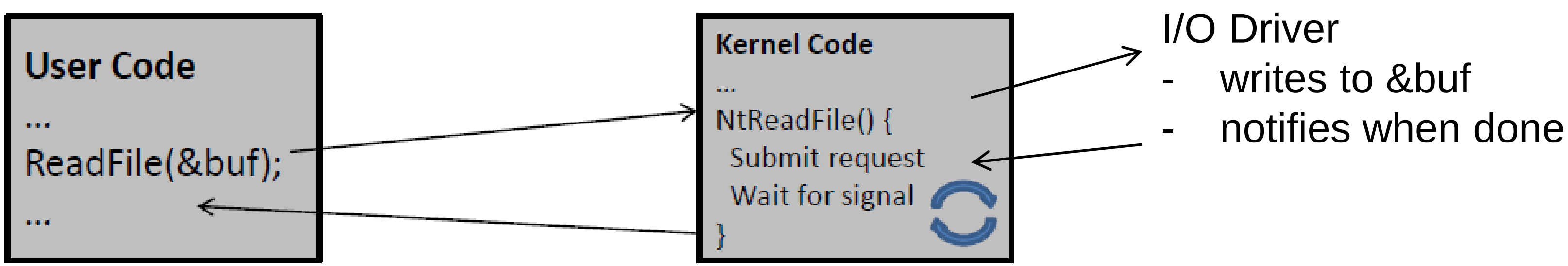
- Advantages of sync I/O?
  - Easier to program
    - Don’t have to explicitly synchronize with I/O driver
- Advantages of async I/O?
  - Potentially more efficient
    - You can overlap work with the I/O request
- How can we make sync I/O go faster?

I/O in Windows

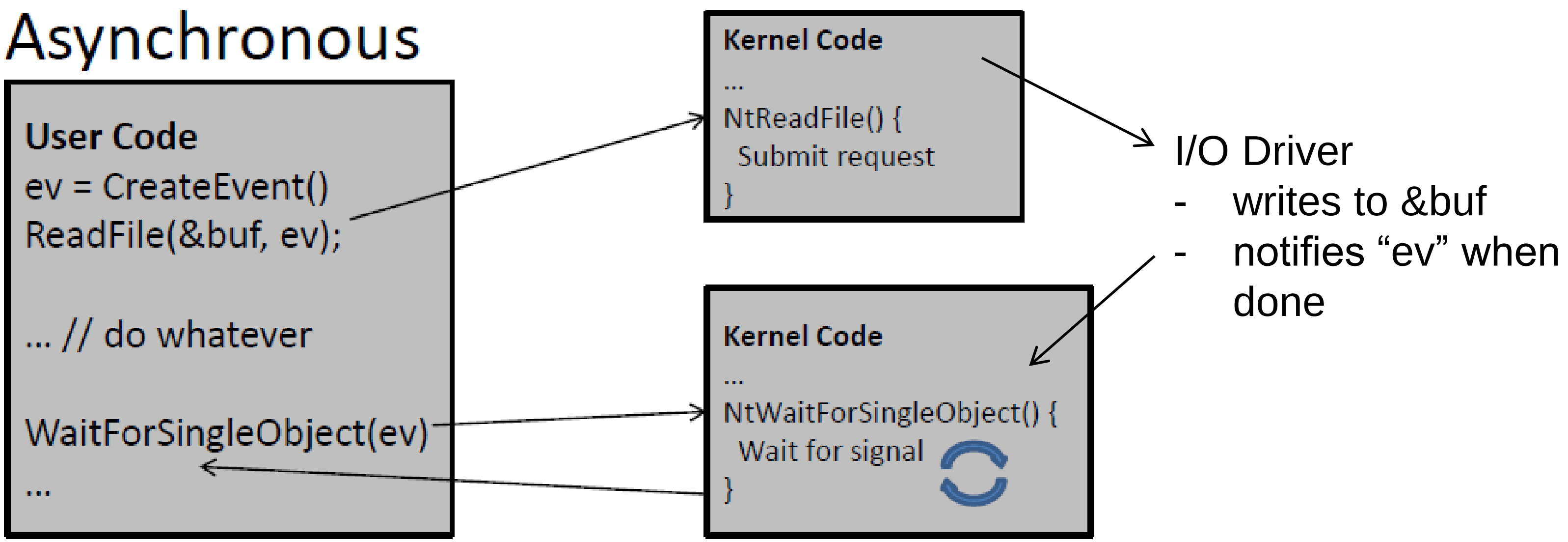
- Advantages of sync I/O?
  - Easier to program
    - Don’t have to explicitly synchronize with I/O driver
- Advantages of async I/O?
  - Potentially more efficient
    - You can overlap work with the I/O request
- How can we make sync I/O go faster?
  - Use more threads!

I/O in Windows

- Synchronous

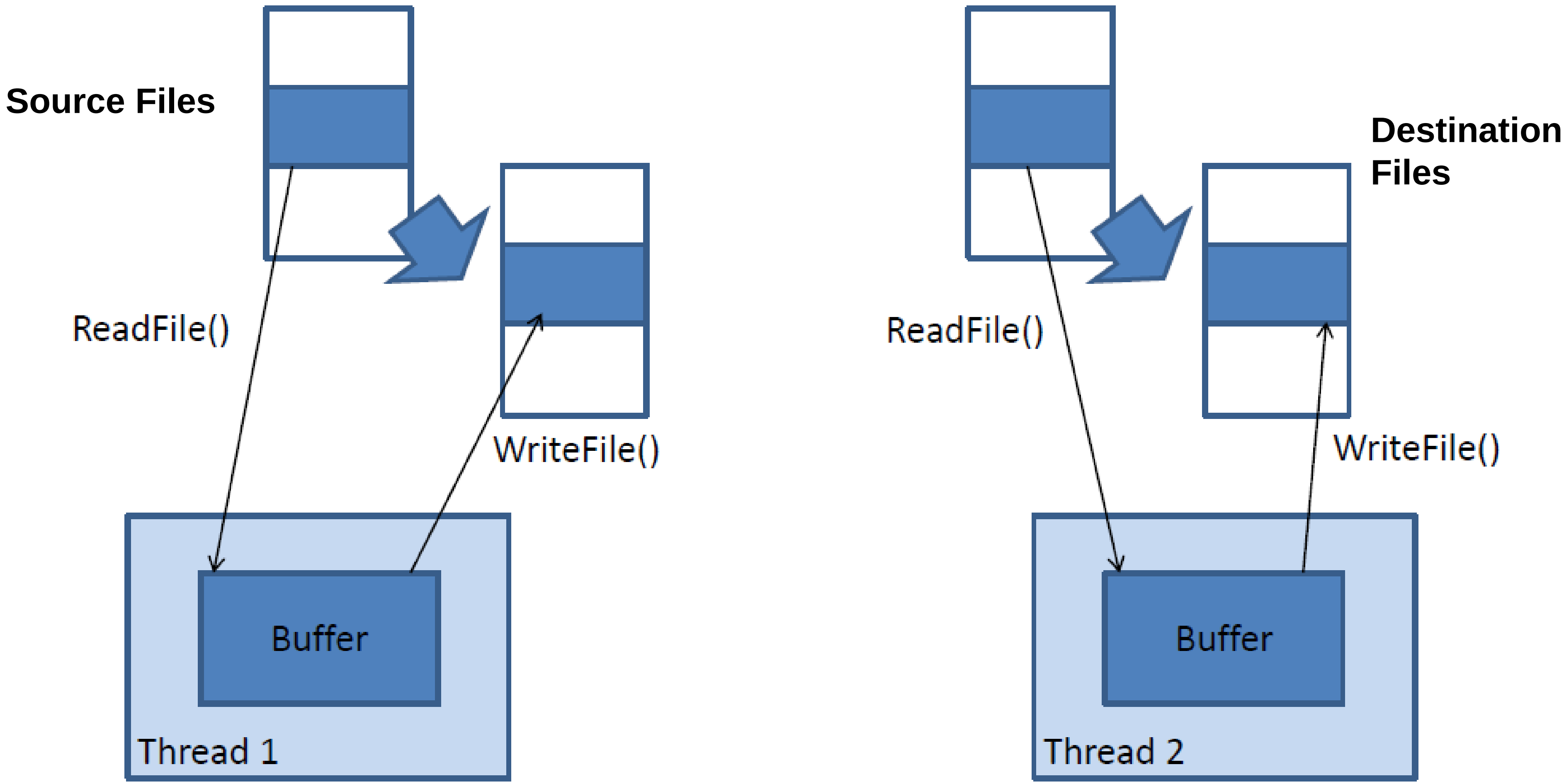


- Asynchronous



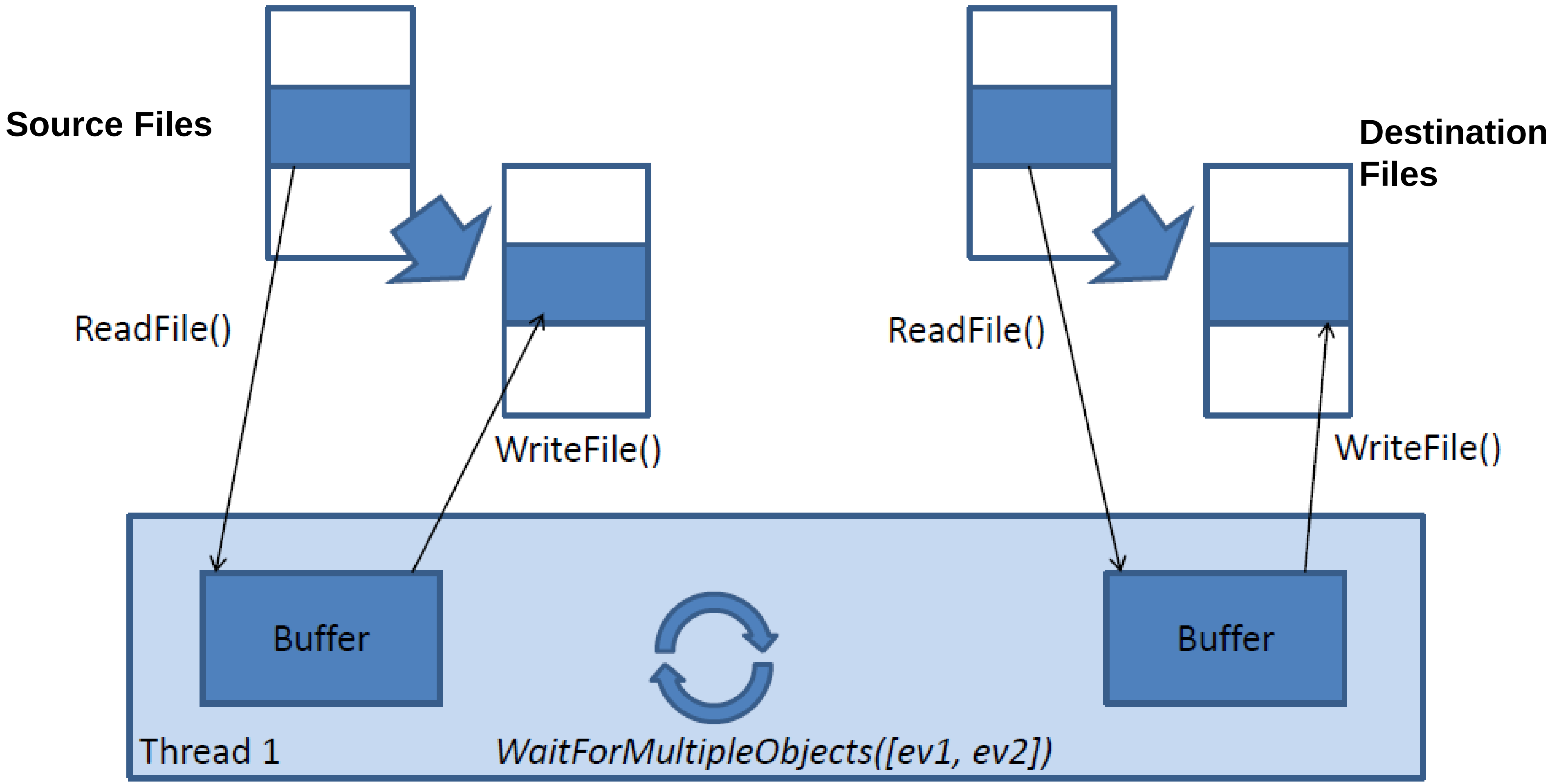
Project 3

Multithreaded + Sync I/O



Project 3

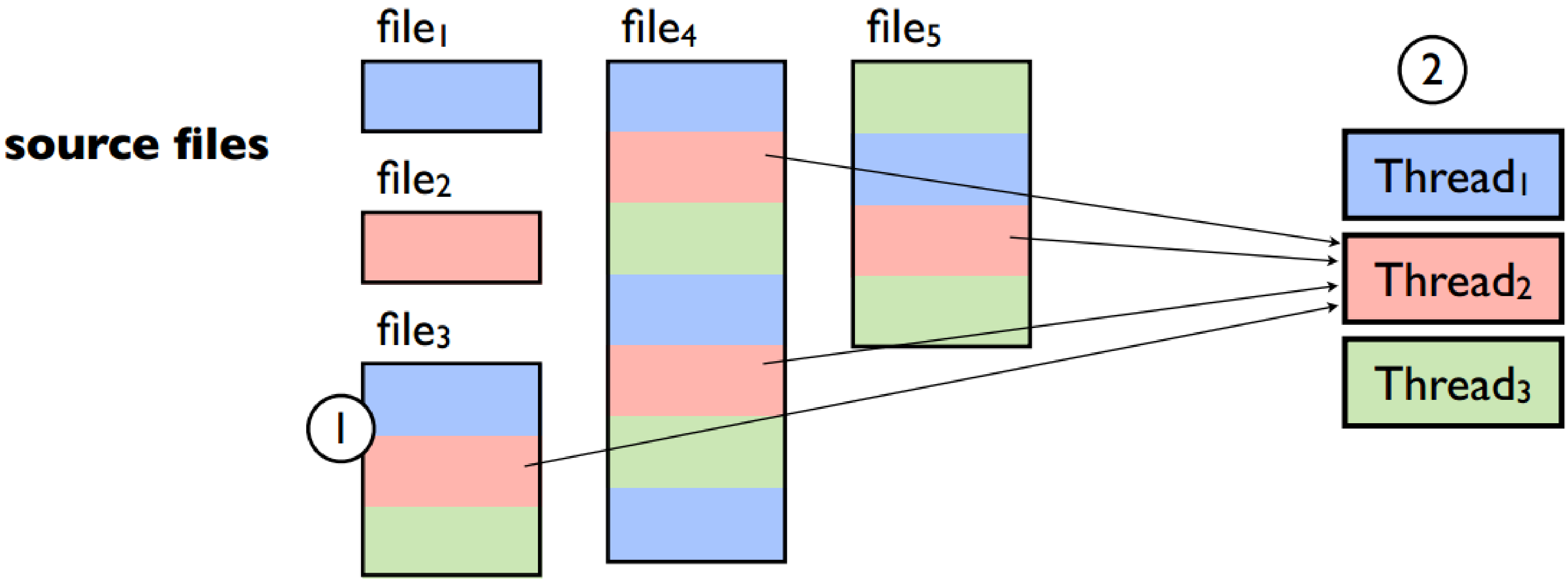
Single Threaded + Async I/O



Project 3

- Three parts
  - Implement MtFileCopy (multithreaded)
  - Implement MtFileCopyAsync (single threaded)
  - Performance analysis

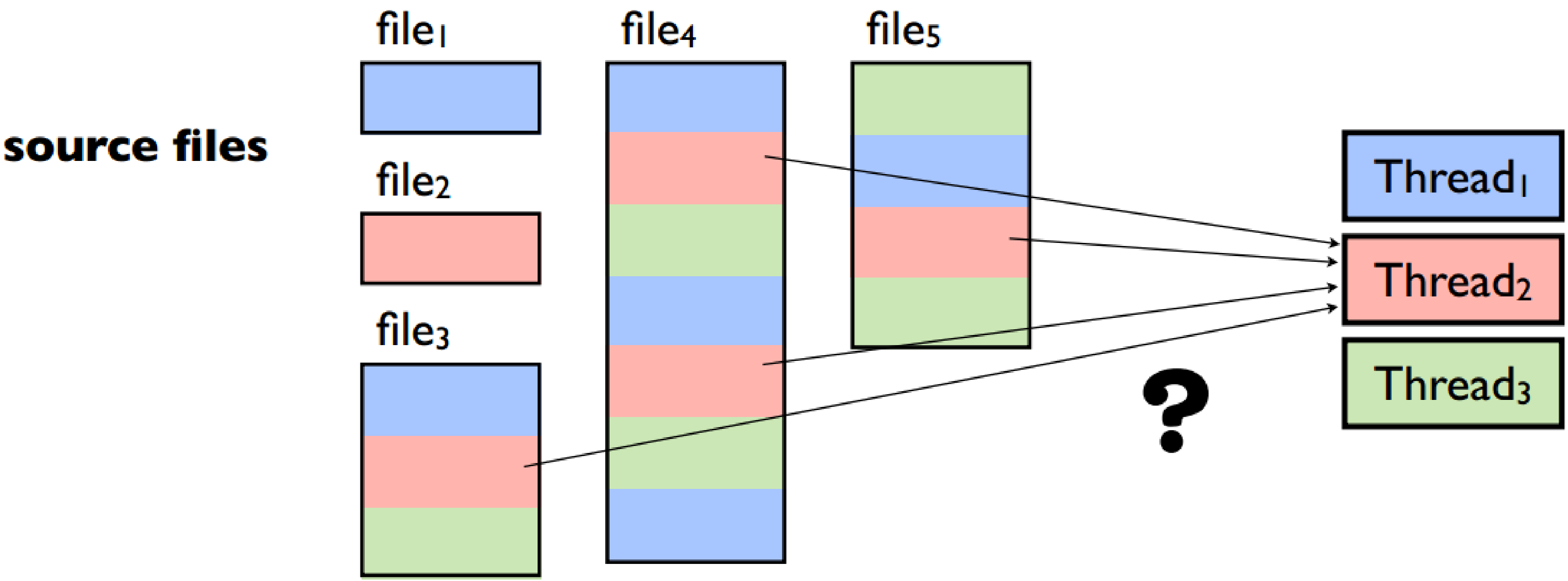
Project 3



`MtFileCopy( ThreadCount=3, BufferSize=4096, files .. )`

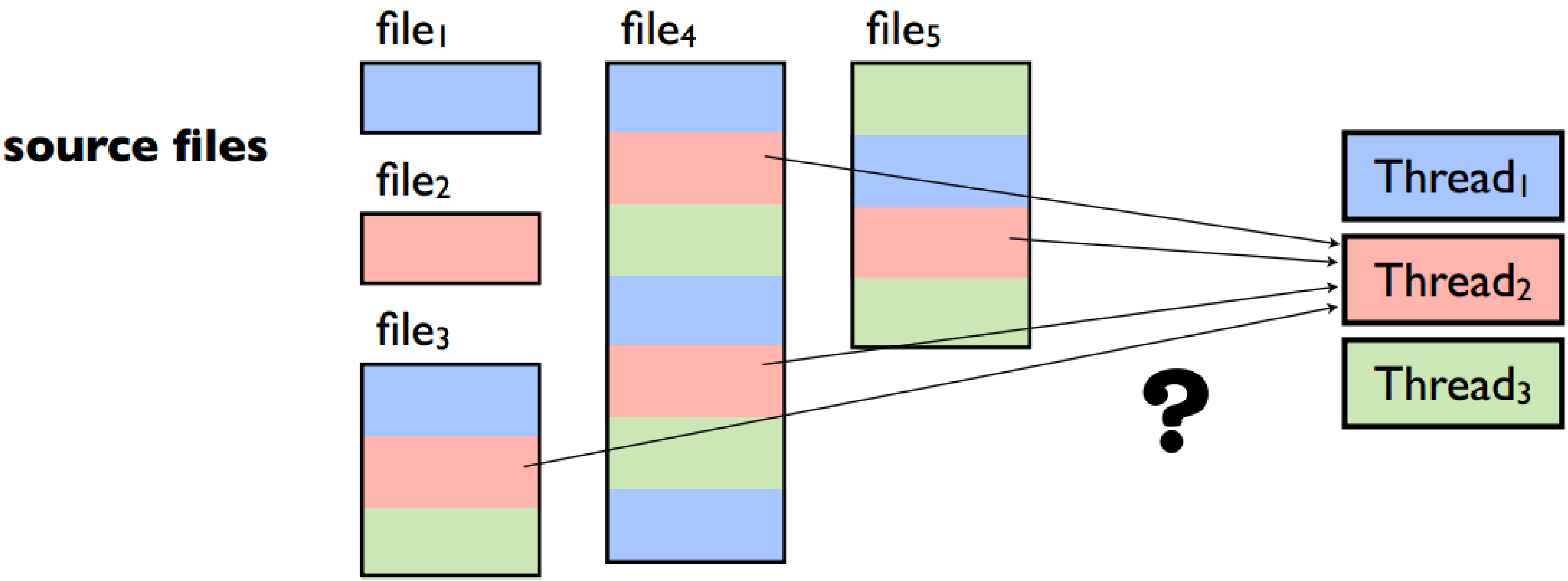
1. Break files into chunks of work (use chunkSize == ?)
2. Schedule chunks to threads (each thread copies one chunk at a time)

Project 3



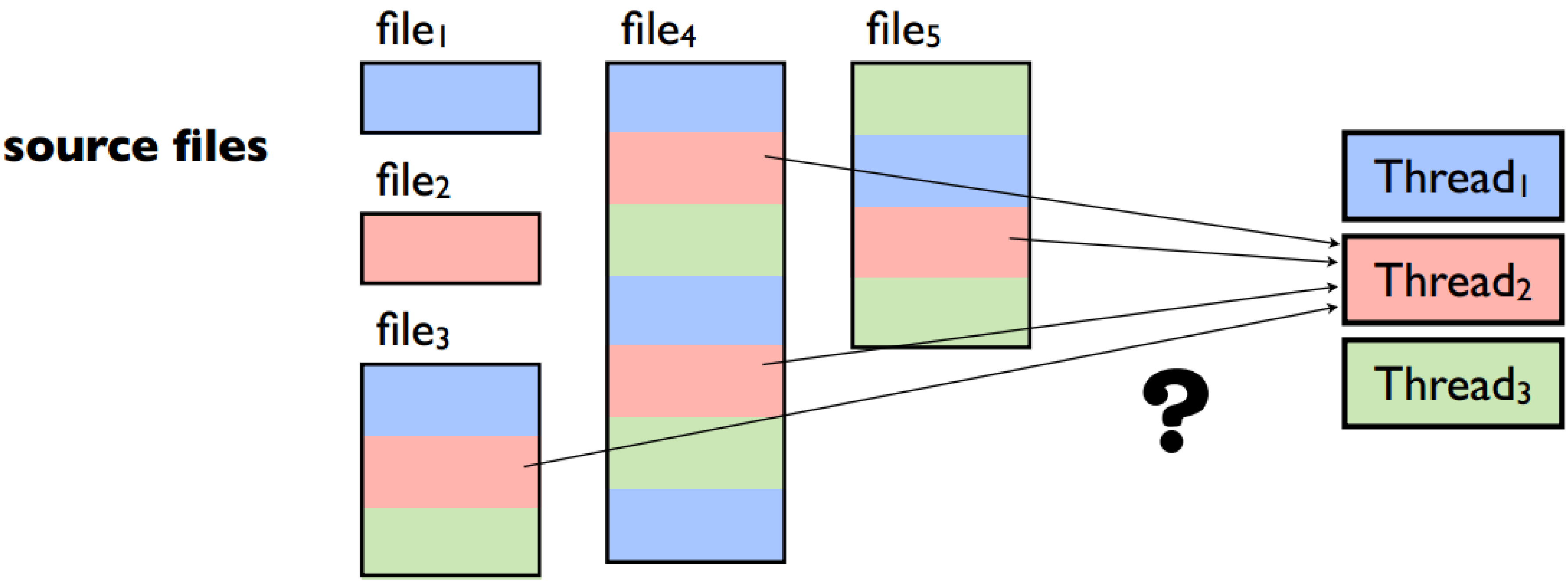
- What goes into efficient schedule?

Project 3



- What goes into efficient schedule?
  - Load balancing (keep threads busy)
  - Locality
    - Assign threads to different files?
    - Have threads team up on the same file?

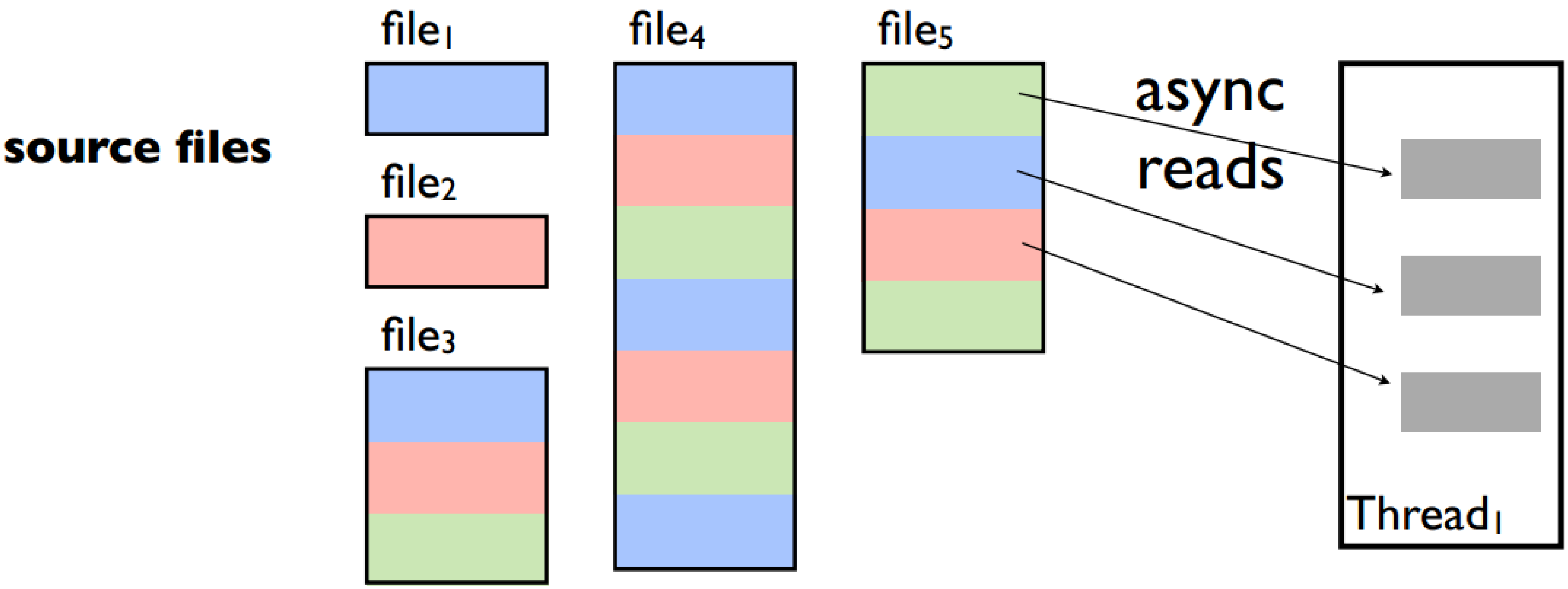
Project 3



- Scheduling approaches
  - Build a schedule up-front (doesn't respond well to performance glitches?)
  - Put chunks in a FIFO queue

Project 3

Async version



`MtFileCopyAsync( BufferCount=3, BufferSize=4096, files .. )`

- Same idea. Except
  - You have just one thread
  - That thread does 3 asynchronous chunk copies at once

Project 3

What experiments could I run?

- Use diverse input sets

Big files	many files	using network drives
Small files	few files	using local hard drives
		using USB drives

- Time your program on each input set
  - Use different values for /T and /B
  - Use sync and async
- Analyze
  - What is the best configuration
  - What is the worst configuration
  - Make graphs

Topics for Today

- ~~Project 3~~
- Processes and Threads

Processes

Recap from lecture

- What is a process?

Processes

Recap from lecture

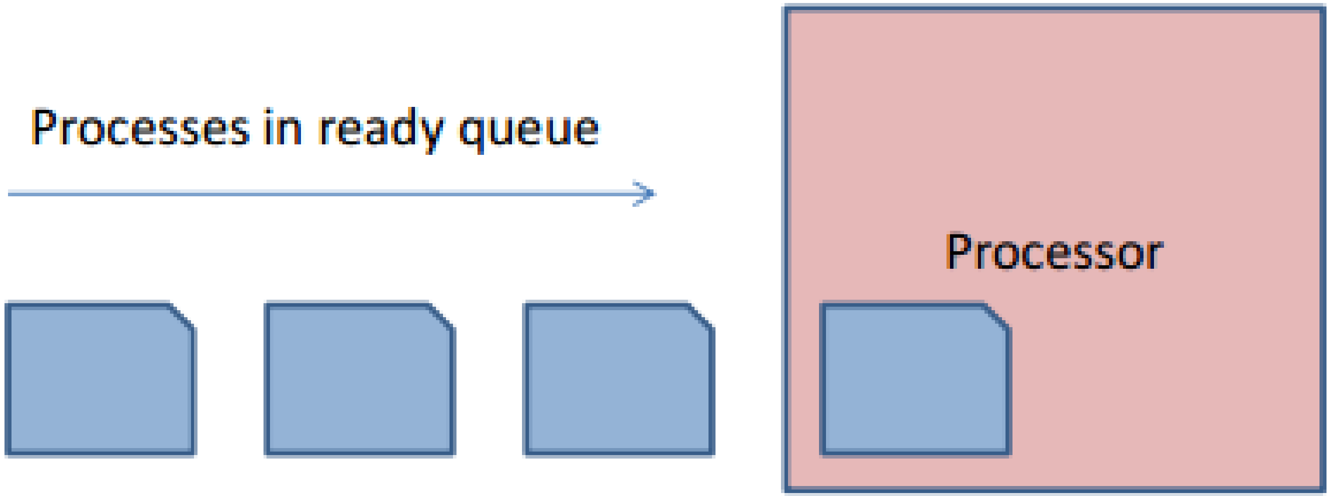
- What is a process?
  - An execution entity
  - A running instance of a program
  - Has at least
    - An address space
    - The code for the running program
    - The data for the running program
    - At least one thread
    - A set of OS resources

Processes

- How does an OS on single processor hardware run multiple processes?

Processes

- How does an OS on single processor hardware run multiple processes?



Processes

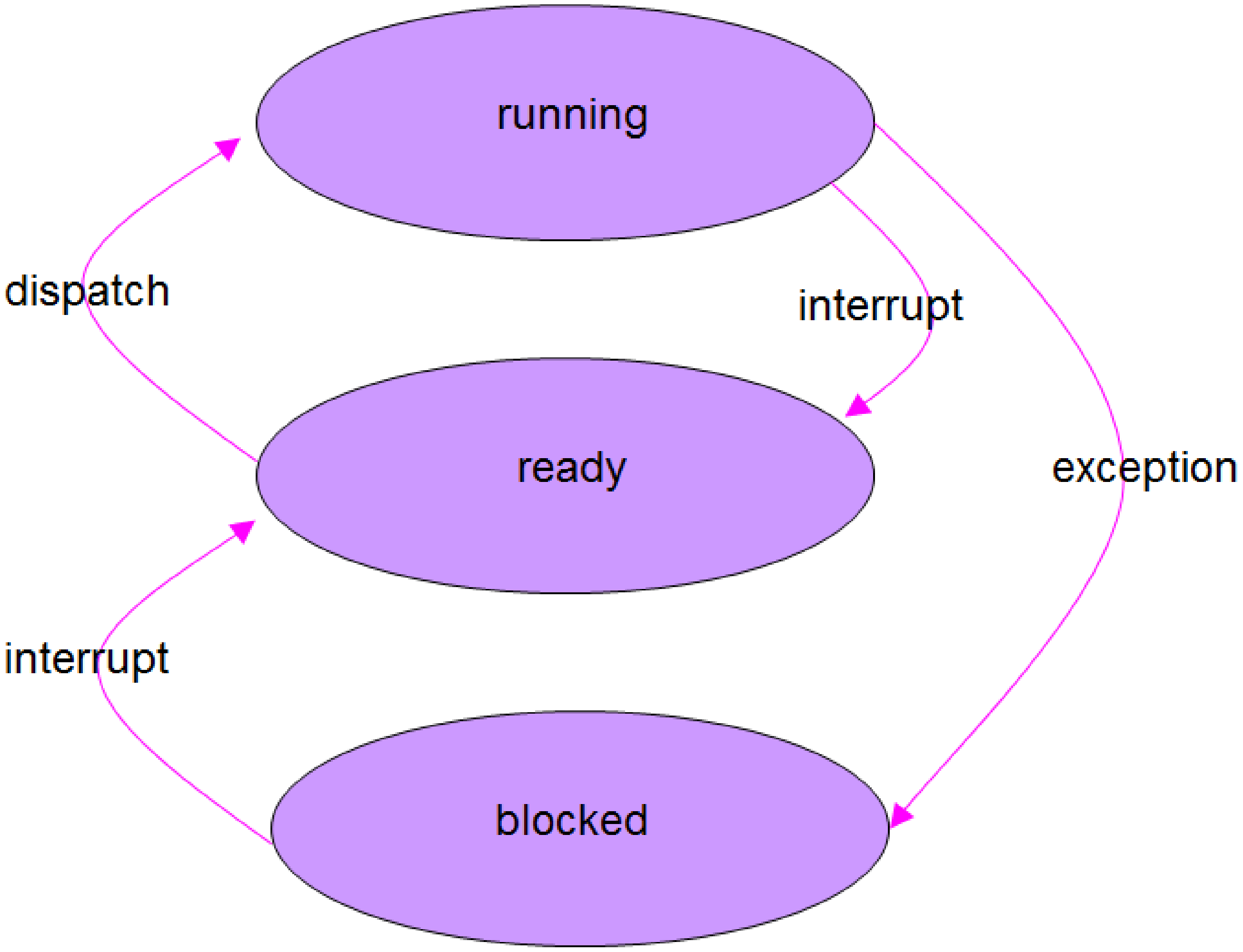
- What does the OS do when there are no processes to run?

Processes

- What does the OS do when there are no processes to run?
  - Run an idle process
    - Periodically checks for any new tasks to run
    - Loops the HLT instruction to save CPU time

Threads

States of a thread



Threads

- Why use threads?

Threads

- Why use threads?
  - Perform multiple tasks at once (reading and writing, computing and receiving input)
  - Take advantage of multiple CPUs
  - More efficiently use resources

Threads v Processes

Overview

- Process
  - Isolated with its own virtual address space
  - Contains process data like file handles
  - Lots of overhead
  - Every process has at least one kernel thread
- Kernel Threads
  - Shared virtual address space
  - Contains running state data
  - Less overhead
  - From the OS's point of view, this is what is scheduled to run on a CPU
- User Threads
  - Shared virtual address space, contains running state data
  - Kernel unaware
  - Even less overhead

Threads v Processes

Trade-offs

- Process
  - Secure and isolated
  - Kernel aware
  - Creating a new process brings lots of overhead (address space)
- Kernel Threads
  - No need to create a new address space
  - No need to change address space in context switch
  - Kernel aware
  - Still need to enter kernel to context switch
- User Threads
  - No new address space, no need to change address space
  - No need to enter kernel to switch
  - Kernel is unaware. No multiprocessing. Synch I/O block all user threads

Threads v Processes

Implicit overheads

- Context switching between processes is very expensive because it changes the address space
  - But changing the address space is simply a register change in the CPU?
  - Requires flush the Translation Look-aside Buffer
- Context switching between threads has a similar overhead. Suddenly the cache will miss a lot.

Threads v Processes

Suppose that a programmer mistakenly creates a local variable v in one thread t1 and passes a pointer to v to another thread t2. Is it possible for a write by t1 to some variable other than v to change the state of v as observed by t2?