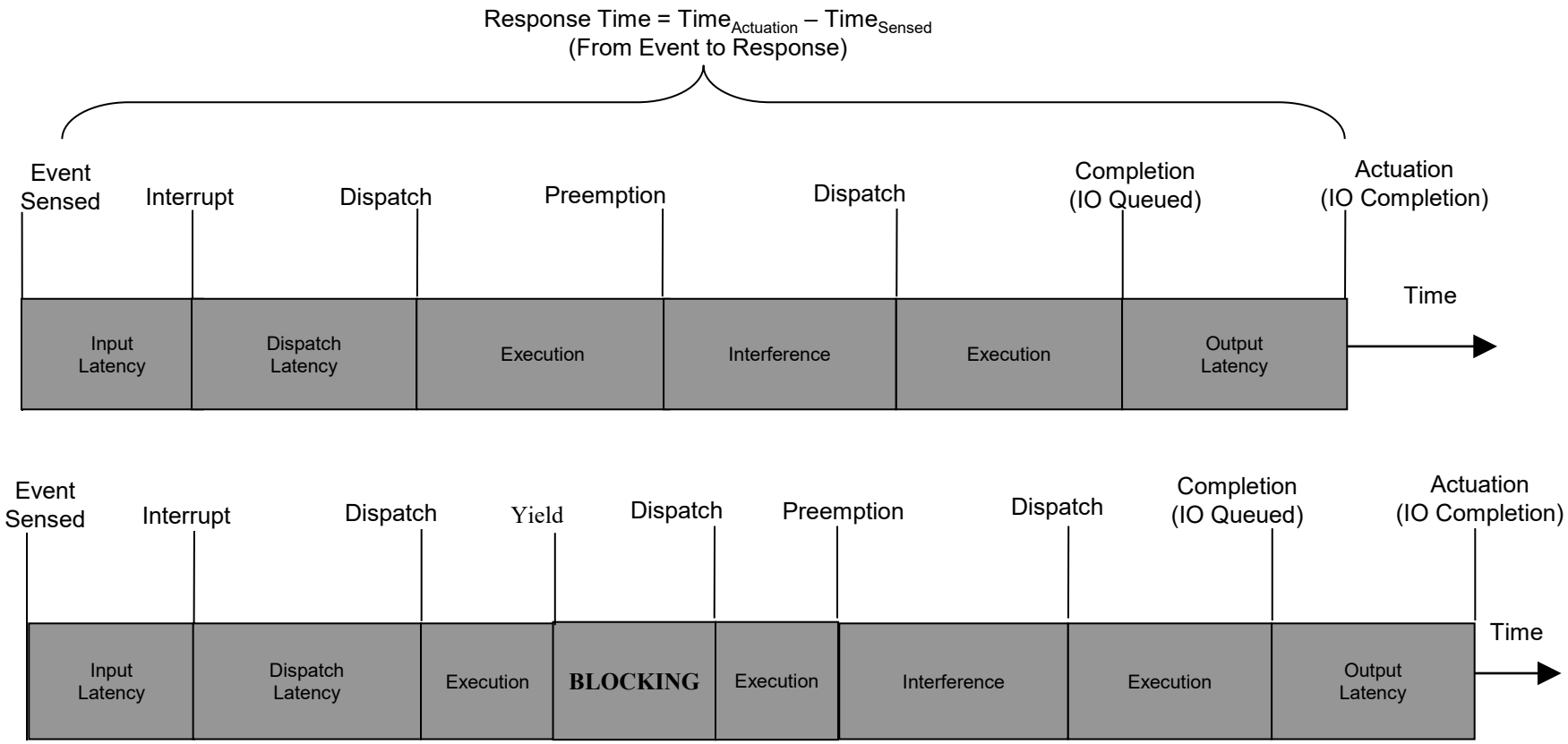
ECEN 5623

IO Services

Be Boulder.

University of Colorado Boulder

Service Response Timeline
(With Intermediate Blocking)

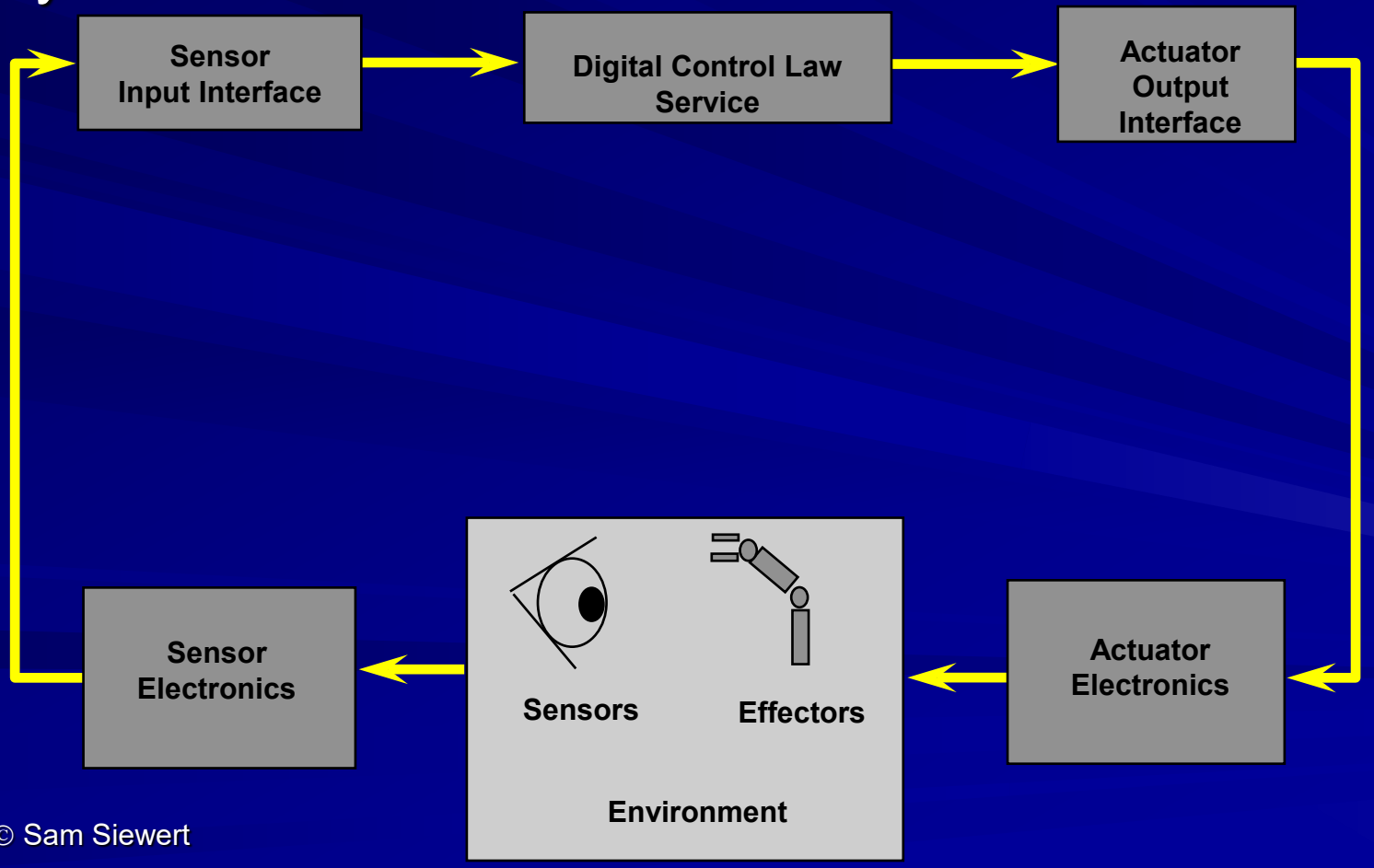


Services and I/O

- Initial Input I/O from Sensors
  - Low-Rate Input: Byte or Word Input from Memory-Mapped Registers or I/O Ports
  - High-Rate Input: Block Input from DMA Channels and Block Reads from FIFOs
- Final Response I/O to Actuators
  - Low-Rate Output: Byte or Word Writes Posted to Output FIFO or Bus Interface Queue
  - High-Rate Output: Block Output on DMA Channels and Block Writes
- Intermediate I/O
  - During Service Execution, Memory Mapped Register I/O
  - External Memory I/O
  - Cache Loads and Write-Backs
  - In Memory Input/Output from Service to Service

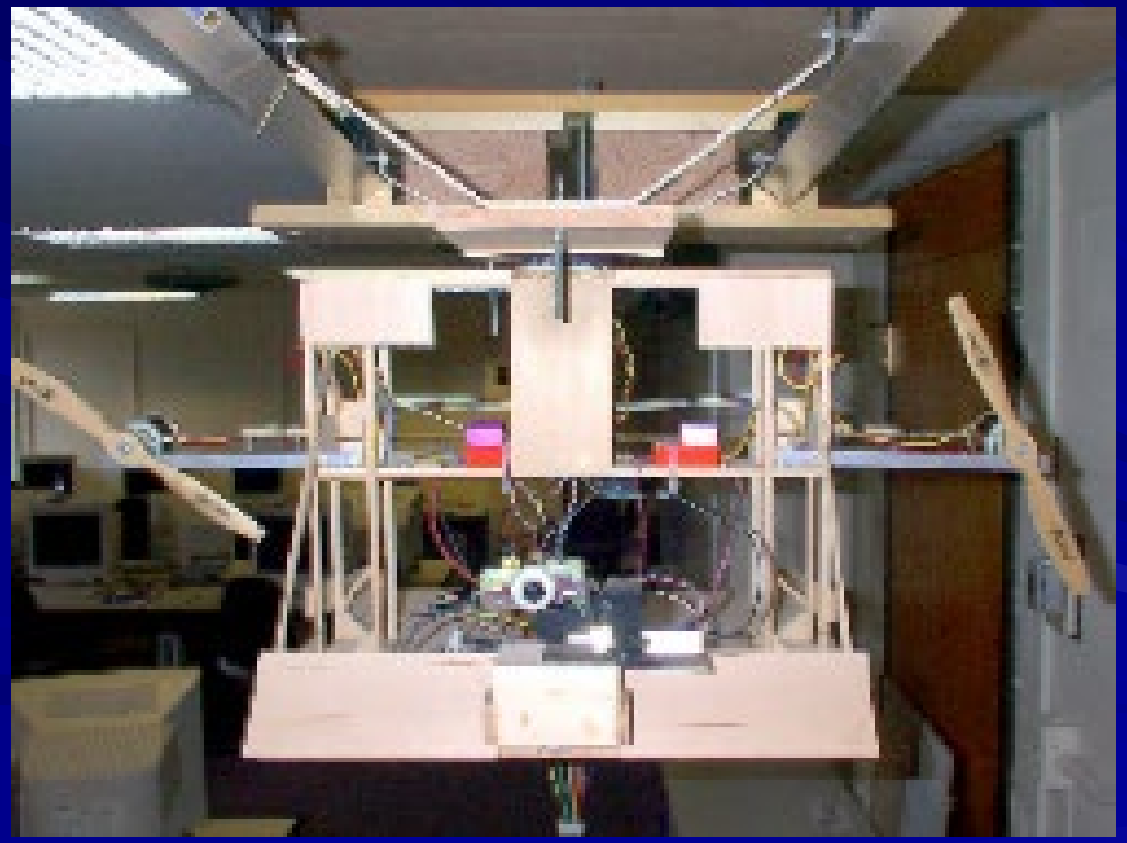
Service Integration Concepts

- Initial Sensor Input and Final Actuator Response Output
  - Device Interface Drivers
- Simple Service Has No Intermediate I/O Latency, No Synchronization

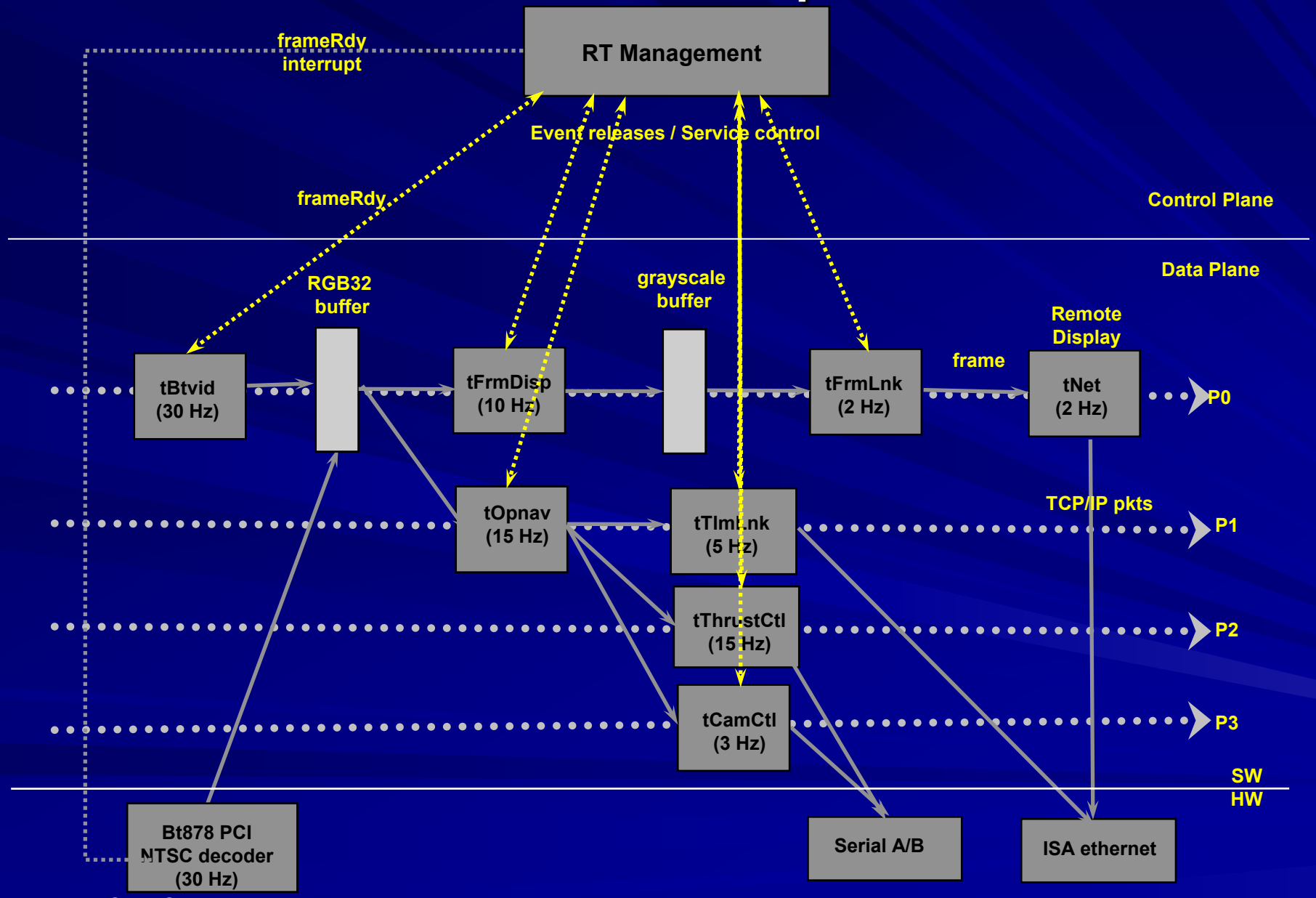


Complex Multi-Service Systems

- Multiple Services
- Synchronization Between Services
- Communication Between Services
- Multiple Sensor Input and Actuator Output Interfaces
- Intermediate I/O, Shared Memory, Messaging

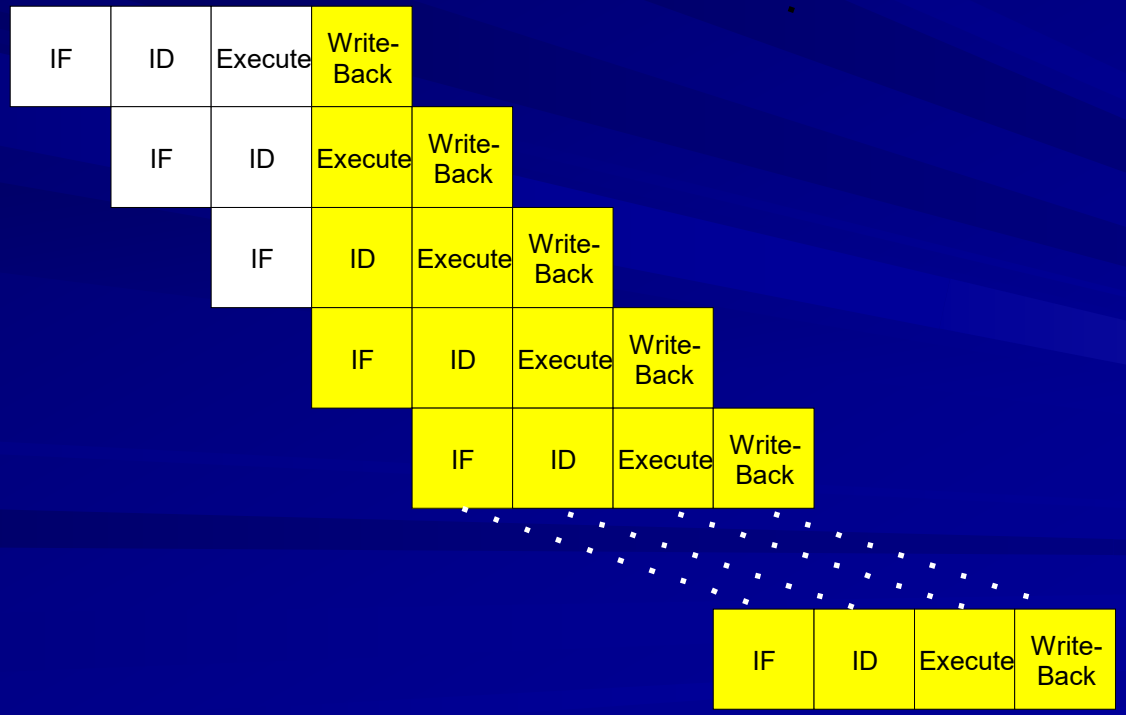


Multi-Service Pipelines



Pipelined Architecture Review

- Recall that Pipeline Yields CPI of 1 or Less
- Instruction Completed Each CPU Clock
- Unless Pipeline Stalls!



Service Execution

$$WCET = \left[(CPI_{best-case} \times Longest\_Path\_Inst\_Count) + Stall\_Cycles \right] \times Clk\_Period$$

- Efficiency of Execution
- Memory Access for Inter-Service Communication
- Bounded Intermediate I/O
- Ideally Lock Data and Code into Cache or Use Tightly Coupled Memory [Scratch Pad, Zero Wait State]

Service Efficiency

- Path Length for a Release
  - Instruction Count
  - Longest Path Given Algorithm
    - Branches
    - Loop Iterations
    - Data Driven?
- Path Execution
  - Number of Cycles to Complete Path
  - Clocks Per Instruction
  - Number of Stall Cycles for Pipeline
    - Data Dependencies (Intermediate IO)
    - Cache Misses (Intermediate Memory Access)
    - Branch Mis-predictions (Small Penalty)

Hiding Intermediate IO Latency (Overlapping CPU and Bus I/O)

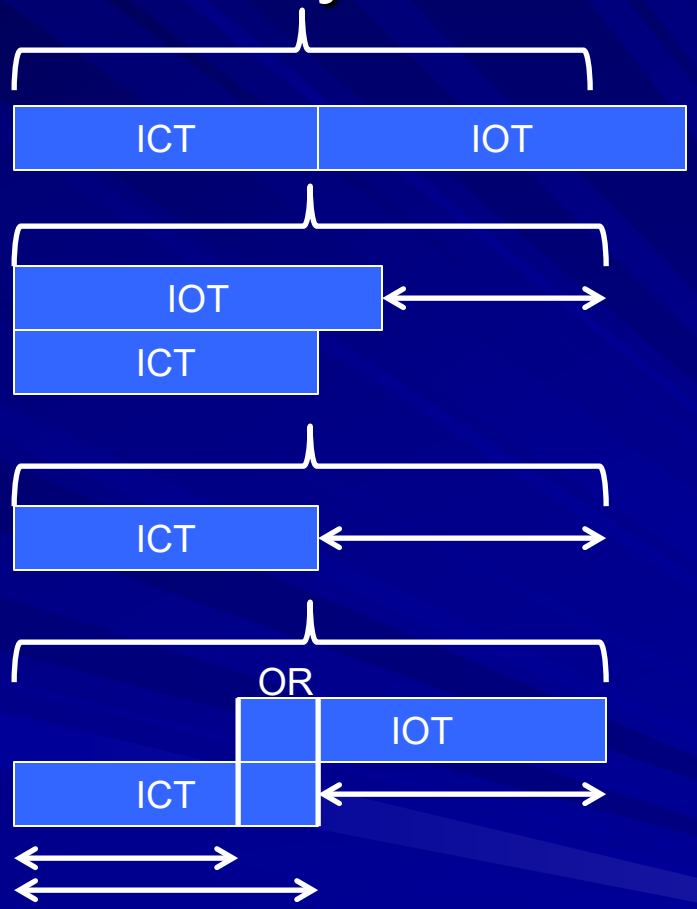
- ICT = Instruction Count Time
  - Time to execute a block of instructions with no stalls
  - CPU Cycles x CPU Clock Period
- IOT = Bus Interface IO Time
  - Bus IO Cycles x Bus Clock Period
- OR = Overlap Required
  - Percentage of CPU cycles that must be concurrent with I/O cycles
- NOA = Non-Overlap Allowable for Si to meet Di
  - Percentage of CPU cycles that can be in addition to IO cycle time without missing service deadline
- Di = Deadline for Service S_i relative to release
  - interrupt or system call initiates S_i request and S_i execution
- CPI = Clocks Per Instruction for a block of instructions
  - IPC is Instructions Per Clock, Also Used, Just the Inverse (Superscalar, Pipelined)

Processing and I/O Time Overlap

- Di >= IOT is required; otherwise if Di < IOT, Si is ***IO-Bound***
- Di >= ICT is required; otherwise if Di < ICT, Si is ***CPU-Bound***
- Di >= (IOT + ICT) **requires no overlap** of IOT with ICT
- if Di < (IOT + ICT) where (Di >= IOT and Di >= ICT), **overlap of IOT with ICT is required**
- if Di < (IOT + ICT) where (Di < IOT or Di < ICT), deadline **Di can't be met regardless of overlap**

Service Execution Efficiency – Waiting on I/O from MMIO Bus and Memory

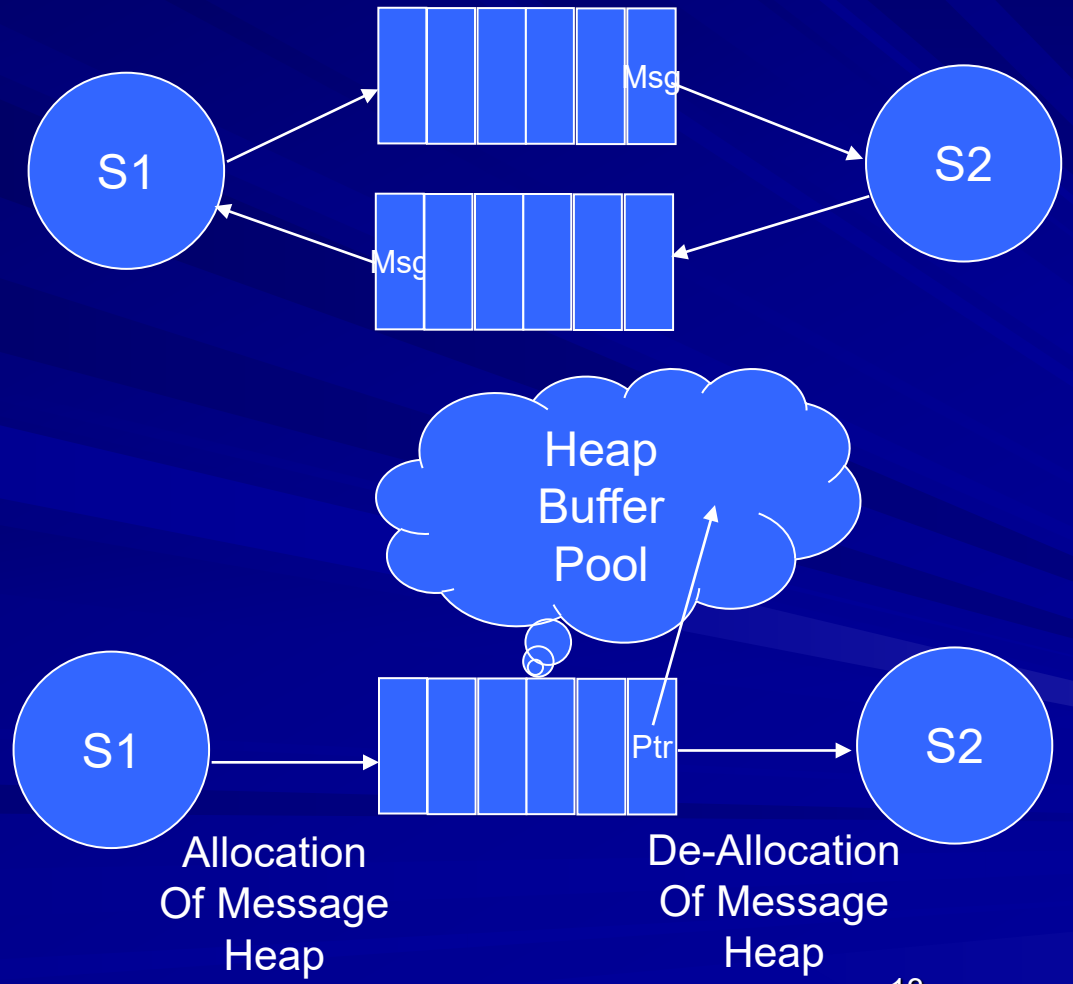
- $CPI_{worst\text{-}case} = (ICT + IOT) / ICT$
- $CPI_{best\text{-}case} = (max(ICT, IOT)) / ICT$
- $CPI_{required} = D_i / ICT$
- $OR = 1 - [(D_i - IOT) / ICT]$
- $CPI_{required} = [ICT(1\text{-}OR) + IOT] / ICT$
- $NOA = (D_i – IOT) / ICT$; $OR + NOA = 1$ (by definition)



Synchronization and Message Passing

- Message Queues
  - Provide Communication
  - Provide Synchronization
- Traditional Message Queue
  - Message Size
  - Internal Fragmentation
- Heap Message Queue
  - Messages Are Pointers
  - Message Data in Heap

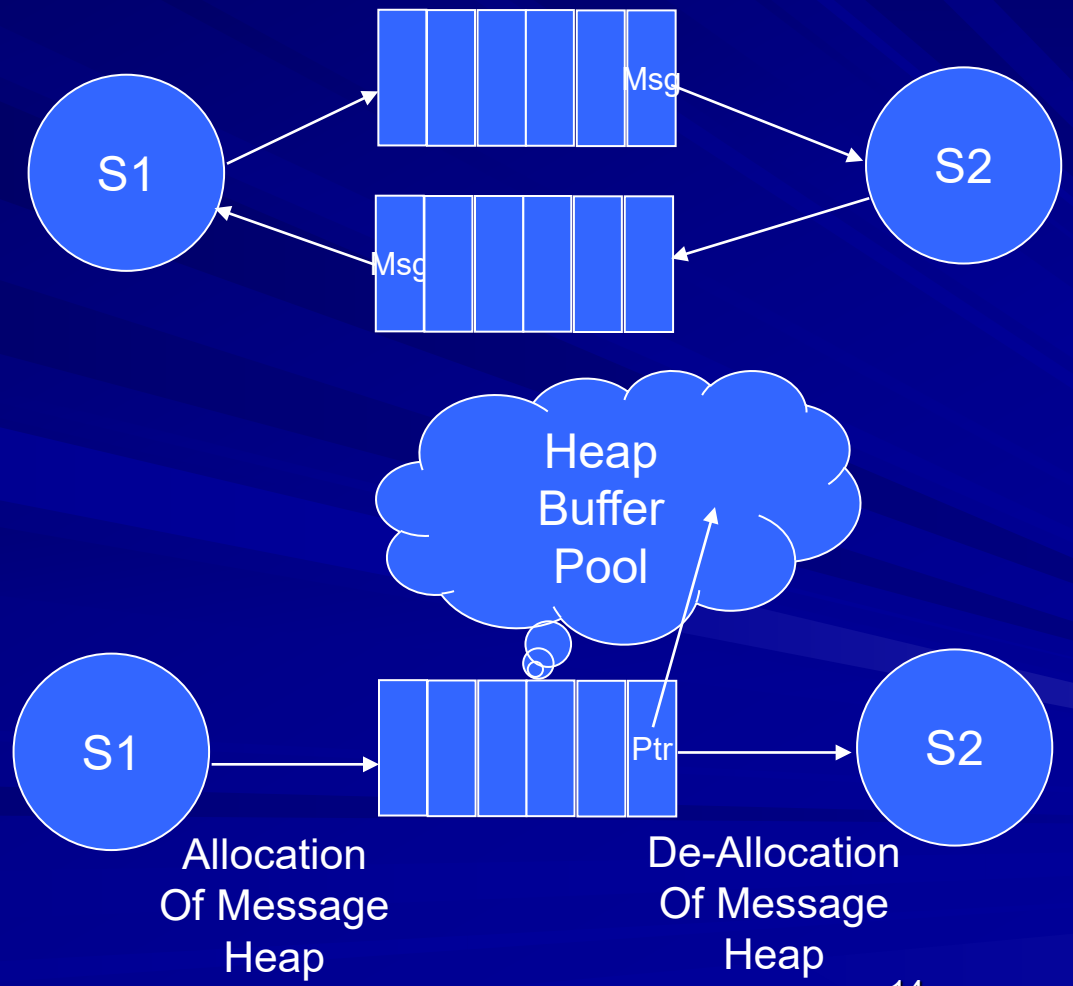
ENEA OSE
- Message Passing RTOS
- Advertised Advantages



Synchronization and Message Passing

ENEA OSE
- Message Passing RTOS
- Advertised Advantages

- Message Queues
  - Provide Communication
  - Provide Synchronization
- Traditional Message Queue
  - Message Size
  - Internal Fragmentation
- Heap Message Queue
  - Messages Are Pointers
  - Message Data in Heap



What About Storage I/O

- Much Higher Latency than Off-Chip Memory or Bus MMIO
- Milliseconds of Latency Compared to GHz Core Clock Rates – 1 million to 1!
- Flash Better, But Still Slower than SRAM or DRAM

- Option #1 – Avoid Disk Drives and Flash I/O

- Option #2 – Pre-Buffer Storage I/O (Read-Ahead Cache), Post Write-backs (Write-Back Cache), MFU/MRU (Most Frequently Used / Most Recently Used) Kept in Read Cache

Parallel Processing Speed-up (Makes I/O Latency Worse?)

- Grid Data Processing Speed-up
  1. Multi-Core, Multi-threaded, Macro-blocks/Frames
  2. SIMD, Vector Instructions Operating over Large Words (Many Times Instruction Set Size)
  3. Co-Processor Operates in Parallel to CPU(s)

- SPMD – GPU or GP-GPU Co-Processor
  - PCI-Express Bus Interfaces
  - Transfer Program and Data to Co-Processor
  - Threads and Blocks to Transform Data Concurrently

- Image Data Processing – Few Data Dependencies
  - Good Speed-up by Amdahl's Law
  - P=Parallel Portion
  - (1-P)=Sequential Portion
  - S=# of Cores (Concurrency)
  - Overhead for Co-Processor
  - IO for Co-Processing

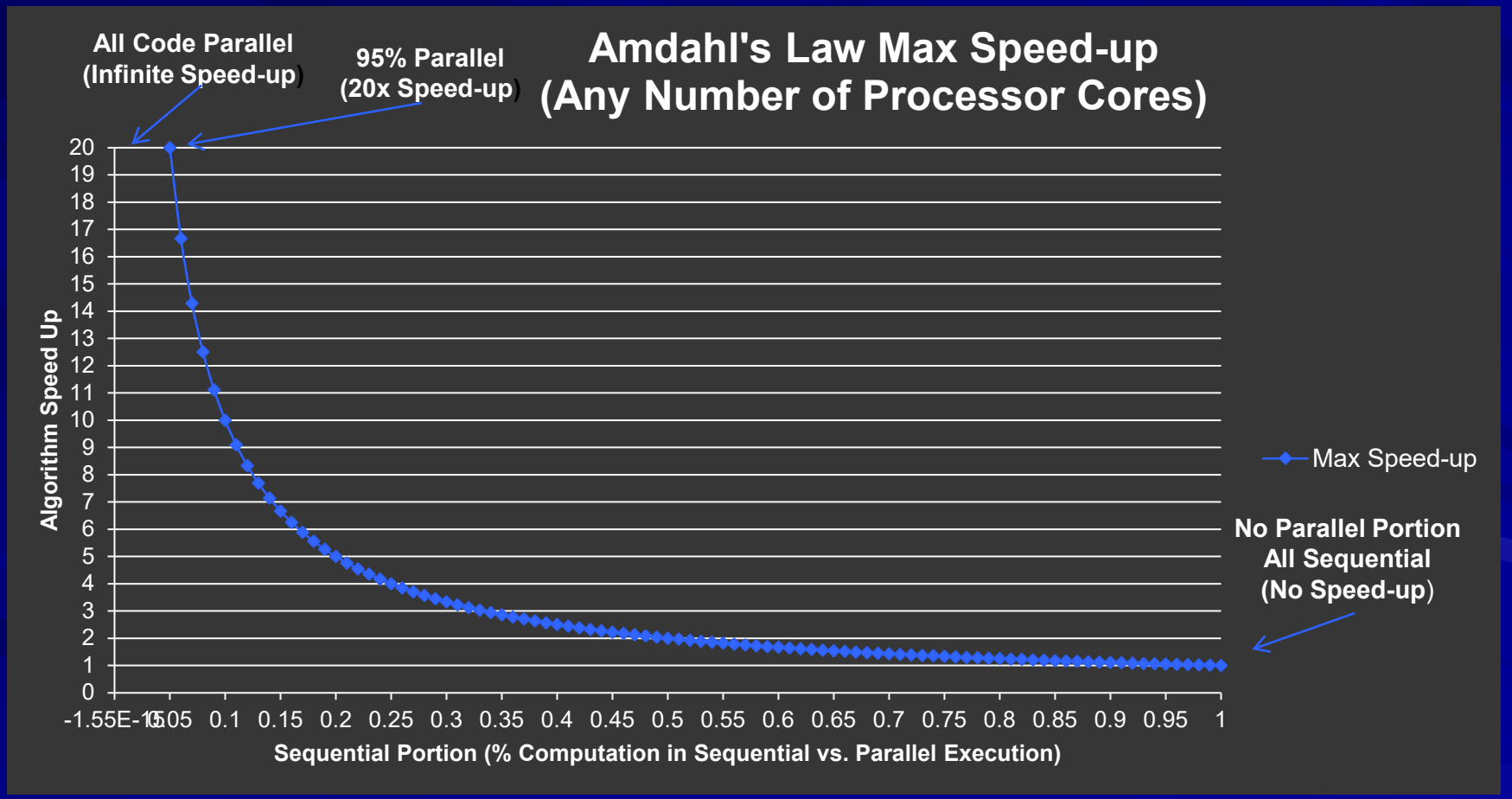
S is infinite here

$$Max\_Speed\_Up = \frac{1}{(1-P)+0}$$

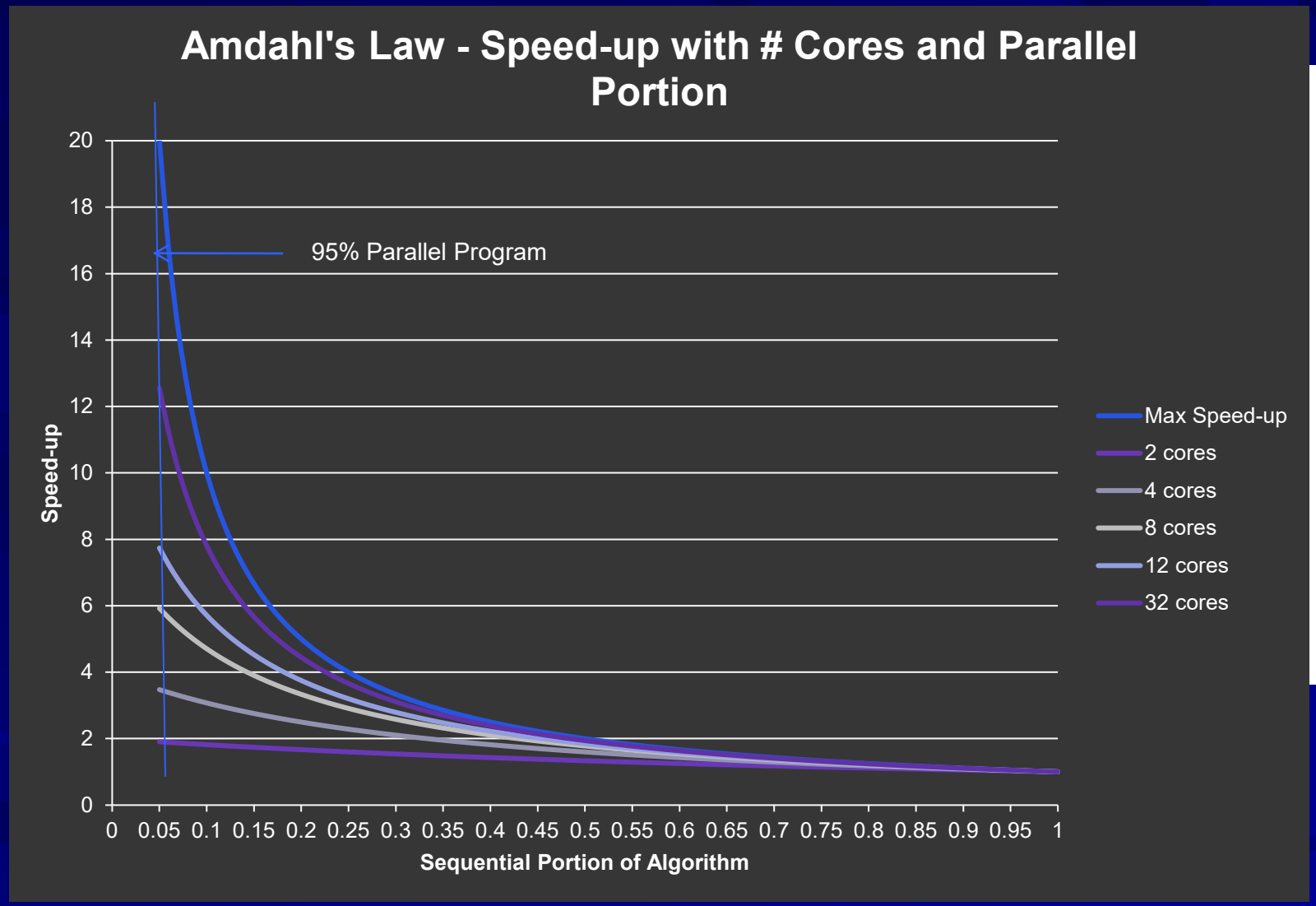
$$Multicore\_Speed\_Up = \frac{1}{(1-P)+P/S}$$

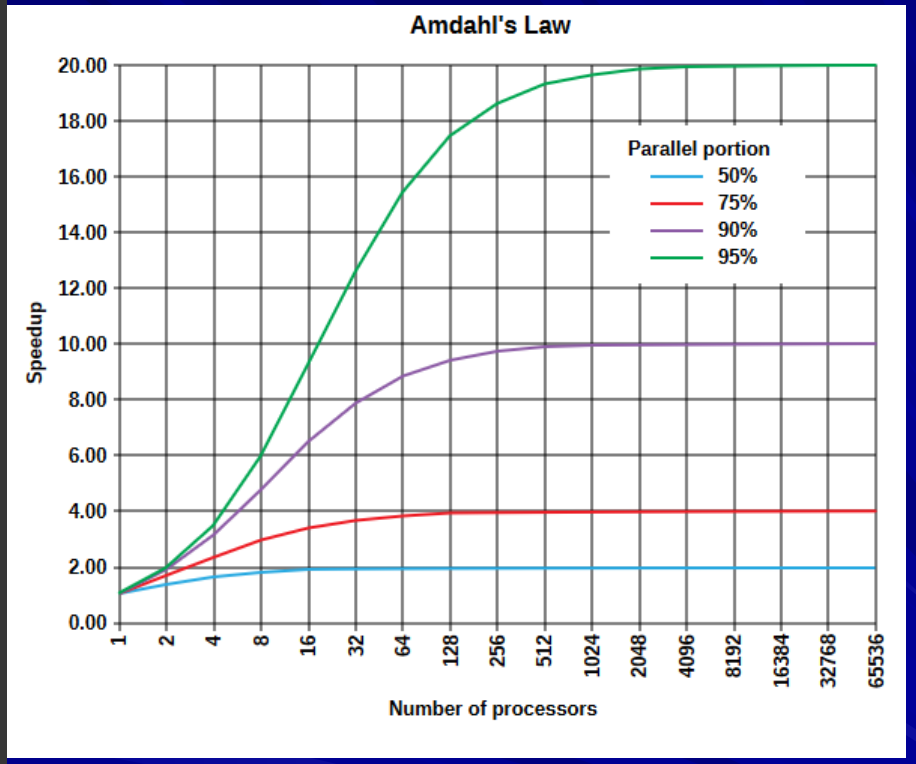
Amdahl's Law – Infinite Cores

- Maximum Speed-up Driven by Sequential and Parallel Portions of Program
  - P = Parallel Portion
  - (1-P) = Sequential Portion
  - Speed-up for Given Multi-core Architecture Function of # of Cores (Speed-up in Parallel Portions)



Multi-Core Speed-Up





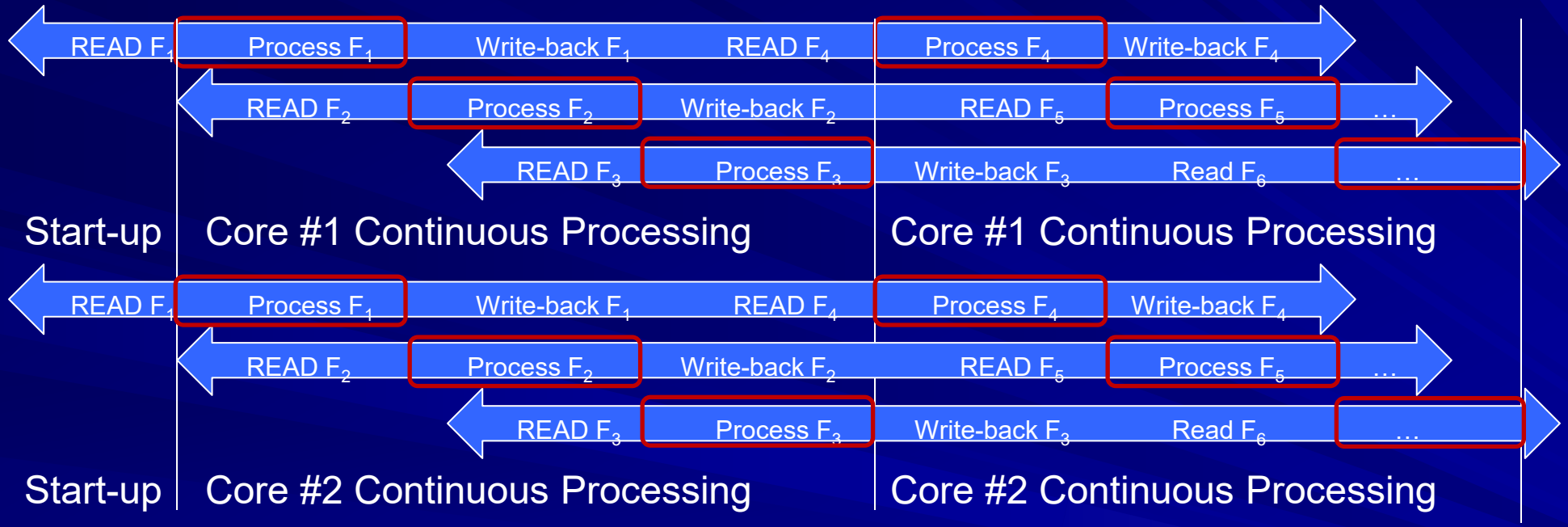
Hiding Storage I/O Latency – Overlapping with Processing

- Simple Design – Each Thread has READ, PROCESS, WRITE-BACK Execution

- Frame rate is READ+PROCESS+WRITE latency – e.g. 10 fps for 100 milliseconds
  - If READ is 70 msec, PROCESS is 10 msec, and WRITE-BACK 20 msec, predominate time is IO time, not processing
  - Disk drive with 100 MB/sec READ rate can only read 16 fps, 62.5 msec READ latency

Hiding Storage I/O Latency

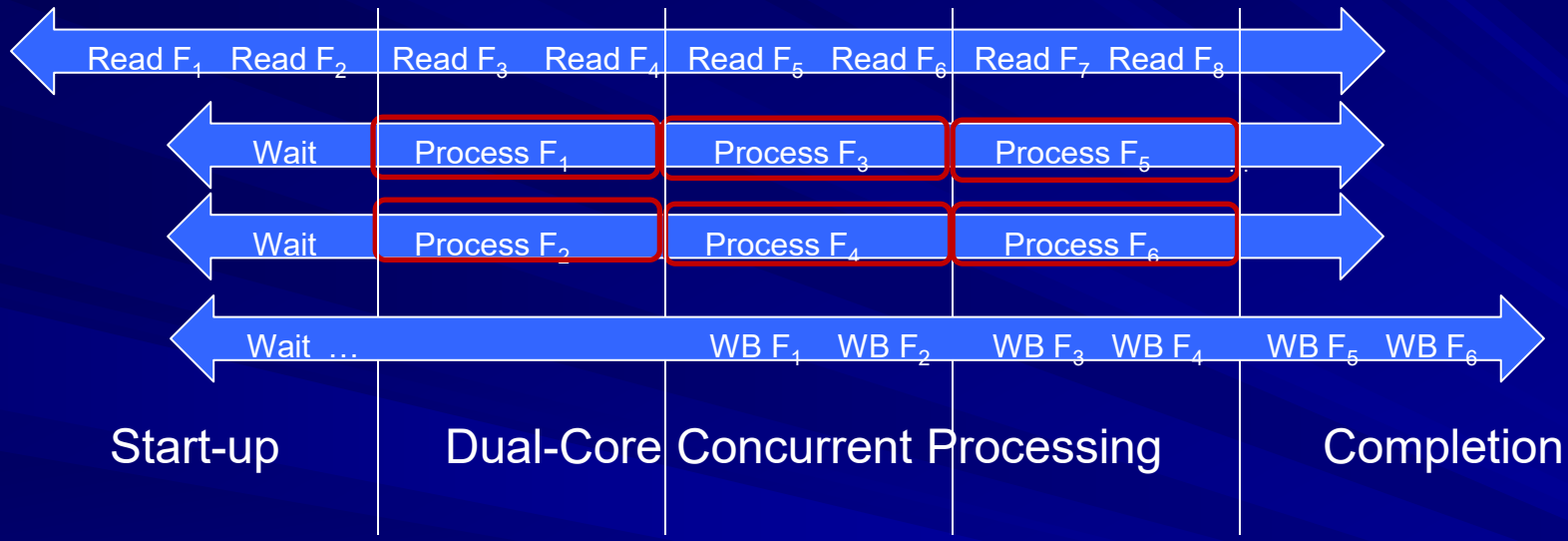
- Schedule Multiple Overlapping Threads?



- Requires $N_{threads} = N_{stages} \times N_{cores}$
- 1.5 to 2x Number of Threads for SMT (Hyper-threading)
- For IO Stage Duration Similar to Processing Time
- More Threads if IO Time (Read+WB+Read) >> 3 x Processing Time

Hiding Latency – Dedicated I/O

- Schedule Reads Ahead of Processing



- Requires $N_{threads} = 2 + N_{cores}$
- Synchronize Frame Ready/Write-backs
- Balance Stage Read/Write-Back Latency to Processing
- 1.5 to 2x Threads for SMT (Hyper-threading)

Processing Latency Alone

- Write Code with Memory Resident Frames
  - Load Frames in Advance
  - Process In-Memory Frames Over and Over
  - Do No IO During Processing
  - Provides Baseline Measurement of Processing Latency per Frame Alone
  - Provides Method of Optimizing Processing Without IO Latency

I/O Latency Alone

- Comment Out Frame Transformation Code or Call Stubbed NULL Function
  - Provides Measurement of IO Frame Rate Alone
  - Essentially Zero Latency Transform
  - No Change Between Input Frames and Output Frames
  - Allows for Tuning of IO Scheduler and Threading

Tips for Linux I/O Scheduling

- blockdev --getra /dev/sda
  - Should return 256
  - Means that reads read-ahead up to 128K
  - Function calls – read, fread should request as much as possible
  - Check “actual bytes read”, re-read as needed in a loop
- blockdev --setra /dev/sda 16384 (8MB)

- Switch CFQ to Deadline
  - Use “lsscsi” to verify your disk is /dev/sda … substitue block driver interface used for file system if not sda
  - cat /sys/block/sda/queue/scheduler
  - echo deadline > /sys/block/sda/queue/scheduler
- Options are noop, cfq, deadline, anticipatory