Revisiting Operating System Mass Storage Presumptions Enables Higher Performance and Efficiency

Robert Gezelter <gezelter@rlgsc.com>

IEEE LISAT Conference * 6 May 2022

NYIT, Old Westbury

Abstract

Operating system mass storage I/O facilities have remained essentially unchanged since their original conception nearly 50 years ago; an era of limited memory capacity, processing power, and device intelligence.

Today’s environment has far larger workloads and more plentiful resources, allowing us to revisit past tradeoffs.

We will examine how increased resources and removal of imposed serialization can be leveraged to increase performance and efficiency.

Overview

- *Changes in computing context*
- *The “knowable future”*
- *How modest changes in I/O Infrastructure can improve optimization*

Today’s differences:

- *Vastly larger main memory*
- *Vastly higher degree of multiprogramming/processing*
- *Deeply queued devices*

Before starting

- While concept was inspired by rotating mass storage; changes benefit all storage technology with any variable timing
- Virtual ≠ Real, e.g., physical optimization can only be determined in the physical world

Caches are not a universal answer

- ***“Working set”****
- ***More streams than available cache lines***
- ***Simultaneous active contexts, e.g., no quantum***
- ***Cache entry fratricide***
- ***Cache thrashing***

* J. Denning (1967) “The working set model for program behavior” Proceedings, ACM Symposium on Operating System Principles

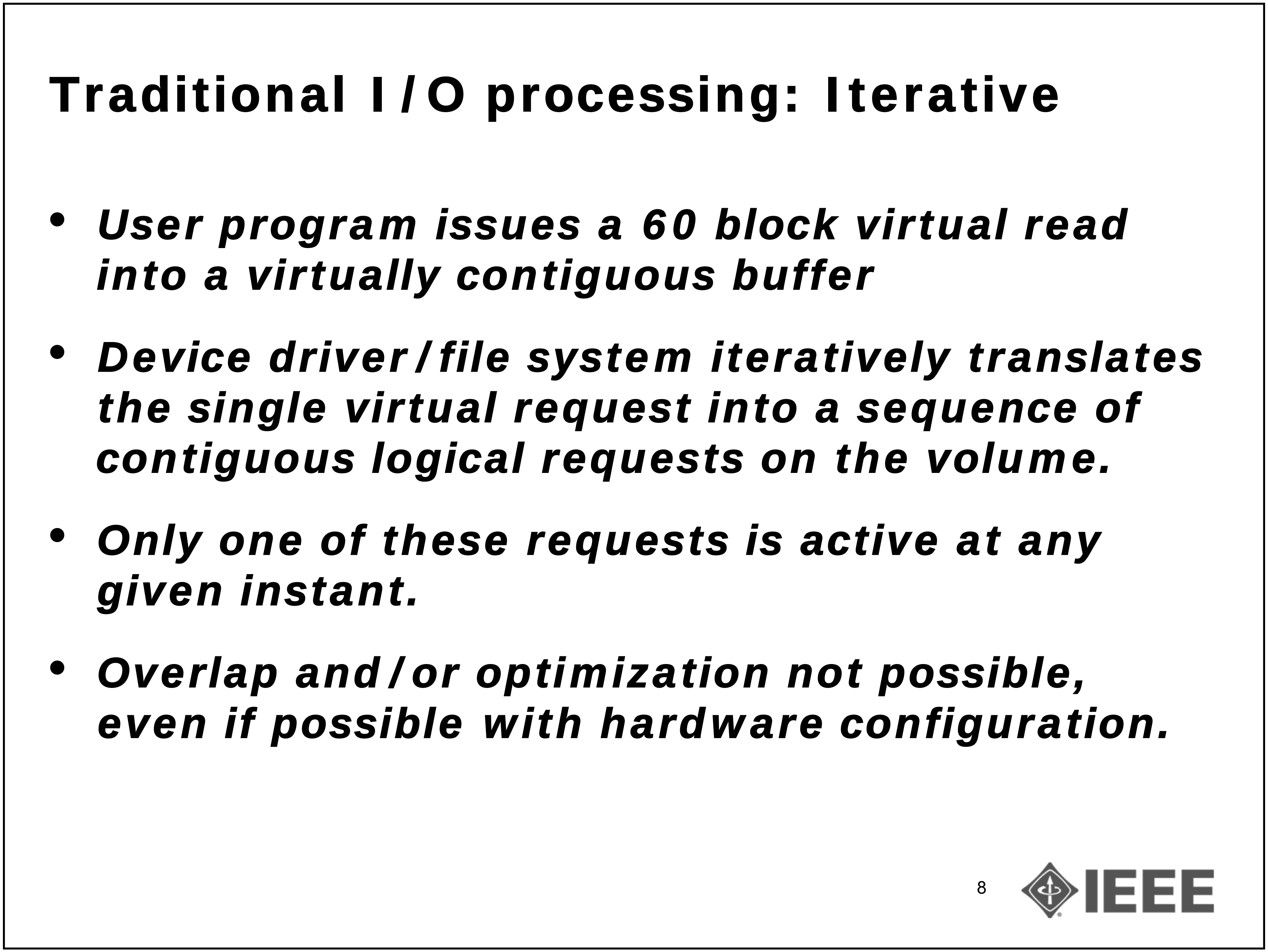
What enables optimization

- *Hoisting [Allen, et al., 1971]*
- *Resequencing*
- *Optimization can only be done on known requests*
- *Wider optimization windows; e.g., basic blocks*
- *Highest correlations are blocks in single files in the same access stream*

A sample request: read / write 60 blocks

Starting at VBN 0, transfer 60 blocks into a virtually contiguous buffer. The file is not contiguous on the storage volume:

Segment	Starting VBN	Starting LBN	Length
1	0	4500	5
2	5	100	10
3	15	20250	20
4	35	450	10
5	45	1000	10
6	55	10000	10



Traditional I/O processing: Iterative

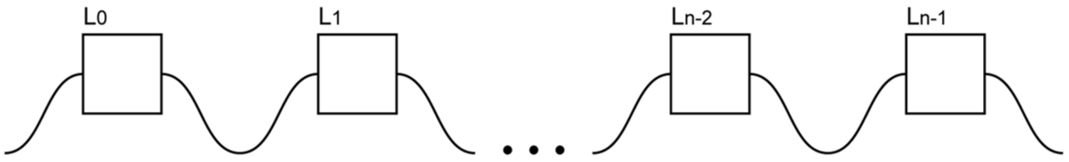
- *User program issues a 60 block virtual read into a virtually contiguous buffer*
- *Device driver/file system iteratively translates the single virtual request into a sequence of contiguous logical requests on the volume.*
- *Only one of these requests is active at any given instant.*
- *Overlap and/or optimization not possible, even if possible with hardware configuration.*

Iterative issue truncates the optimization window

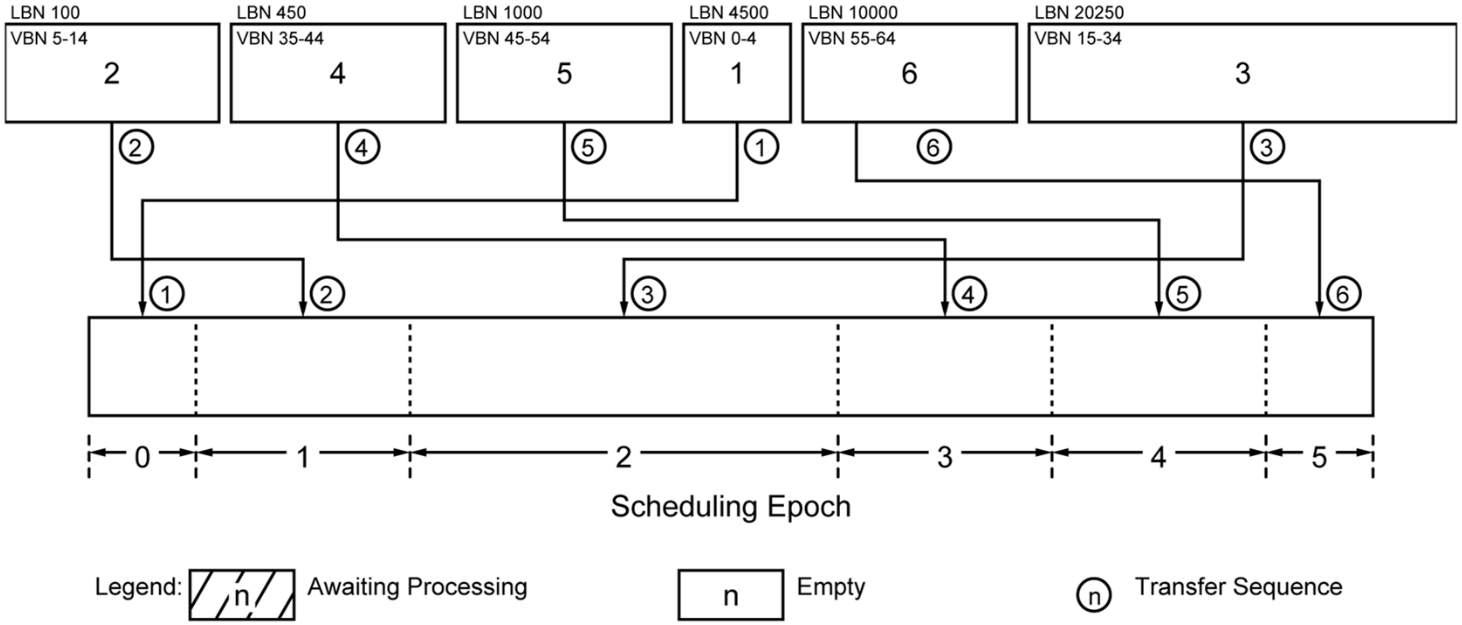
Epoch	LBN	Length
0	4500	5
1	100	10
2	20250	20
3	450	10
4	1000	10
5	10000	5

Issued one LBN range at a time, waiting for completion serializes all of the requests, regardless of physical device reality.

Serialization obscures the “known future”.



Sequential issuance prevents device optimization

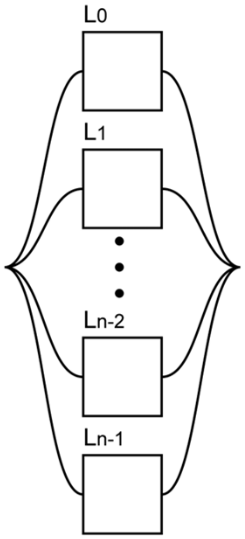


Can issuance be improved?

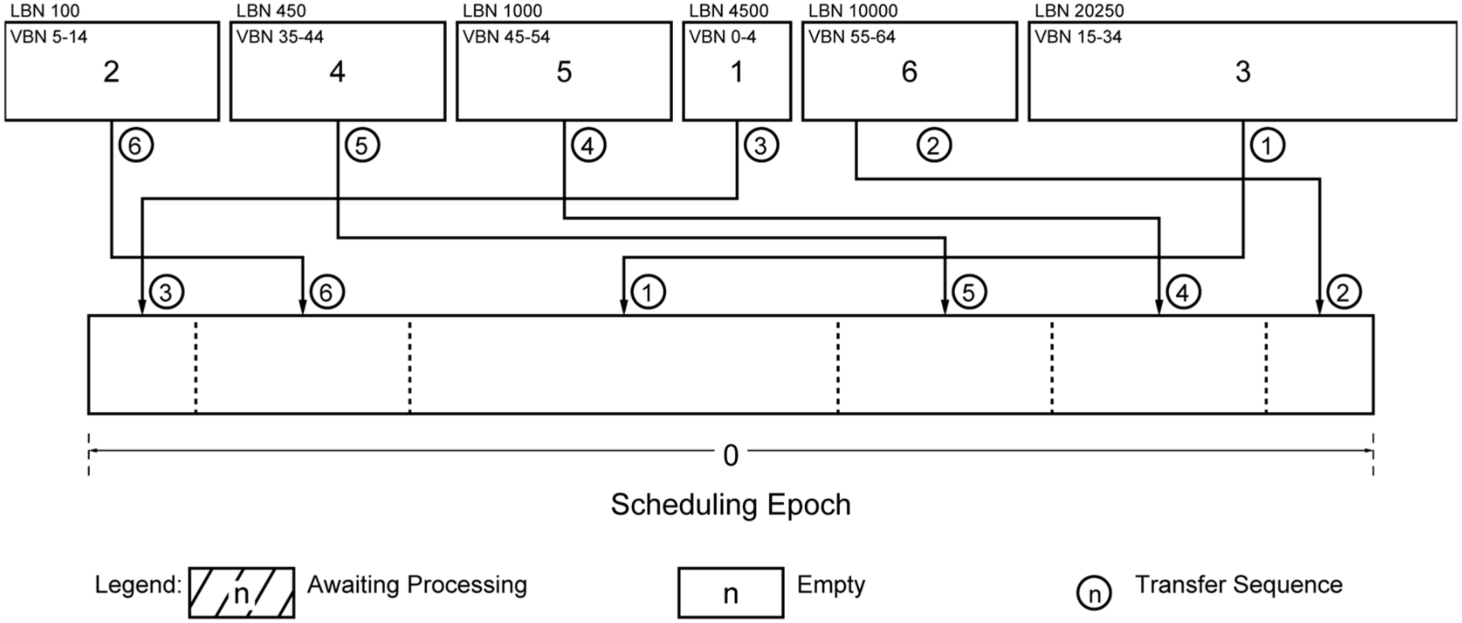
- *Since all of the sub-requests are mutually independent, we can reduce the six optimization windows/epochs to a single epoch*
- *The device(s) gain visibility of the "known future"*
- *Sub-requests can be processed in a device-optimized sequence*

Removing unneeded serialization widens the epoch

Epoch	LBN	Length
0	4500	5
	100	10
	20250	20
	450	10
	1000	10
	10000	5

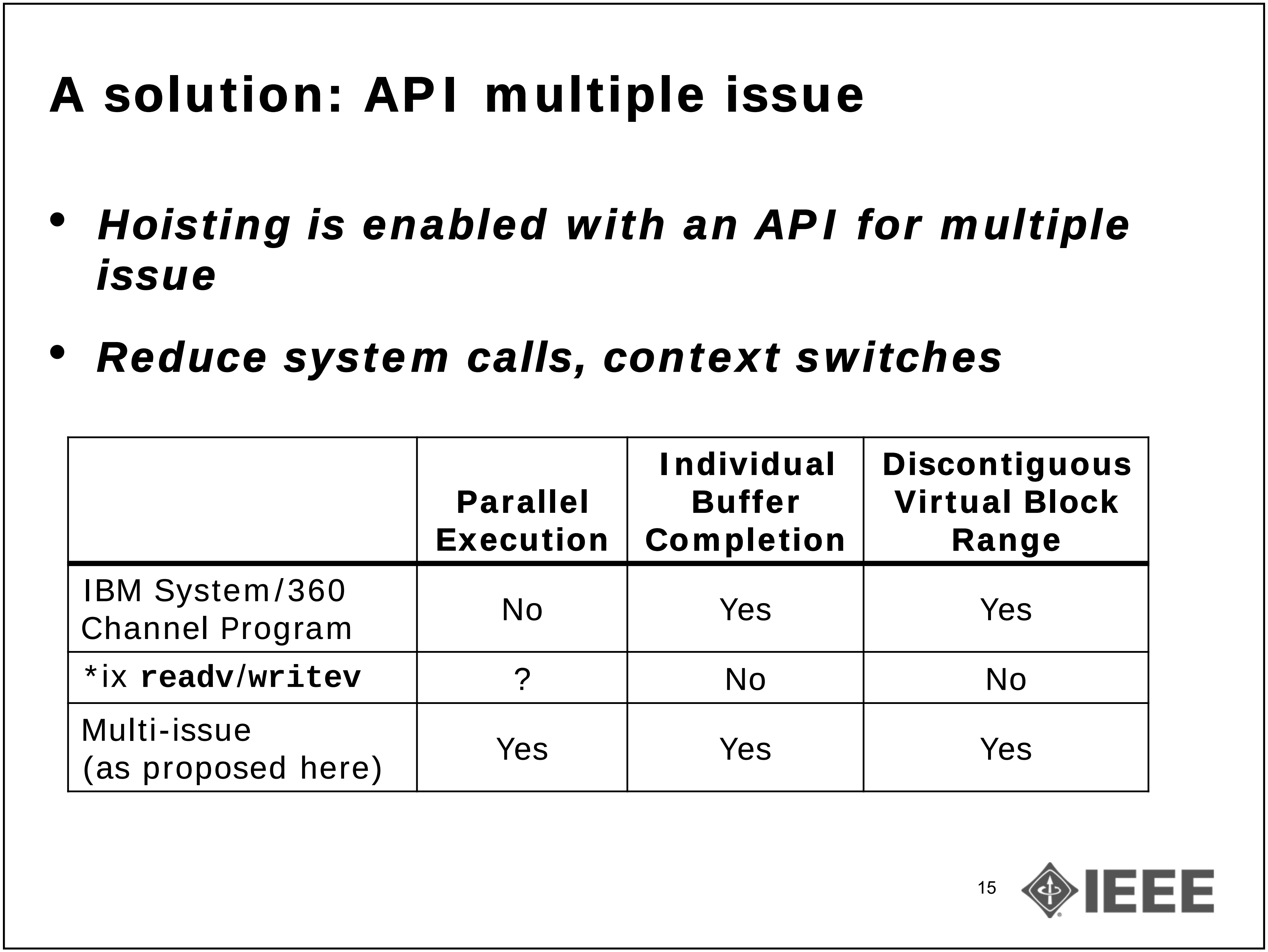


Simultaneous issue enables optimization



Simultaneous issue also applies at user API level

- *Multiple I/O operations on a file are often known at the same time.*
- *OS primitives (OpenVMS QIO; *ix* `readv/writev`*) do not express multiple buffers with disjoint mass storage block locations*
- *User programs can be preempted between successive system calls*



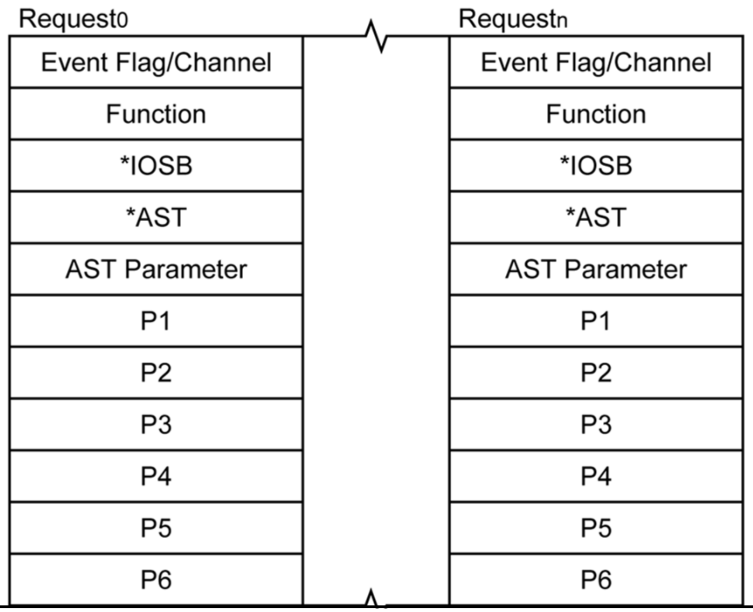
A solution: API multiple issue

- ***Hoisting is enabled with an API for multiple issue***
- ***Reduce system calls, context switches***

	Parallel Execution	**Individual Buffer Completion**	**Discontiguous Virtual Block Range**
IBM System/360 Channel Program	No	Yes	Yes
*ix **readv/writev**	?	No	No
Multi-issue (as proposed here)	Yes	Yes	Yes

A different perspective: multiple issue

- ***Each sub-request of a virtual request is independent of the other sub-requests derived from the same virtual request: enable translate and issue en masse.***
- ***All requests are then within the same epoch***



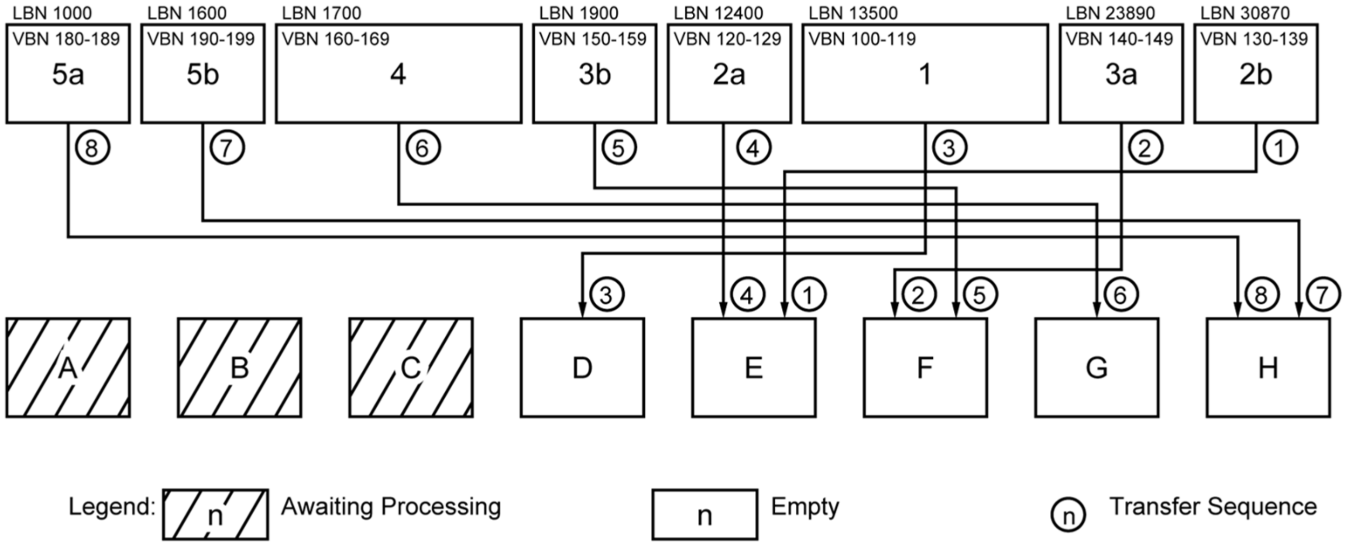
Request$_0$	Request$_n$
Event Flag/Channel	Event Flag/Channel
Function	Function
*IOSB	*IOSB
*AST	*AST
AST Parameter	AST Parameter
P1	P1
P2	P2
P3	P3
P4	P4
P5	P5
P6	P6

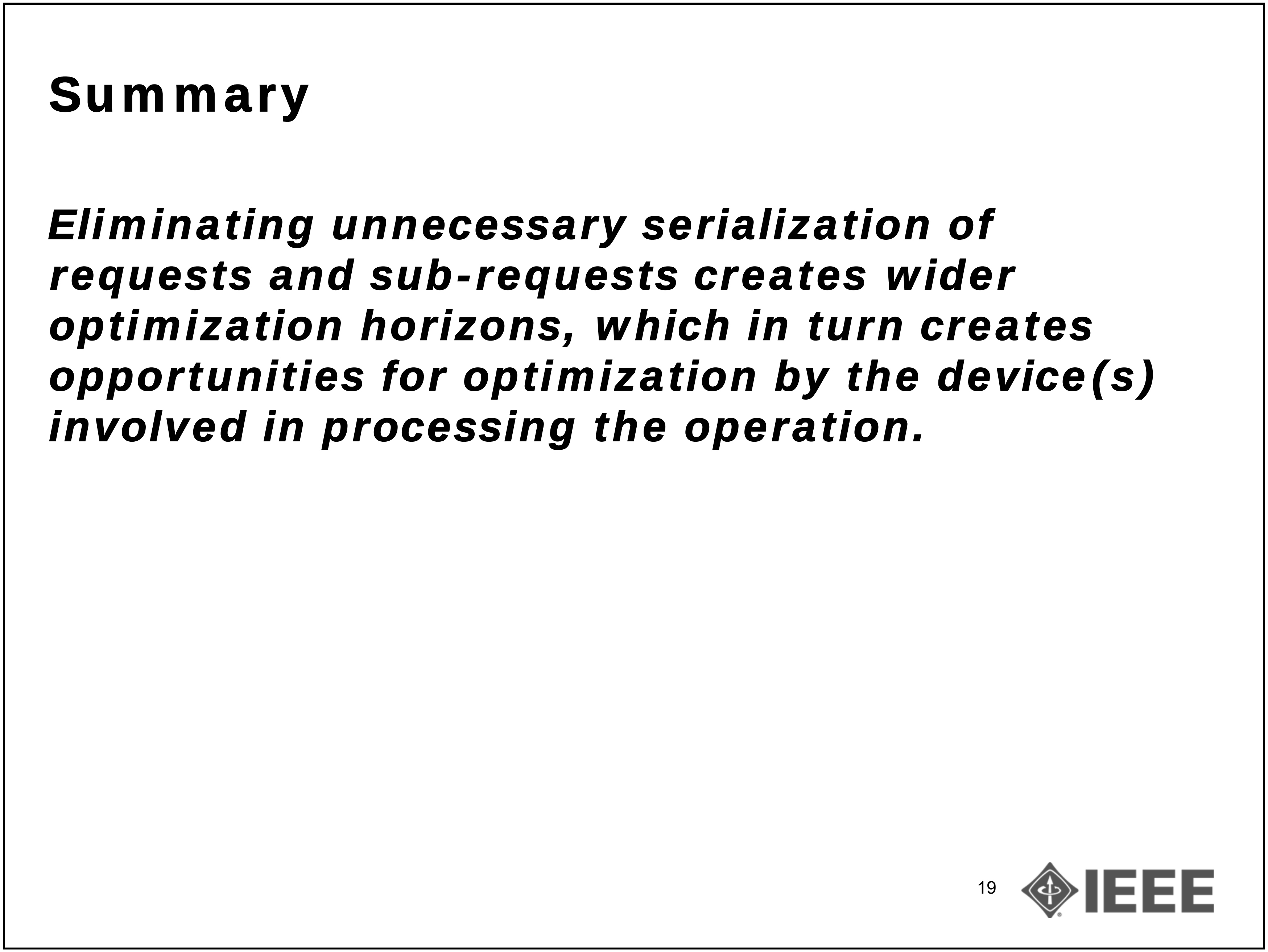
- Virtual/Logical Block Number
- Transfer length
- Buffer Address
- Completion Status
- Completion Exit
- etc.

Eliminating imposed serialization enables more efficient processing

- *Eliminating imposed serialization exposes the ”known future”*
- *For rotating devices, advantageous positioning options are exposed*
- *Wider optimization windows prevent cache fratricide*
- *Ensuring visibility prevents poor decisions*

More optimal processing sequence



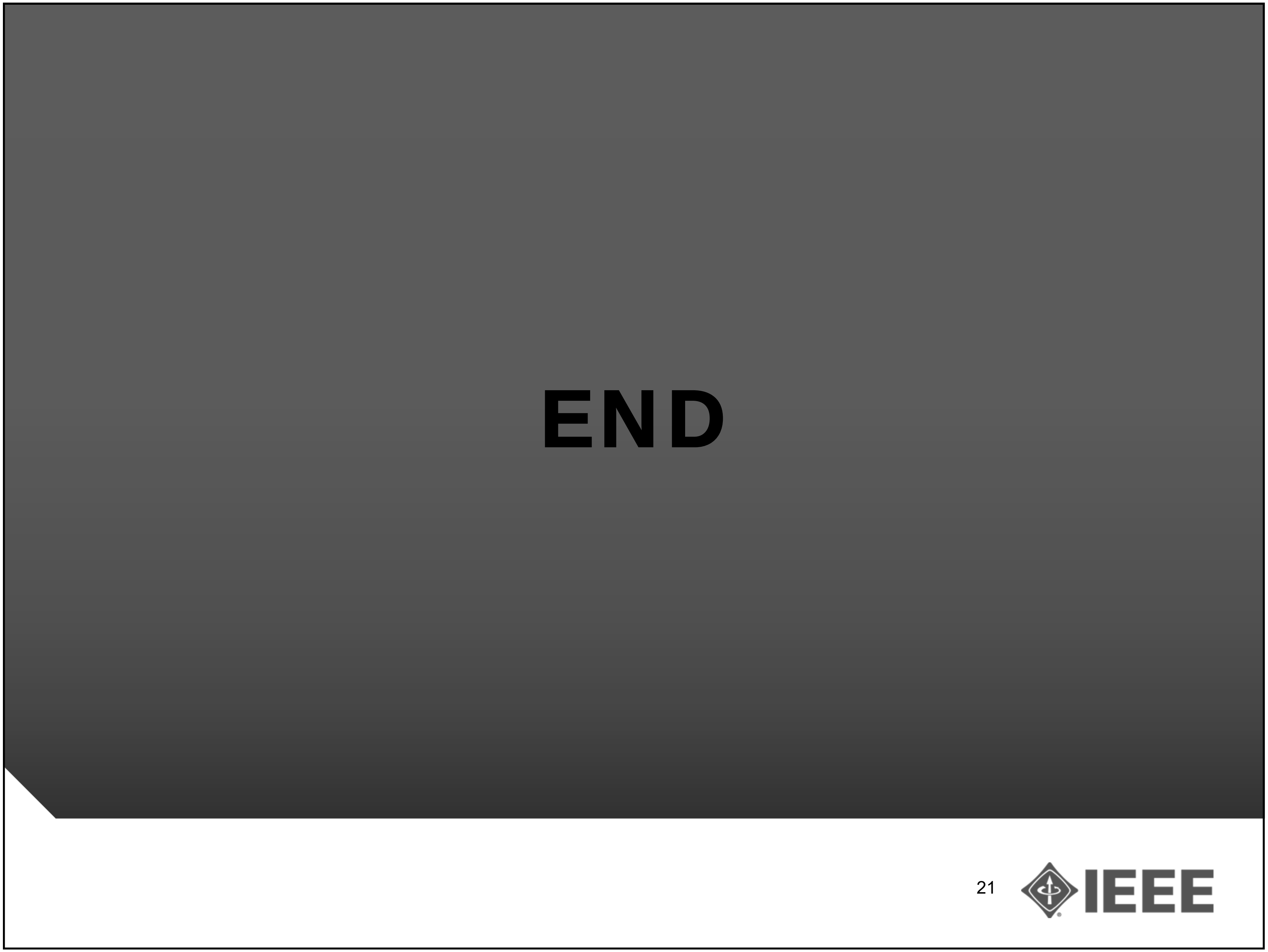


Summary

Eliminating unnecessary serialization of requests and sub-requests creates wider optimization horizons, which in turn creates opportunities for optimization by the device(s) involved in processing the operation.

Questions?

Robert Gezelter
gezelter@rlgsc.com
http://www.rlgsc.com



END