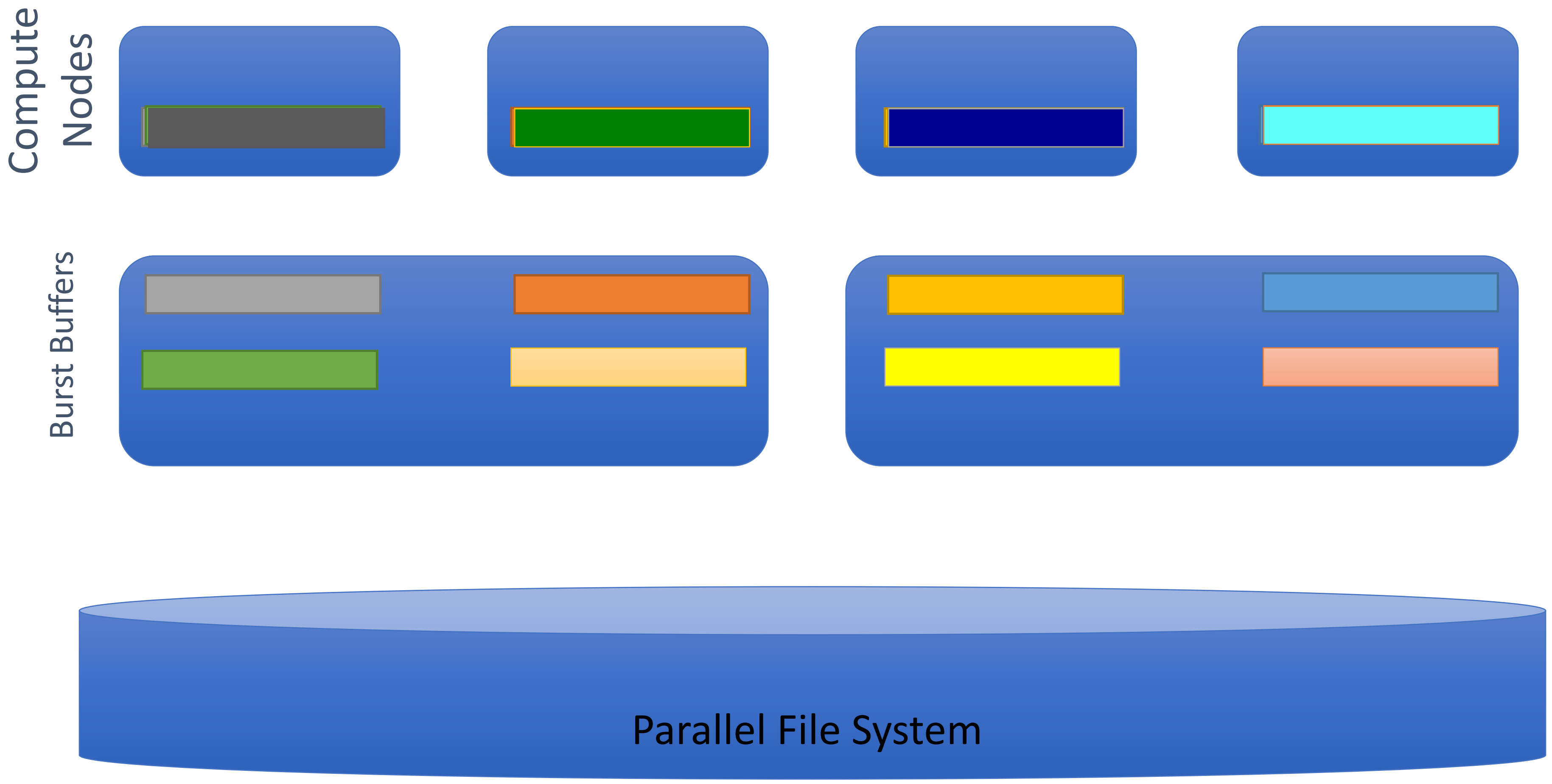
Unraveling Burst Buffers

APIs and Architectures

John Bent, Chief Architect, Seagate Gov

Checkpoint with Burst Buffers



How do applications access tiered storage

- Middleware
  - Argonne FTI
  - LANL HIO
  - Livermore SCR
  - HDF5
  - ORNL ADIOS
  - Intel DAOS
- Direct
  - Cray DataWarp
  - DDN IME
  - EMC 2 Tiers
  - IBM CORAL burst buffer
- Block level
  - DDN SFX Flash Cache
  - Hitachi Dynamic Tiering
  - Seagate L300N/G300N Nytro
  - ZFS Allocation Class

Did not study these. They tend to hide the tiering from the application. [Surely they have hints though which are worthy of study.]

Our focus.

Middleware developers abstract/hide tiering

- Argonne FTI
  - Leo Bautista-Gomez: “FTI is *designed to abstract away the burst buffers's API* and offer a higher level interface for the apps. The objective is to hide that complexity to users and to offer them a way to easily make use of BB for checkpointing. FTI focuses only on checkpointing and does not offer an API for general purpose I/O. Concerning the frequency of flushing data to the PFS, this can be specified on a config file. Also, both modes (synchronous and asynchronous) are offered.”
- LANL HIO
  - Cornell Wright: “HIO operates a a *higher level of abstraction*. It makes use of the sort of function you describe, rather than implementing them.”
- Livermore SCR
  - Adam Moody: “in SCR, the user specifies which files are part of the checkpoint, SCR directs the app to write those files to the burst buffer mount point (which we assume supports POSIX IO), and then when the app says it's finished writing all of its files for that checkpoint, *SCR makes the necessary calls* to the vendor-specific burst buffer API to copy those files to the parallel file system.We also plan to write higher level tools / scripting to help with pre-stage and post-stage.”

Is block–level Tiered Storage?

Yes

- Multiple media with different performance/capacity ratios
- Does tiering

No

- Invisible to users
- Not managable

Maybe

- What if they add control interfaces?
  - Like Intel DSS (Differentiated Storage Services)
- But users only know files/objects; they don’t know blocks
  - Needs integration with the file system

Possible Interfaces to Tiered Storage

- Workload Manager / Job Scheduler
- Command Line Interface
- Run Time
  - MPI-IO
  - POSIX
    - In-kernel
    - FUSE
  - Direct API

Actually where all the action is at.

Ostensibly the focus of my exploration.

Using burst buffers is about the control plane
and not the data plane

What are Interfaces Actually Used For?

- Initialize
  - Set up, initialize, configure
- Policy
  - Set tiering policy
- Mount
  - Set up a mount point
- Staging
  - Initiate staging (pre-job, post-job, during job)
- Directory direct
  - Direct directory operations
- File IO direct
  - Directly read/write files

	Job Submission	Command Line	Application API	POSIX	MPI-IO
IBM	I,P,PS,S,M	I,P,S,M,D	I,S	D,F	?
Cray	I,P,PS,S,M	I,P,S,M	I,S	D,F	?
EMC	?	?	I,S,M	D,F	F
DDN	?	?	?	D,F	F

Most interesting observation to me? No D,F here!
(obviously no PS)

KEY	Unknown	Supported, documentation available	Supported, documentation unavailable	Unsupported	N/A

I=initialize,setup,config; P=set policy; PS=pre/post-stage; S=initiate staging; M=mount; D=directory Ops; F=file read/write

Set Up Tiering: Different Features

- Specify Which Portion of the Namespace (3/3)
    - Specify which directories to stage
- Set throttles
    - Bandwidth (1/3)
    - Concurrency (1/3)
- Set IO limits (3/3 [but each with different combos of the below])
    - Read/write byte limits, IO per time period, max file size, max file creates, quota
- Set up striping/sharing characteristics (1/3)
    - Private or shared
    - Striped or not
- Disable/enable automation (1/3)
    - Implicit or explicit tiering
- Set tiering policy (3/3)
    - Frequency, on sync, on close, at job end
- Set up lifetime (2/3)
    - Persistent or job
- Sizing
    - Set initial size (3/3)
    - Grow (2/3)

Most “cool” features unavailable in the API’s

- Set up the actual allocation/namespace
    - Shared or private, persistent or job
    - Size
    - Tiering policy (sync on close, on sync, at job end, every M minutes, never)
- Request optimization strategy
    - Bandwidth or interference or SSD health

Observation? Intention is to minimize application modifications.
Do most everything through job scheduler, CLI and POSIX.

API mostly for small optimizations: ”tier now”, “tier later”, “kill transfer”

Exception: EMC does have setup in the API.

BB Setup

- IBM
  - CLI: > bbcmd create –options –path –size –target # what are options?
  - API: n/a
  - Job Scheduler: ssd=min[,max]
- EMC
  - CLI: ?
  - API: int burst_b_ns_create(path, quota_bbfs, quota_pfs, *bbns, *e);
  - Job Scheduler: ?
- Cray
  - CLI:
    - > session=`dwcli create session --expiration=e`
    - > instance=`dwcli create instance –session $session –capacity=c –optimization=o`
    - > configuration=`dwcli create configuration –type=t`
    - > dwicl create activation –mount /path –configuration $configuration
  - API: n/a
  - Job Scheduler:
    - #DW jobdw access_mode=m capacity=c type=scratch|cache [options]
    - #DW persistentdw name=n [options]
    - #DW swap size_in_GiB

Stage a File / Directory – Ops on Staging Transfers

- Asynchronous (3/3)
- Query ongoing (3/3)
- Cancel ongoing (2/3)
- List ongoing (1/3)
- Submit batch/list (2/3)
- Block on ongoing (1/3)
- Add key-values (1/3)
- Set concurrency for directory staging (1/3)

Staging Terminology

- EMC
  - Migrate (BB -> PFS)
  - Restore (PFS -> BB)
  - Release (free from PFS)
- Cray
  - Stage in (PFS -> BB)
  - Stage out (BB -> PFS)
- IBM
  - Transfer (both directions)
- Others that I’ve heard
  - Promote (PFS -> BB)
  - Persist (BB -> PFS)
  - Trim (free from PFS)

Interesting Unique Feature: KV attached to transfer

- Tom Gooding:
  - Add metadata to a transfer.
  - Labs asked for this
  - “The BB_AddKeys() API allows users to add metadata about their transfers. This metadata can be queried via the BB_GetTransferKeys() API. The Labs wanted a means to add customized information about their checkpoints, which would be retained in the BB metadata store. The example we were given was adding a "checkpoint generation #" or "checkpoint version", which could be subsequently queried. It can also be very useful information for administration and debugging.”

Interesting unique feature: dw_get_mds_path()

- char *dw_get_mds_path(const char **dw_root, uint64_t key)

- DW can create multiple shared namespaces each with its own MDS
- How ranks can balance place/locate data

A Closer Look at SSD Protection

- IBM
  - int BB_SetUsageLimit (const char *mountpoint, BBUsage_t *usage)
  - int BB_GetDeviceUsage (uint32_t devicenum, BBDeviceUsage_t *usage)
    - (temperature, OP, % used, IO counts, etc)

- Cray
  - Disallow more than 10*222GiB writes within 10 seconds
  - #DW jobdw type=scratch access_mode=striped capacity=222GiB \
  - #DW write_window_length=10 write_window_multiplier=10

  - Disallow files larger than MFS, more than 12 file creates, limit total size
  - #DW jobdw type=scratch access_mode=striped(MFS=16777216,MFC=12) capacity=222GiB

  - Can also ask for scheduler SSD lifetime prioritization

A closer look at API throttling

- int dw_set_stage_concurrency(const char *path, unsigned int num)
    - On an instance (mount point)
    - Set number of streams

- int BB_SetThrottleRate (BBTransferHandle_t handle, uint64_t rate)
    - On a (set of) transfer(s)
    - Set bandwidth rate
    - Use 0 to pause

Job Submission Commands Pretty Straightforward

#DW jobdw	- Create and configure access to a DataWarp job instance	Cray
#DW persistentdw	- Configure access to an existing persistent DataWarp instance	
#DW stage_in	- Stage files into the DataWarp instance at job start	
#DW stage_out	- Stage files from the DataWarp instance at job end	
#DW swap	- Create swap space for each compute node in a job	
#BB [a bunch more]	- SLURM adds a bunch more (https://www.slurm.schedmd.com/burst_buffer.html)	

ssd=min,[max]	min = Initial size of the LV to create (per compute node) max = Size of the LV during runscript phase (per compute node)	IBM
usr_stage_in		
usr_stage_out	Paths to stagein and stageout scripts.	

Then environment variables define the paths:

IBM: SSDPATH

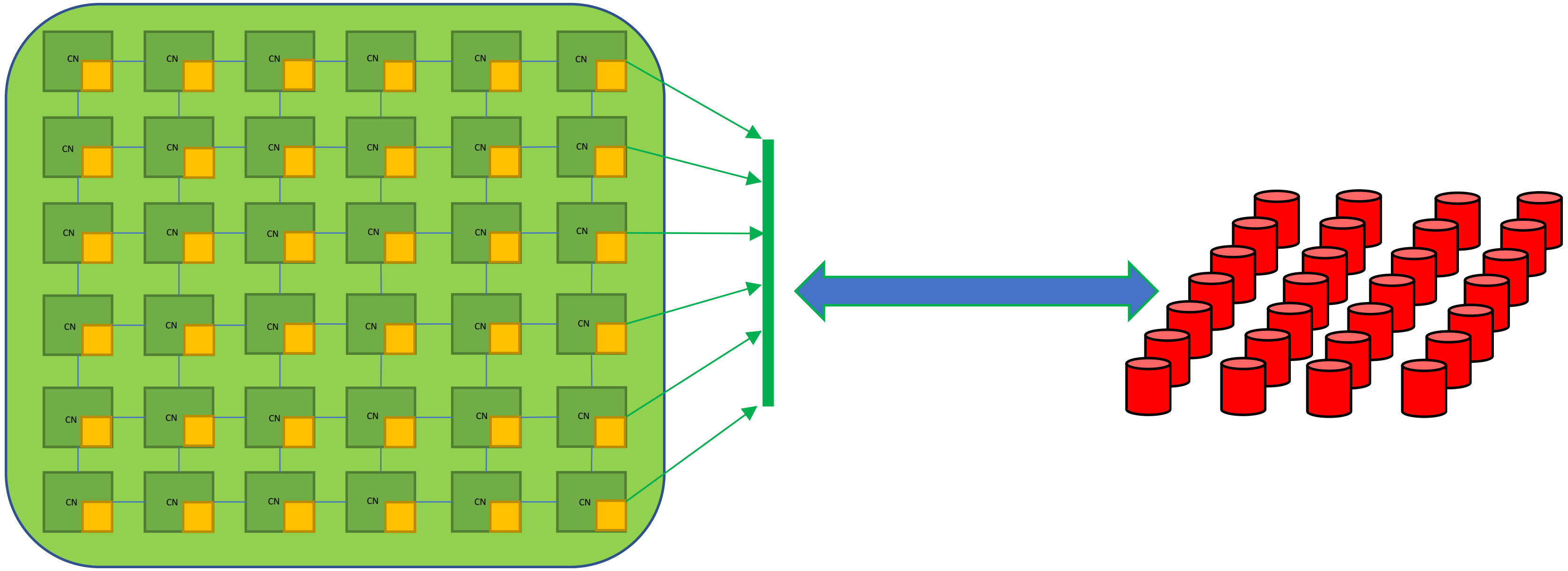
Cray: DW_JOB_PRIVATE, DW_JOB_STRIPED, DW_JOB_STRIPED_CACHE,
DW_PERSISTENT_STRIPED_CACHE_{name}, DW_PERSISTENT_LDBAL_CACHE_{name},
DW_PERSISTENT_STRIPED_{name}

to share or not to share

a comparison of burst buffer architectures

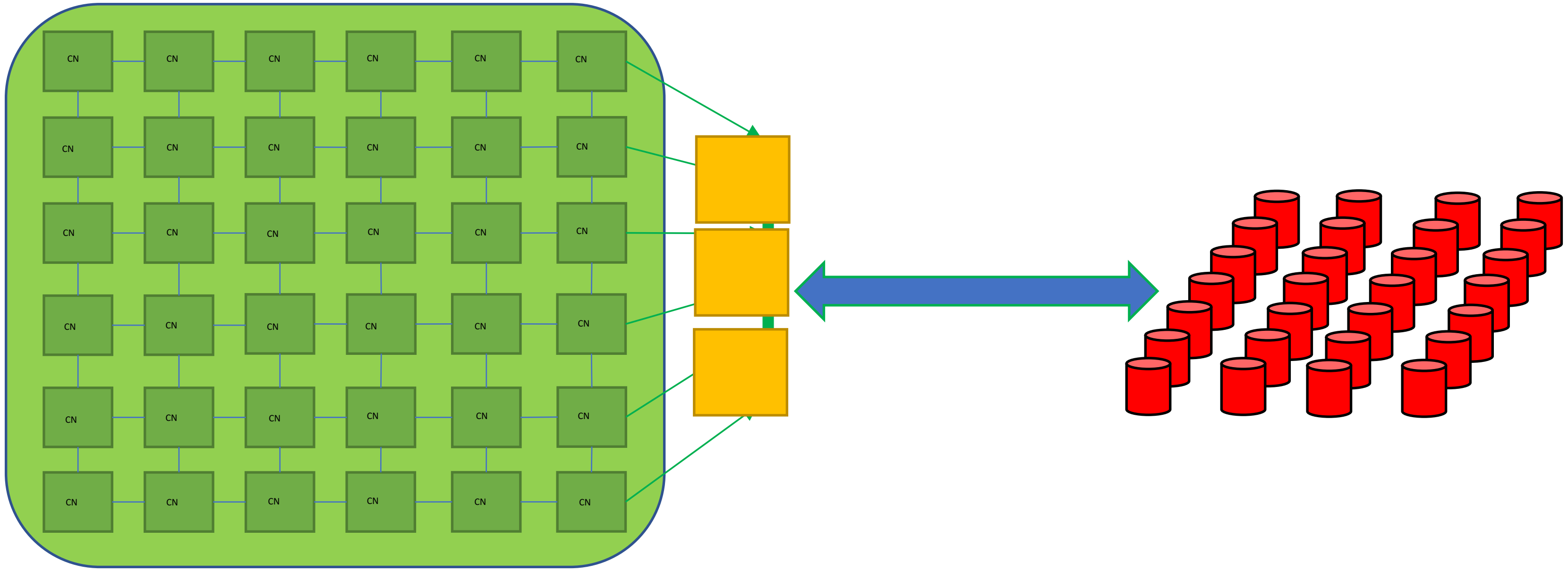
bent, settlemeyer, cao

three places to add burst buffers



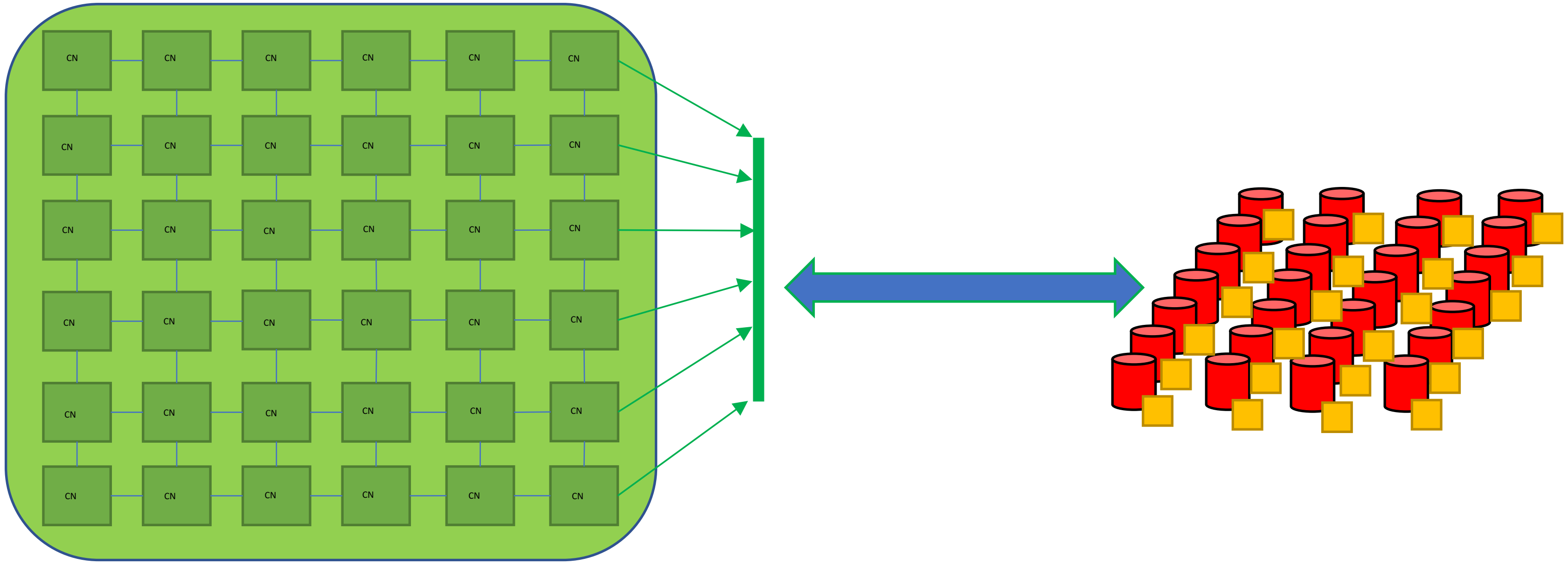
private, e.g. Cray/Intel Aurora @ Argonne

three places to add burst buffers



shared, e.g. Cray Trinity @ LANL

three places to add burst buffers



embedded, e.g. Seagate Nytro NXD

private

no contention
linear scaling
low cost
no network bandwidth

coupled failure domain
single shared file is difficult
small jobs cannot use them all

shared

n-1 easy
data can outlive job
temporary storage if pfs offline
small jobs can use it all
decoupled failure domain
most flexible ratio btwn compute, burst, pfs

most expensive
interference possible

embedded

n-1 easy
data outlives job
small jobs can use it all
decoupled failure domain
low cost
most transparent

SAN must be provisioned for burst
interference possible
most transparent

coupled failure domain

when job fails, last *bursted* checkpoint is lost

must restart from last *drained* checkpoint

–or– must parity protect burst buffer

decoupled failure domains

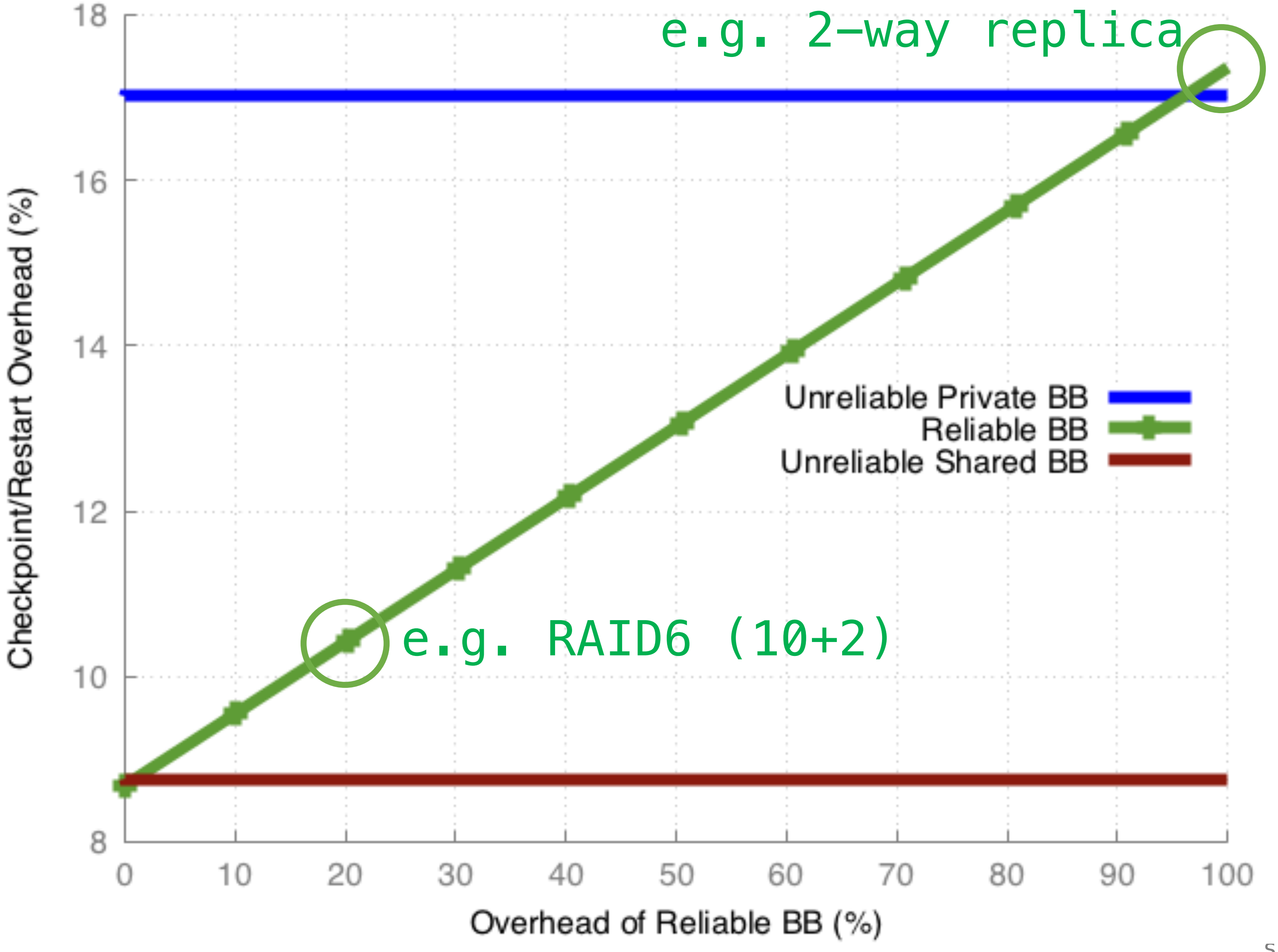
when job fails, restart from last *bursted*

when bb fails, job just continues

[assumes some failure resilience not yet avail]

decoupled does not need parity

the value of decoupled failure domains



observations

shared doesn't need parity
private does
...
but then the
high private perf is lost

Bent, Settlemyer, et al.
On the non-suitability of
non-volatility.
HotStorage '15.

the value of shared for bandwidth

	Local Unreliable	Local 20% Parity	Shared Unreliable
Mean Ckpt Bw	206.8 GB/s		

simulation of APEX workflows running on Trinity

Lei Cao, Bradley Settlemyer, and John Bent. To share or not to share: Comparing burst buffer architectures. SpringSim 2017.

the value of shared for bandwidth

	Local Unreliable	Local 20% Parity	Shared Unreliable
Mean Ckpt Bw	206.8 GB/s	165.6 GB/s	

simulation of APEX workflows running on Trinity

Lei Cao, Bradley Settlemyer, and John Bent. To share or not to share: Comparing burst buffer architectures. SpringSim 2017.

the value of shared for bandwidth

	Local Unreliable	Local 20% Parity	Shared Unreliable
Mean Ckpt Bw	206.8 GB/s	165.6 GB/s	614.54 GB/s

simulation of APEX workflows running on Trinity

observation: capacity machines need shared burst buffers

Lei Cao, Bradley Settlemyer, and John Bent. To share or not to share: Comparing burst buffer architectures. SpringSim 2017.

Thanks!

john.bent@seagategov.com