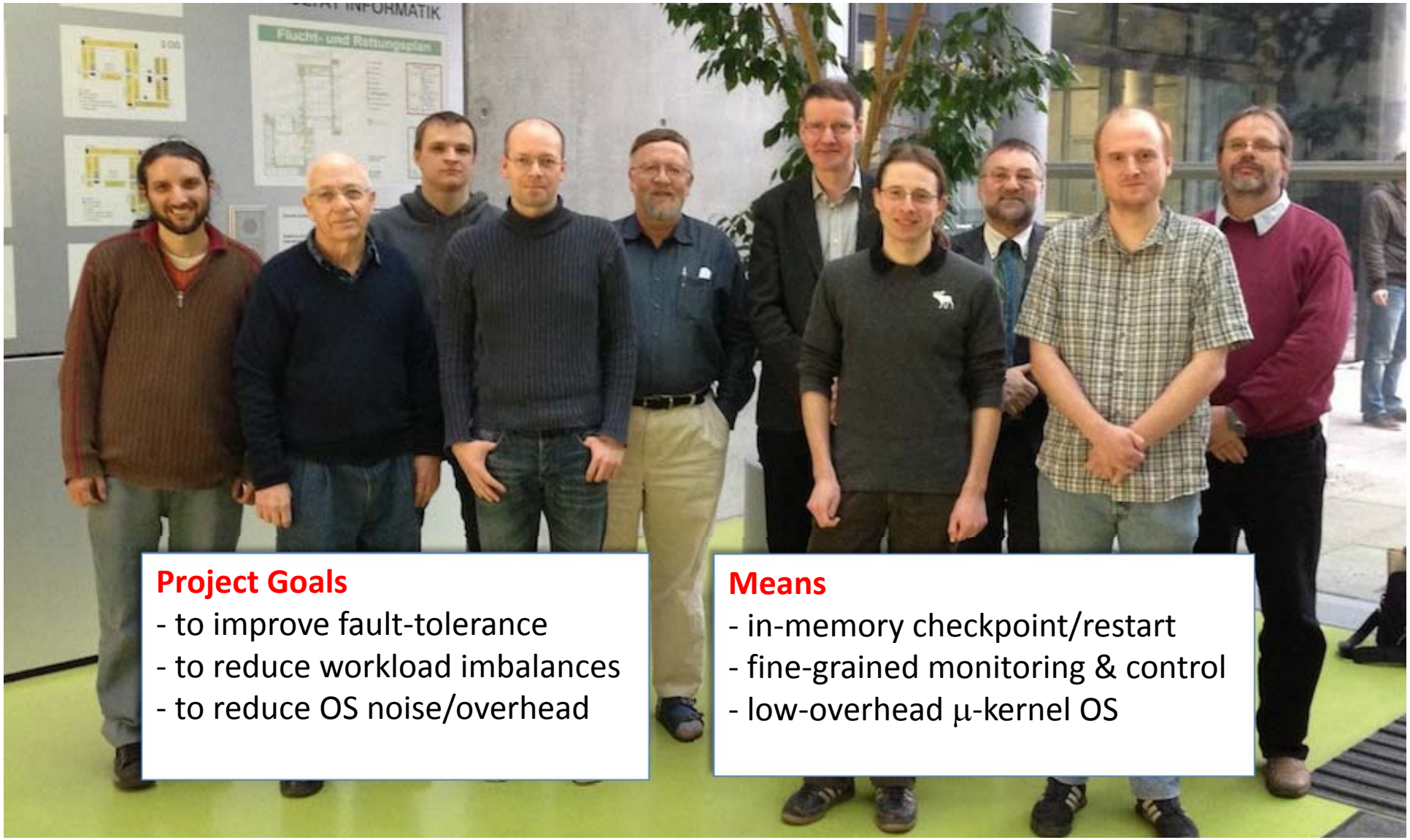
Fast In-Memory Checkpointing with POSIX API for Legacy Exascale-Applications

Jan Fajerski, Matthias Noack, Alexander Reinefeld,
Florian Schintke, Thorsten Schütt, Thomas Steinke

Zuse Institute Berlin

SPPEXA Symposium, 26.01.2016

FFMK: A Fast and Fault-tolerant Microkernel-based Operating System for Exascale Computing

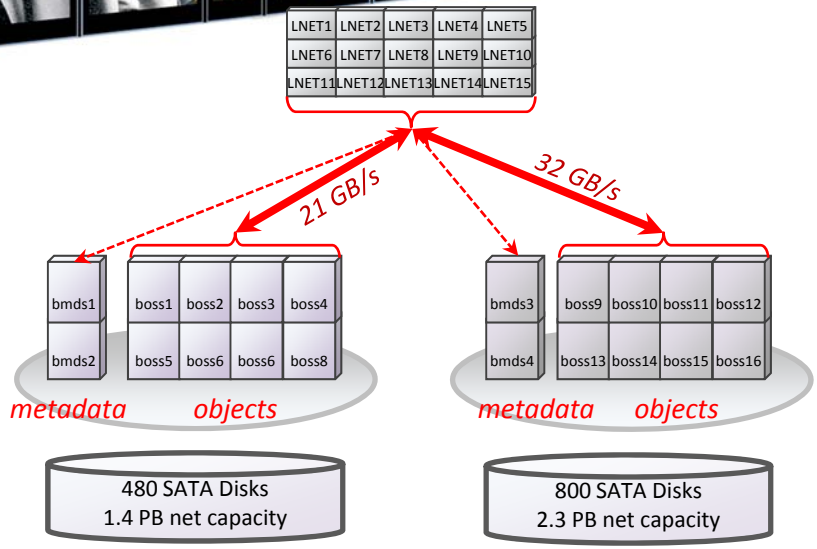


Outline

1. In-memory C/R
   - First Results on Cray XC40
2. Reed-Solomon Erasure Codes in Distributed, Unreliable Systems
3. Restart
   - Process placement after restart
   - Oversubscription after restart

Checkpointing Today

Cray XC40 “Konrad” @ZIB
- 1 Petaflop/s peak
- 1872 nodes
- 44928 cores
- 117 TB memory



Need **60 minutes** to write
Konrad‘s main memory
to Lustre 2 (at peak bandwidth)

Checkpointing on ExaScale

- Exascale systems
  - more components
  - more complexity

  → With today's methods the time to save the checkpoint **may be larger than MTBF.**

- Must reduce …
  - **data size** → **application-level checkpointing**
    - can use multilevel checkpointing
    - utilize storage layers, e.g. NVRAM
  - **writing time** → **stripe over many remote DRAMs**
    - via fast interconnect
    - but need redundant storage (EC)

Goal: Checkpoint frequency of a few minutes

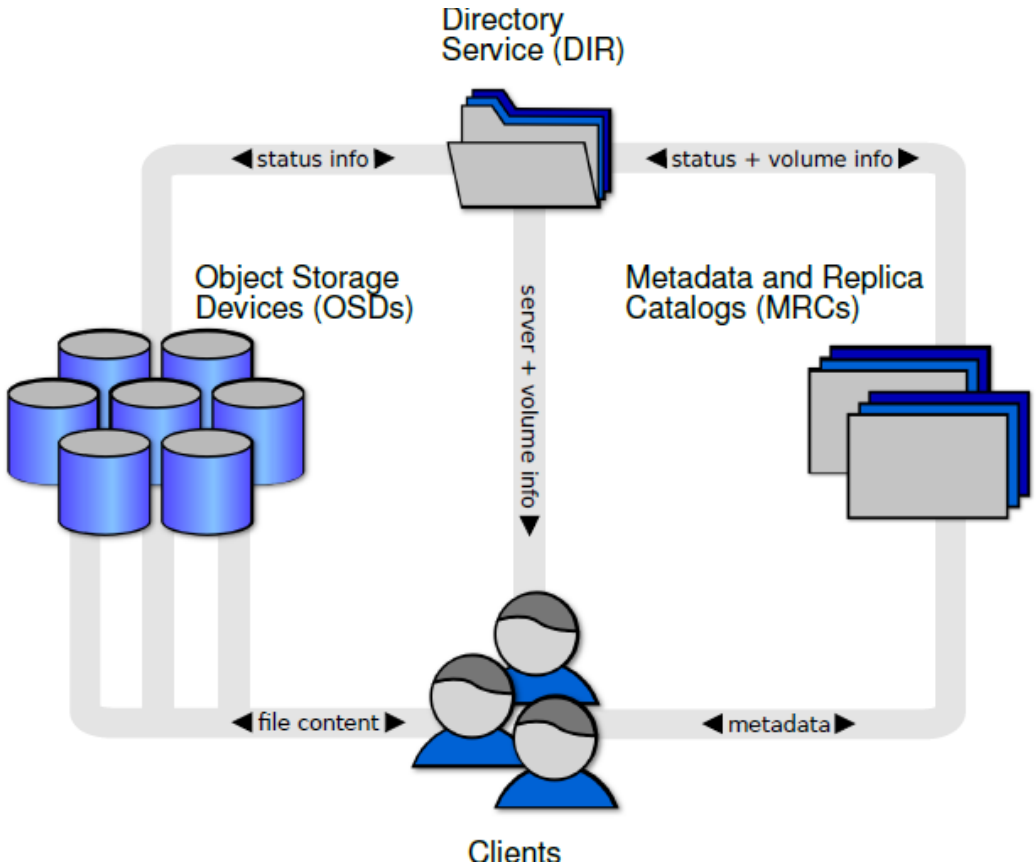
- Need POSIX compliance
  - for legacy applications

In-Memory C/R

Cray XC40 implementation and first results

In-Memory C/R with XtreemFS

- We use **XtreemFS**, a *user space* file system.
  - OSD data is stored in the *tmpfs* file system (RAM disk)



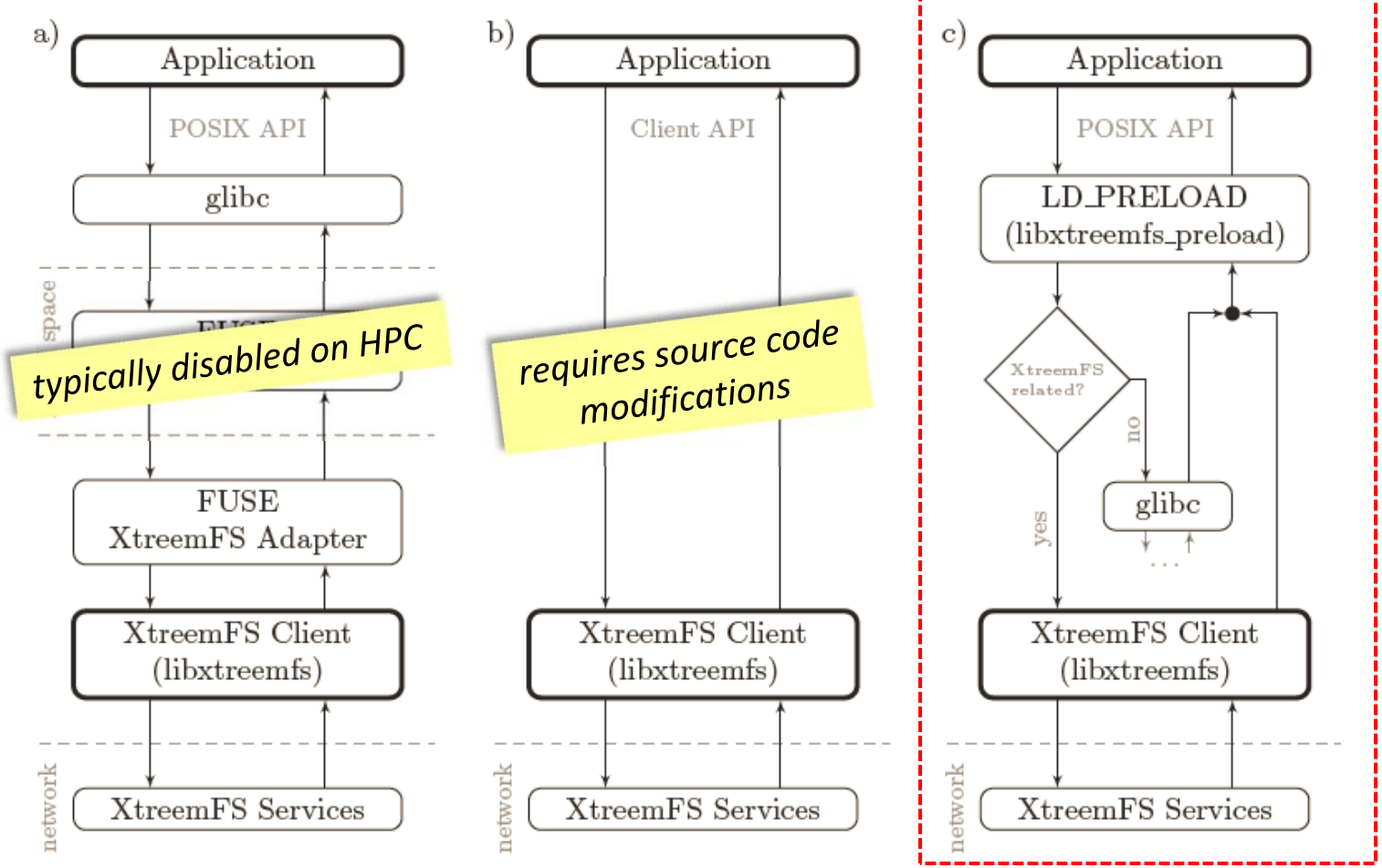
Stender, Berlin, Reinefeld. *XtreemFS - a File System for the Cloud*, In: Data Intensive Storage Services for Cloud Environments, IGI Global, 2013.

Accessing Remote RAM File System: 3 Options

FUSE (file system in user space)
requires kernel modules

libxtreemfs
link the library
to client code

LD_PRELOAD
intercepts calls to DLLs

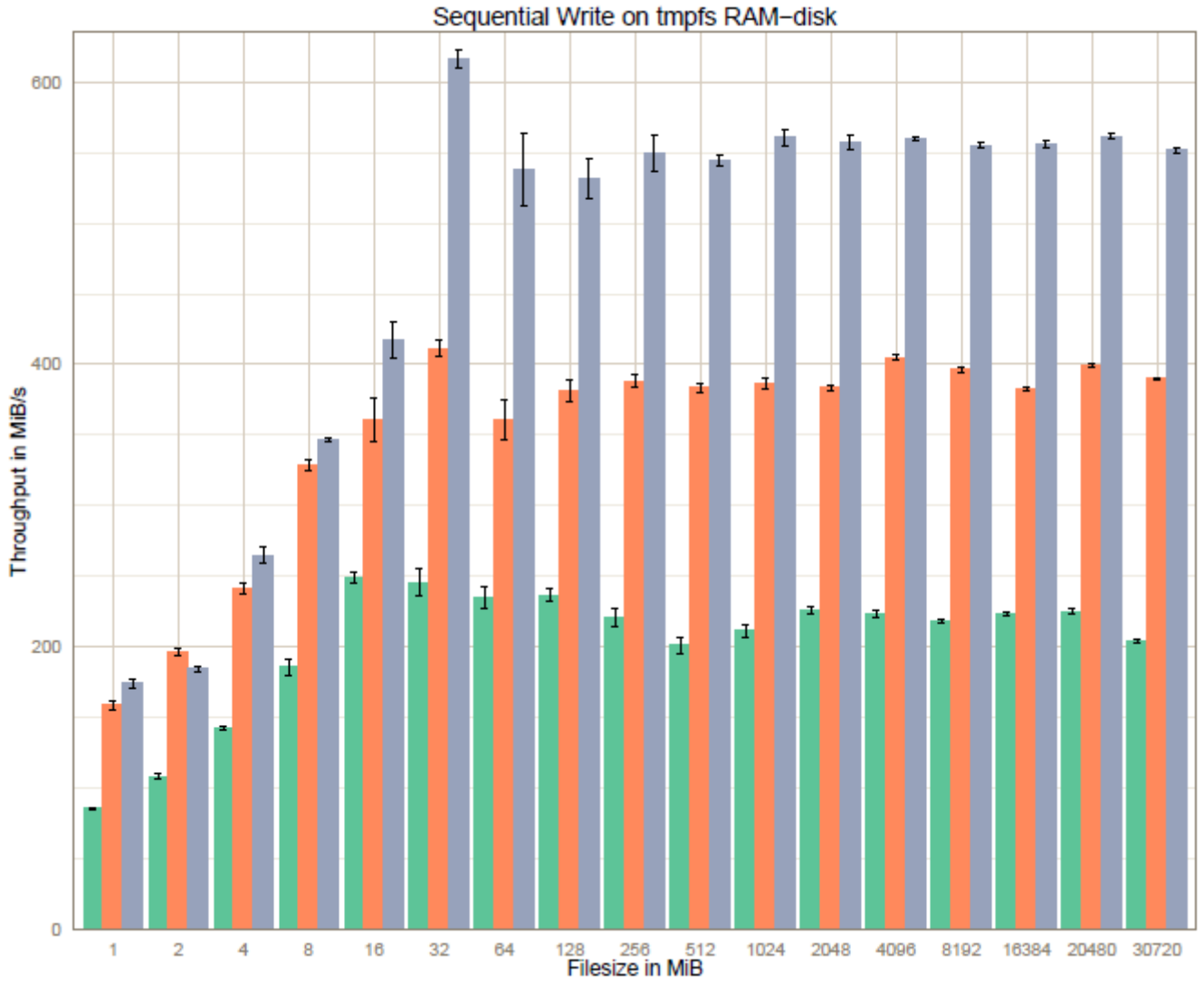


Deployment on a Cray XC40

- Experimental results with **BQCD** code with application-level C/R
  - OpenMPI
    - Node 1: DIR, MRC
    - Node 1...n: 1 OSD each
  - We manually killed the job end restarted it from the checkpoint.

- Used CCM mode (cluster compatibility mode) on Cray
  - need to run XtreemFS and application side-by-side
  - need to be able to restart the checkpoint
  - ibverbs (emulates IB on Aries)

Results (1): Sequential Write on Remote RAM Disk



Client uses:

libxtreemfs

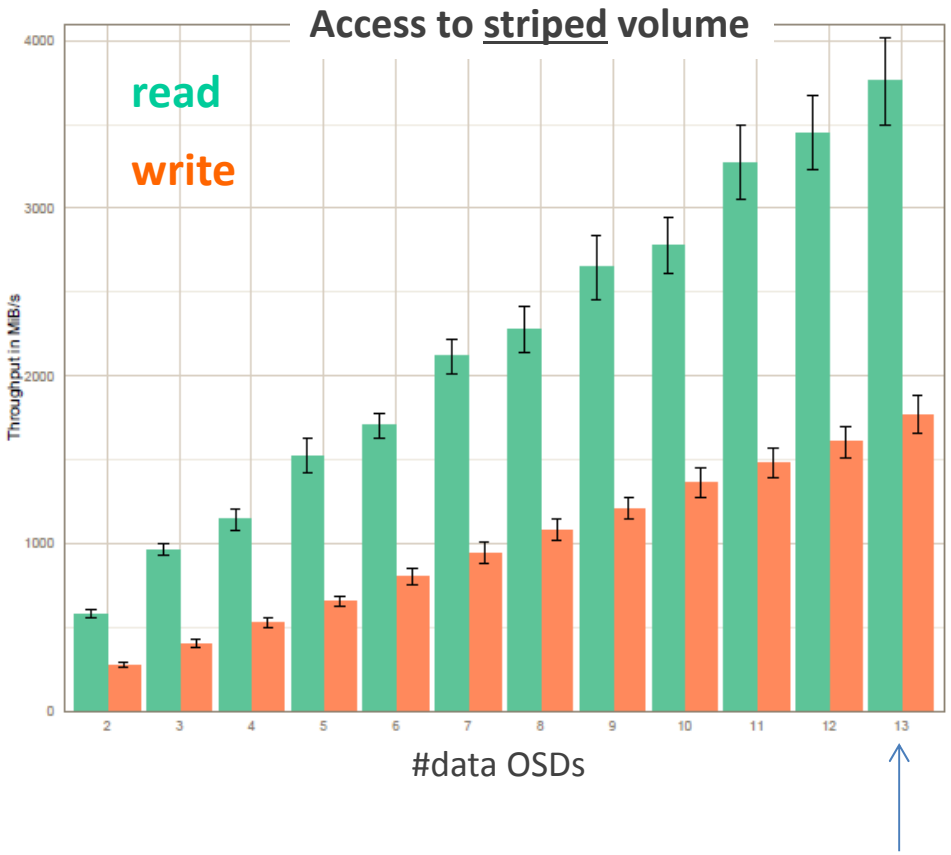
LD_PRELOAD

FUSE

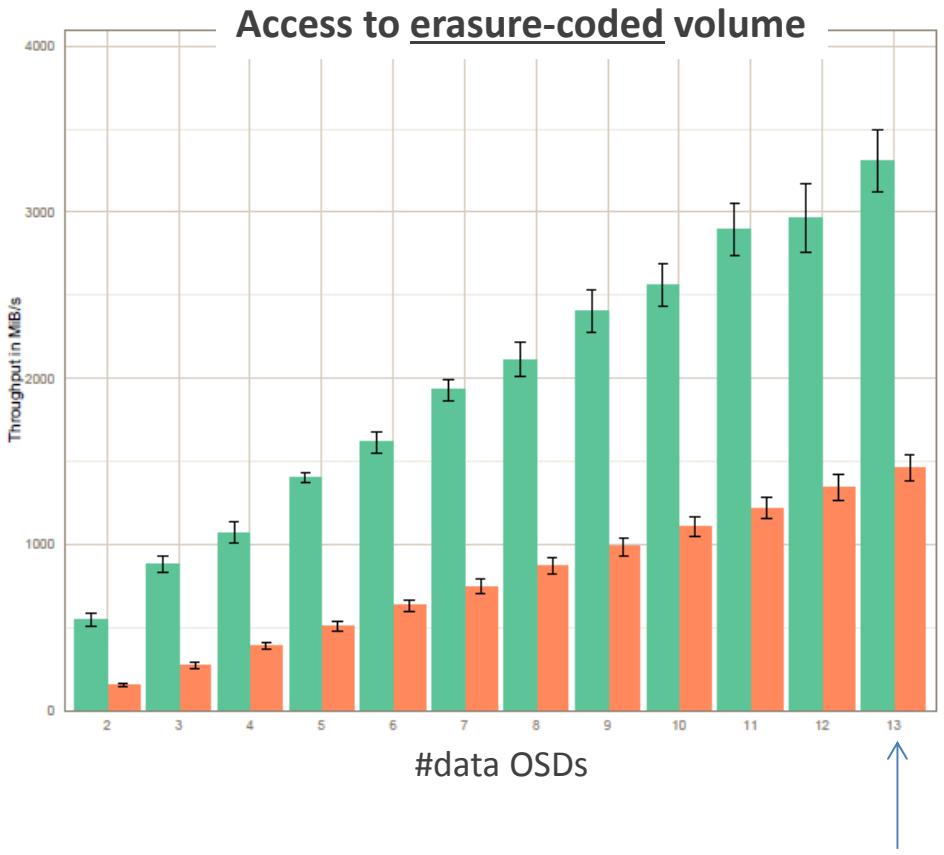
Limited bandwidth because (1) synchronous access pattern, (2) single data stream, (3) single client

Results (2): Writing Erasure Coded Data

n = 2 ... 13 clients, each writes 1 GB



13 nodes with 1 OSD and 1 client each



13 nodes with 1 OSD and 1 client each **+ 2 redundancy nodes**

Writing encoded data is **17% to 49% slower**
than writing striped data (15 .. 100% more data)

Using Reed-Solomon Erasure Codes in Distributed, Unreliable Systems

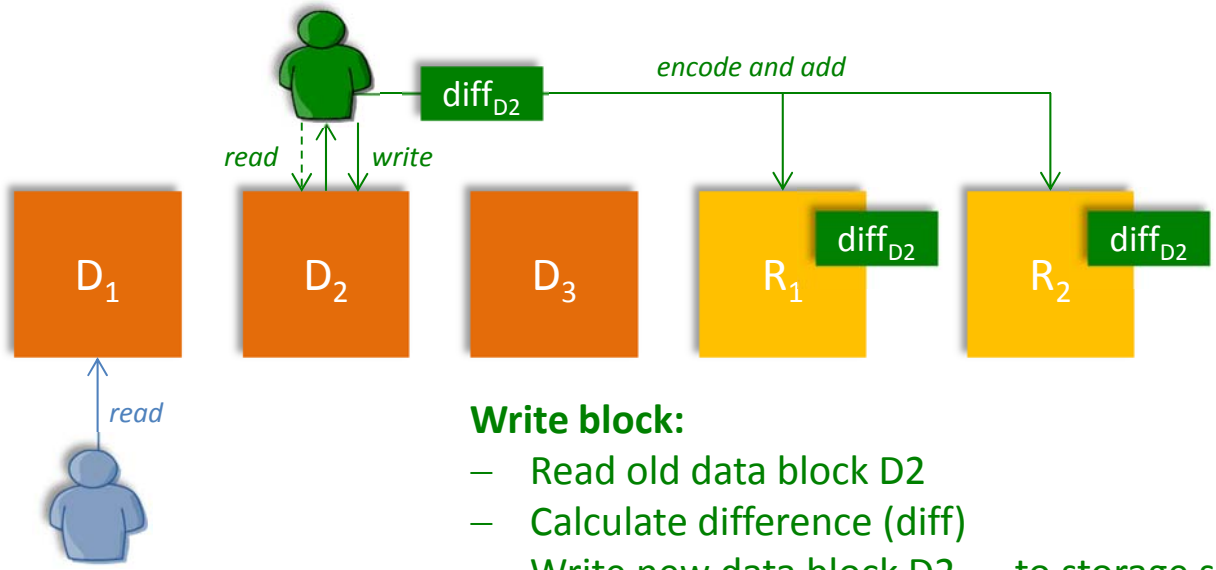
Update Order Problem

RS Erasure Codes

- RS Erasure Codes provide more flexibility than RAID
  - $n = k + m$, for various k , m
- But blocks in one stripe are dependent on each other
  - not so in replication
- Must ensure **sequential consistency** for client accesses
  - w/ failing servers
  - w/ failing (other) clients

RS Erasure Codes

MDS erasure code with k=3 original blocks and m=2 redundancy blocks



Read block:

- Read data block D1

Write block:

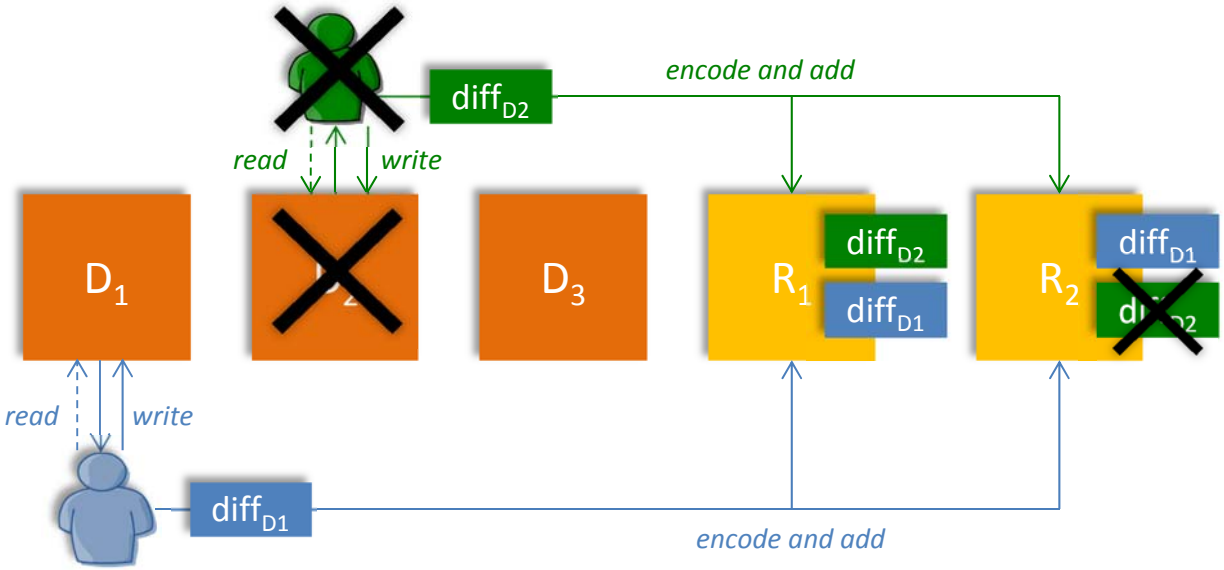
- Read old data block D2
- Calculate difference (diff)
- Write new data block $D2_{new}$ to storage server
- Encode and add at redundancy servers R_1, R_2 (commutative)

Recovery:

- Read any k=3 blocks out of k + m, recover lost blocks

Update Order Problem

Concurrent Updates of data blocks D1 and D2



Problem:

- Update order of redundancy blocks R1, R2
- Inconsistency in case of failures (data block or client failure)

2 Solutions for Update Order Problem

- **PSW**: **P**essimistic Protocol with **S**equential **W**riting
  - a master (sequencer) enforces the total order of updates on R_i
  - use separate master per file for scaling
- **OCW**: **O**ptimistic protocol with **C**oncurrent **W**riting
  - uses buffers and replicated state machine
- Performance
  - OCW is as fast as replication (RAID-1), but needs additional buffer space
  - PSW is slower, but no buffers

K. Peter, A. Reinefeld. *Consistency and fault tolerance for erasure-coded distributed storage systems*, DIDC 2012.

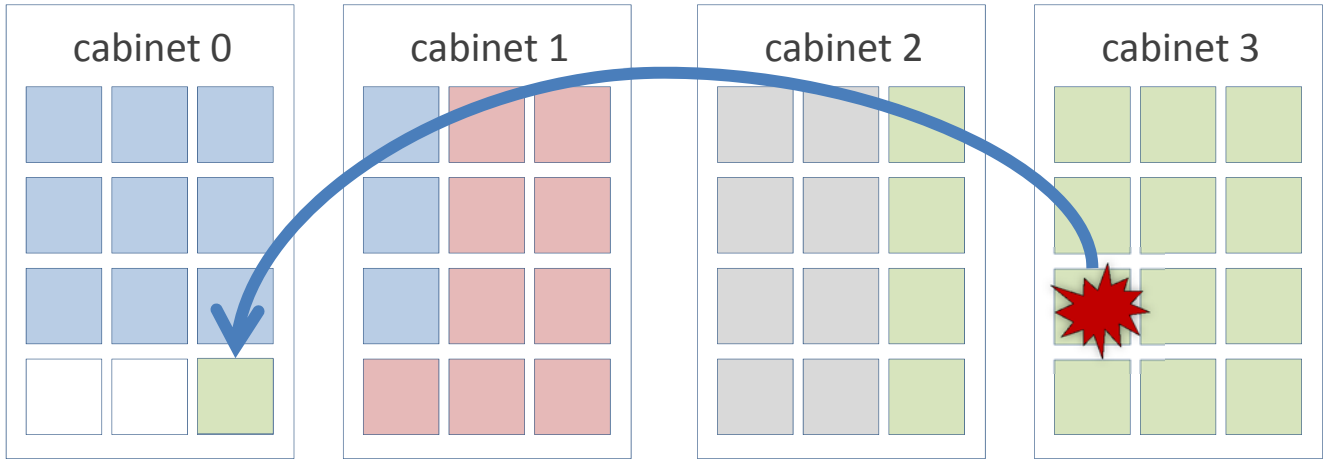
RESTART

Q1: Where to restart?

Q2: Oversubscribing or Downscaling?

Restart

Does it matter where to restart a crashed process?

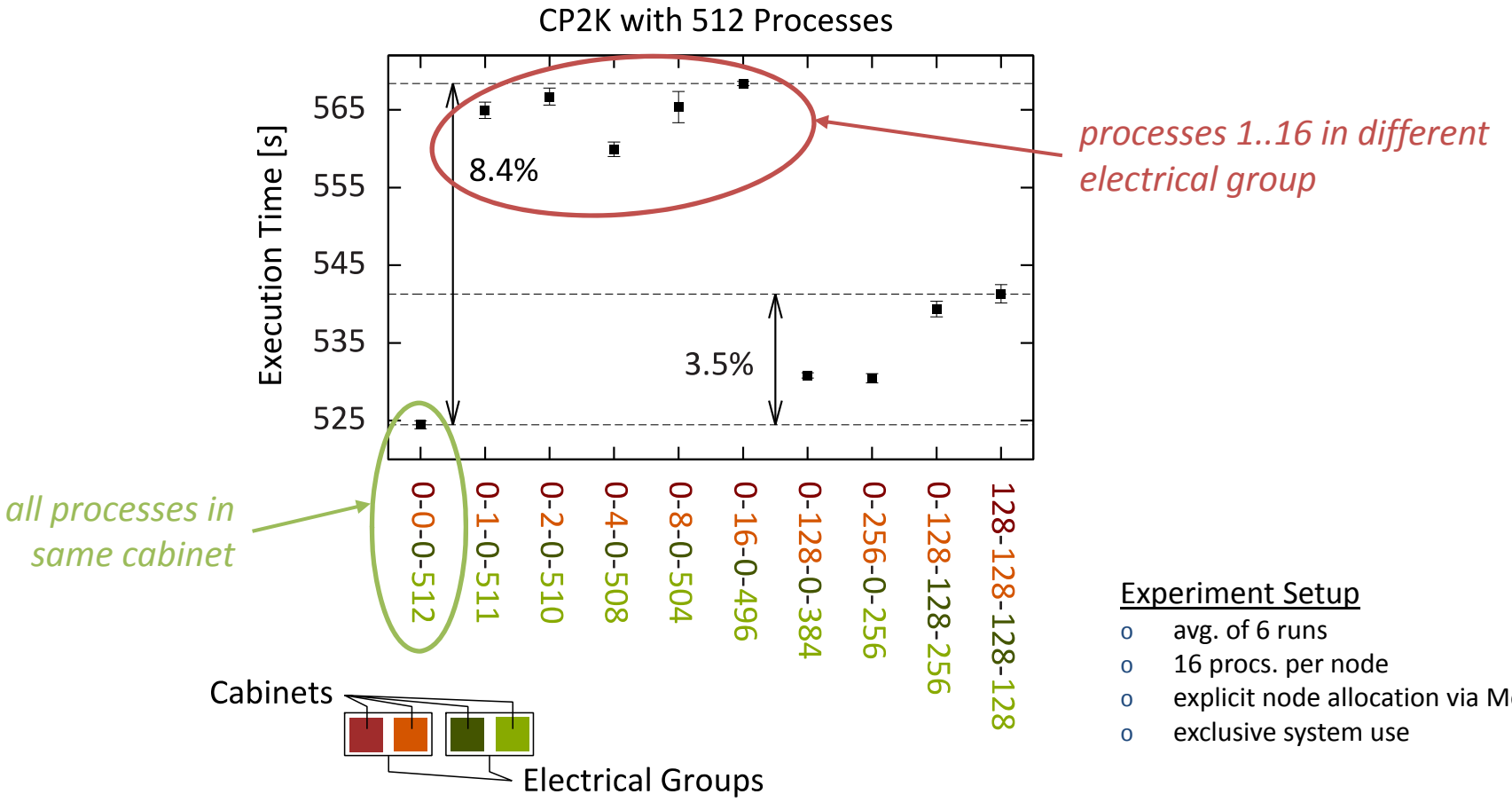


process restart after
node failure in cabinet 3

F. Wende, Th. Steinke, A. Reinefeld, *The Impact of Process Placement and Oversubscription on Application Performance: A Case Study for Exascale Computing,* EASC-2015.

Process Placement: **CP2K on Cray XC40**

- CP2K setup: H_2O-1024 with 5 MD steps
- Placement across 4 cabinets is (color)encoded into string C1-C2-C3-C4

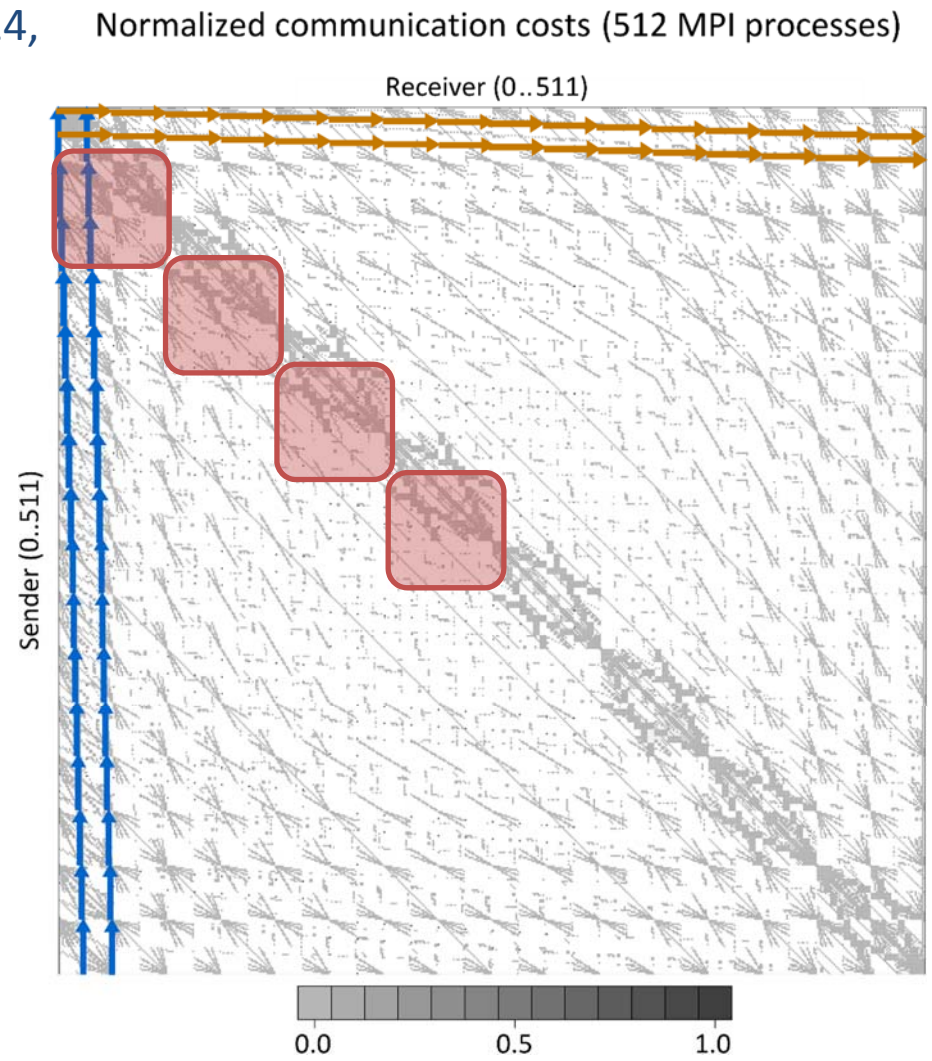


Experiment Setup

- avg. of 6 runs
- 16 procs. per node
- explicit node allocation via Moab
- exclusive system use

Process Placement: **CP2K on Cray XC40**

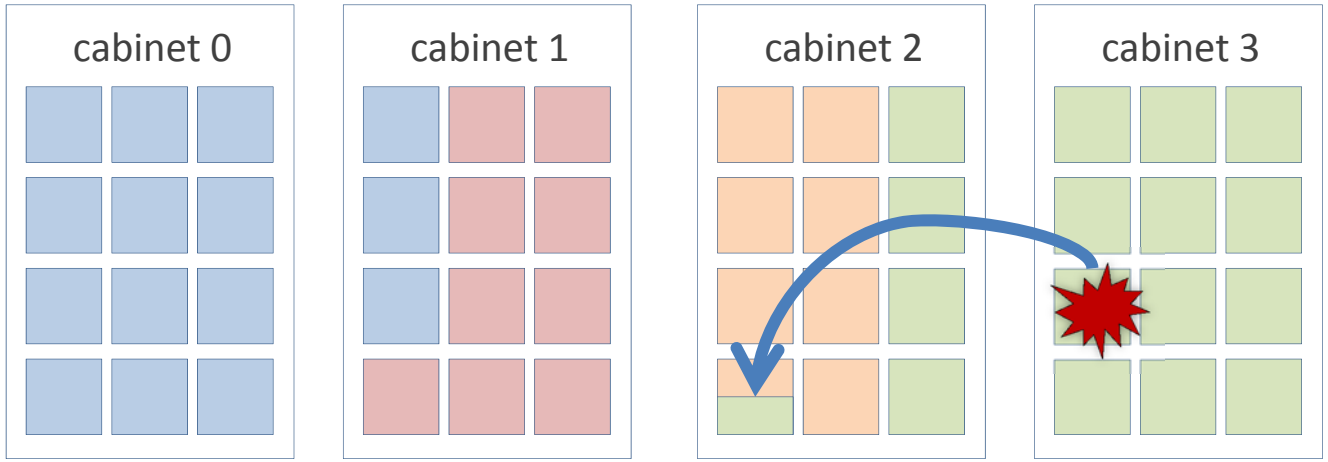
- Communication matrix for H_2O-1024, 512 MPI processes
  - Some MPI ranks are src./dest. of **gather** and **scatter** operations → Placing them far away from other processes may cause performance decrease
  - **Intra-group** and **nearest neighbor** communication



Notes:
- *tracing* experiment with CrayPAT
- some comm. paths pruned away

Restart without free nodes

Oversubscribing or downscaling the application?

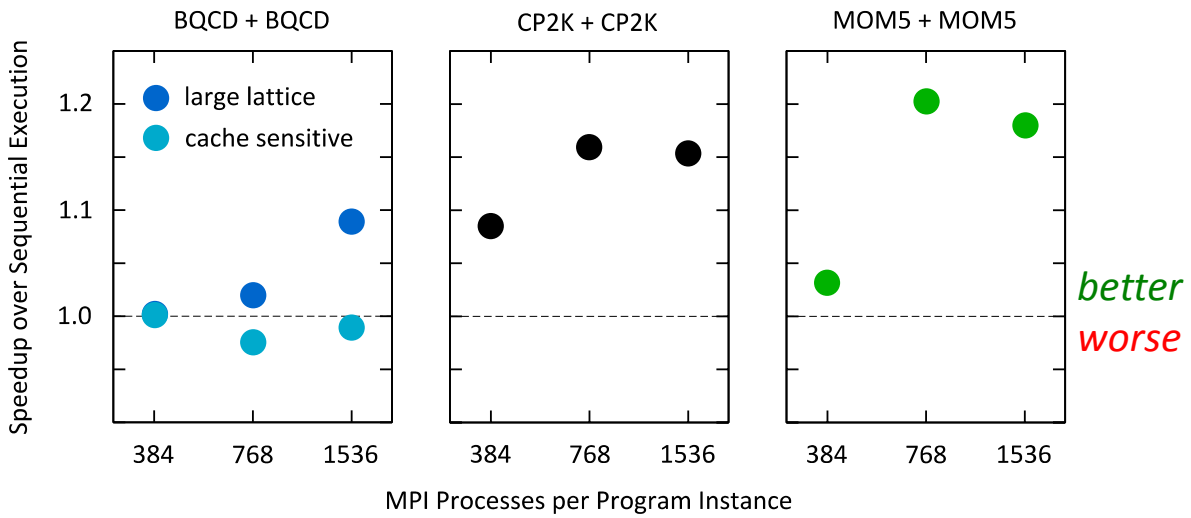


Oversubscribing in cabinet 2
after node failure in cabinet 3 ?

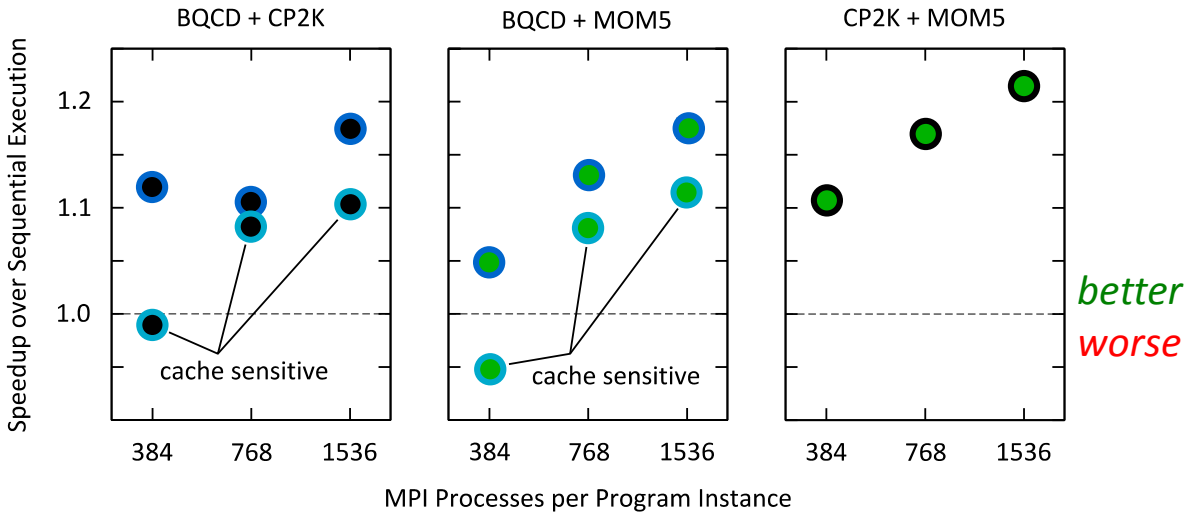
F. Wende, Th. Steinke, A. Reinefeld, *The Impact of Process Placement and Oversubscription on Application Performance: A Case Study for Exascale Computing,* EASC-2015.

Oversubscription

Oversubscription with the **same application**



Oversubscription with **different applications**



Oversubscribing for the whole runtime on Cray XC40, ALPS_APP_PE

Summary & Outlook

- We demonstrated the feasibility of in-memory erasure-coded C/R on HPC
  - supports legacy applications (POSIX)

- Next steps
  - implement **full Reed Solomon** erasure codes with *Jerasure* lib
  - **data scheduling**: checkpointing should not impact running applications
  - **improve coding speed** on manycore, coprocessor, …

- Publications
  - F. Wende, Th. Steinke, A. Reinefeld, *The Impact of Process Placement and Oversubscription on Application Performance: A Case Study for Exascale Computing*, EASC-2015.
  - Ch. Kleineweber, A. Reinefeld, T. Schütt. *QoS-Aware Storage Virtualization for Cloud File Systems.* 1st Programmable File Systems Workshop, HPDC'14
  - K. Peter, A. Reinefeld. *Consistency and fault tolerance for erasure-coded distributed storage systems*, DIDC 2012.
  - H. Härtig, S. Matsuoka, F. Müller, A. Reinefeld. *Resilience in Exascale Computing*, Dagstuhl Reports, vol. 4, no. 9, pp.124-139, doi: 10.4230/DagRep.4.9.124