AI for HPC:

- Data Compression and System Software Optimization -

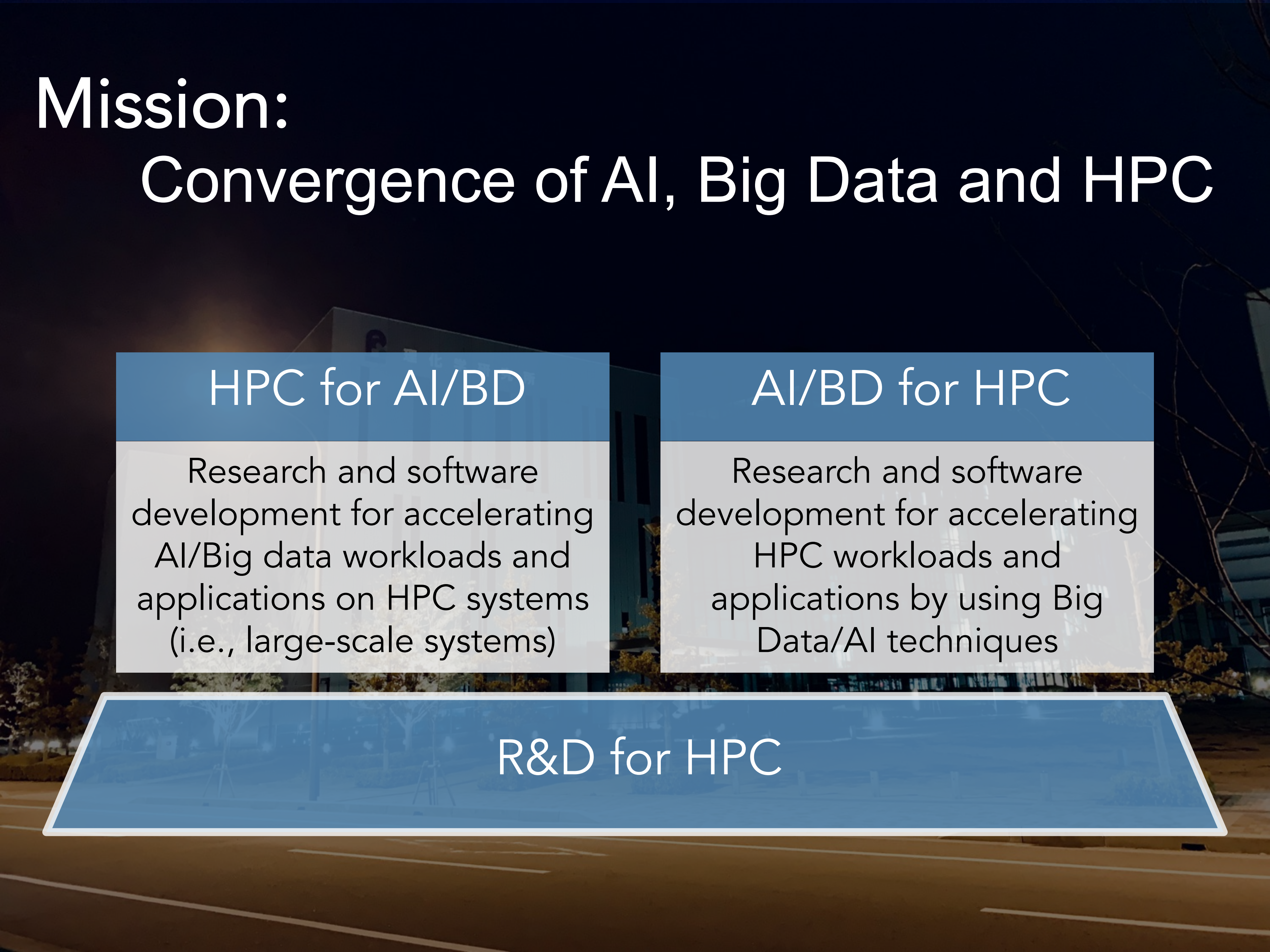
France-Japan-Germany trilateral workshop
Convergence of HPC and Bata Science
for Future Extreme Scale Intelligent Applications
November 6th-8th, 2019

Kento Sato

High Performance Big data Research Team

RIKEN R-CCS

Mission:
Convergence of AI, Big Data and HPC



HPC for AI/BD

Research and software development for accelerating AI/Big data workloads and applications on HPC systems (i.e., large-scale systems)

AI/BD for HPC

Research and software development for accelerating HPC workloads and applications by using Big Data/AI techniques

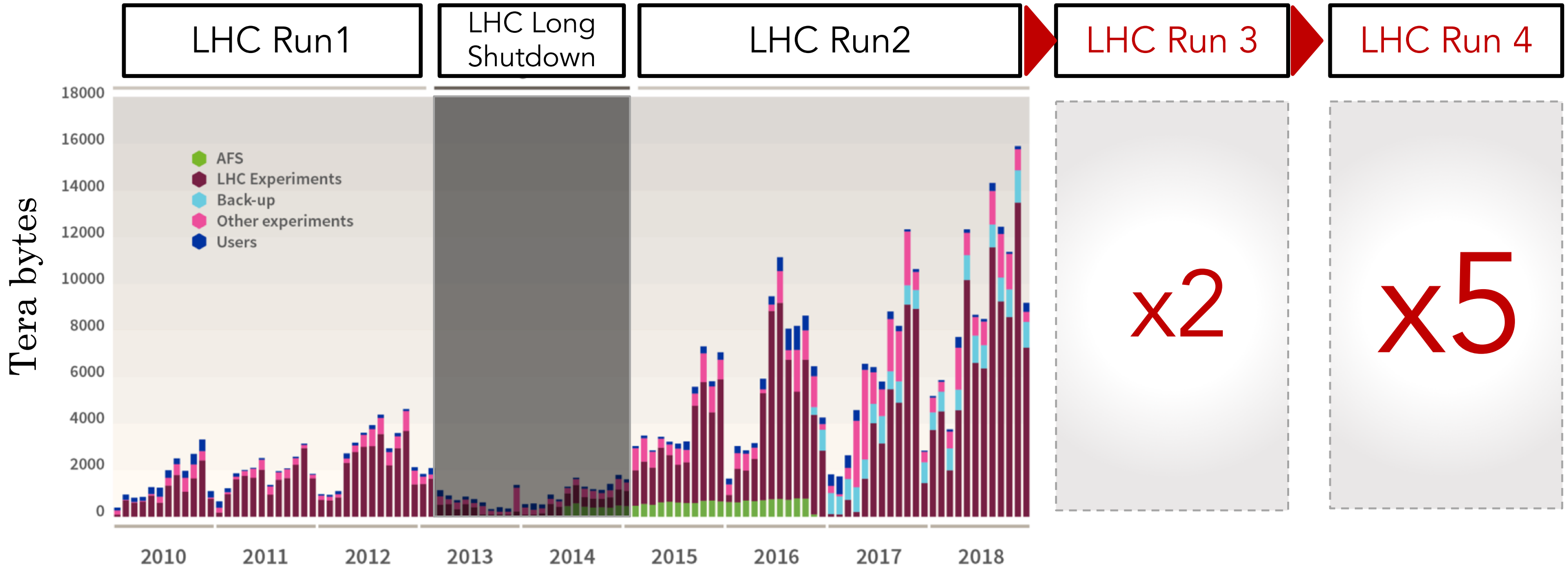
R&D for HPC

Current research topics

- R&D for HPC
  - Reproducibility in MPI/OpenMP applications
  - Design space exploration for the Post-Moore era
- AI/BD for HPC
  - Big data compression with AI techniques
  - System software optimization with AI techniques
  - System log analysis and prediction
- HPC for AI/Big data
  - Deep learning framework tuning on HPC systems

Big Data Generation and Transfer

- **Generation**: Scientific big data is generated every day all over the world
  - LHC (Large Hadron Collider) in CERN generated about 88PB of data in 2018 [1]
    - *"Data archival is expected to be two-times higher during Run 3 and five-times higher or more during Run 4 (foreseen for 2026 to 2029). "*

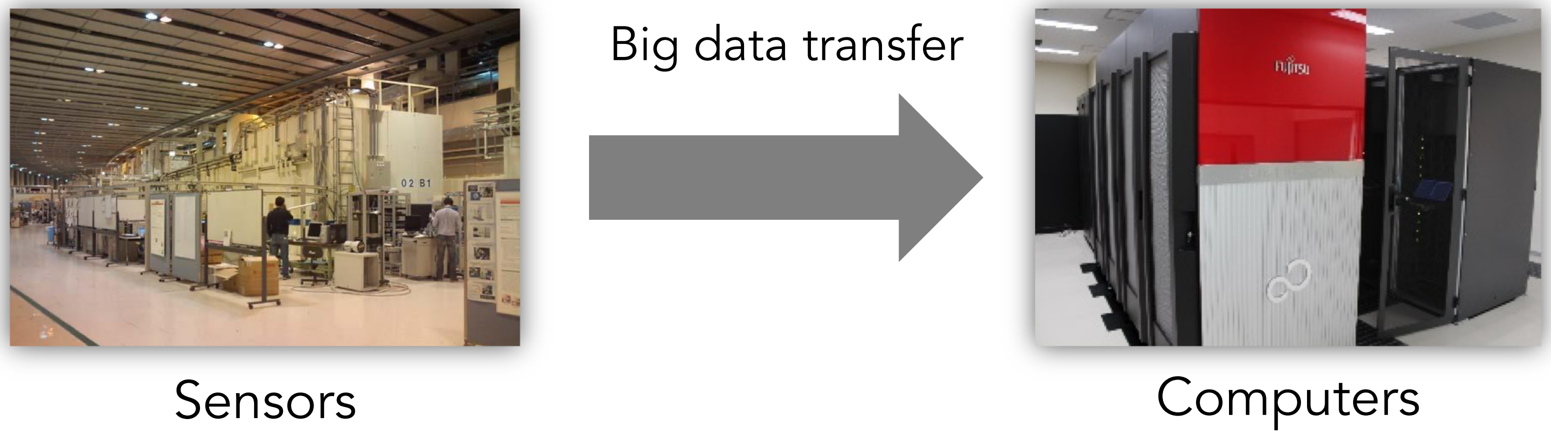


---

[1] Esra Ozcesmeci, "LHC: pushing computing to the limits", https://home.cern/news/news/computing/lhc-pushing-computing-limits

Big Data Generation and Transfer (Cont'd)

- **Transfer**: Data transfer is an essential part of data analytics
  - Big data transfer from sensors to computers
    - Generated data from sensors must be transferred to internal/external computers for the analysis
    - The facilities needs to transfer the data to external collaborators via WAN
      - e.g.) In LHC, 830 PB of data and 1.1 billion files were transferred all over the world [1]



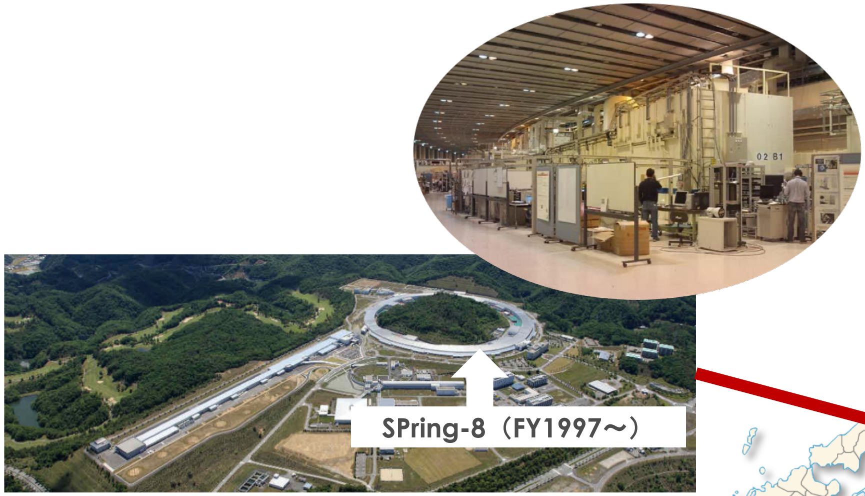
Efficient data transfer is important in big data analysis

---

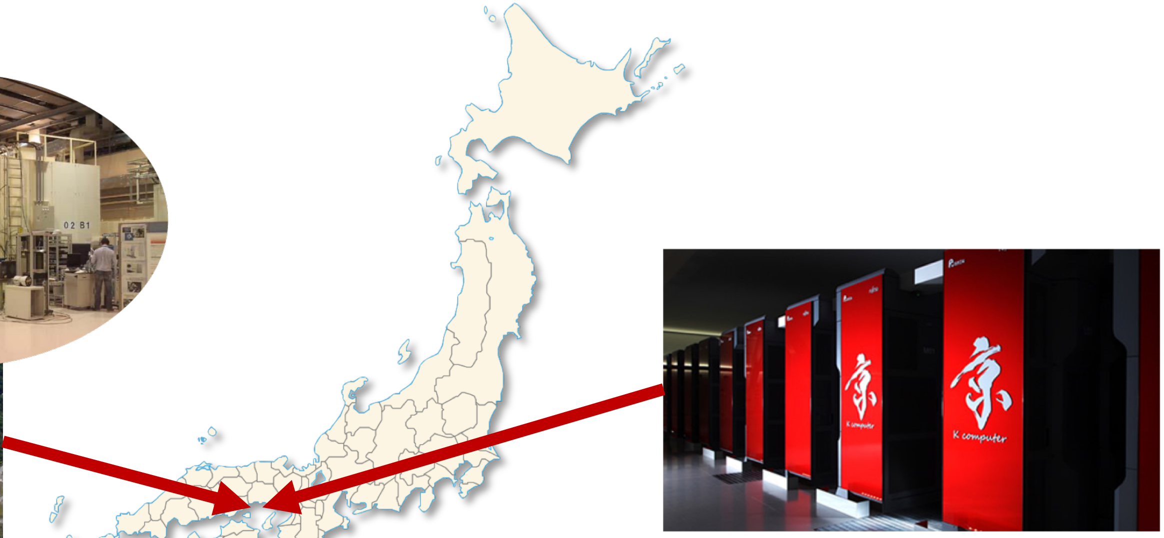
[1] Esra Ozcesmeci, "LHC: pushing computing to the limits", https://home.cern/news/news/computing/lhc-pushing-computing-limits

SPring-8

- RIKEN has SPring-8 large synchrotron radiation facility generating PB-order of big data
  - Opened in 1997 in Harima, located in the same Hyogo prefecture as R-CCS
  - Managed by RIKEN, with Japan synchrotron radiation research institute (JASRI)
  - SPring-8 stands for Super Photon ring-8 GeV
    - 8 GeV (giga electron volts) is the energy of electron beam circulation in the storage ring



SPring-8 RIKEN SPring-8 Center (RSC)



RIKEN R-CCS RIKEN Center for Computational Science

Big data transfer in SPring-8

- SPring-8 public beamline (26 BLs) generated 0.32 PB/year in 2017
- With the next generation detector (CITIUS), it is projected that the facility will generate 1.3 ExaB of raw data per year in 2025
  - Actual transfer size can be reduced to 100-400 PB by
    - Image averaging/extraction
    - Reducing duty ratio to throttle data generation rate

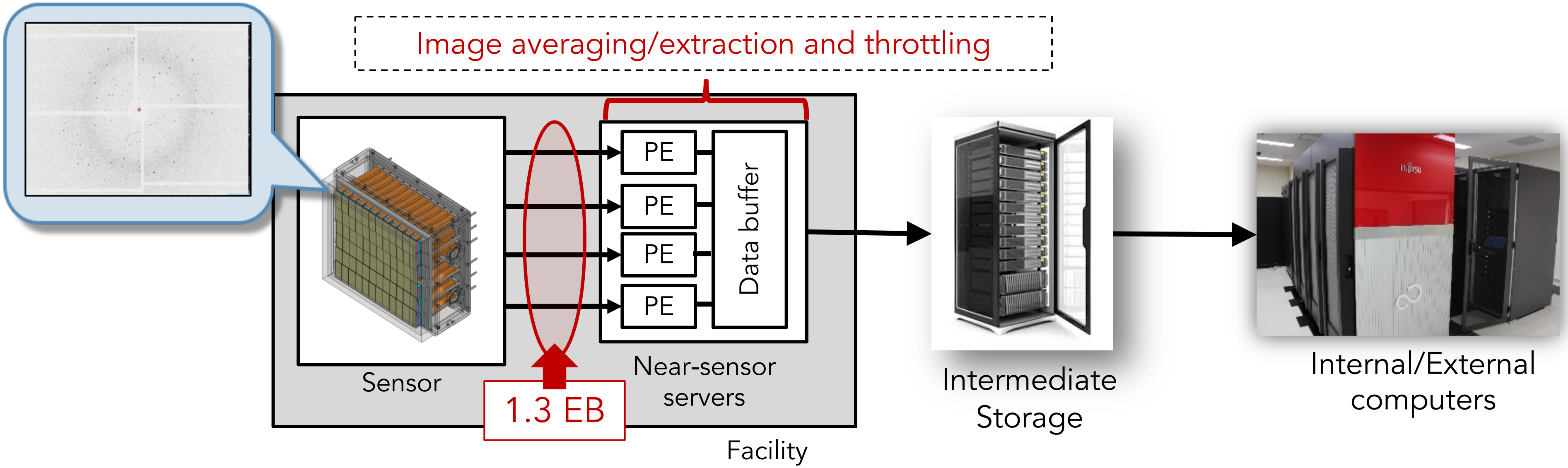


Image averaging/extraction and throttling

Sensor

PE

PE

PE

PE

Data buffer

1.3 EB

Near-sensor servers

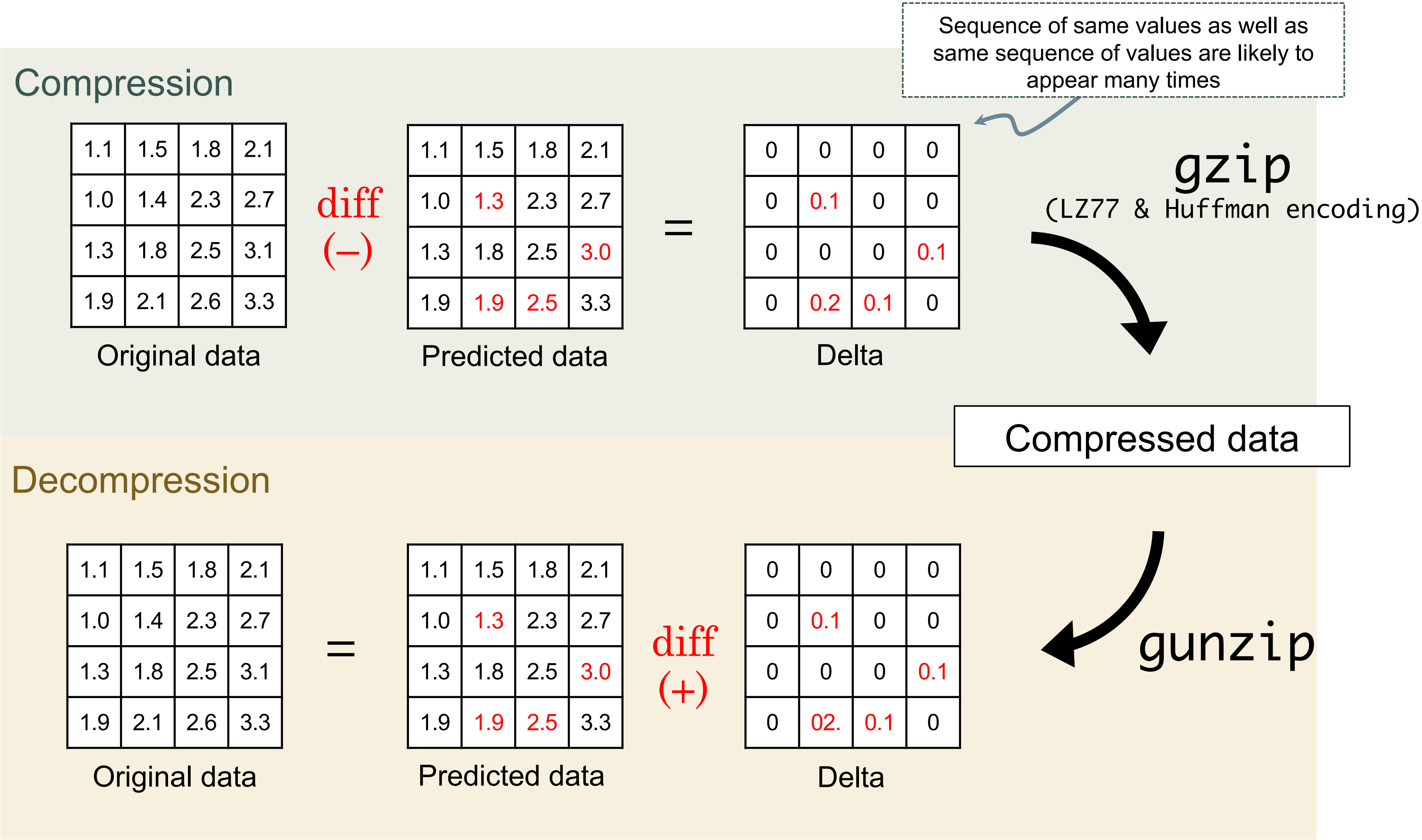
Facility

Intermediate Storage

Internal/External computers

We are trying to compress big data
to accelerate data transfer from sensors to HPC systems

Prediction is one of keys for good compression



Sequence of same values as well as same sequence of values are likely to appear many times

Compression

Original data

1.1	1.5	1.8	2.1
1.0	1.4	2.3	2.7
1.3	1.8	2.5	3.1
1.9	2.1	2.6	3.3

diff (–)

Predicted data

1.1	1.5	1.8	2.1
1.0	1.3	2.3	2.7
1.3	1.8	2.5	3.0
1.9	1.9	2.5	3.3

=

Delta

0	0	0	0
0	0.1	0	0
0	0	0	0.1
0	0.2	0.1	0

`gzip`

`(LZ77 & Huffman encoding)`

Compressed data

Decompression

`gunzip`

Delta

0	0	0	0
0	0.1	0	0
0	0	0	0.1
0	02.	0.1	0

diff (+)

Predicted data

1.1	1.5	1.8	2.1
1.0	1.3	2.3	2.7
1.3	1.8	2.5	3.0
1.9	1.9	2.5	3.3

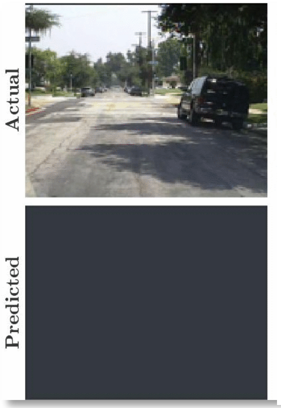
=

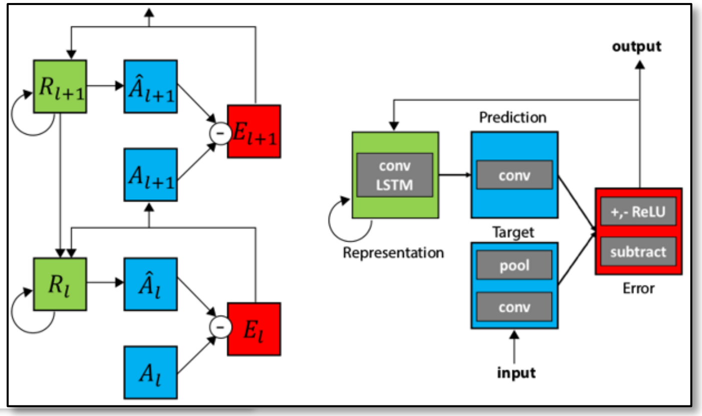
Original data

1.1	1.5	1.8	2.1
1.0	1.4	2.3	2.7
1.3	1.8	2.5	3.1
1.9	2.1	2.6	3.3

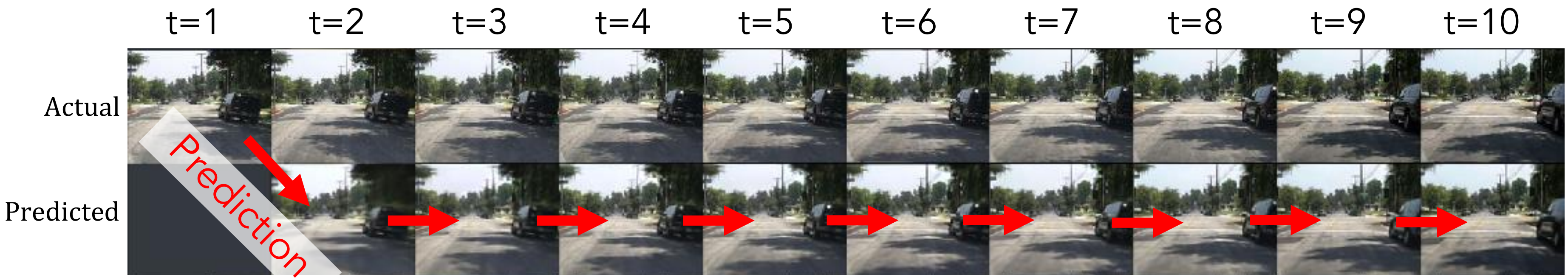
We use deep neural network (PredNet) for prediction

- PredNet [1]
  - Deep recurrent convolutional neural network
  - Given an frame of a picture, this NN can predict multiple future frames





https://coxlab.github.io/prednet/



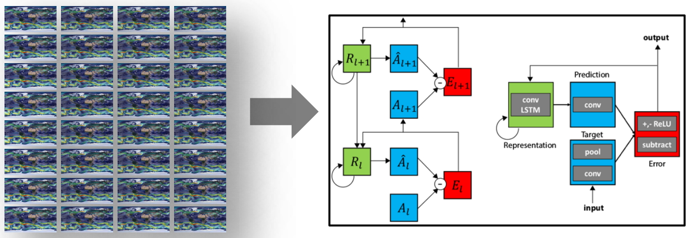
[1] Lotter, W., Kreiman, G., Cox, D.: Deep predictive coding networks for video prediction and unsupervised learning. arXiv preprint arXiv:1605.08104 (2016)

Predict future frames and compress data

- We train PredNet to learn how pixels move and how fast
  - i.e.) Giving a number of time evolutional frames to PredNet

- Example
  - When compressing frames from t=2 to t=5, we predict future frames from original data (t=1)
  - We compute diff, apply series of encoding
  - We only store (1) base frame data and (2) compressed data

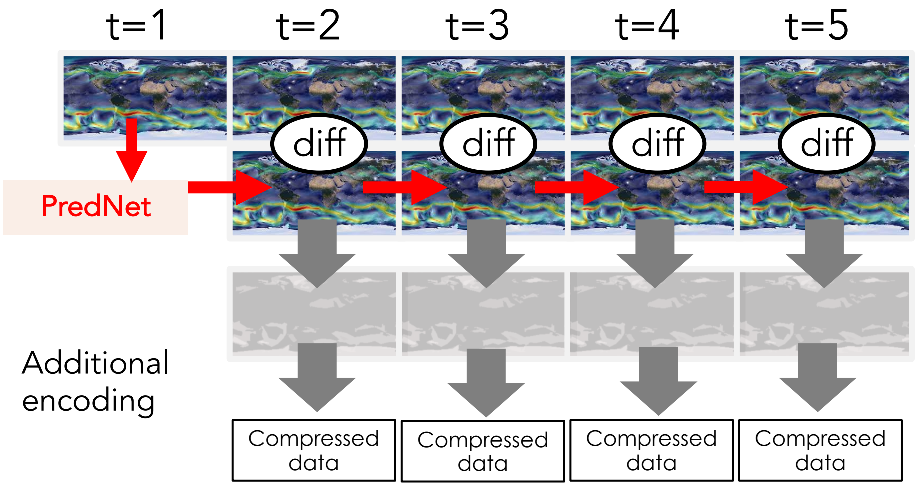
Training



Training data:
Time evolutional
frame data set

PredNet

Inference + Data compression



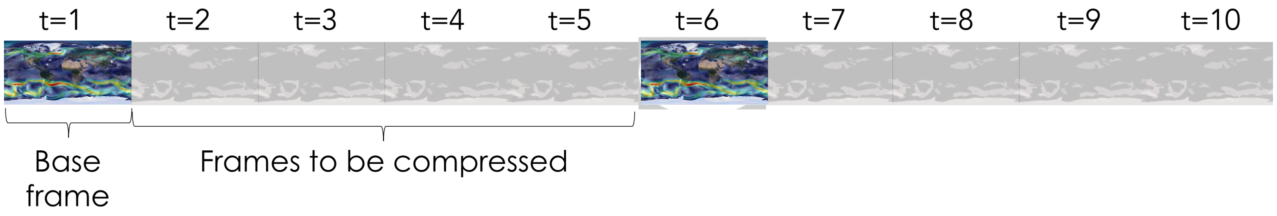
Other things to do

- Accelerate compression time with distributed GPUs

Compression time + I/O time (of compressed data) < I/O time (of original data)

GPU acceleration at large-scale

- Develop new predictive encoding NN to predict more future frames with higher accuracy
  - We use PredNet as a black box
  - We will extend PredNet for more accurate prediction
  - e.g.) Interval=5 → We store original data every 5 step and apply NN-based compression to the rest of frames.

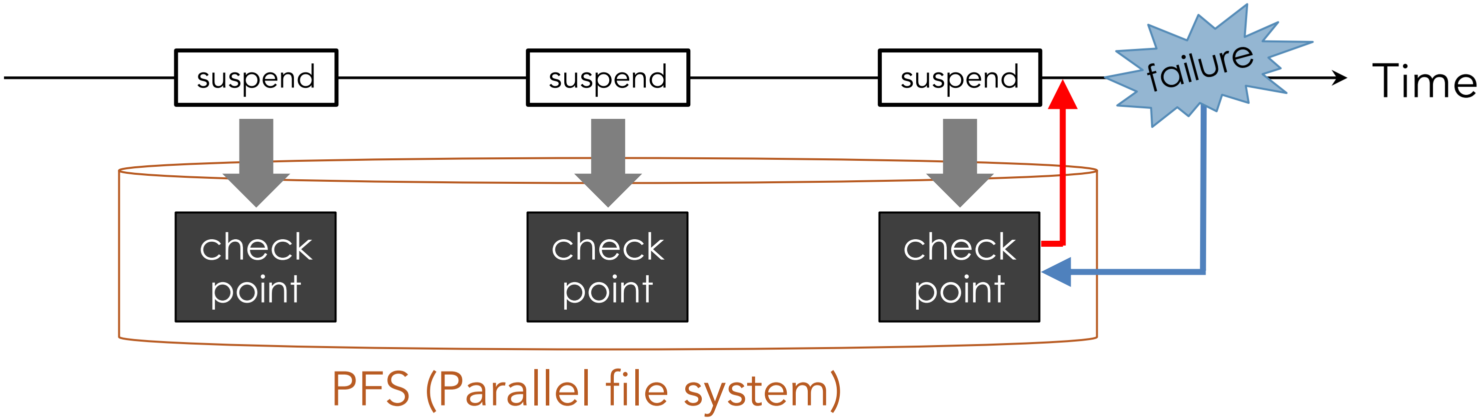


Current research topics

- R&D for HPC
  - Reproducibility in MPI applications
  - Design space exploration for the Post-Moore era
- AI/BD for HPC
  - Big data compression with AI techniques
  - System software optimization with AI techniques
  - System log analysis and prediction
- HPC for AI/Big data
  - Deep learning framework tuning on HPC systems

Checkpoint/restart

- Checkpoint-and-Restart is commonly used technique for large-scale applications running for long time
- Checkpoint/Restart
  - Write a snapshot of an application
  - On a failure, the application can restart from the last checkpoint
- Checkpoint/restart is one of major I/O workloads in HPC systems
  - PFS checkpoint: 10 – 30 mins



Efficient use of C/R is important for large-scale execution

Many configurations in C/R libraries

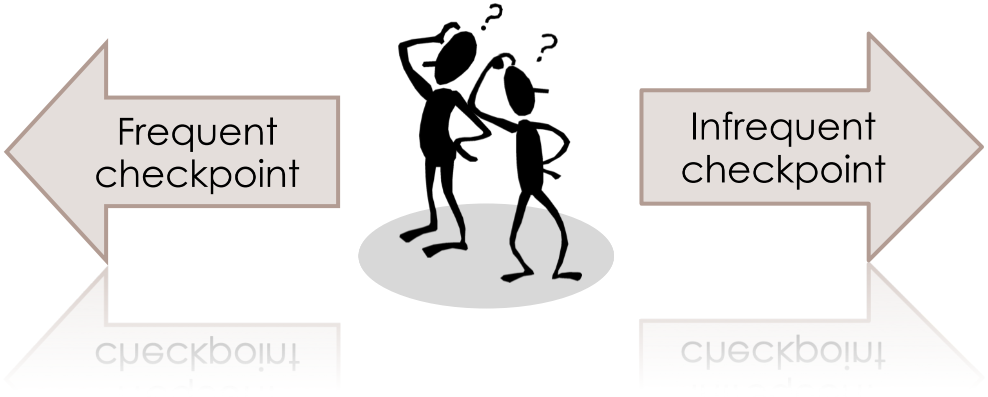
- Checkpoint location
  - Capacity v.s. Performance v.s. Reliability
- Checkpoint interval
  - Each level of checkpoint interval in multi-level checkpointing
- Erasure encoding
  - What erasure encoding should be used ?
  - Group size (or Failure group size)
- Many others …

Given an execution environment,
finding optimal configurations is challenging
as C/R scheme becomes more complicated

Finding optimal interval for efficient checkpointing

- Tradeoff
  - Frequent checkpoint: Unnecessarily spend more I/O time for checkpointing
  - Infrequent checkpoint: You may lose much more useful computation on a failure
- Even if you use state-of-the-art C/R techniques, poorly determined checkpoint intervals make system resilience worse than simple C/R

⇒ Finding optimal checkpoint interval is important for efficient C/R

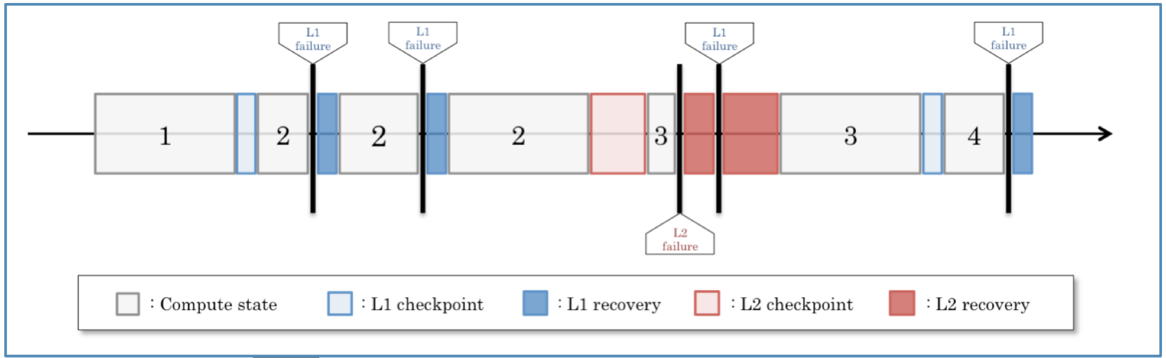


more resilient but …
huge overhead

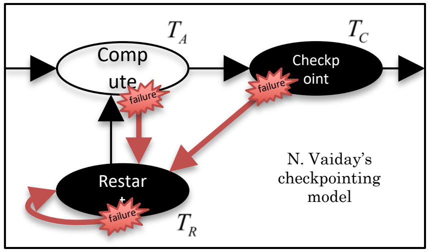
Low overhead but …
less resilient

Simple checkpoint model [1]

- One of approaches is modeling checkpointing behaviors
- Execution states can be categorized into compute, checkpoint and recovery state
- Transitions from one state to another can be described as transition diagram
- Assuming transitions occur based on PDF, easily compute expected time



T	Checkpoint interval
C	Checkpoint time
R	Restart time
λ	Failure rate

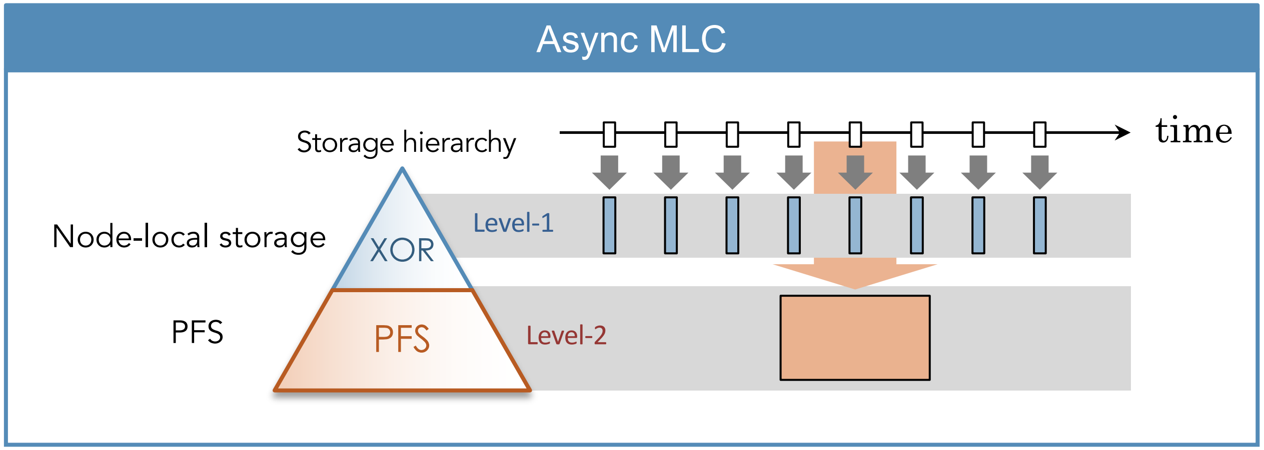
Markov model	Formulation(Efficiency)	Analytical solution (Optimal interval)
 N. Vaidya's checkpointing model	$\frac{T}{\lambda^{-1} e^{\lambda(L-C+R)}(e^{\lambda(T+C)}-1)}$	$\sqrt{2\times C/\lambda}$

[1] Nitin H. Vaidya. 1995. On Checkpoint Latency. Technical Report

While this model is simple to compute opt. interval, writing all checkpoints into PFS introduces huge I/O overhead

Asynchronous multi-level checkpointing (Async. MLC) [2]

- With the emergence of fast local-storage (e.g., NVM), MLC has become a common C/R approach
- Hierarchically write checkpoints
  - XOR: Frequently for a single-node failure
  - PFS : Infrequently for multi-node failure in the background
- With this more complicated C/R, finding each level of checkpointing intervals becomes more challenging, but important

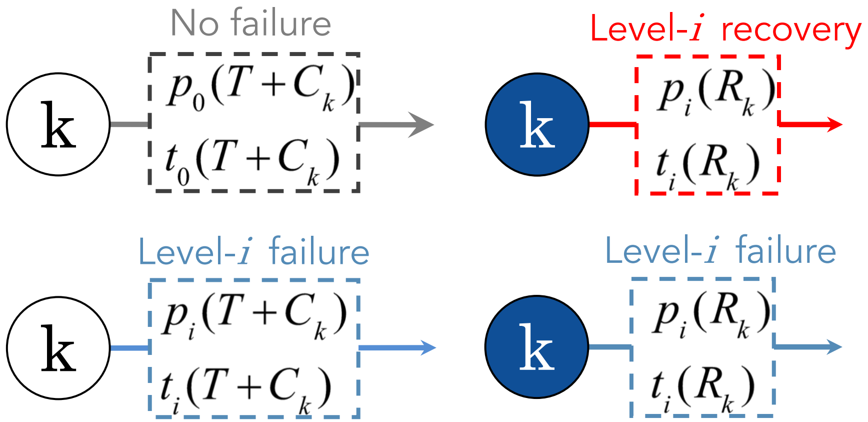


[2] **Kento Sato**, Adam Moody, Kathryn Mohror, Todd Gamblin, Bronis R. de Supinski, Naoya Maruyama and Satoshi Matsuoka, "Design and Modeling of a Non-blocking Checkpointing System", SC12, Salt Lake, USA, Nov, 2012

We modeled this async MLC for finding the optimal ckpt intervals

Markov model of async MLC

Transition probabilities ($p_i(t)$) and the expected times ($t_i(t)$)



Poison distribution

Probability (No failure): $p_0(t)=e^{-\lambda t}$

Expected time (No failure): $t_0(t)=t$

Probability (Level-i failure): $p_i(t)=\frac{\lambda_i}{\lambda}(1-e^{-\lambda t})$

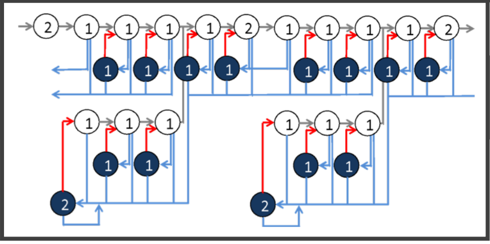
Expected time (Level-i failure): $t_i(t)=\frac{1-(\lambda t+1)\cdot e^{-\lambda t}}{\lambda\cdot(1-e^{-\lambda t})}$

$\left(\lambda=\sum_i \lambda_i\right)$

Input

T_k	Level-k checkpoint interval
C_k	Level-k checkpoint time
R_k	Level-k restart time
λ_k	Level-k failure rate

Markov model of async MLC



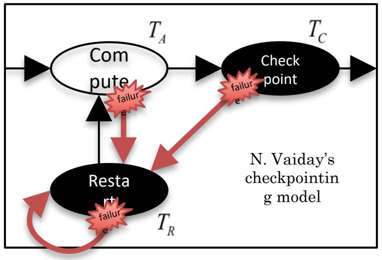
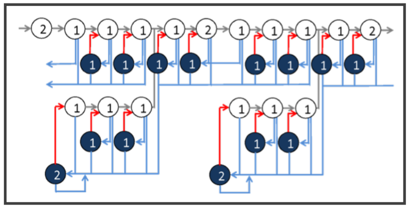
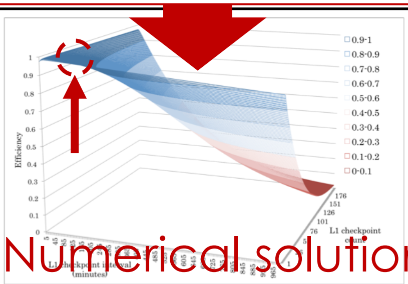
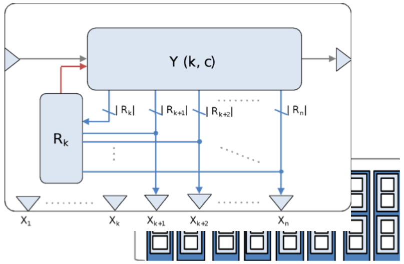
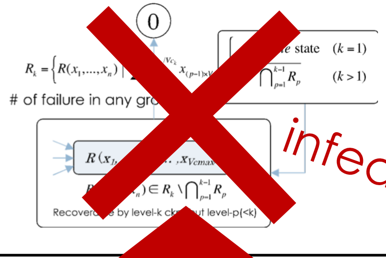
Output

E	Efficiency

(A ratio of time an application can spend for its useful computation)

$$E=\frac{\text{Exec. time w/o failures}}{\text{Exec. time w/o failures} + \text{C/R time} + \text{Re-exec. time}}$$

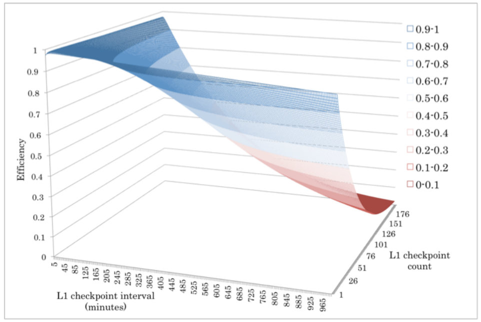
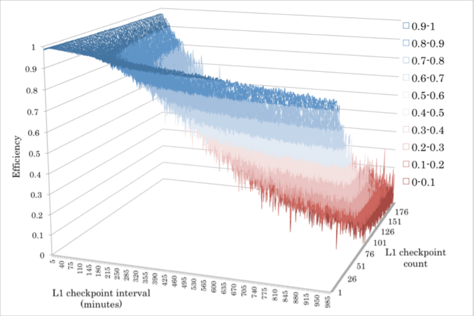
Modeling for optimal checkpointing

Checkpointing model		Formulation (Efficiency)	Analytical solution (Optimal interval)
Single level checkpointing		$\frac{T}{\lambda^{-1}e^{\lambda(L-C+R)}(e^{\lambda(T+C)}-1)}$	Analytical solution $\sqrt{2\times C/\lambda}$
Mutil-level Checkpointing (SCR)		$\begin{aligned} T_{W0} &= p_{YL}^{0}\cdot p_{Y0}\cdot(T_{ZY}+0\cdot(t_{YL}+T_{RZ}+T_{ZY})+t_{Y0})) \\ &\quad + p_{YL}^{1}\cdot p_{Y0}\cdot(T_{ZY}+1\cdot(t_{YL}+T_{RZ}+T_{ZY})+t_{Y0})+\ldots \\ &= p_{Y0}\cdot\{(t_{YL}+T_{RZ}+T_{ZY})\cdot\sum_{i=1}^{\infty} i\cdot p_{YL}^{i}+(T_{ZY}+t_{Y0})\cdot\sum_{i=1}^{\infty} p_{YL}^{i}\} \\ &= p_{Y0}\cdot\{\frac{(t_{YL}+T_{RZ}+T_{ZY})\cdot p_{YL}}{(1-p_{YL})^{2}}+\frac{(T_{ZY}+t_{Y0})}{1-p_{YL}}\} \\ &= \frac{(t_{YL}+T_{RZ}+T_{ZY})\cdot p_{YL}}{1-p_{YL}}+(T_{ZY}+t_{Y0}) \end{aligned}$	 Numerical solution
Mutil-level Checkpointing (FTI)		 infeasible	infeasible

Complicated C/R models have finding analytical solution harder

We tried to model to evaluate resiliency of more complicated erasure encodings
We found that it is significantly difficult to formulate C/R models
unless we simply the model and/or make strong assumption

Simulation for optimal checkpointing in multi-level checkpointing in SCR

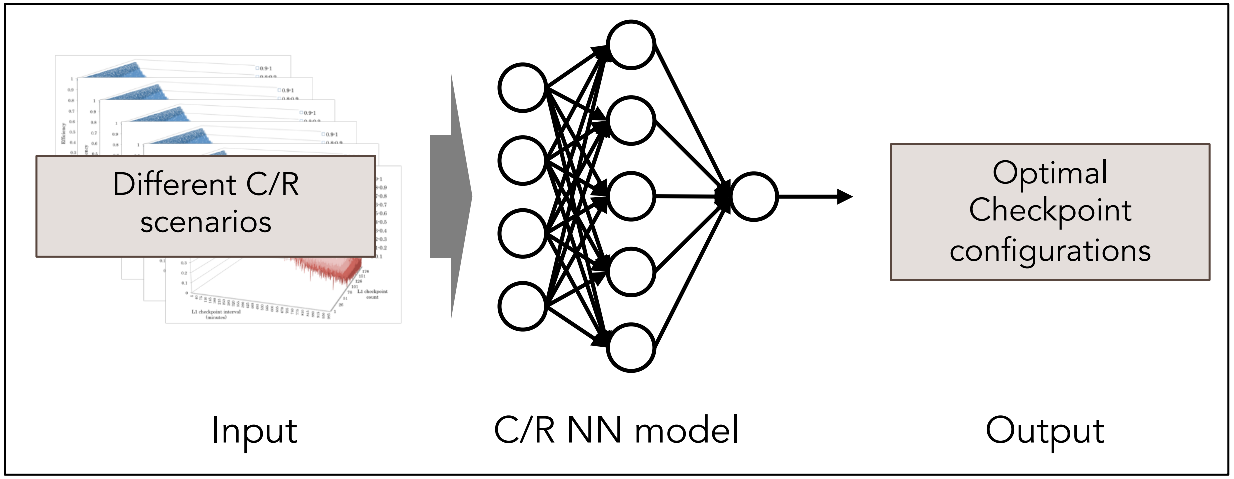
Modeling	Simulation
	

- We are shifting from modeling approach to simulation
  - (Simulation is also important to validate the model)
- Pros
  - Simulation can be applied to more complicated
  - Simulation can estimate expected execution time much more accurately than modeling approach
- Cons
  - Simulation takes time to explore different C/R parameters and find optimal checkpoint interval

While simulation is useful when evaluating efficient of C/R
If one wants to know the optimal checkpoint interval
when submitting a job, Simulation is not practical approach

AI for C/R

- Combine simulation with AI techniques
- Generate training data consisting input/output data by running simulator in many different scenarios
  - Checkpoint and recovery time, failure rates, type of erasure encodings (partner, XOR, RS etc.), node allocation, network topology (fat tree, torus etc.)
- Train and Build a C/R NN model to find optimal configurations (e.g., checkpoint location, checkpoint intervals and a type of erasure coding)



Different C/R scenarios

Optimal Checkpoint configurations

Input

C/R NN model

Output

Summary:
Convergence of AI, Big Data and HPC

1. AI for data compression
2. AI for checkpointing optimization

→ Poster

SC19
Denver, CO
hpc is now.

HPC for AI/BD

Research and software development for accelerating AI/Big data workloads and applications on HPC systems (i.e., large-scale systems)

AI/BD for HPC

Research and software development for accelerating HPC workloads and applications by using Big Data/AI techniques

R&D for HPC