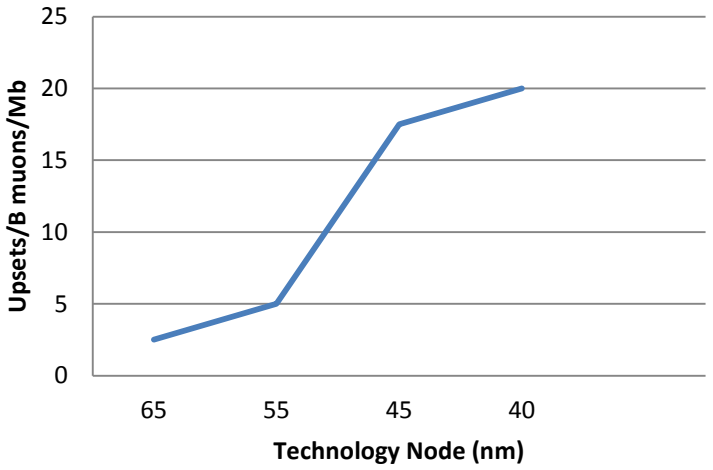
Parallel Streaming Computation on Error-Prone Processors

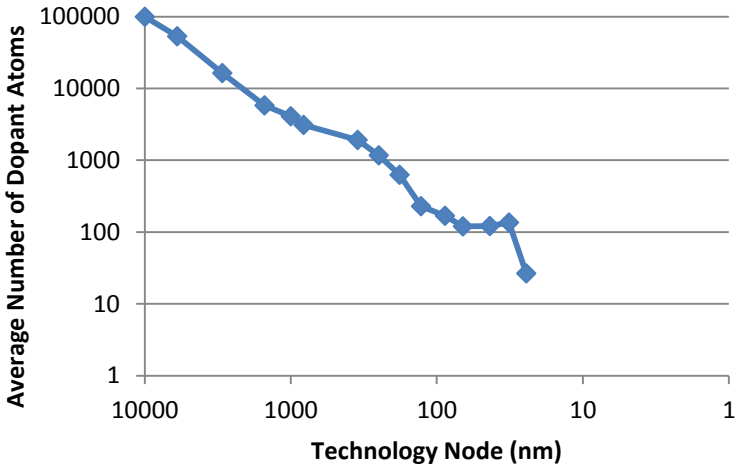
Yavuz Yetim, Margaret Martonosi, Sharad Malik

Hardware Errors on the Rise

Soft Errors Due to Cosmic Rays [Sierawski et al., 2011]

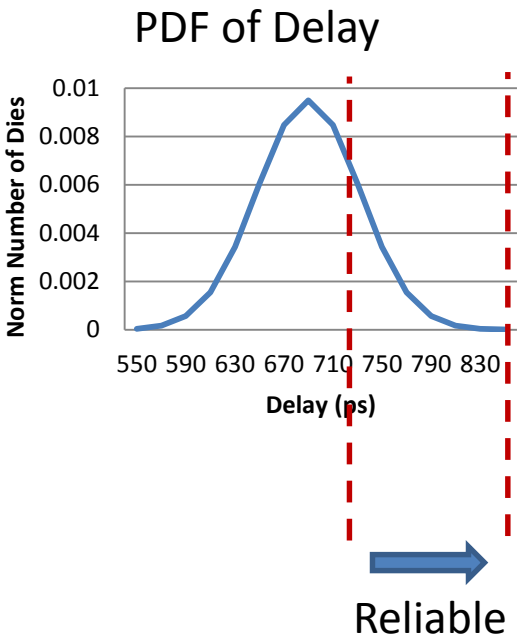


Random Process Variation [Khun et al., 2011]

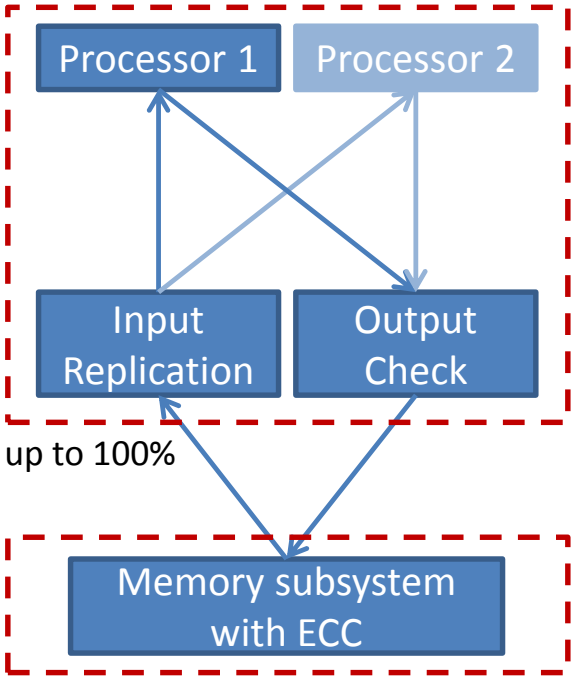


Traditional Solutions

Higher Latencies or Voltage Margins



Redundancy

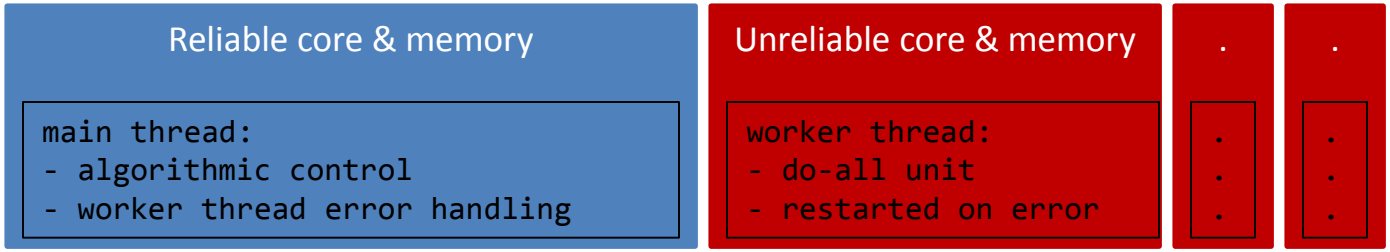


SECDED: 1-cycle latency, ~10k gates
4EC5ED: 14-cycle latency, ~100k gates

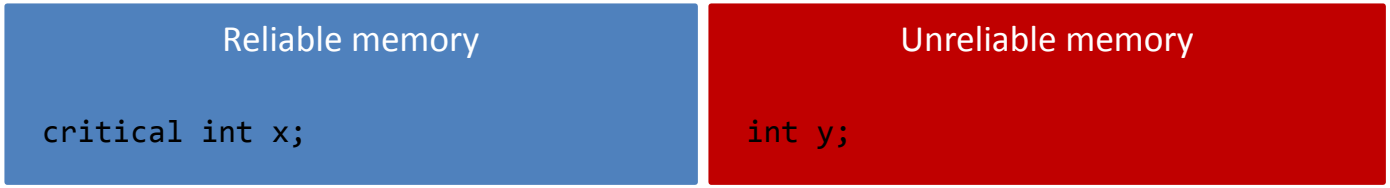
High Power, Performance and Area Overhead

Architectures for Error-Prone Computing

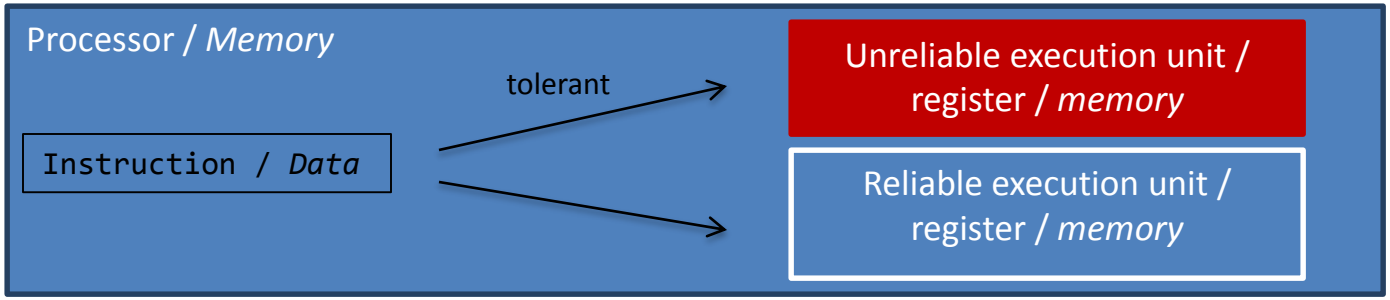
ERSA
[Leem et al., 2010]



Flikker
[Liu et al., 2011]



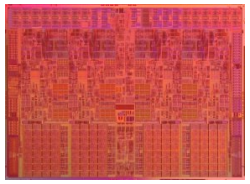
EnerJ
[Sampson et al., 2011]



* Reliable Unreliable

To Minimal Reliable Hardware

Error-tolerant application



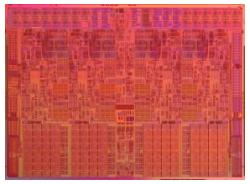
Error-prone processor

Output:

- Crashes due to memory errors
- Hangs due to control-flow errors

To Minimal Reliable Hardware

Error-tolerant application

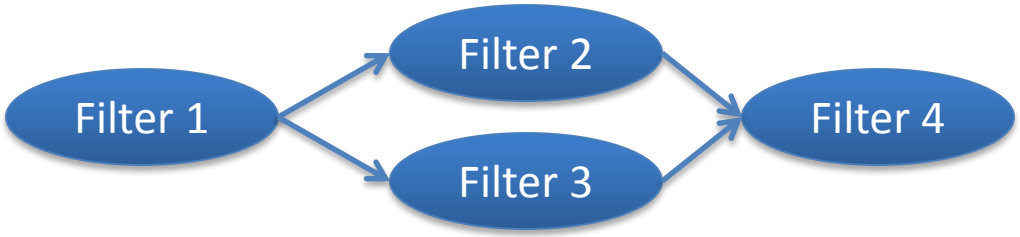


Error-prone processor

Output:

- Crashes due to memory errors
- Hangs due to control-flow errors

StreamIt programming model
+ memory segmentation



Regions with R/W/X permissions

Control-flow with scopes:

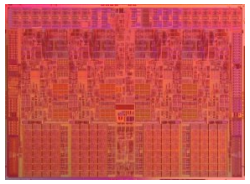
- Known run-times of modular control-flow regions determine timeout limits
- Coarse-grain sequencing of computation

Memory:

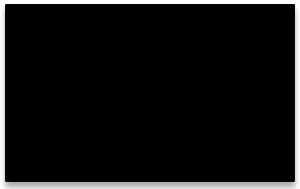
- Only allowed accesses are allowed, other dropped

To Minimal Reliable Hardware

Error-tolerant application



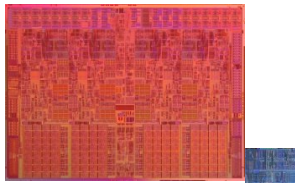
Error-prone processor



Output:

- Crashes due to memory errors
- Hangs due to control-flow errors

Error-tolerant application



Error-prone processor

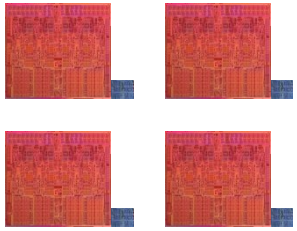
+ coarse-grain control-flow, memory, I/O management

Output: Graceful quality degradation with errors

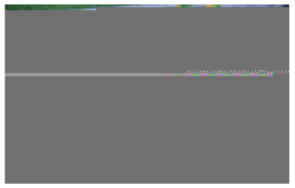
*Extracting Useful Computation From Error-Prone Processors [Yetim et al, 2013]

Communication Errors For Parallel Streaming Applications

Error-tolerant application



Multiple processing nodes with single-threaded protection



Output: Unacceptable quality

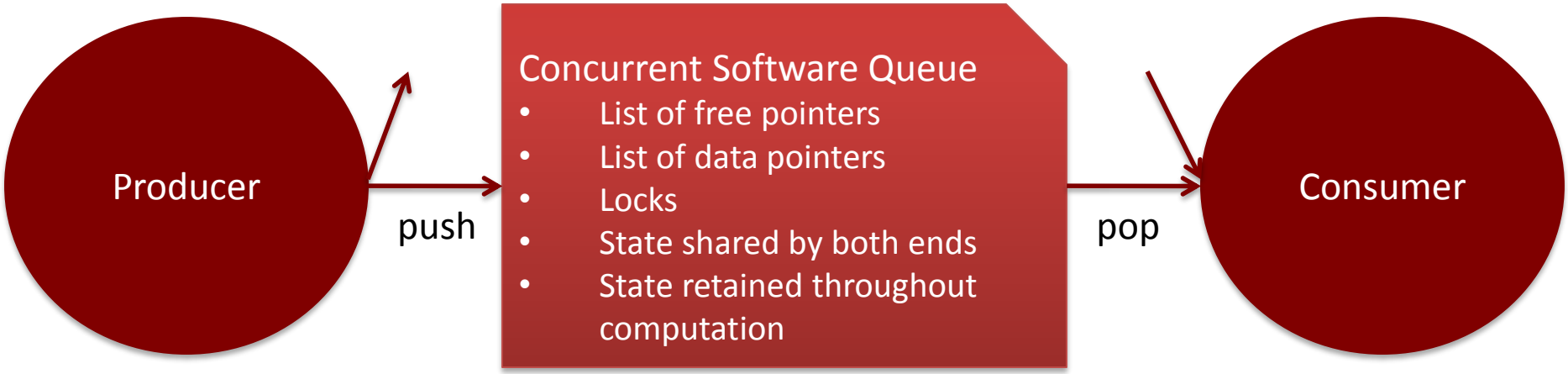
This work

- Communication errors
  - Unrecoverable corruption of the communication mechanism
  - Data misalignment among producer/consumer threads
- CommGuard
  - Application-level communication information
  - Low overhead recovery from communication errors

Outline

- Motivation
- Communication Errors in Parallel Streaming Applications
- CommGuard System Overview
- Experimental Methodology and Results
- Conclusions

Communication Errors
Transmission Failure

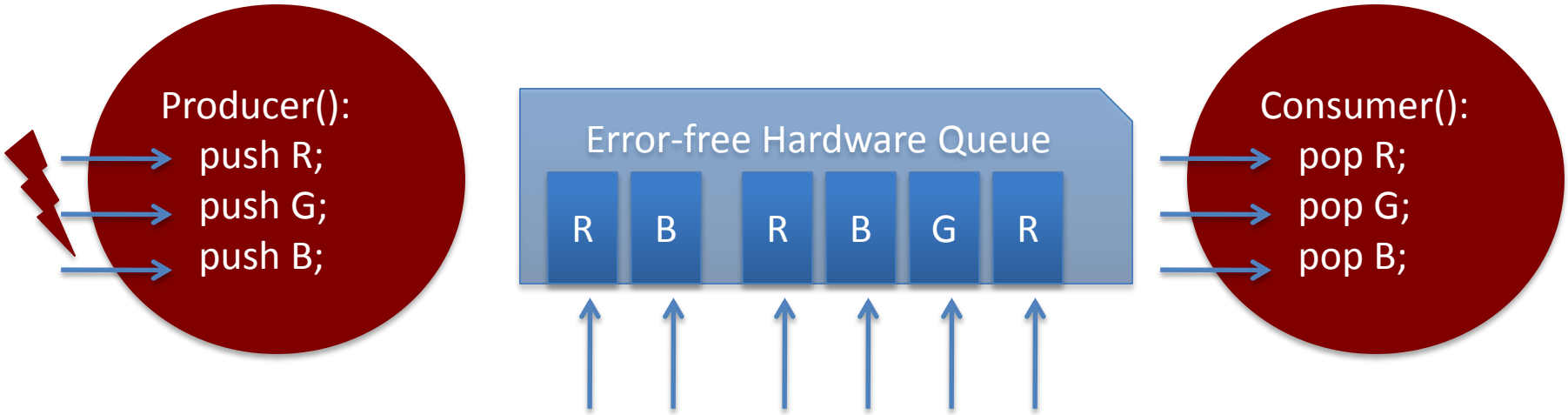


Corruption in lists, pointers and locks are permanent

Communication Errors
Transmission Failure

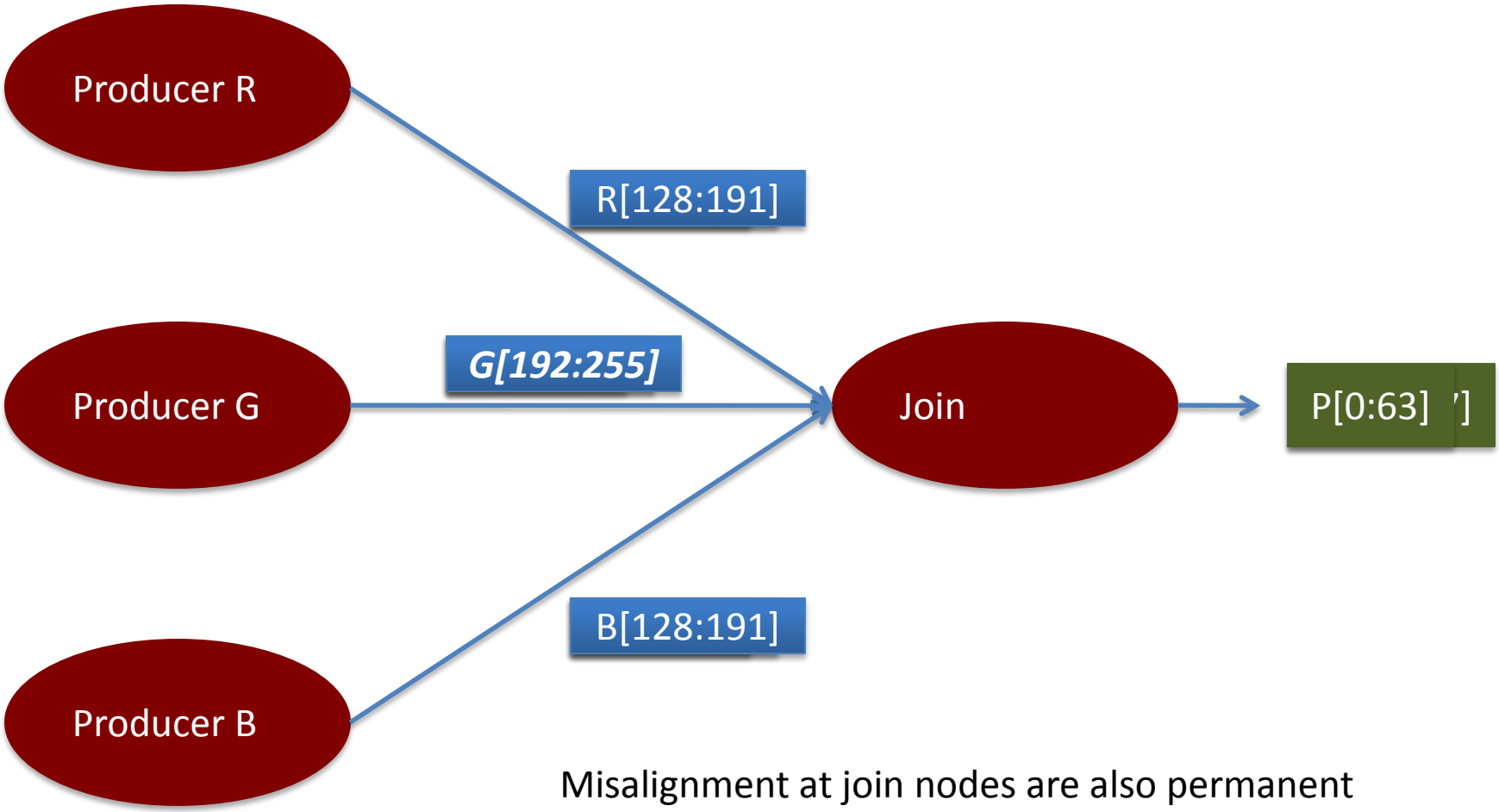
- Data items are flowing
- Image is not coherent

Communication Errors
Misalignment I



Misalignment due to a control-flow error is permanent

Communication Errors
Misalignment II

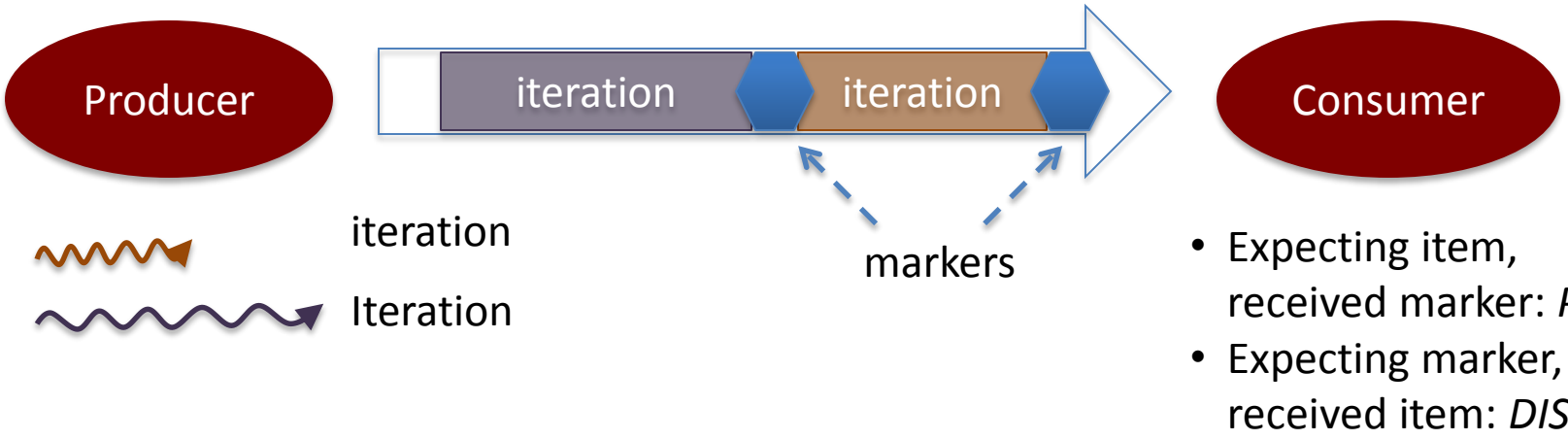


Misalignment at join nodes are also permanent

Outline

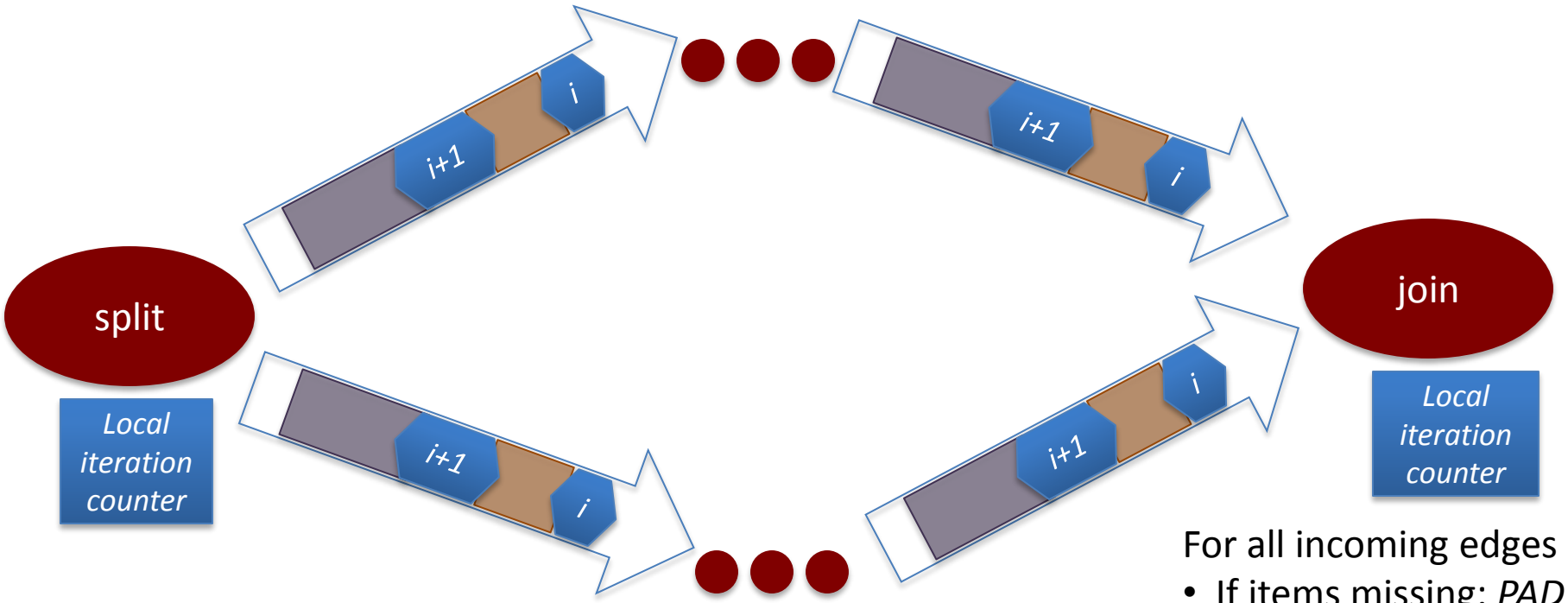
- Motivation
- Communication Errors in Parallel Streaming Applications
- CommGuard System Overview
- Experimental Methodology and Results
- Conclusions

CommGuard Overview



- Expecting item, received marker: *PAD*
- Expecting marker, received item: *DISCARD*

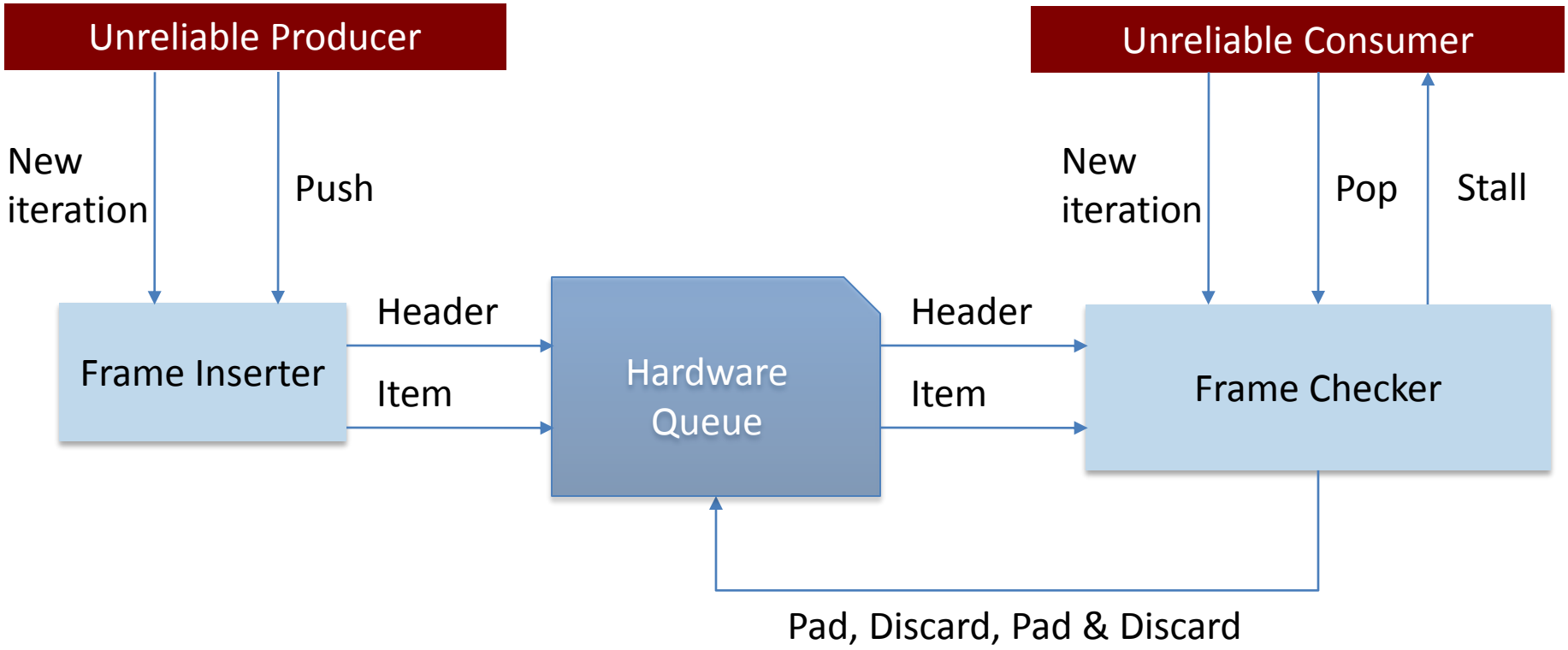
CommGuard Overview



For all incoming edges

- If items missing: *PAD*
- If items extra: *DISCARD*

CommGuard System Overview



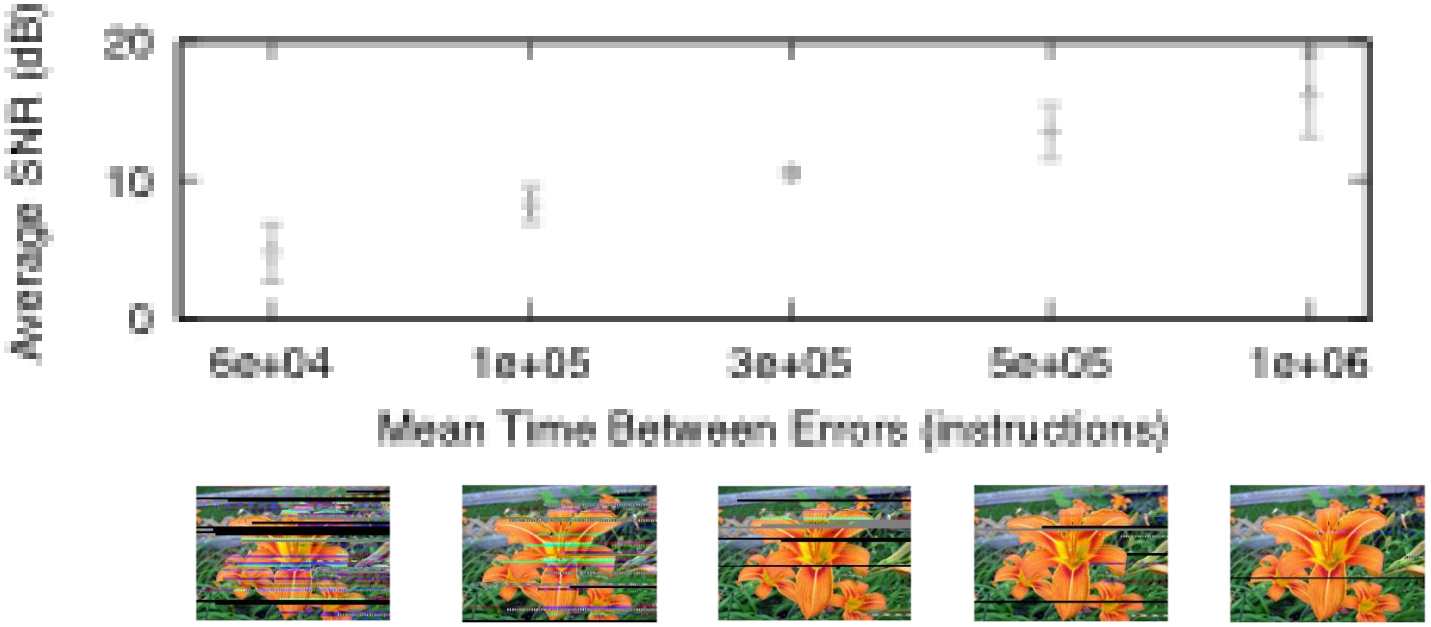
Outline

- Motivation
- Communication Errors in Parallel Streaming Applications
- CommGuard System Overview
- Experimental Methodology and Results
- Conclusions

Experimental Methodology

- Built on prior simulation Infrastructure by [Yetim et al, DATE 2013]
  - Virtutech Simics modeling 32-bit Intel x86
  - Error injection capabilities
  - Protection modules for sequential streaming applications
  - Architecturally visible errors following distribution with given mean time between errors (**MTBE**)
    - Pick error injection cycle
    - Picks random register, pick random bit
    - Flip bit, repeat
- Extensions for multi-core simulation
  - Monitor scheduling of selected threads
  - Pin threads to processor cores
  - Per-core error injection
  - Protection modules implemented for every core
- Modeled frame checker and frame inserter
- JPEG Decoder as a streaming application

Output at Different Error Rates

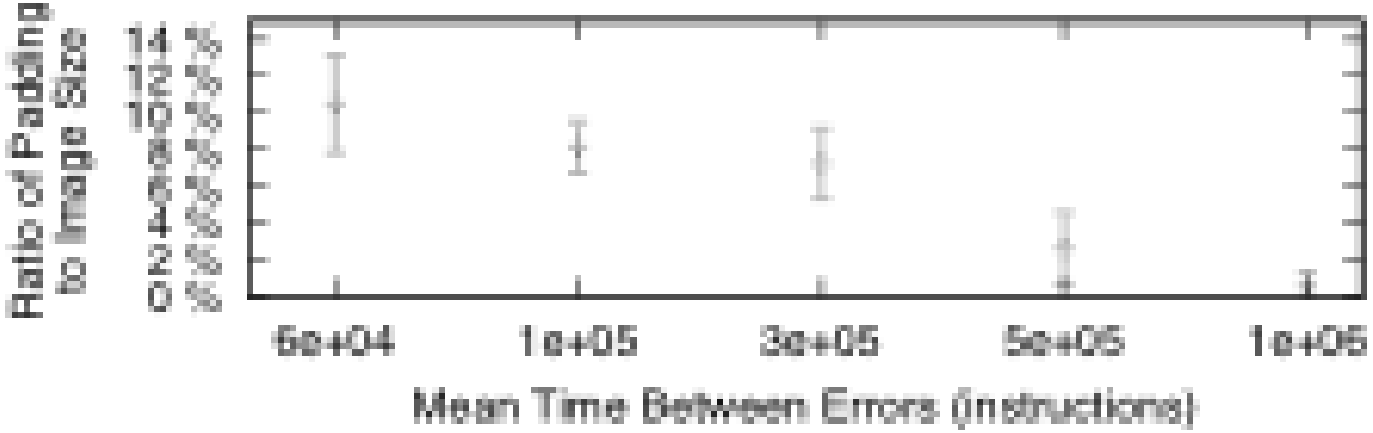


- Output quality restored after misalignment through CommGuard
- Graceful output degradation with increasing errors

Run-time Overhead Due to Stalls

- Run-time increases due to stalls caused by misalignments
- Only 2% even at high error-rates

Amount of Padding



- Padding to resolve misalignments is observed even at low error rates

Outline

- Motivation
- Communication Errors in Parallel Streaming Applications
- CommGuard System Overview
- Experimental Methodology and Results
- Conclusions

Conclusions

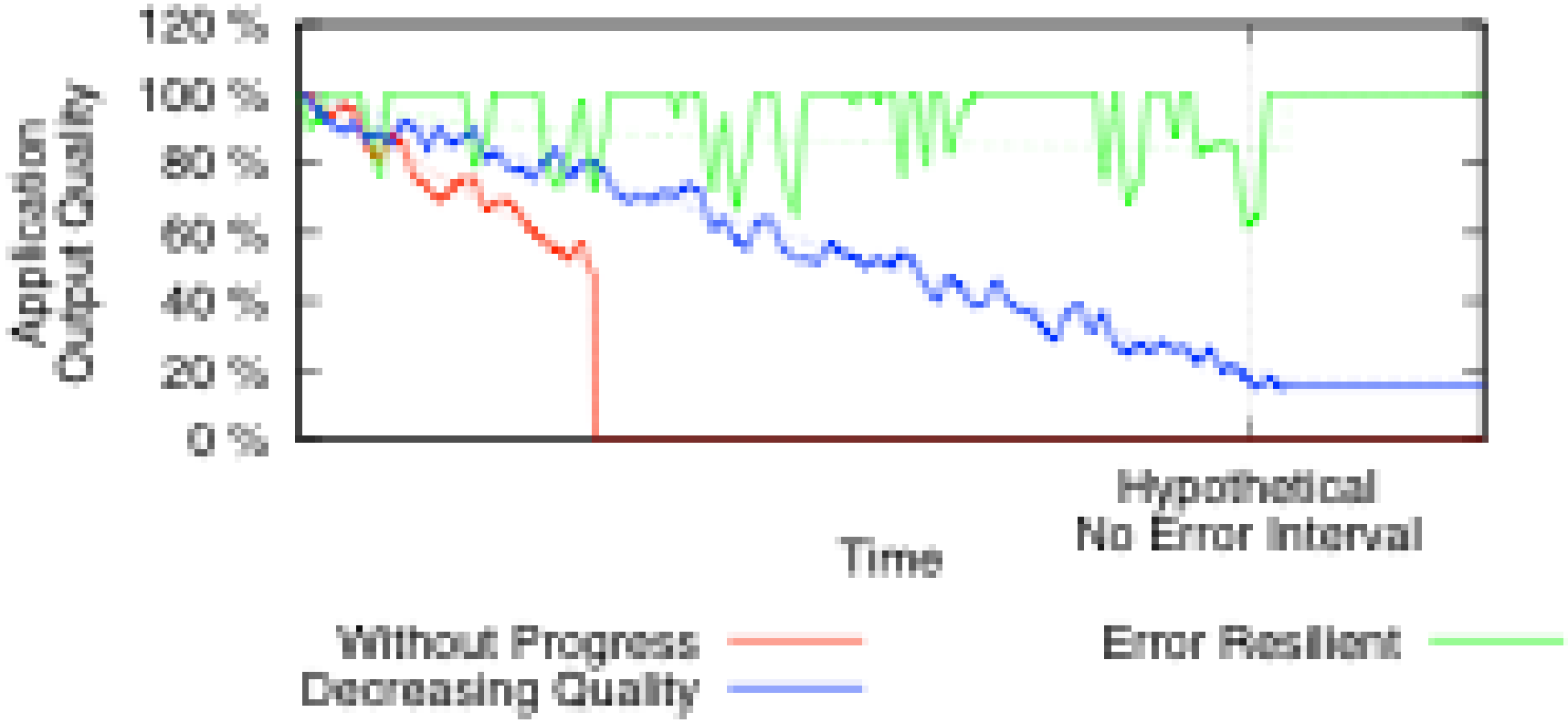
- Communication in parallel applications add fragility
  - Error-prone communication subsystem
  - Data misalignments due to asynchronous threads
- Explicit communication & control-flow can be used
  - Encapsulate coarse-grain data units
  - Use small checker circuitry to recover from communication errors
- Low overhead solutions to sustain quality
  - Only ~150B of reliable state per core and less than 2% run-time overhead even at high error rate
  - 16dB can be sustained for errors as frequent as every 1ms

Parallel Streaming Computation on Error-Prone Processors

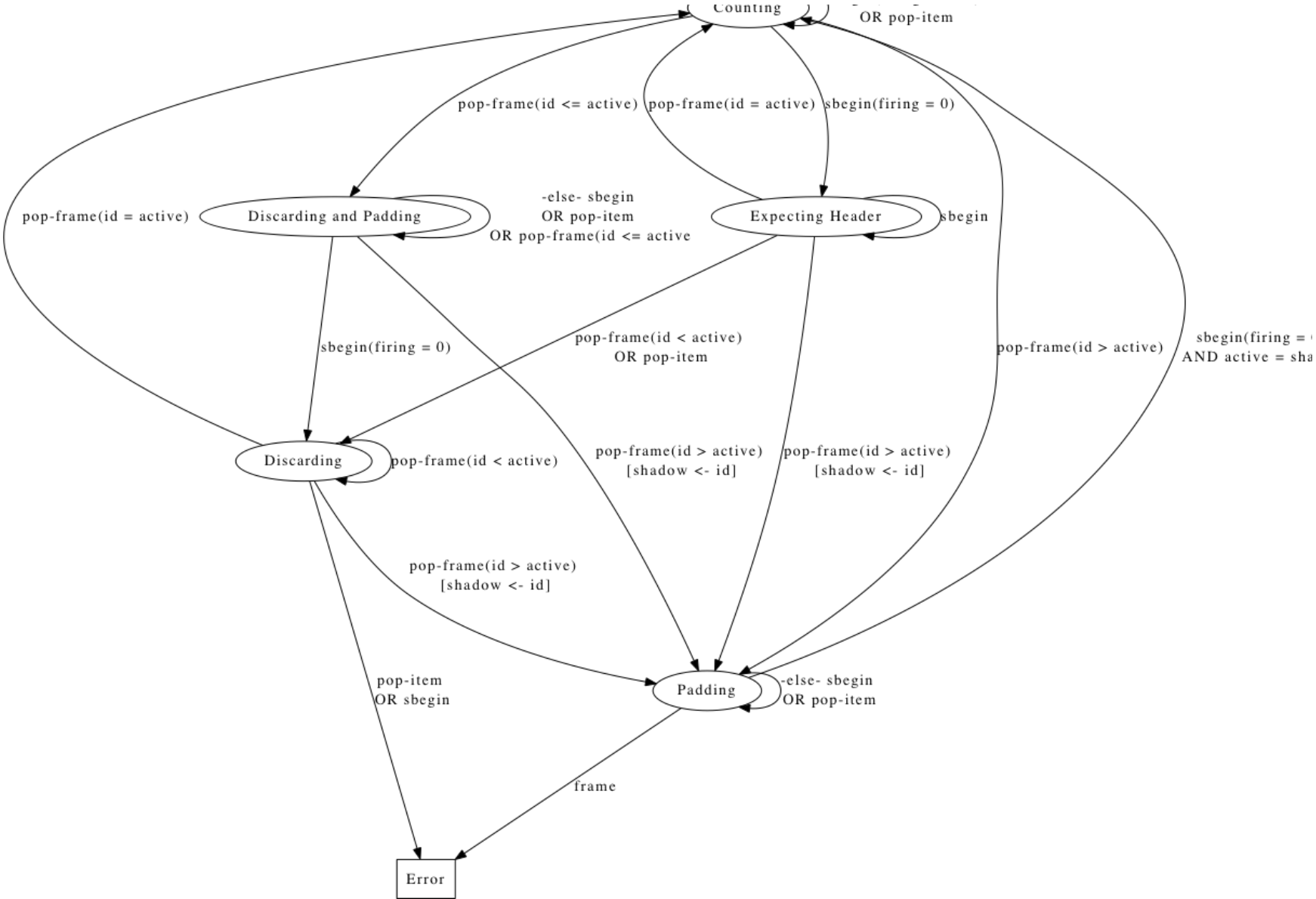
Yavuz Yetim, Margaret Martonosi, Sharad Malik

Backup Slides

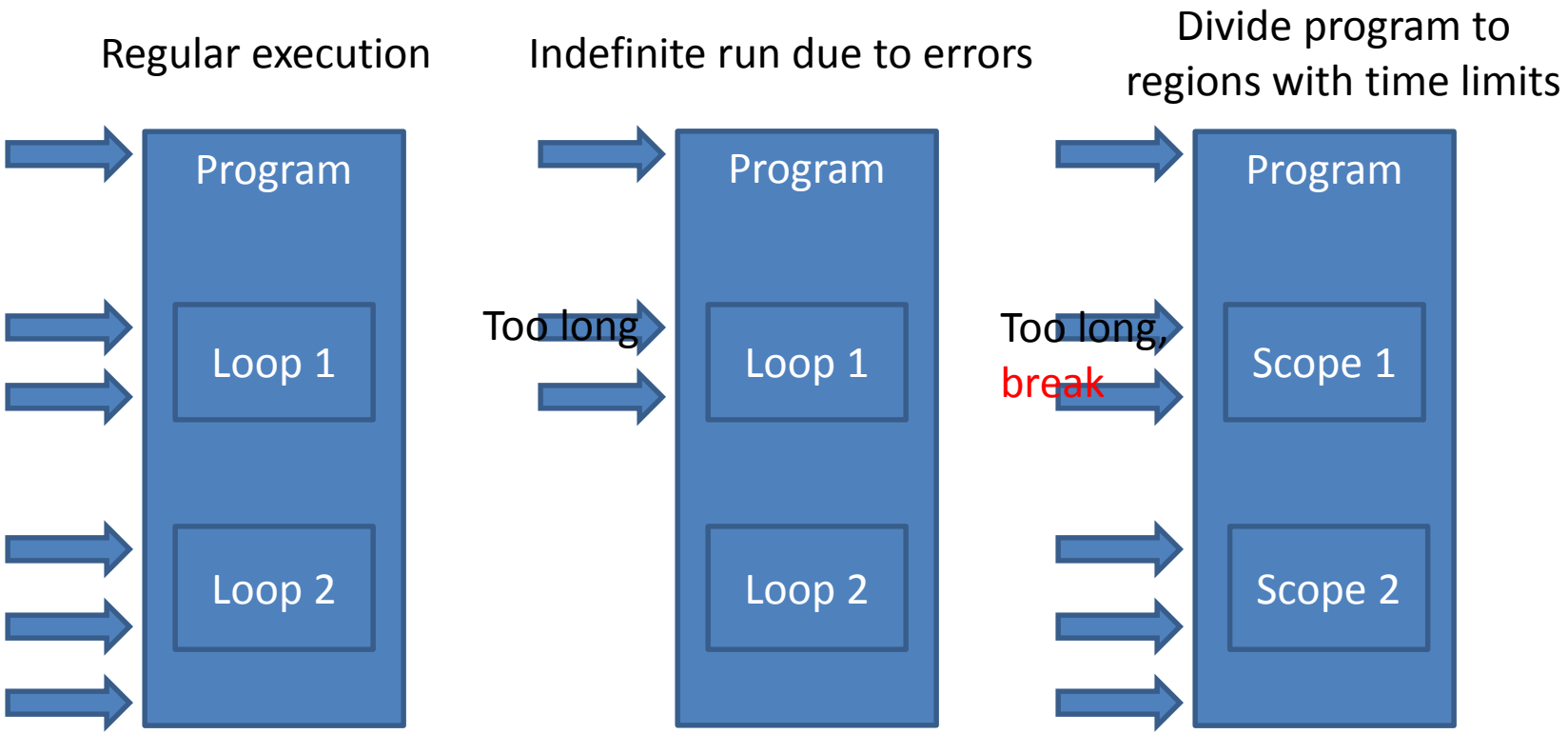
Suitably Error Tolerant



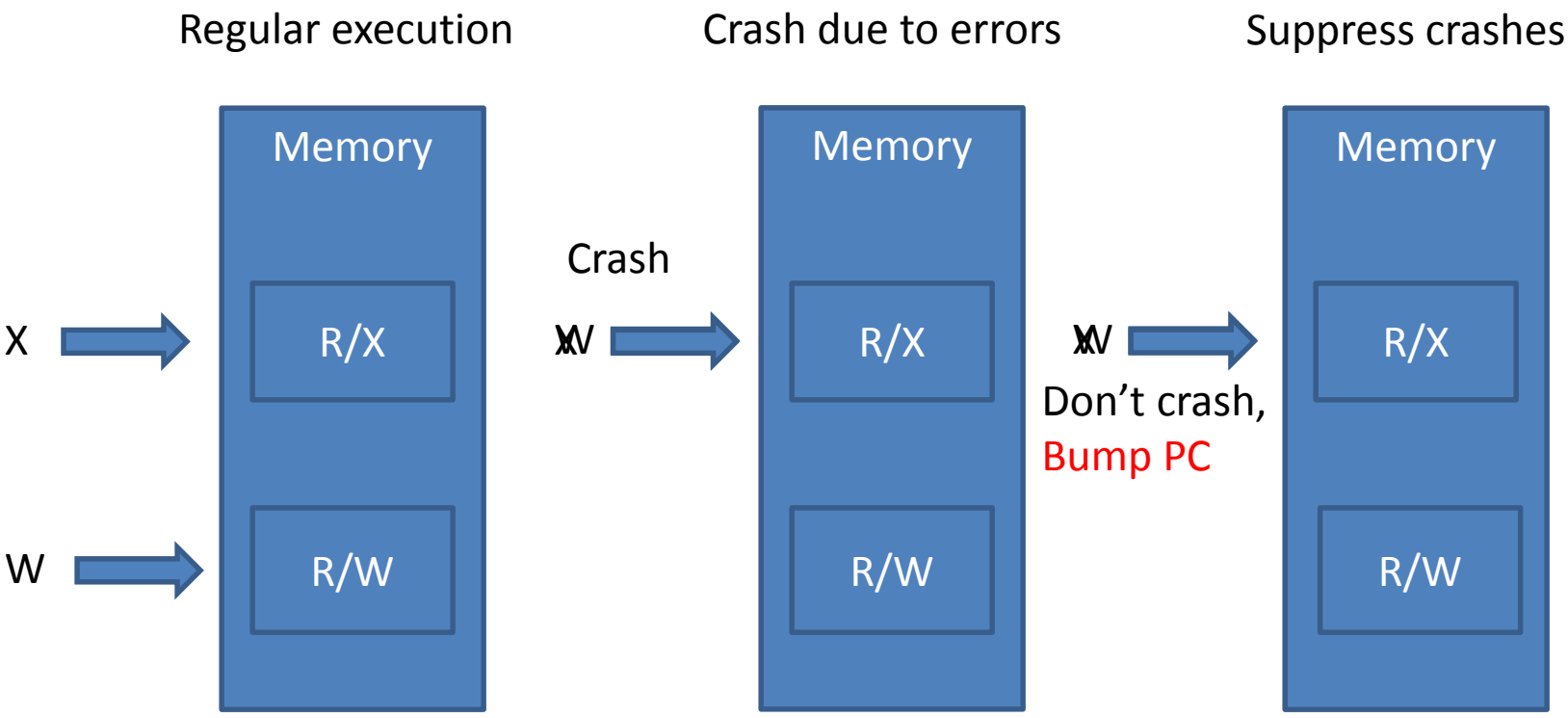
Frame Checker FSM



Avoid Running Indefinitely



Disallowed Memory Accesses

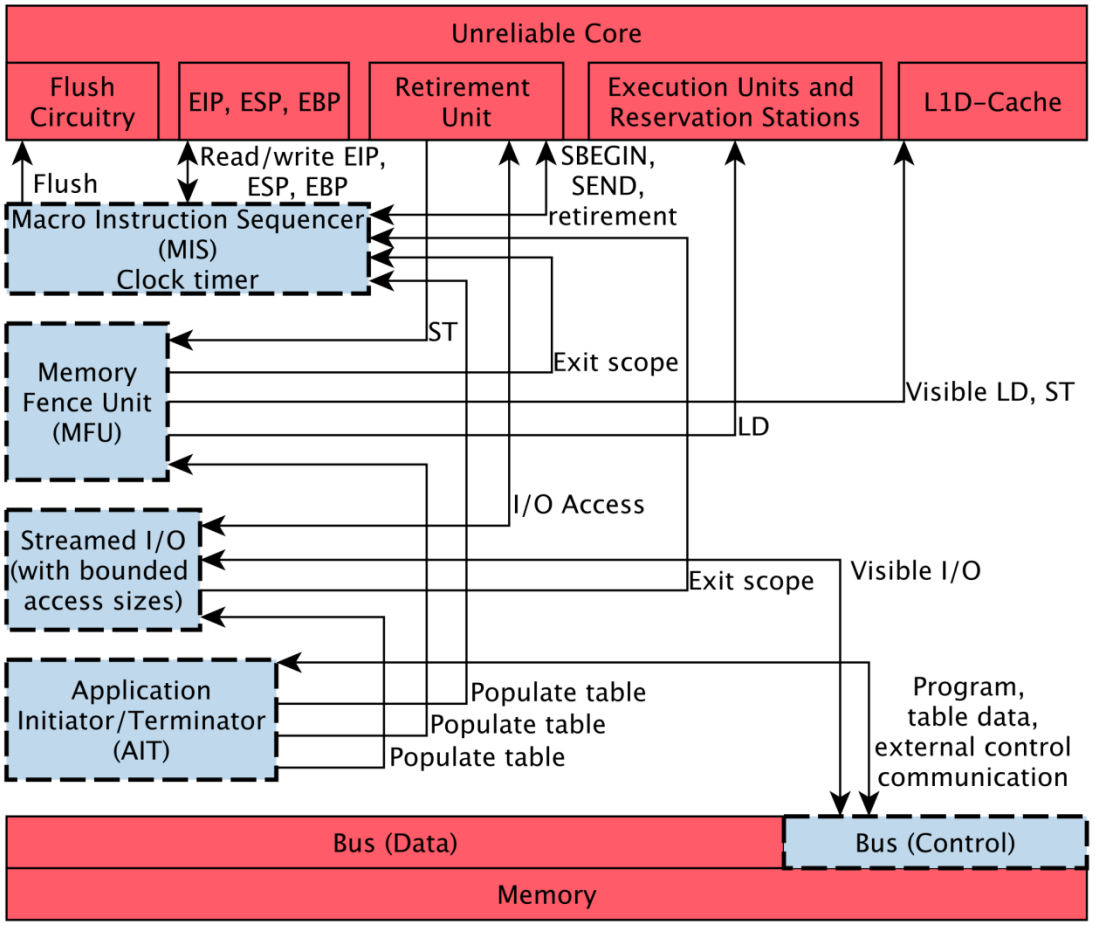


Overall Design

MIS: Coarse-grained control flow constraints and recovery

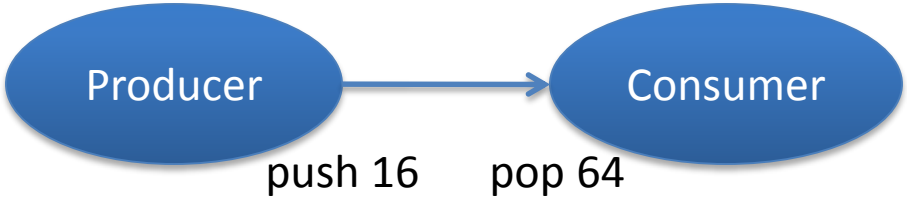
MFU: Coarse-grained constrains on memory accesses

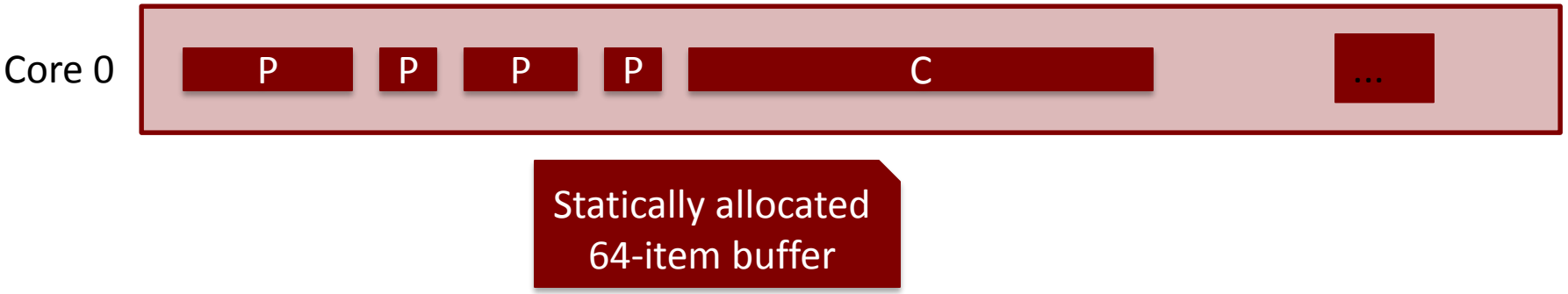
Streamed I/O: Manages bounded data streams



Communication Errors
Single-threaded

Toy producer-consumer streaming application





- Static location is preserved in reliable I-Cache throughout the computation
- Every new [P] or [C] iteration recovers the pointer values
- Communication never halts indefinitely

Shared State

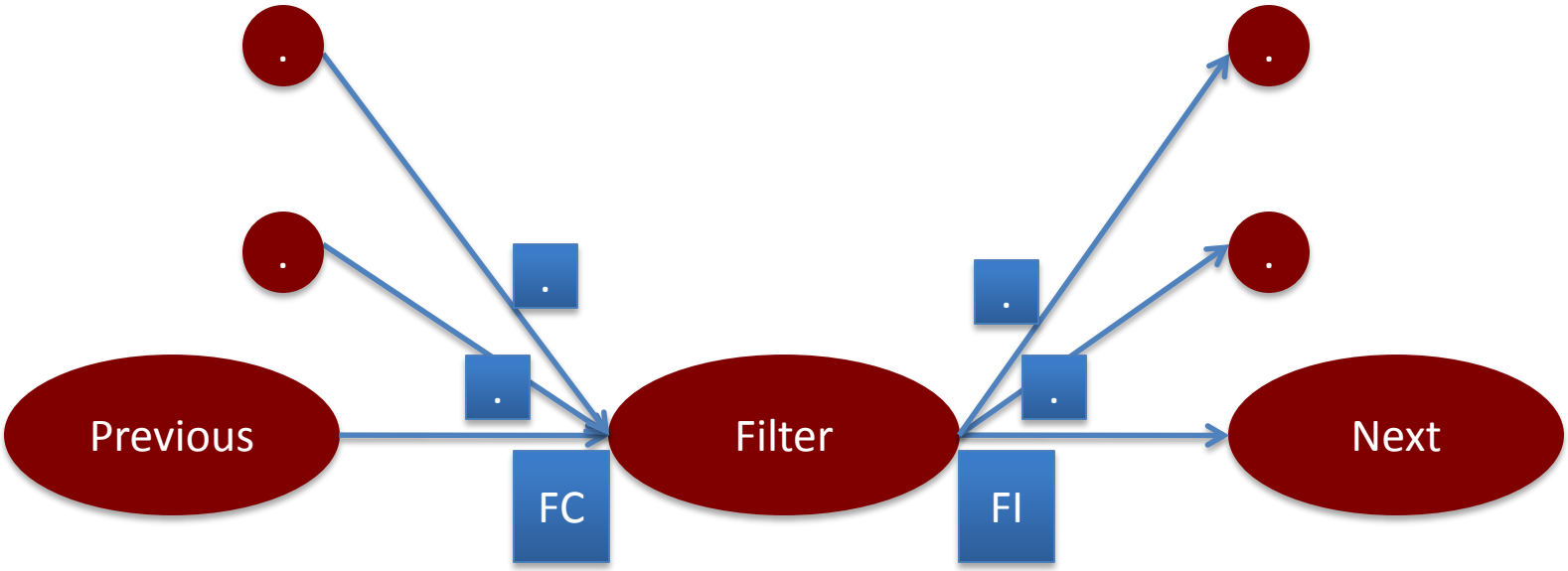
- The *inserter* and the *checker* need to keep state to operate
- State below is shared by every *inserter* and *checker* belonging to a node

Value	Details	(S)tatic or (D)ynamic
Firing per frame	How many times a node needs to fire before the computation starts for the next *frame*	S
Frame limit	Number of total frames the application needs to process	S
Active frame	How many frames have been processed so far	D
Active firing	How many times the node has fired for the active frame	D

Additional Frame Checker State

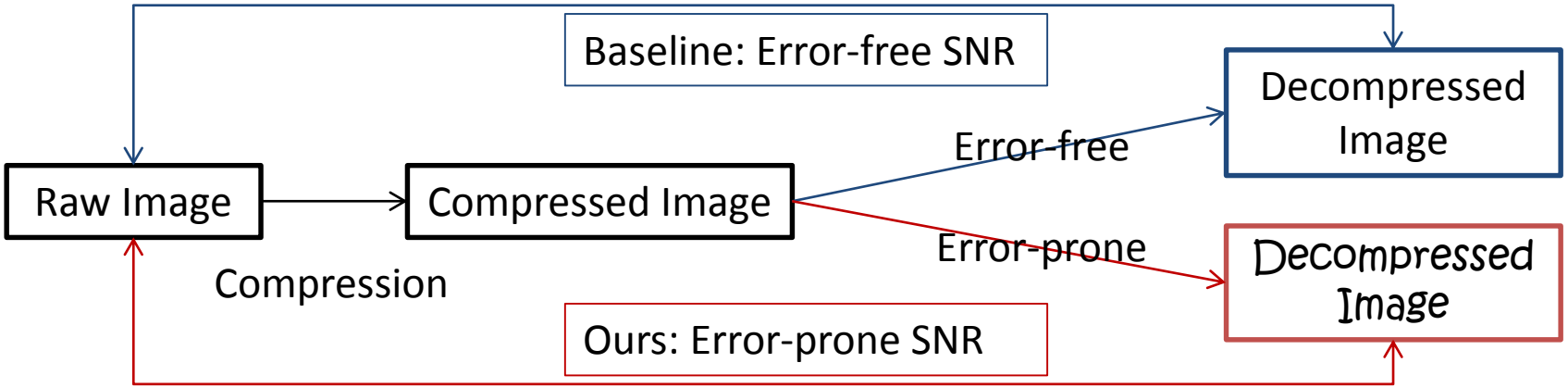
State	Details	(E)rroneous (N)ormal
Receiving items	Node is receiving items for the active frame	N
Expecting a header	Node has started new frame computationally hence the next item in the queue should be a header	N
Discarding	The computation in the node is ahead of the communication of the edge	E
Padding	The communication of the edge is ahead of the computation in the node	E

CommGuard Placement



Output Quality For Varying MTBEs

- Compare *lossy compression* to *error-prone decompression*
- For raw image file I, encoded file E and decoded files F or P:



- This study was performed for MP3 and JPEG decoder benchmarks
  - Widely used
  - Full-runs
  - Each experimental setting: 10 times