ECE 6930-004 HPC Fault Tolerance

COURSE SUMMARY

DR. JON C. CALHOUN

Course outline

Part 1: Introduction to HPC and reliability

Part 2: Checkpoint-restart

Part 3: Soft errors

Part 4: Approximate computing and lossy compression

Part 1: Introduction to HPC and reliability

Why HPC?

Why is resilience an issue for HPC

Terminology

Modeling Reliability

Availability

Formulas

Probability modeling

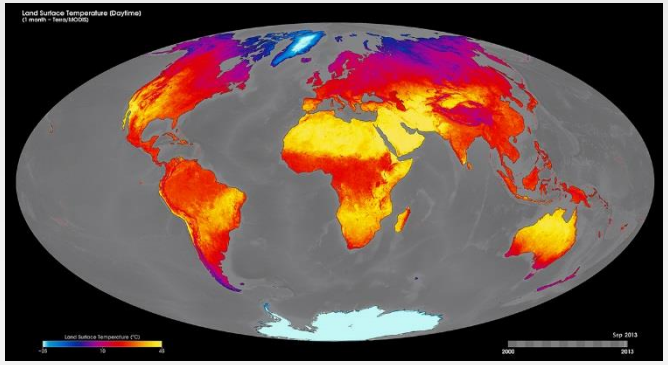
Series/parallel circuit

Failure detection

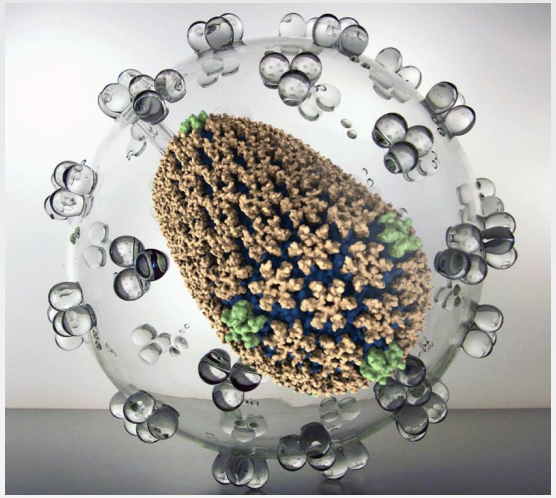
What is high-performance computing?

High-performance computing (HPC) focuses massive compute capabilities for exploration and solving of previously intractable computational problems

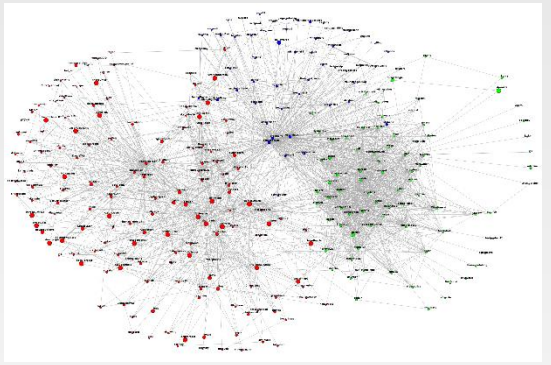
HPC is an essential tool of science and discovery in many fields of science, engineering, and industry



Land surface temperature [NASA]



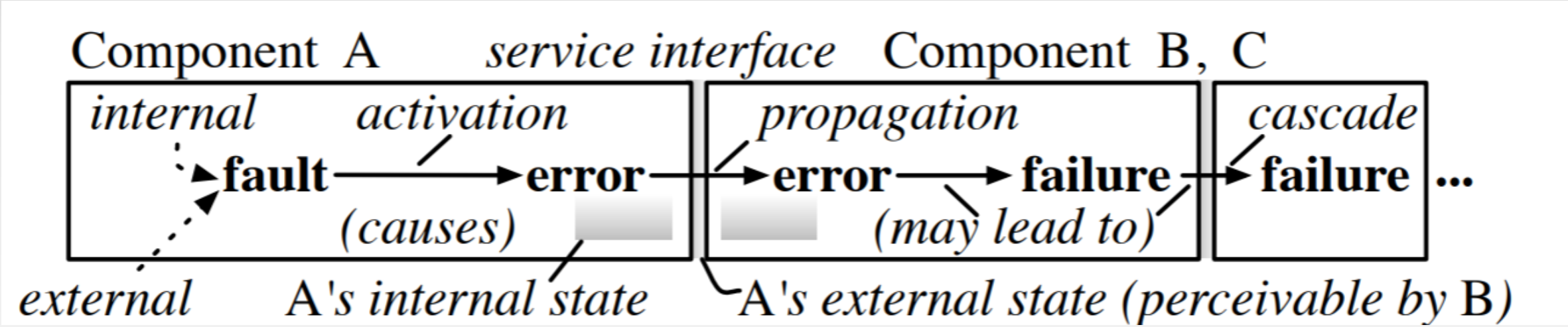
HIV capsid [Beckman Institute]



Twitter social graph [Olin College]

"Why does high-performance computing matter? Because science matters! Discovery matters! … HPC extends our reach, putting more knowledge, more discovery, and more innovation within our grasp" - Dr. Shishir Pandya NASA Aerospace Engineer

What is the relationship?



Error propagation and cascading failures [Snir et al. 2013]

Series system

$$R_{series}(t) = \prod_{i=1}^{n} R_i(t)$$

$R_i(t)$ is the reliability of component i

All components need to survive for the system to function

For exponentially distributed failures:

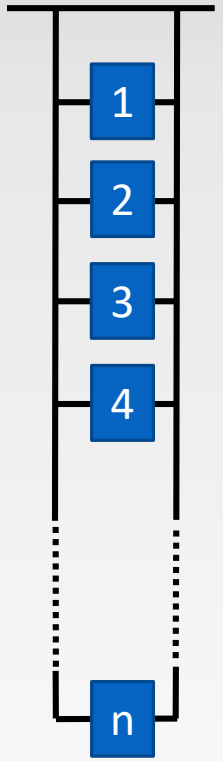
$$R_{series}(t) = \prod_{i=1}^{n} R_i(t) = e^{-\sum_{i=1}^{n} \lambda_i t} = e^{-\lambda_{system} t}$$

Parallel system

System with spares

Assume that when a failure occurs, the spare assumes responsibility immediately

$$R_{parallel}(t) = 1.0 - \prod_{i=1}^{n}(1.0 - R_i(t))$$

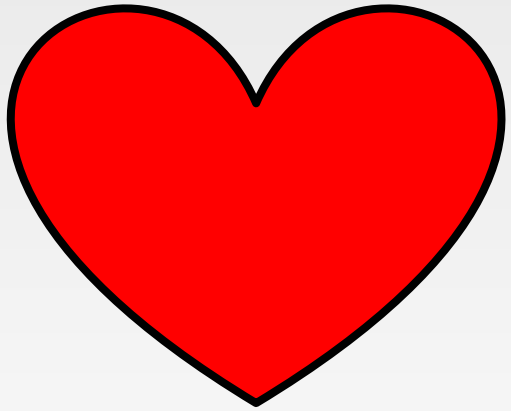


For the system to work only one component needs to operate correctly

- Probability of module i to survive: R_i
- Probability module i does not survive: $(1 - R_i)$
- Probability of no modules survive: $(1 - R_1)(1 - R_2) \ldots (1 - R_n)$

How do we know a human is ALIVE?

HEARTBEATS



Part 2: Checkpoint-restart

Why is checkpointing important?

Difference between system and application level

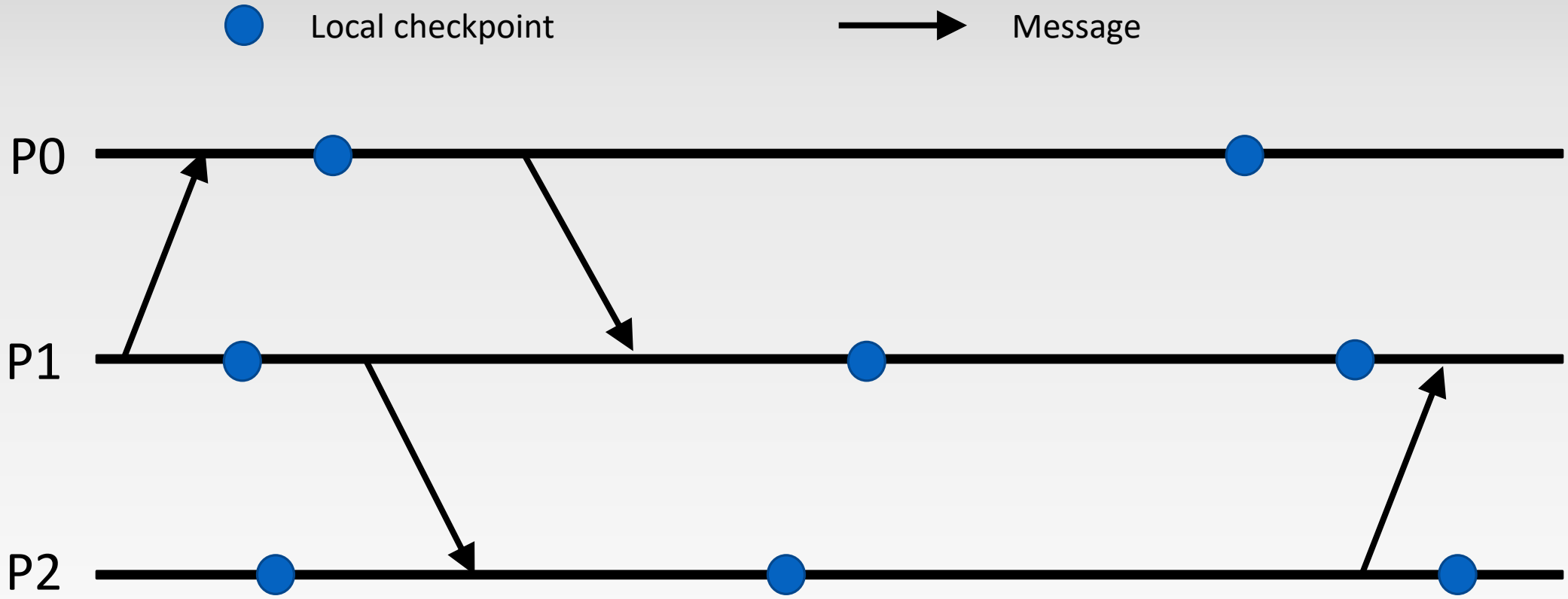
- Pros/cons

Difference between synchronous/blocking and asynchronous/non-blocking

Why message logging?

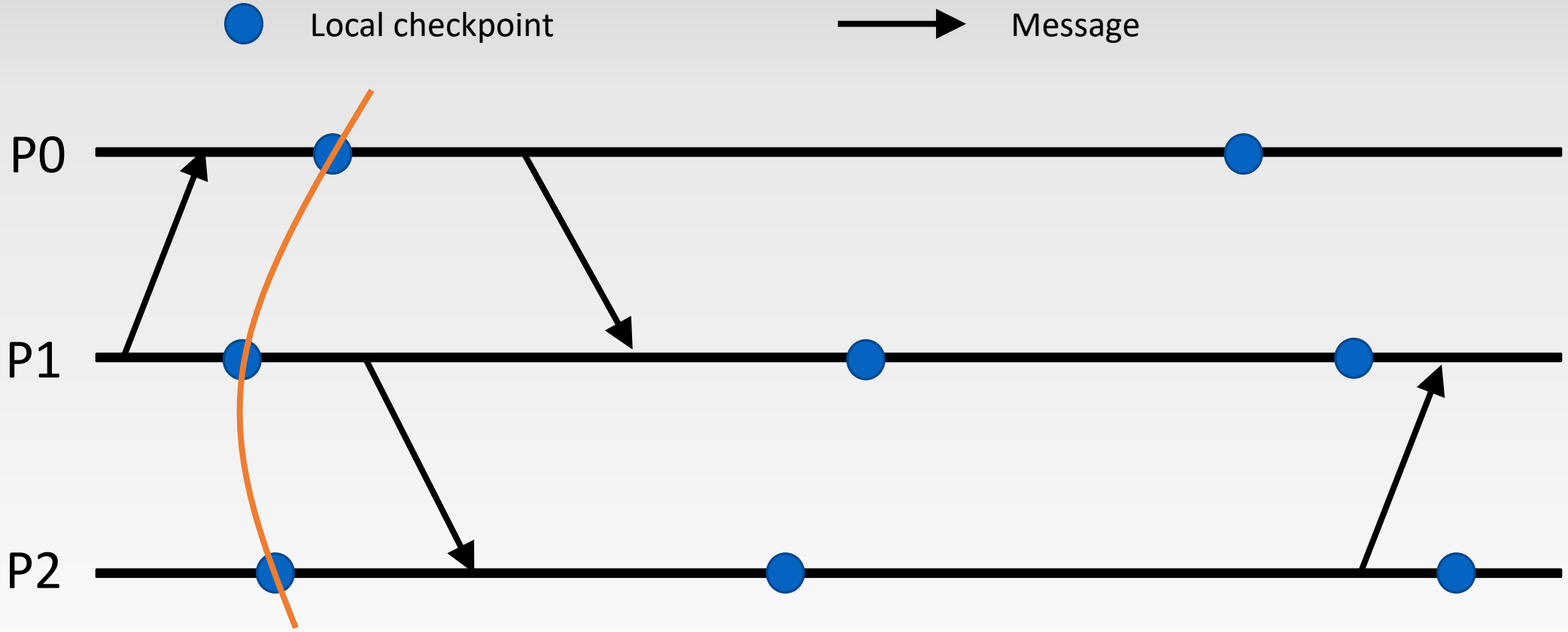
What is the Young/Daley Formula? (High level)

Example



What global state make sense?

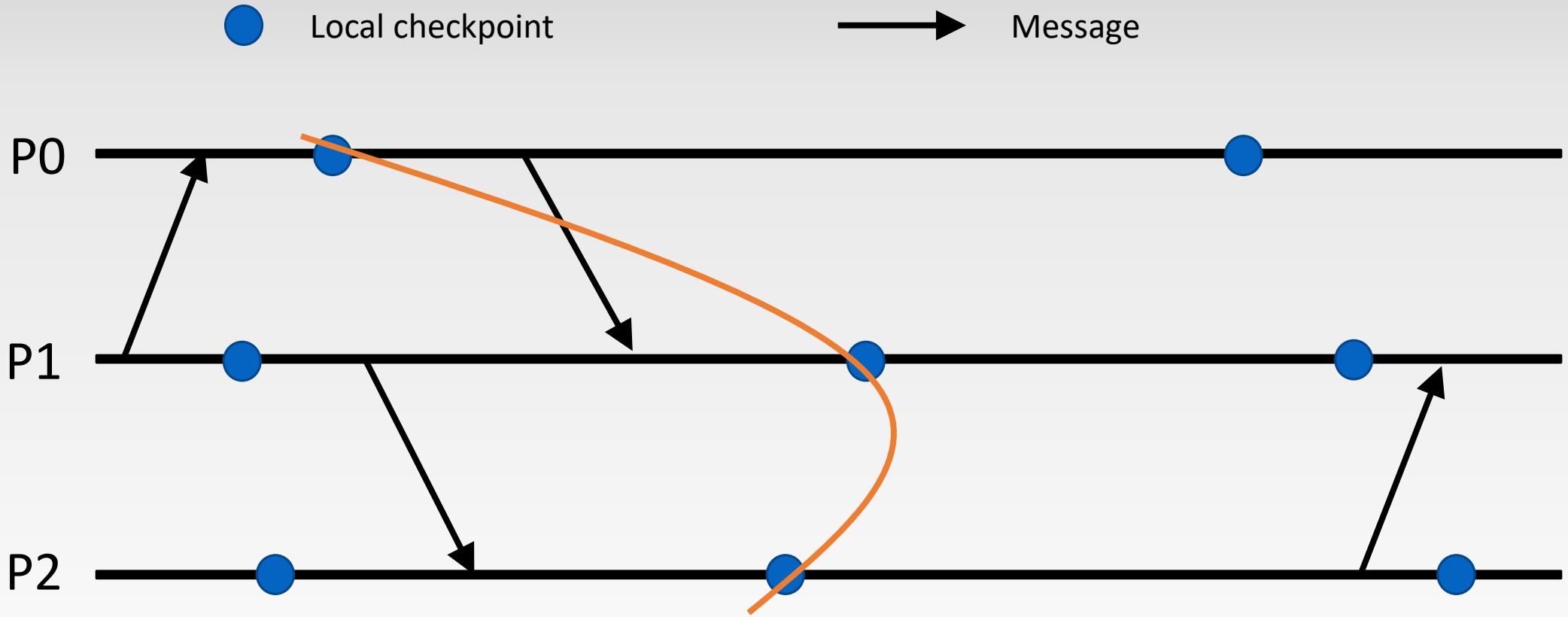
Example



$GS = \{LS_{0,0}, LS_{1,0}, LS_{2,0}\}$

Strongly Consistent

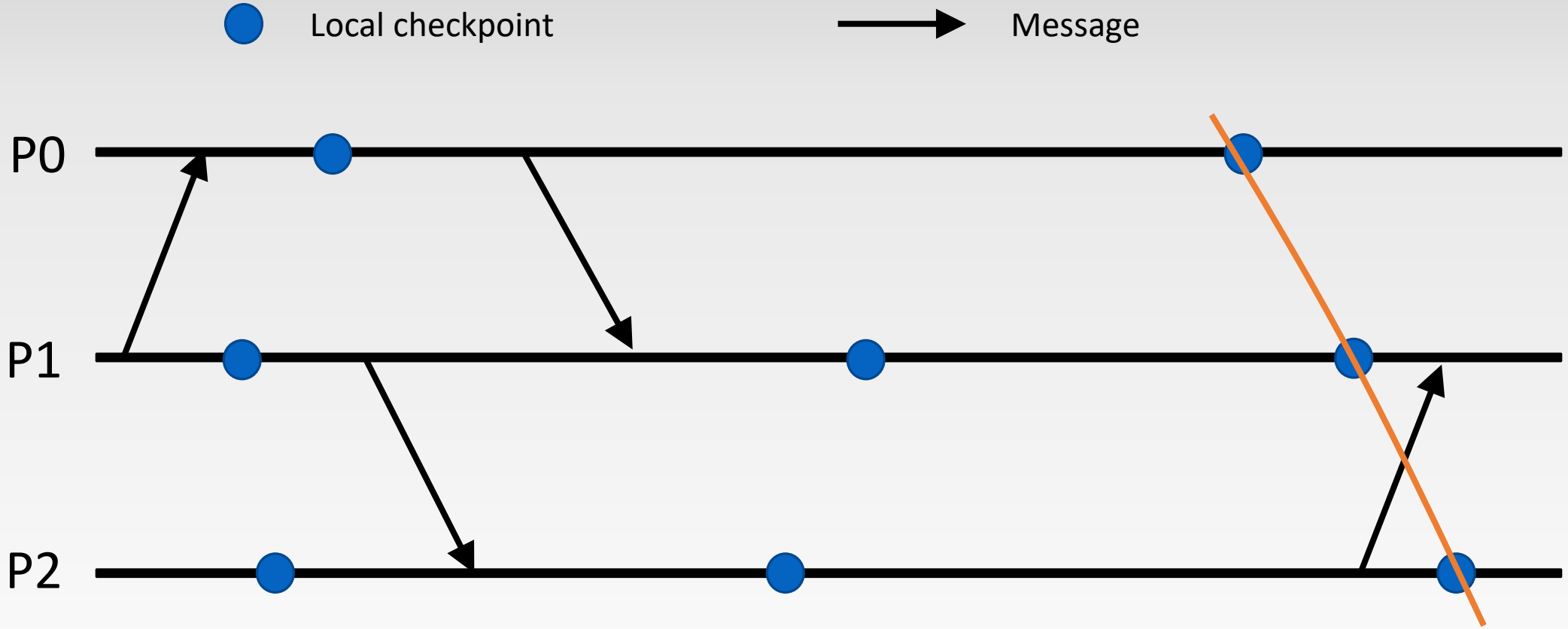
Example



$GS = \{LS_{0,0}, LS_{1,1}, LS_{2,1}\}$

Inconsistent

Example



$GS = \{LS_{0,1}, LS_{1,2}, LS_{2,2}\}$

Consistent

Determining the cut

To have a valid application checkpoint we need at least consistent cut of checkpoints

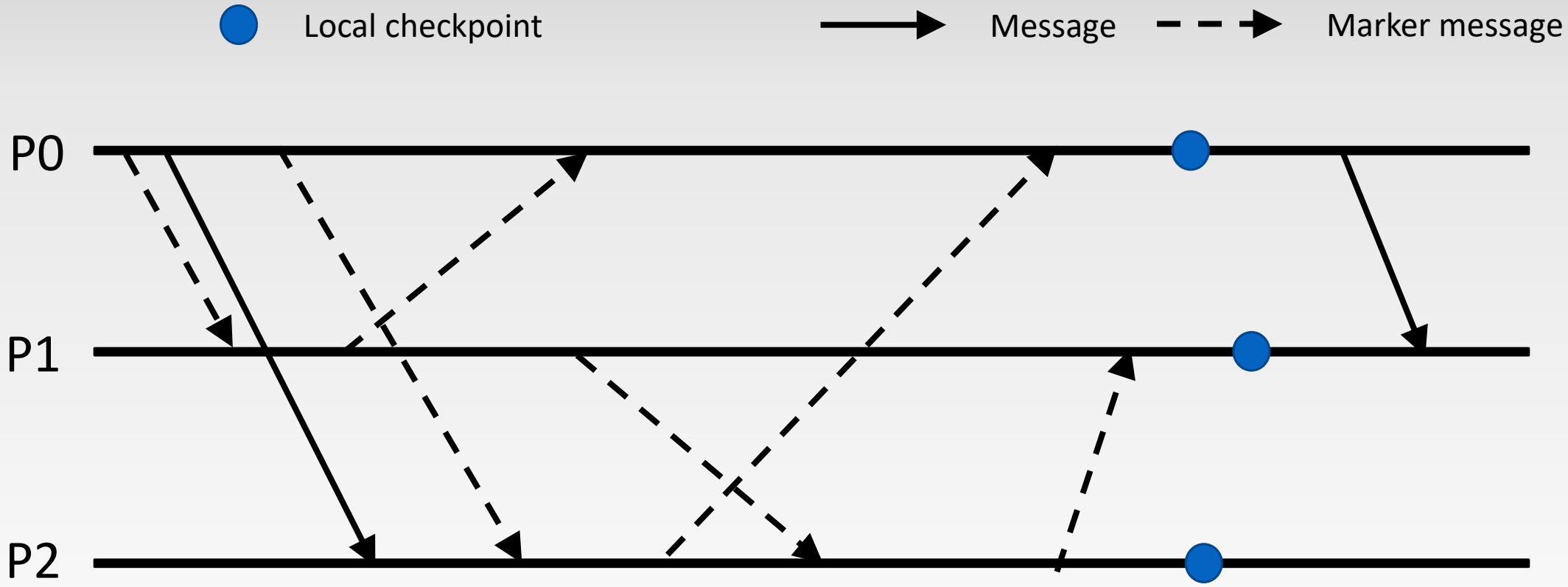
How to make the cut?

Coordinated approach: Use marker messages to indicate that a checkpoint is being taken

Uncoordinated approach: Attempt to form a consistent global cut at recovery time

- Domino effect must be overcome

Example – blocking



Why is there a message sent to P2 after P1 gets a marker message?

Non-blocking coordinated checkpointing

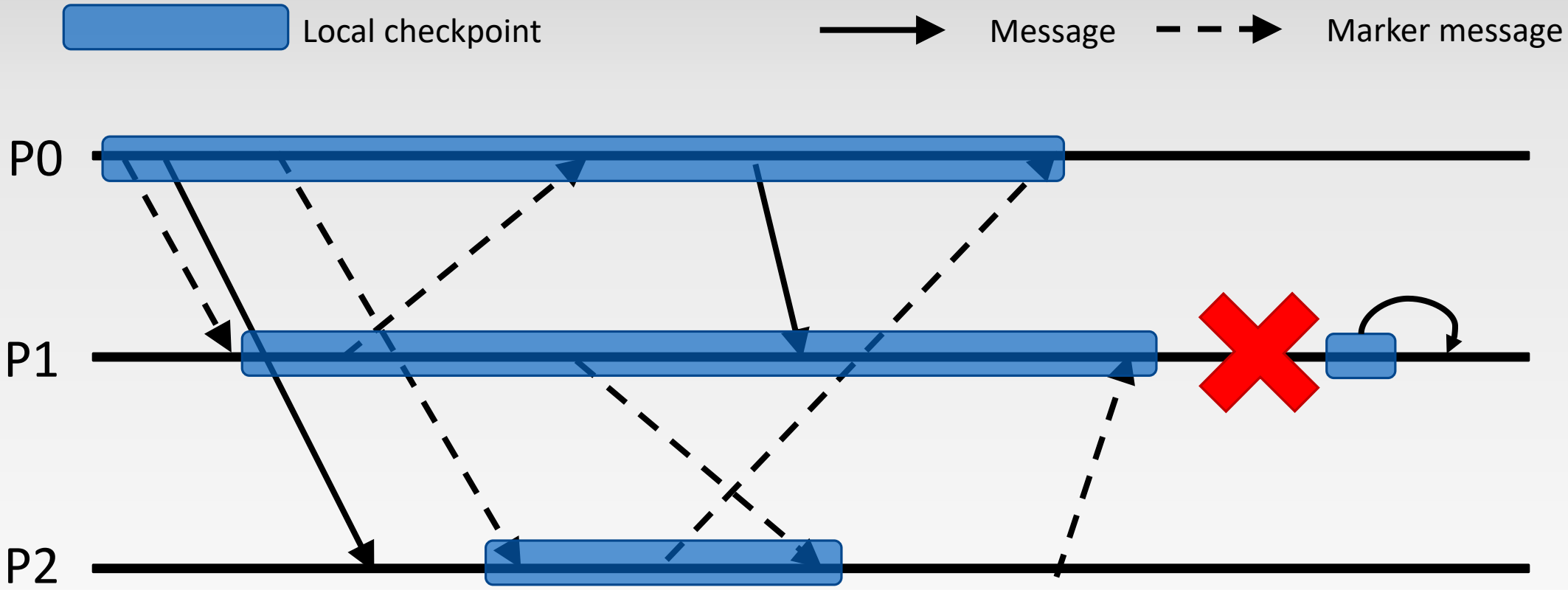
Marker message is an All-to-all operation

After receiving the first marker message

- Checkpoint local state
- Log all incoming messages into the checkpoint until checkpoint is complete
  - Received all marker messages

Checkpoint finishes when all marker messages are received

Example – nonblocking



Message from P0 to P1 stores in P1's checkpoint (receiver logging)

Logging messages

Logging incoming messages on each process helps minimize the amount of computation during a restart

Pessimistic:

- Log incoming message before it is processed
  - Slows down computation

Optimistic:

- Processes does not stop computing
- Incoming messages are stored in volatile storage and logged at certain intervals
  - Messages that have yet to be logged to stable storage are lost if the process fails
  - Does not slow down computation

Optimistic message logging

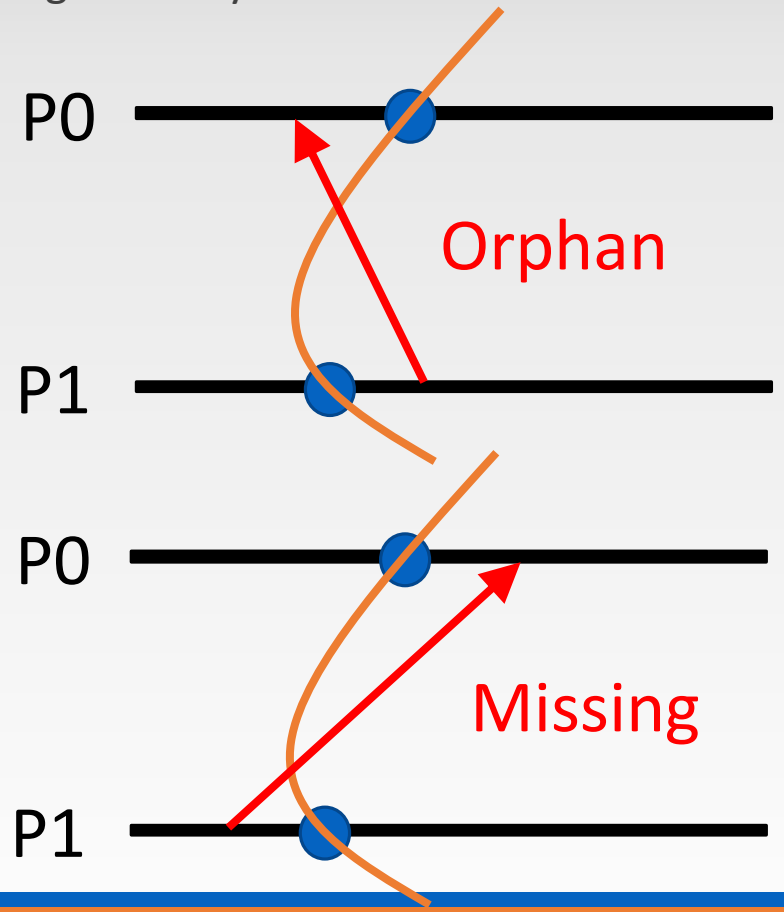
Messages that don't reside in non-volatile storage are not available during recovery

Other process state may causally depend on lost messages

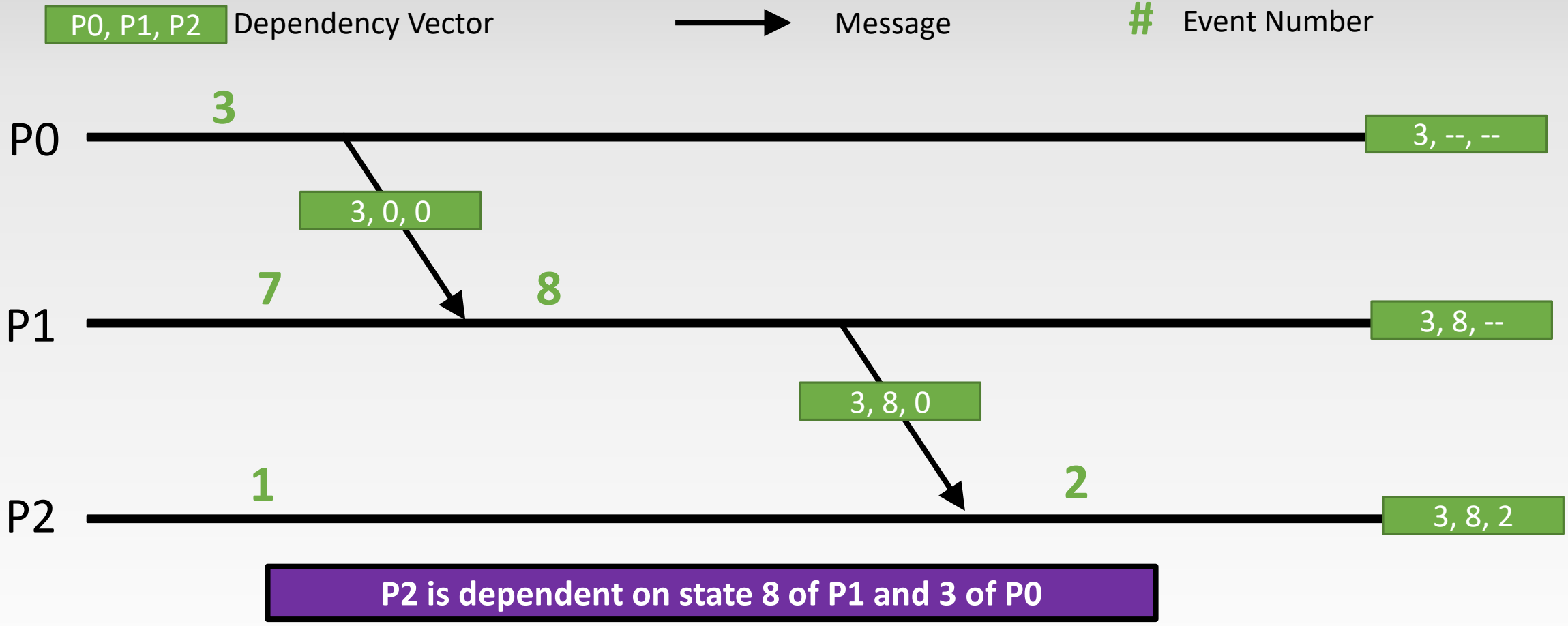
- Creates orphan messages
- Creates missing messages

Orphan and missing messages are determined by tracking event state

- Process consists of sequence of events
- Receipt of message starts a new event
- Outgoing messages dependent upon current event state of a process



Dependency example

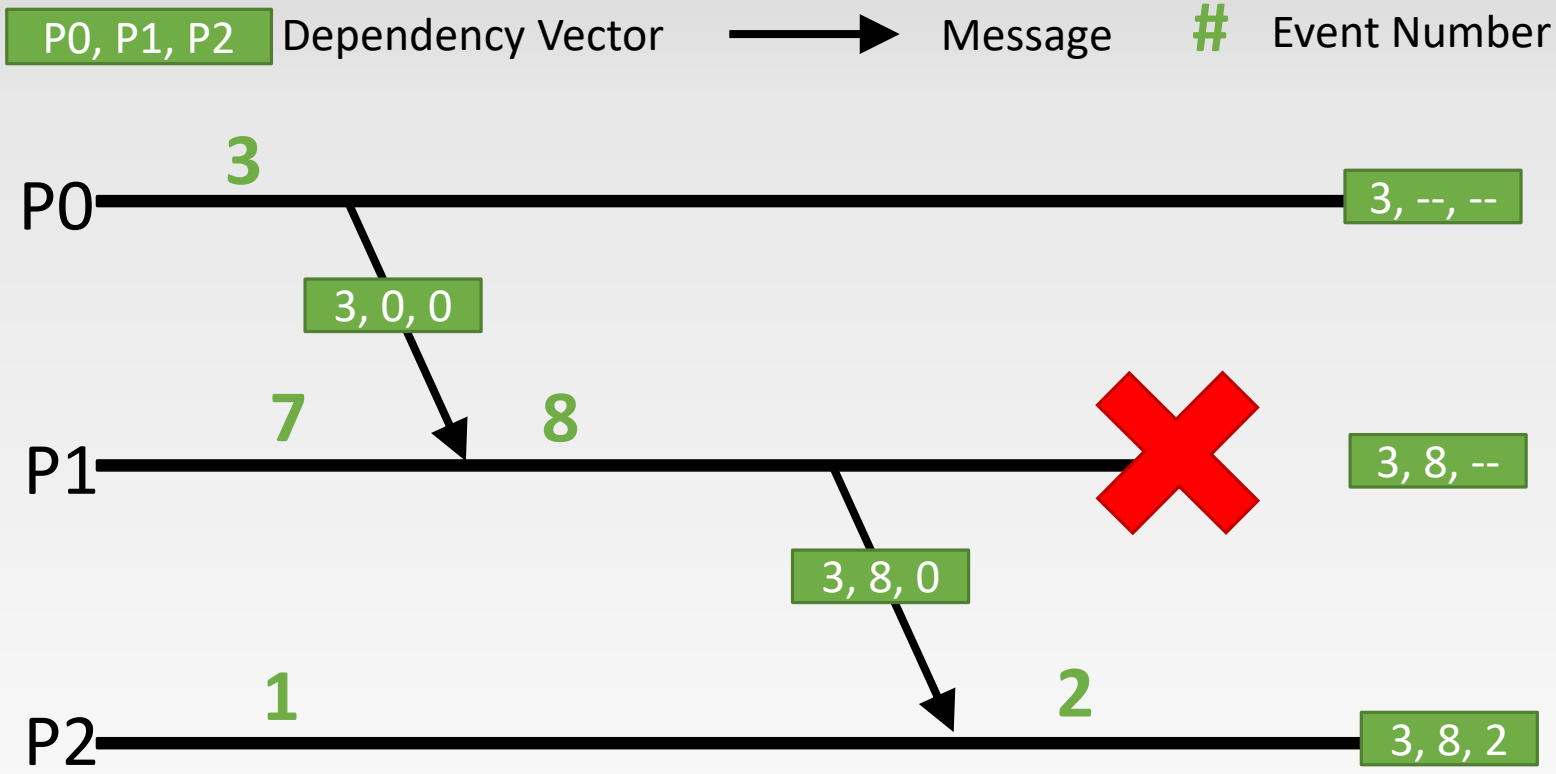


Recovery

If P1 has not logged message from P0, then state **8** is not valid

Restore from state 7

Any process where v[1] >7 must rollback



Part 3: Soft errors

Why is a soft error?

Where do they come from?

What are some solutions to the issue?

Basic coding

Review: Error characteristics

Errors indicate the presence of incorrect state in the system

Faults -> Errors -> Failures

Errors are classified as

- **Detected:** indicated by error message or signal
- **Latent/silent:** not detected
- **Masked:** not causing a failure
- **Soft:** due to a transient fault

What is an error?

What causes errors?

How do we classify errors?

Where do errors come from?

HPC systems are extremely complex and can come from any component or sub-component. At a low level errors can be mapped to:

- Dielectric breakdown and electrical breakdown
- Temperature (extremes and variations)
- Aging
- Manufacturing defects
  Stress

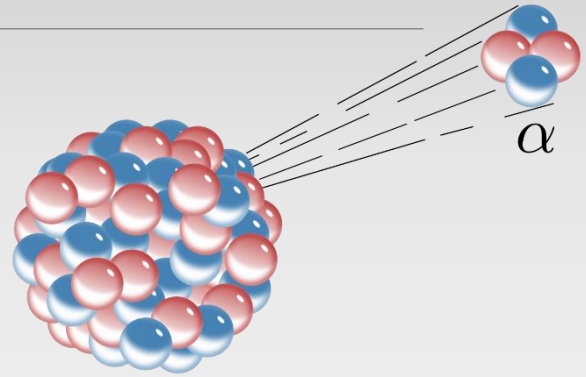
- Extreme conditions
- Voltage fluctuation
- Electro-magnetic interference
- Terrestrial neutrons
- Cosmic radiation
- Alpha particles

Primary Focus

More Radiation

Alpha particles are another source of soft errors

- Alpha particles have the same make up as a Helium atom
- Emitted from radioactive decay



Two main sources:

- **Manufacturing**
  - 1978 Intel failed to deliver chips to AT&T due to trace amounts of uranium in the source materials
  - 1986 IBM used radioactive contaminate to clean bottles used to store an acid needed in chip manufacturing
- **Cosmic Radiation:**
  - Charged alpha particles interact directly with electrons forming “electron-hole pairs”

Outcome of cosmic radiation

Crashes

- Hardware/Software crashes and much be restarted/reset
  - Use checkpoint-restart!

Performance variation

- Wastes time/energy
- What is the difference between this and system noise?

Data corruption

- Wrong results!
- What if the results are only slightly off?
  - 2.0 vs 2.000001

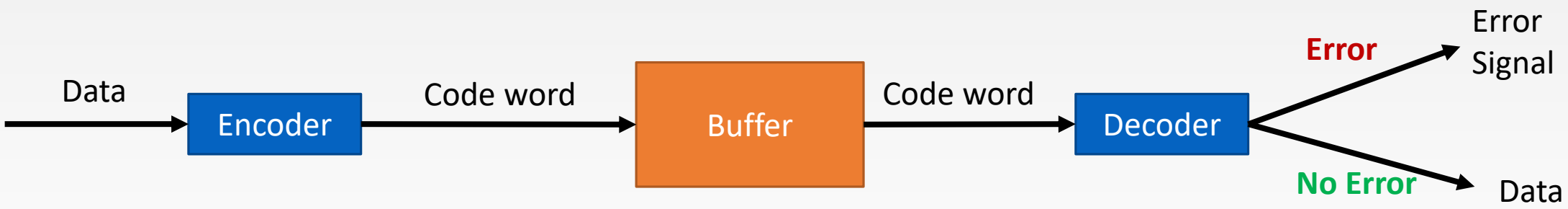
If data corruption is "silent", how do you know it happened?

Coding

Popular and powerful error protection tool used in

- Hardware
- Software

Can be used to detect and correct errors



Our first code

Goal: Protect 1-bit of data

Solution: Add a code bit to the data forming a code word

< data bit, code bit >

Encoding Scheme: code bit = data bit

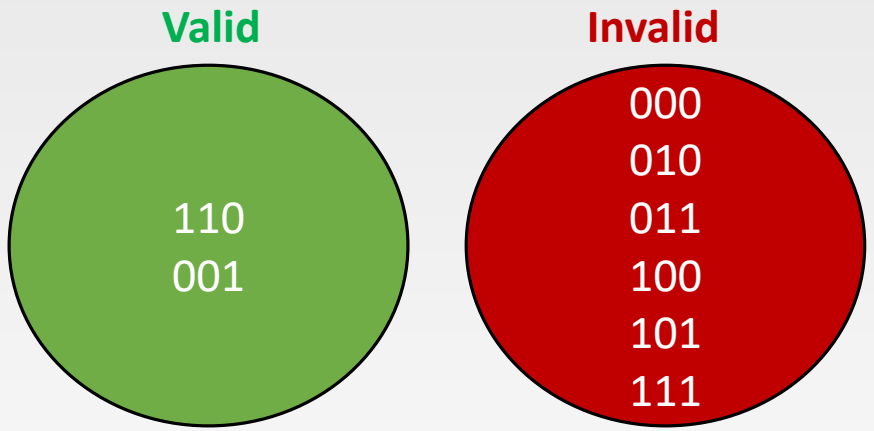
Note: This code is ***separable*** as the data bits and code bits can be separated into two distinct groups

Multi-bit codes

Previous code allows for single error correction and does not allow detection of multi-bit errors

Codewords exist in two groups valid and invalid

The number of bits a coding scheme can detect or correct is determined by the minimum Hamming Distance



Hamming distance

Minimum Hamming distance of a code word determines:

- Number of bit errors that can be detected and/or
- Corrected by the code word

Let:

$\alpha =$ maximum number of bit errors to detect

$\beta =$ maximum number of bit errors to correct

Then:

- Minimum Hamming distance of a code word must be $\alpha + 1$ to detect α or fewer bit errors
- Minimum Hamming distance of a code word must be $2\beta + 1$ to correct β or fewer bit errors
- Minimum Hamming distance of a code word must be $\alpha + \beta + 1$ where $\alpha \geq \beta$ to detect α or fewer bit errors and correct β or fewer bit errors

Minimum Hamming distance

$\alpha + \beta + 1$ where $\alpha \geq \beta$ to detect α or fewer bit errors and correct β or fewer bit errors

- Example: Single Error Correction – Double Error Detection (SEC-DED)

What is the minimum Hamming distance of a code that can detect two faults and correct one?

Detect $\alpha = 2$ and correct $\beta = 1$:
$\alpha + \beta + 1 = 4$

Parity Codes

One of the simplest forms of error detection codes

- A single code bit protects k bits of data
- Rate is 1 for sufficiently large k

Even Parity: code bit = 1 if odd number of bits set

- Even Parity is computed by **XOR**ing all of the k bits

Odd Parity: code bit = 1 if and even number of bits set

Parity Codes

Can parity codes detect multi-bit errors?

Odd number of bit errors results in parity inversion

Even number of bit errors results in masking!!!

Parity Codes

Burst errors result in several consecutive bits becoming in error

How can we detect spatially contiguous multi-bit errors?

Interleaving

Let’s interleave the 4 bits code word A and code word B:

Parity Codes

Let's interleave the 4 bits word A and word B:

Is there a limit to how much we can interleave or protect with parity?

More interleaving means more distance between bits which can increase time and power for the circuit

Extending Parity

Parity has a very low overhead (rate = 1), but is not by itself very powerful

- Only able to detect
- Can not correct

Let us build a more complex protection scheme that uses parity, but is able to detect and recover from bit errors

Single-error correction (SEC) codes

Given a set of k data bits arrange them into a 2D matrix and compute row and column parity

0	0	1	0	1
1	0	0	1	0
1	1	1	1	0
0	1	0	0	

Single-error correction (SEC) codes

If error occurs in data bits, row and column parity can locate incorrect entry

- Similar logic can be used to detect and correct errors in parity
- If one parity entry is in error (shown below) replace with recomputed parity

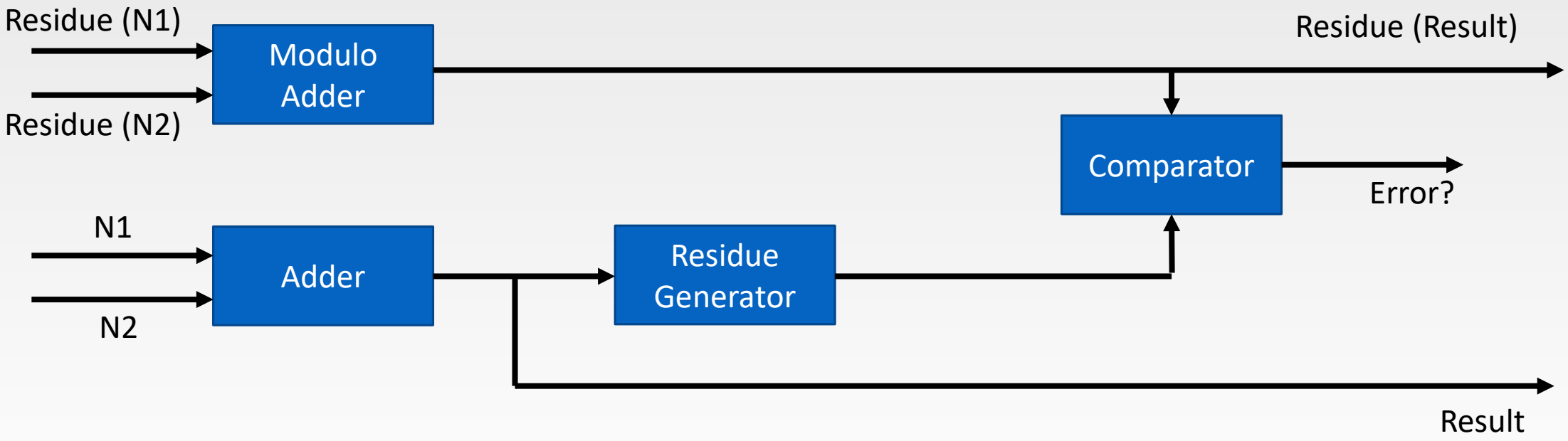
0	0	1	0	1
1	0	0	1	1
1	1	1	1	0
0	1	0	0	

What is an issue with this approach?

Rate not optimal

Residue Codes

Based on the idea of modular arithmetic



Part 4: Approximate computing and lossy compression

What does it mean to be accurate?

Why types of errors exist?

Describe techniques

What is lossy compression?

How has it been used?

Accuracy

What does it mean to be accurate? Give an example.

Various definitions of **accurate**:

- Bit-reproduceable
- High precision / High recall
- Solution works
- Solution is optimal

Accuracy

How are integers represented?

Bit vectors of a fixed length (e.g, 32 or 64 bits)

Each bit x_i represents the coefficient (1 or 0) a power of 2 in the following function

$v = \sum_{i=0}^{n} x_i * 2^i$

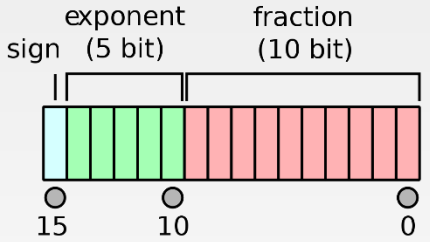
Inverting bit i perturbs v by $\pm 2^i$

How is floating-point represented?

Bit vectors of a fixed length (e.g, 32 or 64 bits)

Each bit x_i represents either:

- Sign bit (s)
- Exponent bits (E)
- Mantissa bits (M)



$v = -1^s * 2^{E-127} * (1 + M/2^{23})$

Inverting bit i perturbs v differently depending on which component it is in

Encoding data

How would you encode the number 0-17?

5 bits where each bit is a power of 2

How would you encode the capital letters A-Z?

5 bits where each bit is a power of 2

Fixed-Rate Codes

The ASCII table is an example of a fixed-rate encoding scheme

Each symbol is given a fixed sized code word

Size of the stream is therefore: #symbols * sizeof(code word)

What can we do to represent the stream with fewer bits?

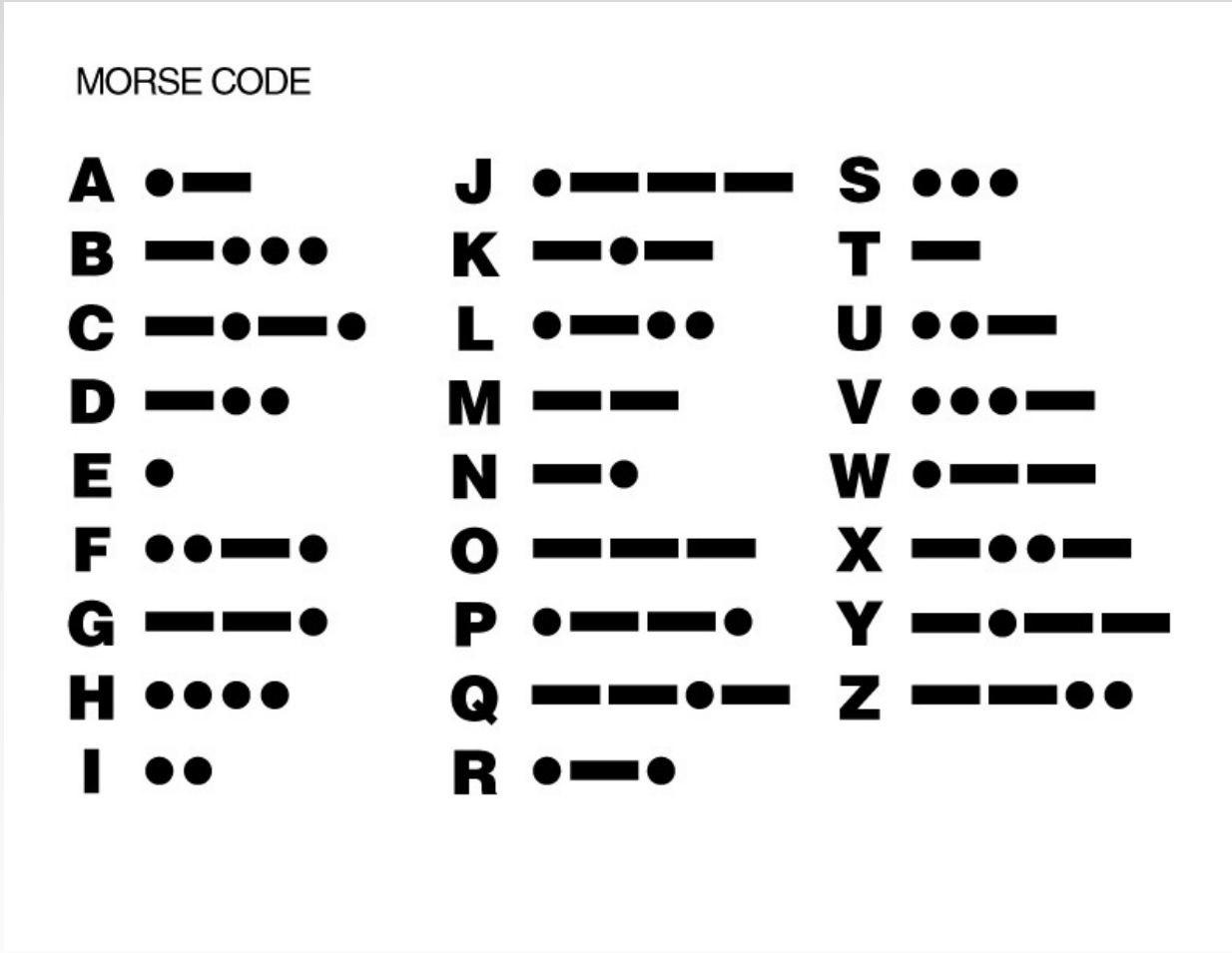
Encode with fewer bits

How many bits do we need to encode the capitals A-Z?

5 bits with a fixed rate code

What is a well known encoding scheme used for A-Z?

Morse Code



Variable length code

Morse Code is an example of a variable length code (VLC)

The code words from a VLC have different sizes

Variable length codes assign the shortest code words to the most frequent symbols

Morse Code uses the inverse frequency of each letter when assigning code words

Example VLC

81% savings

Encode the string "CDACCCCD" using ASCII

01000011 01000100
01000001 01000011
01000011 01000011
01000011 01000100

Symbol	Probability	Codeword
A	5%	100
B	0%	101
C	80%	0
D	15%	11

Encode the string "CDACCCCD" using a VLC

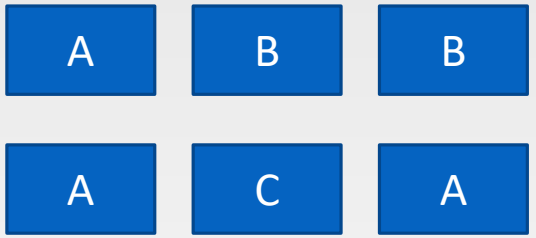
0 11 100 0 0 0 0 11

VLC example

Symbol	Codeword
A	0
B	10
C	101

Read **1 bit** at a time until you can make a code word

Decode the string: 01010



Prefix property: Once a code is assigned to a given symbol, no other code can start with that pattern

Summary

Fault tolerance is important for efficient use of HPC resources

Checkpoint restart and redundancy are two fundamental techniques

Soft errors can be harmful or helpful depending on how they are generated