# Corrected Trees for Reliable Group Communication

Martin Küttler[1] Maksym Planeta[1] Jan Bierbaum[1] Carsten Weinhold[1] Hermann Härtig[1]
Amnon Barak[2] Torsten Hoefler[3]

[1]TU Dresden [2]The Hebrew University of Jerusalem [3] ETH Zürich

Principles and Practice of Parallel Programming, Washington D.C.

2019-02-19

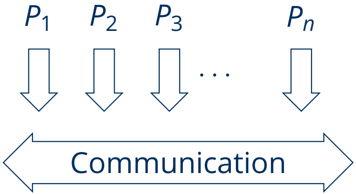
# Motivation

# Motivation

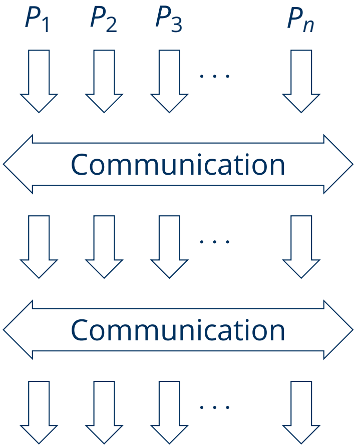
$P_1$ $P_2$ $P_3$ $\ldots$ $P_n$

$\Downarrow$ $\Downarrow$ $\Downarrow$ $\ldots$ $\Downarrow$

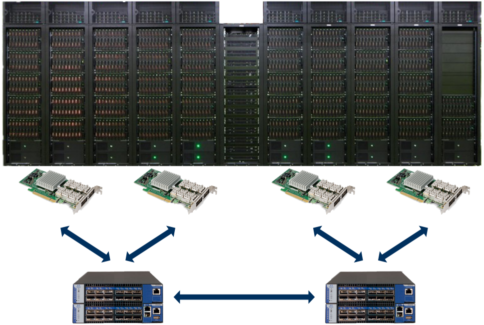
# Motivation

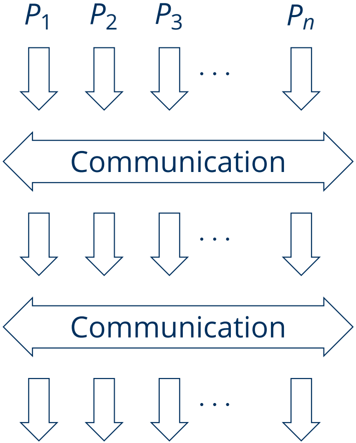


# Motivation

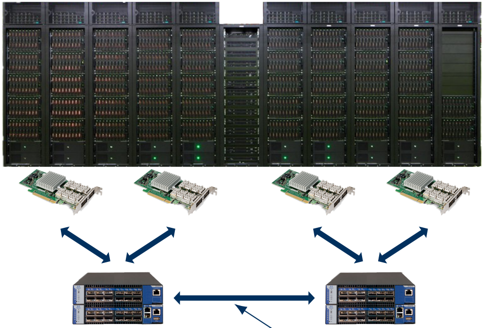


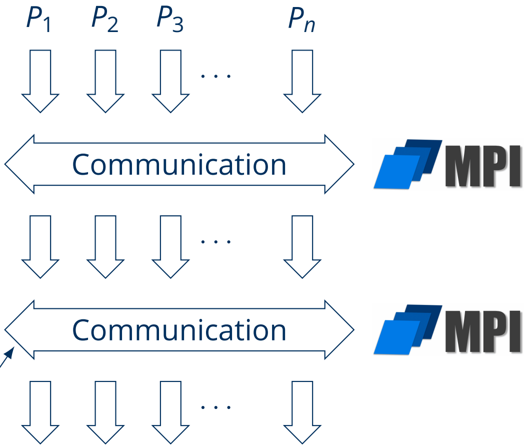
# Motivation





# Motivation






Scalability bottleneck

# Race to Exascale


200 PFLOPS
2 million cores

# Race to Exascale


200 PFLOPS
2 million cores

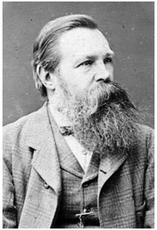
1000 PFLOPS

# Race to Exascale


1000 PFLOPS

# Race to Exascale

*"...quantitative change alters quality"*
Friedrich Engels

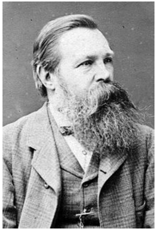



1000 PFLOPS

# Race to Exascale

*"...quantitative change alters quality"*
Friedrich Engels


1000 PFLOPS



Scalability ⟺ Fault Tolerance

# Broadcast

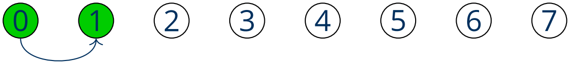
- Collective operation
- $N$ processes

# Broadcast

0 1 2 3 4 5 6 7

- Collective operation
- *N* processes
- Designated root

# Broadcast

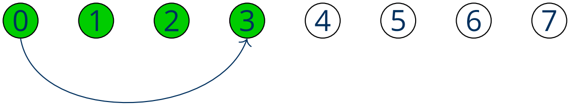



- Collective operation
- *N* processes
- Designated root
- Point-to-point messages

# Broadcast

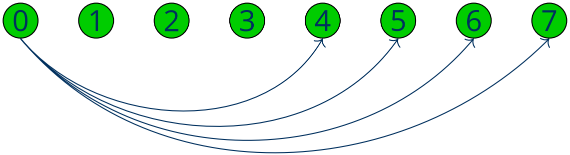

- Collective operation
- *N* processes
- Designated root
- Point-to-point messages
- Messages color processes

# Broadcast



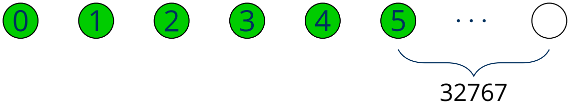

- Collective operation
- *N* processes
- Designated root
- Point-to-point messages
- Messages color processes

# Broadcast




- Collective operation
- *N* processes
- Designated root
- Point-to-point messages
- Messages color processes
- Linear broadcast

# Broadcast




- Collective operation
- *N* processes
- Designated root
- Point-to-point messages
- Messages color processes
- Linear broadcast
- Bad scalability

# Outline

- Scalable broadcast
- Fault tolerant broadcast

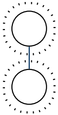
# Outline

- Scalable broadcast
- Fault tolerant broadcast
- Our contribution: Corrected Trees
    1. Construction for several variants
    2. Theoretical analysis
    3. Study with network simulation
    4. Proof-of-concept implementation

# Tree-based Broadcast

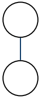
- Binomial tree
  - Start from the root

# Tree-based Broadcast



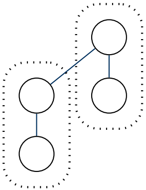
- Binomial tree
  - Start from the root
  - Copy the tree

# Tree-based Broadcast



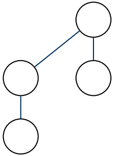
- Binomial tree
  - Start from the root
  - Copy the tree

# Tree-based Broadcast



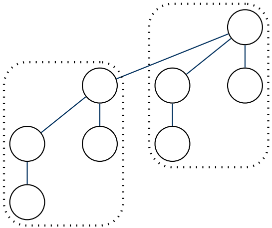
- Binomial tree
  - Start from the root
  - Copy the tree
  - Repeat

# Tree-based Broadcast



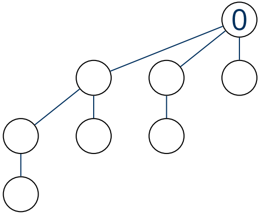
- Binomial tree
  - Start from the root
  - Copy the tree
  - Repeat

# Tree-based Broadcast



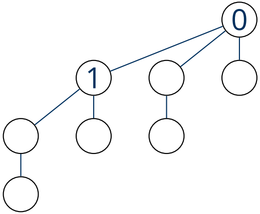
- Binomial tree
  - Start from the root
  - Copy the tree
  - Repeat

# Tree-based Broadcast



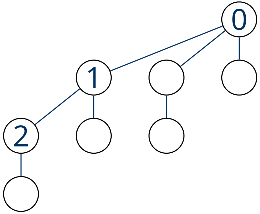

- Binomial tree
  - Start from the root
  - Copy the tree
  - Repeat
  - Assign processes in Depth First order

# Tree-based Broadcast



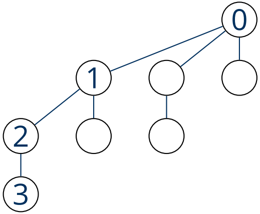

- Binomial tree
  - Start from the root
  - Copy the tree
  - Repeat
  - Assign processes in Depth First order

# Tree-based Broadcast



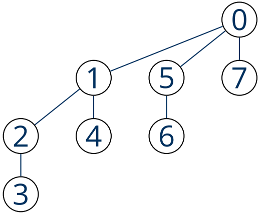

- Binomial tree
  - Start from the root
  - Copy the tree
  - Repeat
  - Assign processes in Depth First order

# Tree-based Broadcast



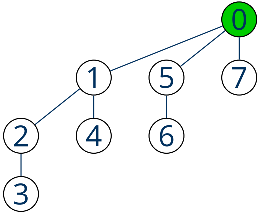

- Binomial tree
  - Start from the root
  - Copy the tree
  - Repeat
  - Assign processes in Depth First order

# Tree-based Broadcast



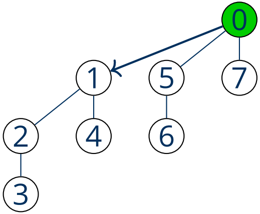

- Binomial tree
  - Start from the root
  - Copy the tree
  - Repeat
  - Assign processes in Depth First order

# Tree-based Broadcast



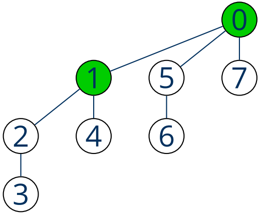

- Binomial tree
  - Start from the root
  - Copy the tree
  - Repeat
  - Assign processes in Depth First order
- Broadcast
  - Root has the message

# Tree-based Broadcast



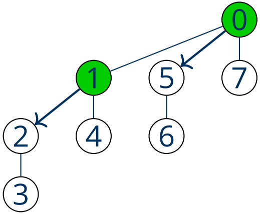

- Binomial tree
  - Start from the root
  - Copy the tree
  - Repeat
  - Assign processes in Depth First order
- Broadcast
  - Root has the message

# Tree-based Broadcast



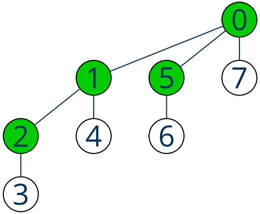

- Binomial tree
  - Start from the root
  - Copy the tree
  - Repeat
  - Assign processes in Depth First order
- Broadcast
  - Root has the message

# Tree-based Broadcast



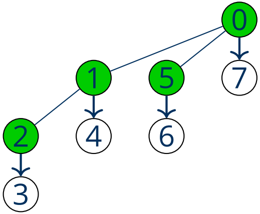

- Binomial tree
  - Start from the root
  - Copy the tree
  - Repeat
  - Assign processes in Depth First order
- Broadcast
  - Root has the message
  - Send in parallel

# Tree-based Broadcast



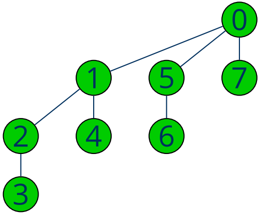

- Binomial tree
  - Start from the root
  - Copy the tree
  - Repeat
  - Assign processes in Depth First order
- Broadcast
  - Root has the message
  - Send in parallel

# Tree-based Broadcast



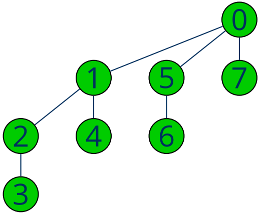

- Binomial tree
  - Start from the root
  - Copy the tree
  - Repeat
  - Assign processes in Depth First order
- Broadcast
  - Root has the message
  - Send in parallel
  - Double every time

# Tree-based Broadcast



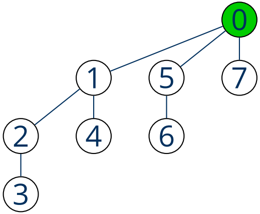

- Binomial tree
  - Start from the root
  - Copy the tree
  - Repeat
  - Assign processes in Depth First order
- Broadcast
  - Root has the message
  - Send in parallel
  - Double every time

# Tree-based Broadcast

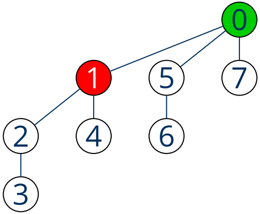



- Binomial tree
  - Start from the root
  - Copy the tree
  - Repeat
  - Assign processes in Depth First order
- Broadcast
  - Root has the message
  - Send in parallel
  - Double every time
  - $O(\log(n))$ depth

# Broadcast with a Failed Process

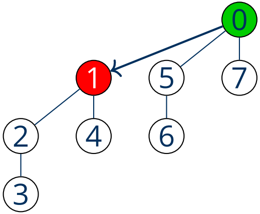


# Broadcast with a Failed Process



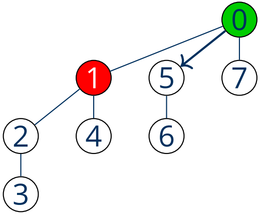

- Non-leaf failure

# Broadcast with a Failed Process



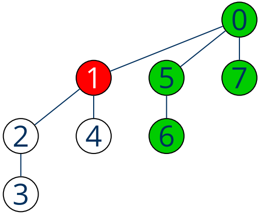

- Non-leaf failure
- Message is silently lost

# Broadcast with a Failed Process



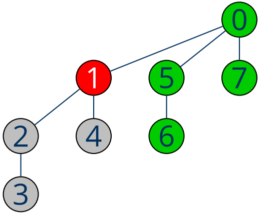

- Non-leaf failure
- Message is silently lost

# Broadcast with a Failed Process




- Non-leaf failure
- Message is silently lost
- Partial coloring

# Broadcast with a Failed Process




- Non-leaf failure
- Message is silently lost
- Partial coloring
- Healthy processes remain uncolored

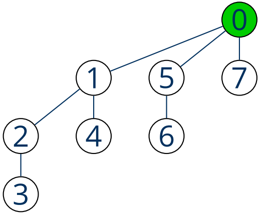
# Failures

- Every live process receives a message (like linear broadcast)
- Low latency (like tree broadcast)
- Fail-stop model
- Reliable messages
- The root is alive
- Multiple failures: details in the paper

# Failures

- Every live process receives a message (like linear broadcast)
- Low latency (like tree broadcast)
- Fail-stop model
- Reliable messages
- The root is alive
- Multiple failures: details in the paper
- Future MPI standards will handle failures (example: ULFM)

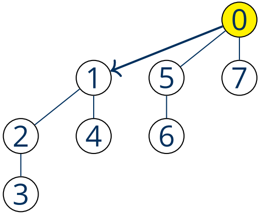
# Acknowledged Broadcast[1]




- Broadcast along the tree

[1] Buntinas, “Scalable Distributed Consensus to Support MPI Fault Tolerance.”

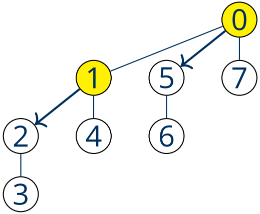
# Acknowledged Broadcast[1]




- Broadcast along the tree

[1] Buntinas, "Scalable Distributed Consensus to Support MPI Fault Tolerance."

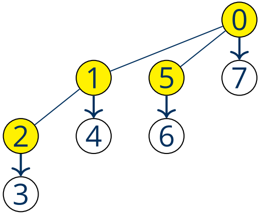
# Acknowledged Broadcast[1]




- Broadcast along the tree
- Waiting for ACKs

[1] Buntinas, "Scalable Distributed Consensus to Support MPI Fault Tolerance."

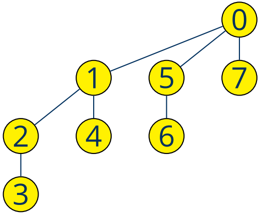
# Acknowledged Broadcast[1]




- Broadcast along the tree
- Waiting for ACKs

[1] Buntinas, "Scalable Distributed Consensus to Support MPI Fault Tolerance."

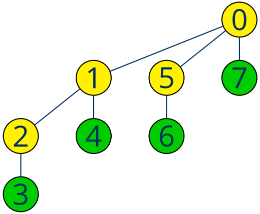
# Acknowledged Broadcast[1]




- Broadcast along the tree
- Waiting for ACKs

[1] Buntinas, “Scalable Distributed Consensus to Support MPI Fault Tolerance.”

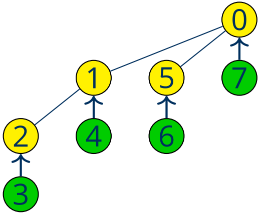
# Acknowledged Broadcast[1]




- Broadcast along the tree
- Waiting for ACKs
- Ready when no ACK pending

[1] Buntinas, “Scalable Distributed Consensus to Support MPI Fault Tolerance.”

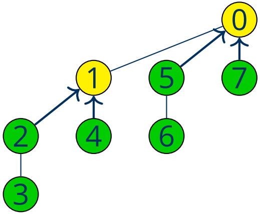
# Acknowledged Broadcast[1]




- Broadcast along the tree
- Waiting for ACKs
- Ready when no ACK pending
- Children send ACKs

[1] Buntinas, “Scalable Distributed Consensus to Support MPI Fault Tolerance.”

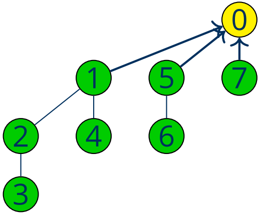
# Acknowledged Broadcast[1]




- Broadcast along the tree
- Waiting for ACKs
- Ready when no ACK pending
- Children send ACKs

[1] Buntinas, "Scalable Distributed Consensus to Support MPI Fault Tolerance."

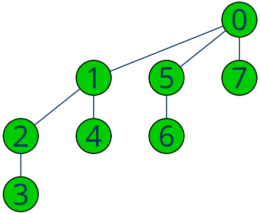
# Acknowledged Broadcast[1]




- Broadcast along the tree
- Waiting for ACKs
- Ready when no ACK pending
- Children send ACKs

[1] Buntinas, "Scalable Distributed Consensus to Support MPI Fault Tolerance."

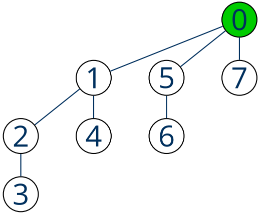
# Acknowledged Broadcast[1]




- Broadcast along the tree
- Waiting for ACKs
- Ready when no ACK pending
- Children send ACKs
- The root waits for all ACKs

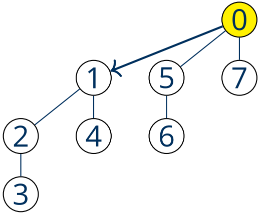
[1] Buntinas, "Scalable Distributed Consensus to Support MPI Fault Tolerance."

# Acknowledged Broadcast with Failures



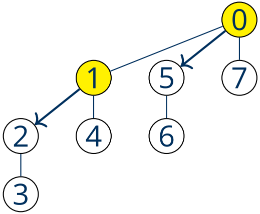

- Run as usual

# Acknowledged Broadcast with Failures



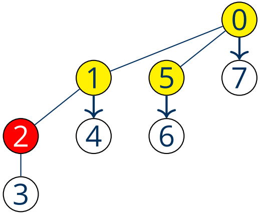

- Run as usual

# Acknowledged Broadcast with Failures



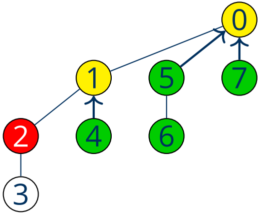

- Run as usual

# Acknowledged Broadcast with Failures



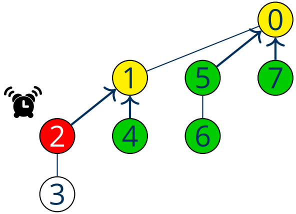

- Run as usual
- Message is lost

# Acknowledged Broadcast with Failures



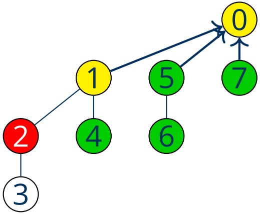

- Run as usual
- Message is lost

# Acknowledged Broadcast with Failures



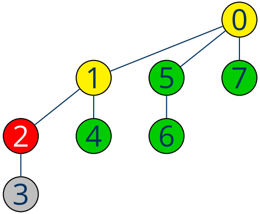

- Run as usual
- Message is lost
- Failure detection

# Acknowledged Broadcast with Failures



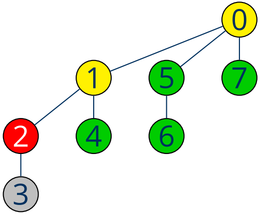

- Run as usual
- Message is lost
- Failure detection
- Notify the root

# Acknowledged Broadcast with Failures



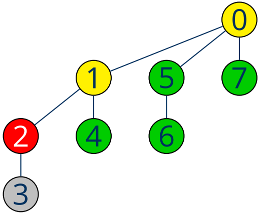

- Run as usual
- Message is lost
- Failure detection
- Notify the root
- Restructure the tree

# Acknowledged Broadcast with Failures



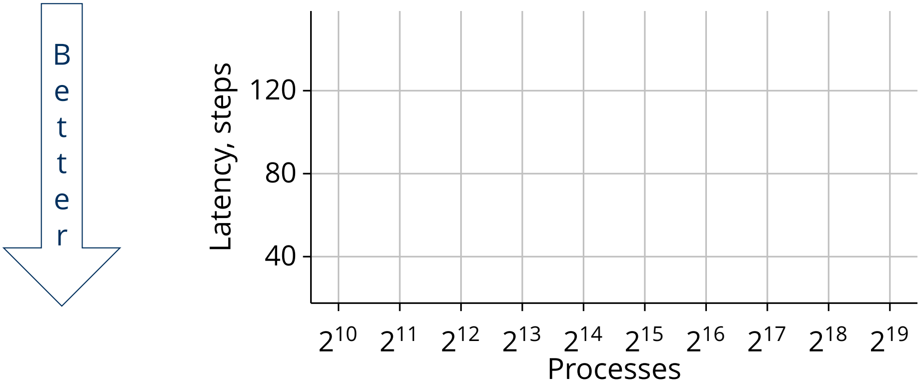

- Run as usual
- Message is lost
- Failure detection
- Notify the root
- Restructure the tree
- Double the latency

# Acknowledged Broadcast with Failures

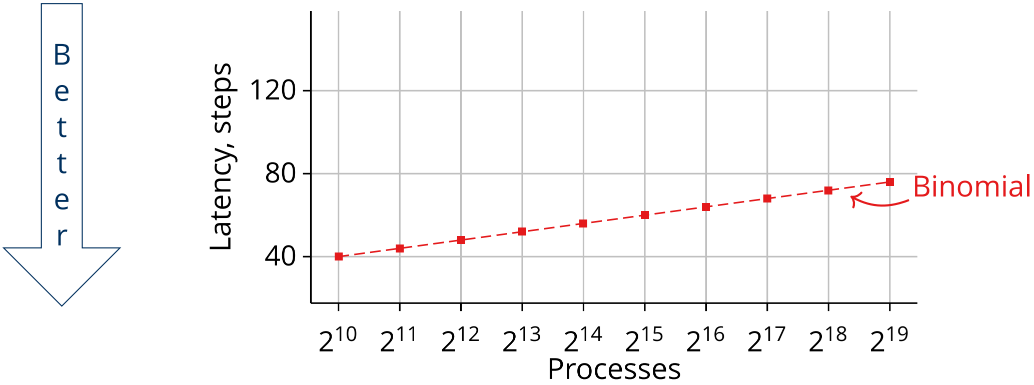



- Run as usual
- Message is lost
- Failure detection
- Notify the root
- Restructure the tree
- Double the latency
- Unnecessary waiting

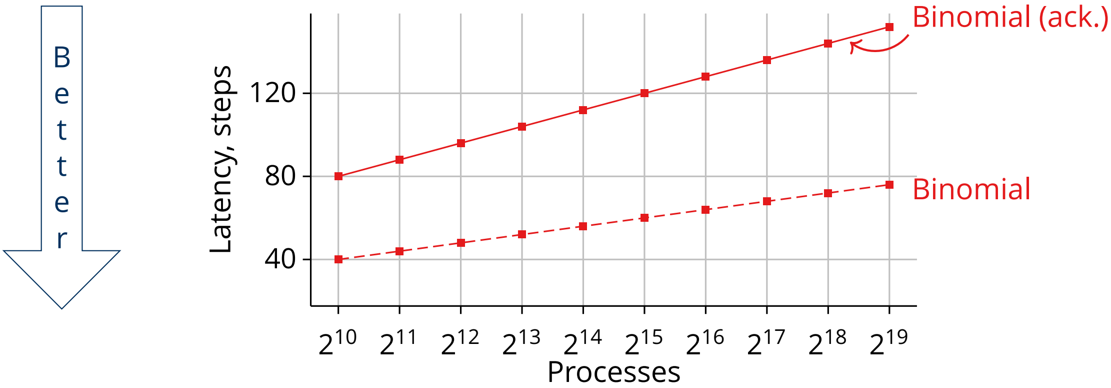
# Simulated Latency: Acknowledged Broadcast



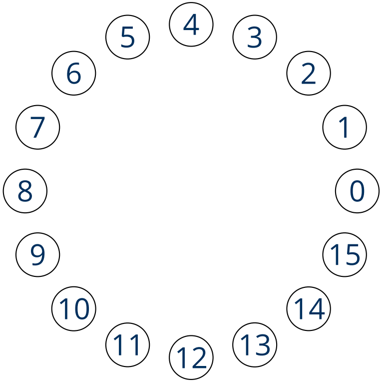
# Simulated Latency: Acknowledged Broadcast



# Simulated Latency: Acknowledged Broadcast



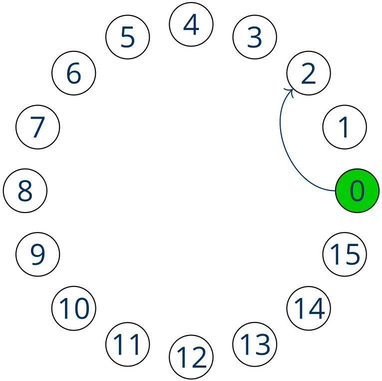
# Gossip-based Broadcast[2]




- No tree structure

[2] Birman et al., “Bimodal Multicast.”

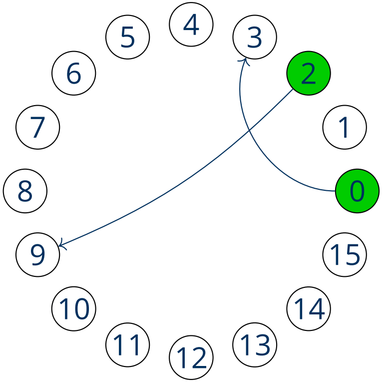
# Gossip-based Broadcast[2]




- No tree structure
- Send to random process (Gossip)

[2]Birman et al., "Bimodal Multicast."

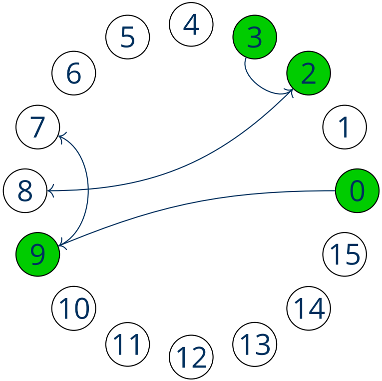
# Gossip-based Broadcast[2]




- No tree structure
- Send to random process (Gossip)

[2] Birman et al., "Bimodal Multicast."

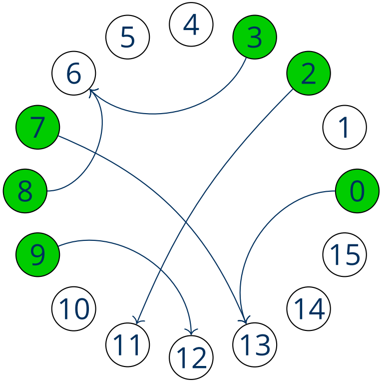
# Gossip-based Broadcast[2]




- No tree structure
- Send to random process (Gossip)
- Multiple rounds

[2] Birman et al., “Bimodal Multicast.”

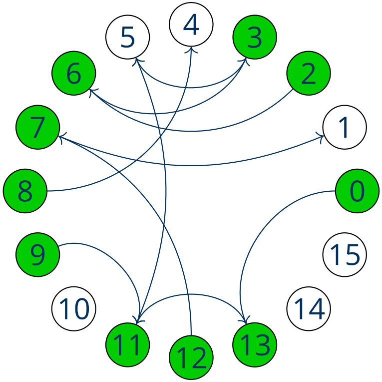
# Gossip-based Broadcast[2]




- No tree structure
- Send to random process (Gossip)
- Multiple rounds

[2] Birman et al., "Bimodal Multicast."

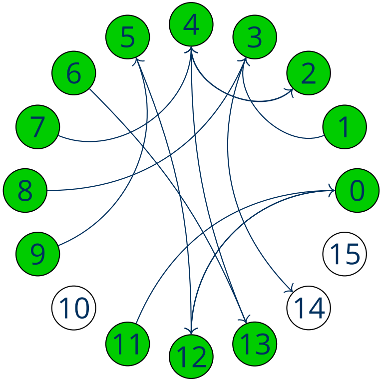
# Gossip-based Broadcast[2]




- No tree structure
- Send to random process (Gossip)
- Multiple rounds

[2] Birman et al., "Bimodal Multicast."

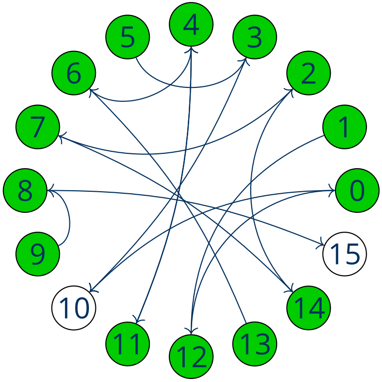
# Gossip-based Broadcast[2]




- No tree structure
- Send to random process (Gossip)
- Multiple rounds

[2] Birman et al., “Bimodal Multicast.”

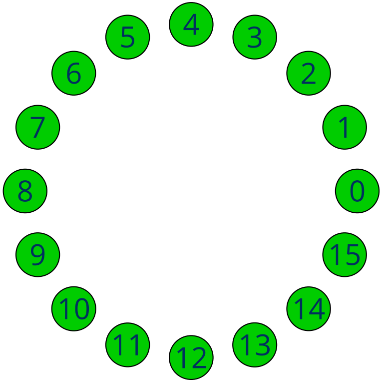
# Gossip-based Broadcast[2]




- No tree structure
- Send to random process (Gossip)
- Multiple rounds

[2] Birman et al., “Bimodal Multicast.”

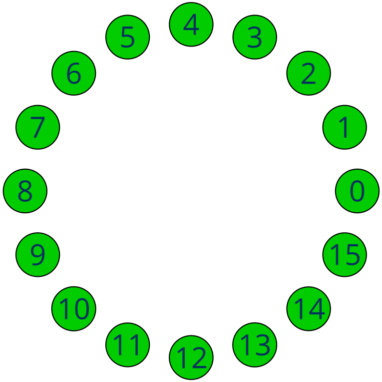
# Gossip-based Broadcast[2]




- No tree structure
- Send to random process (Gossip)
- Multiple rounds
- Full reachability with high probability

[2] Birman et al., “Bimodal Multicast.”

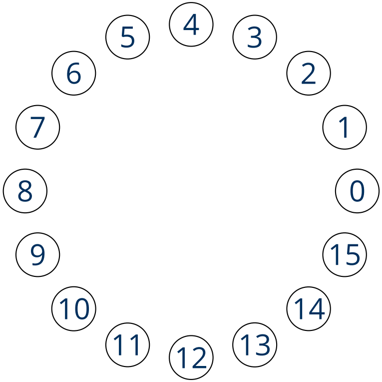
# Gossip-based Broadcast[2]




- No tree structure
- Send to random process (Gossip)
- Multiple rounds
- Full reachability with high probability
- Inherently fault tolerant
- No need for failure detector

[2] Birman et al., “Bimodal Multicast.”

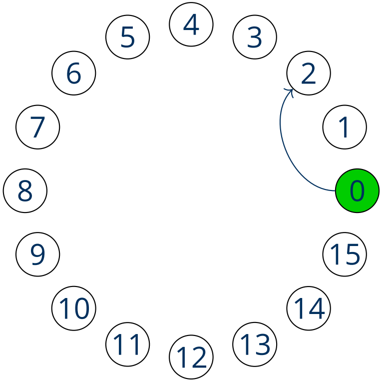
# Corrected Gossip-based Broadcast[3]




- Keep processes in a ring

[3] Hoefler et al., "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems."

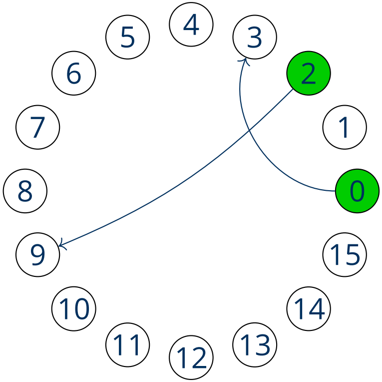
# Corrected Gossip-based Broadcast[3]




- Keep processes in a ring
- Gossip phase

[3] Hoefler et al., "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems."

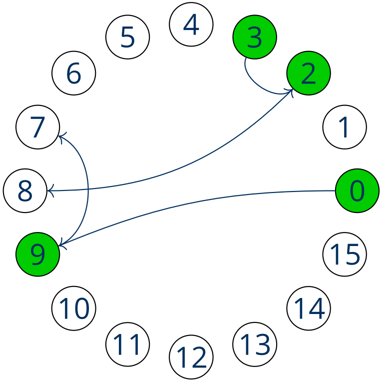
# Corrected Gossip-based Broadcast[3]




- Keep processes in a ring
- Gossip phase

[3] Hoefler et al., “Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems.”

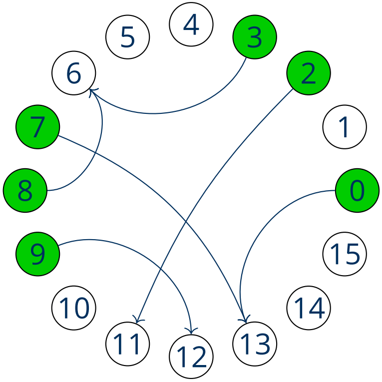
# Corrected Gossip-based Broadcast[3]




- Keep processes in a ring
- Gossip phase

[3] Hoefler et al., "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems."

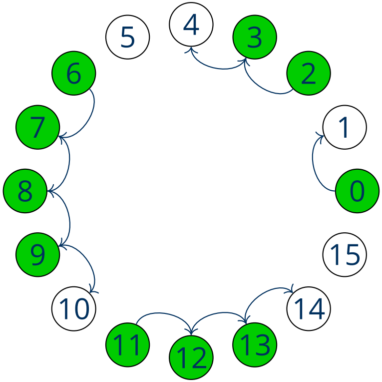
# Corrected Gossip-based Broadcast[3]




- Keep processes in a ring
- Gossip phase

[3] Hoefler et al., "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems."

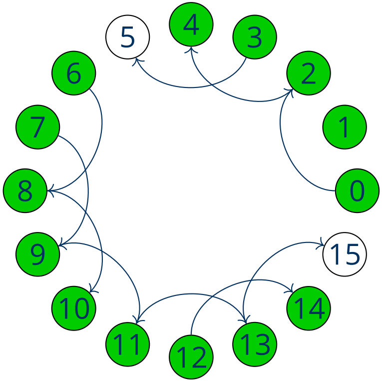
# Corrected Gossip-based Broadcast[3]




- Keep processes in a ring
- Gossip phase
- Correction phase
  - Send along the ring

[3] Hoefler et al., "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems."

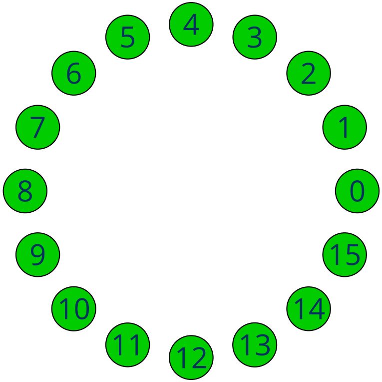
# Corrected Gossip-based Broadcast[3]




- Keep processes in a ring
- Gossip phase
- Correction phase
  - Send along the ring

[3] Hoefler et al., “Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems.”

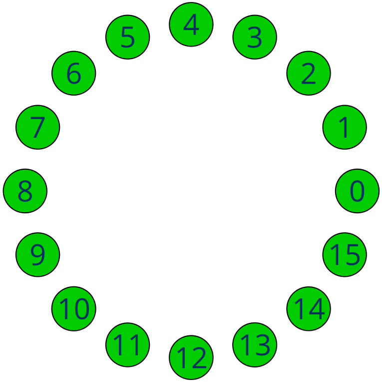
# Corrected Gossip-based Broadcast[3]




- Keep processes in a ring
- Gossip phase
- Correction phase
  - Send along the ring
- Better coloring guarantee

[3] Hoefler et al., “Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems.”

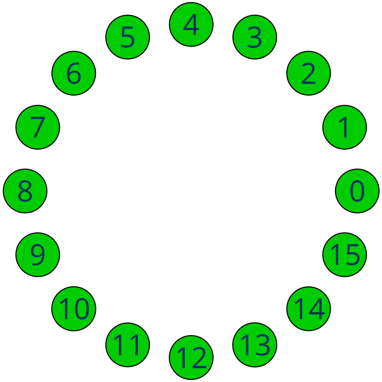
# Corrected Gossip-based Broadcast[3]




- Keep processes in a ring
- Gossip phase
- Correction phase
  - Send along the ring
- Better coloring guarantee
- Inherently fault tolerant

[3] Hoefler et al., "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems."

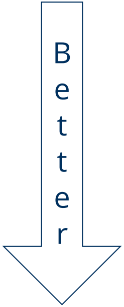
# Corrected Gossip-based Broadcast[3]

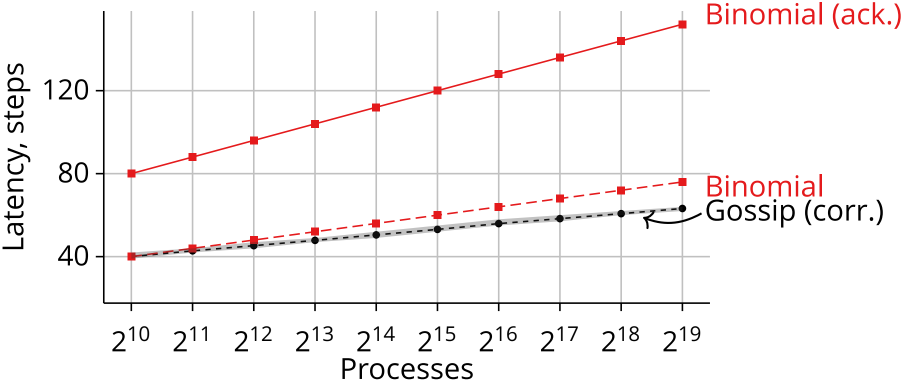



- Keep processes in a ring
- Gossip phase
- Correction phase
  - Send along the ring
- Better coloring guarantee
- Inherently fault tolerant
- Different correction types

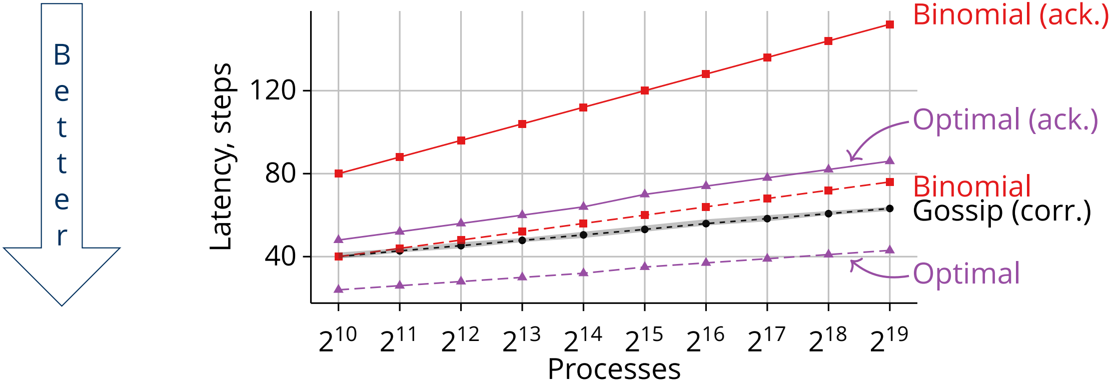
[3] Hoefler et al., “Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems.”

# Simulated Latency: Corrected Gossip

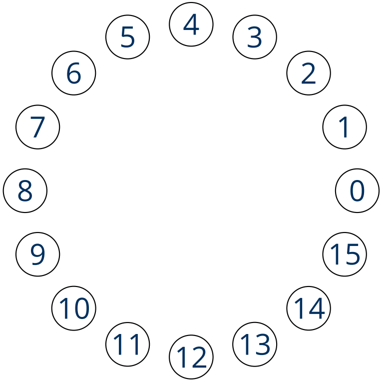


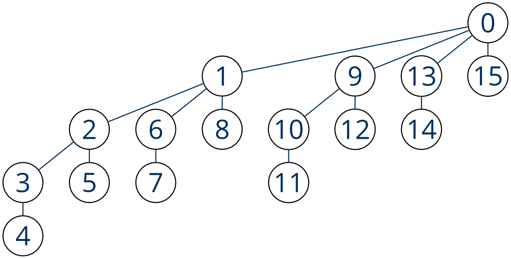



# Simulated Latency: Corrected Gossip

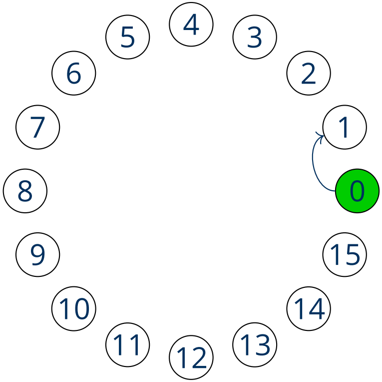


# Correcting a Broadcast Tree

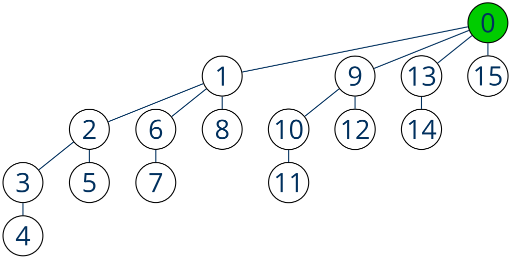


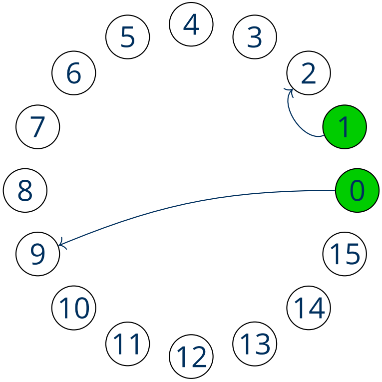
# Correcting a Broadcast Tree



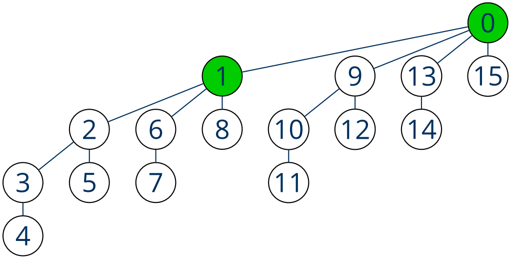

- Tree phase



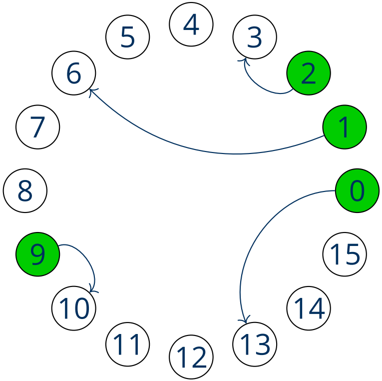
# Correcting a Broadcast Tree



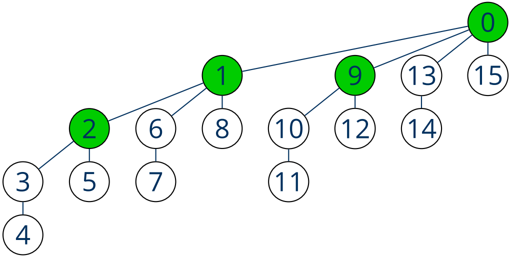

- Tree phase



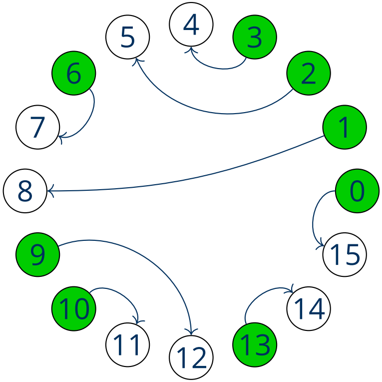
# Correcting a Broadcast Tree



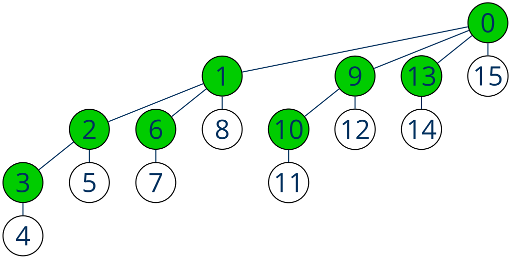

- Tree phase
  - Binomial tree



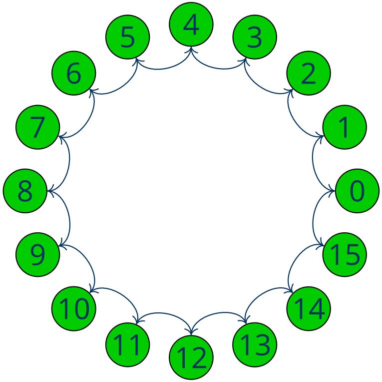
# Correcting a Broadcast Tree



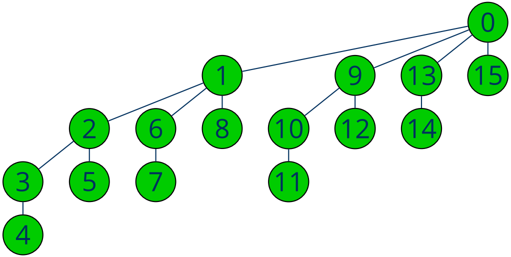

- Tree phase
  - Binomial tree



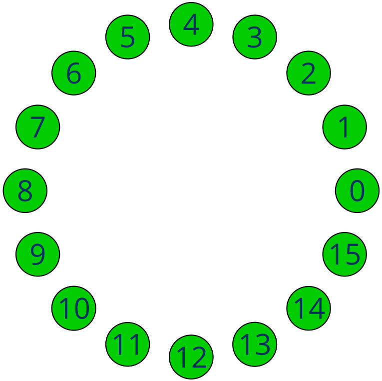
# Correcting a Broadcast Tree



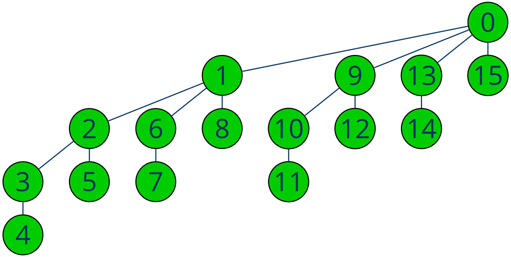

- Tree phase
  - Binomial tree
- Correction phase



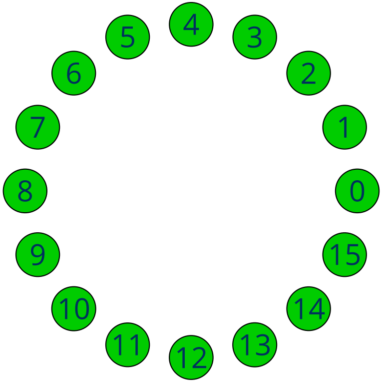
# Correcting a Broadcast Tree



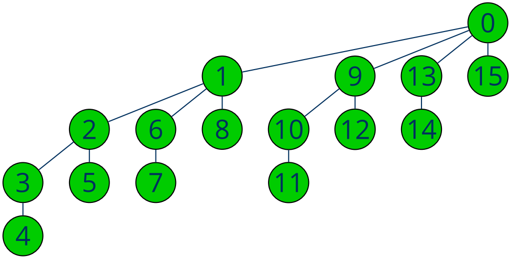

- Tree phase
  - Binomial tree
- Correction phase



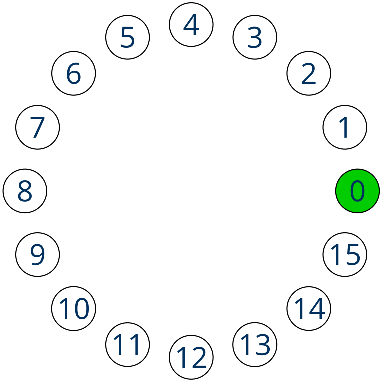
# Correcting a Broadcast Tree

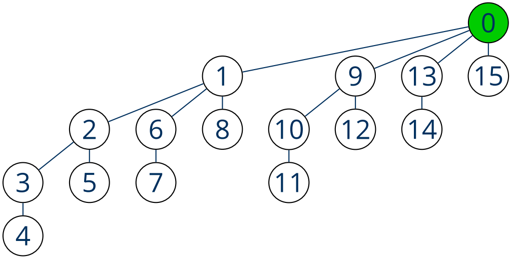



- Tree phase
  - Binomial tree
- Correction phase
- Guaranteed coloring

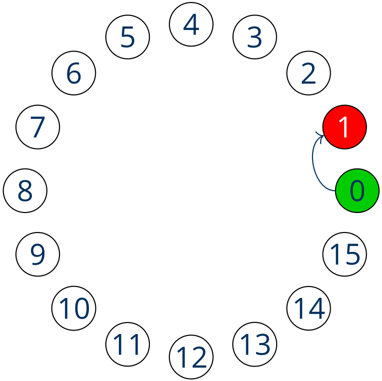


# Correcting a Broadcast Tree with a Failure

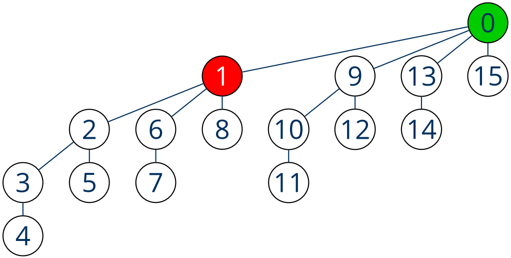


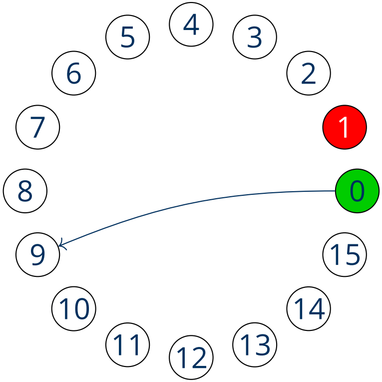
# Correcting a Broadcast Tree with a Failure



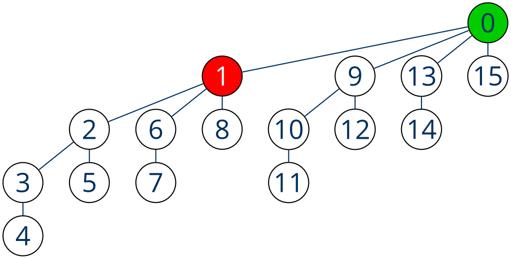

- Non-leaf failure



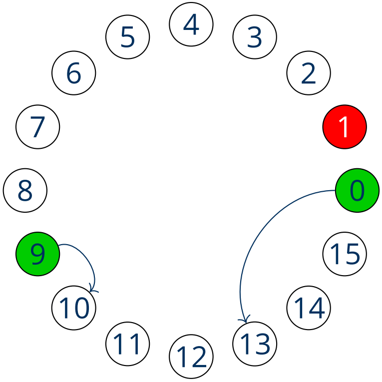
# Correcting a Broadcast Tree with a Failure



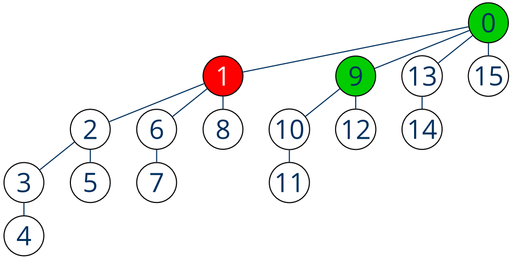

- Non-leaf failure



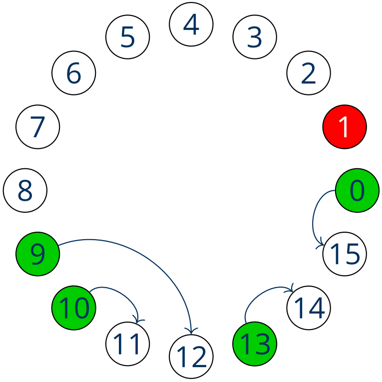
# Correcting a Broadcast Tree with a Failure



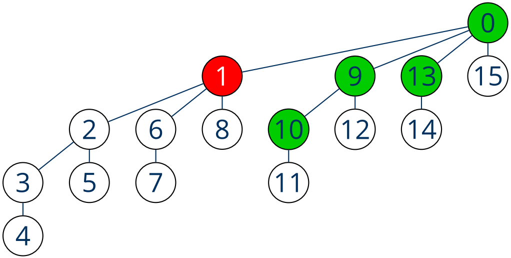

- Non-leaf failure



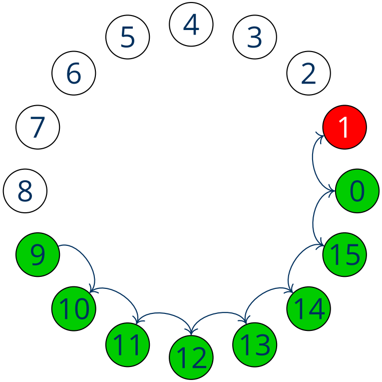
# Correcting a Broadcast Tree with a Failure



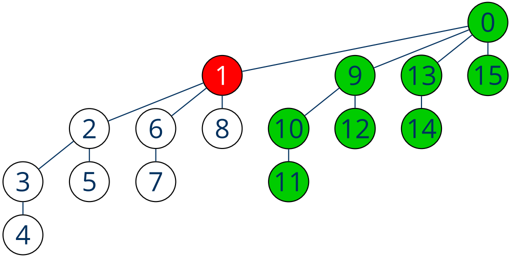

- Non-leaf failure
- Partial coloring



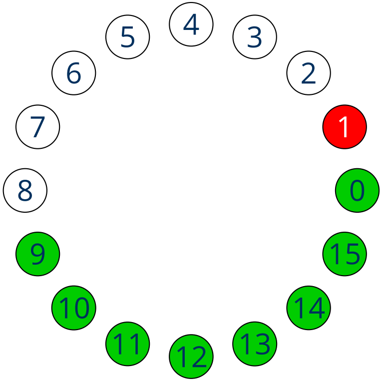
# Correcting a Broadcast Tree with a Failure



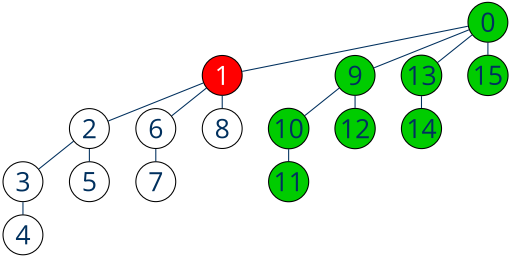

- Non-leaf failure
- Partial coloring
- Correction is inefficient



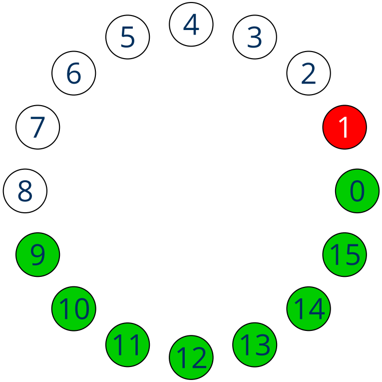
# Correcting a Broadcast Tree with a Failure



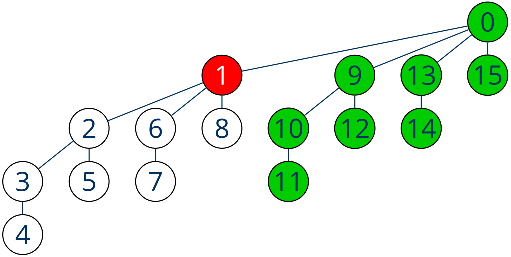

- Non-leaf failure
- Partial coloring
- Correction is inefficient



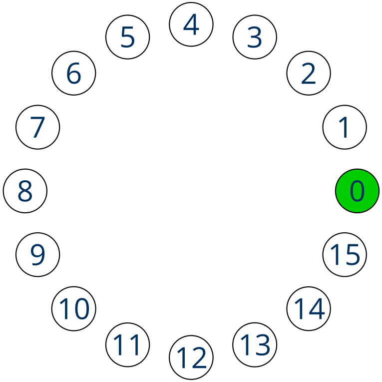
# Correcting a Broadcast Tree with a Failure



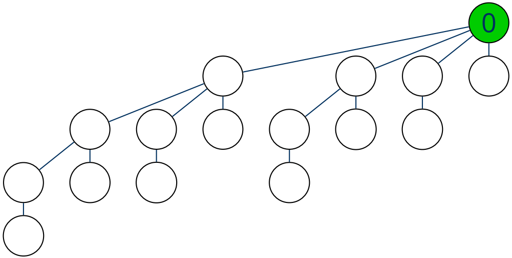

- Non-leaf failure
- Partial coloring
- Correction is inefficient
- Live processes stay uncolored



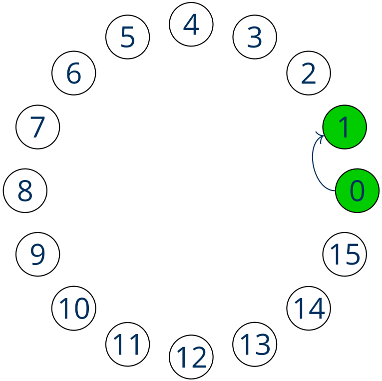
# Corrected Tree-based Broadcast



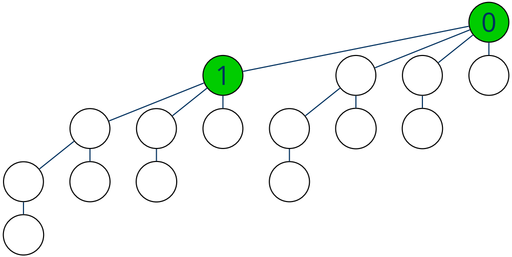

1. Tree phase



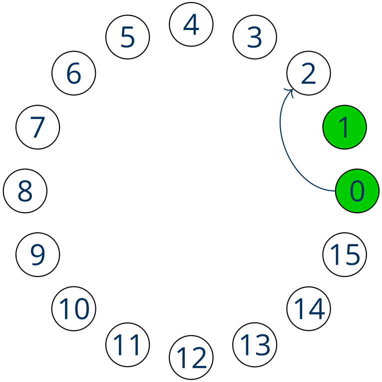
# Corrected Tree-based Broadcast



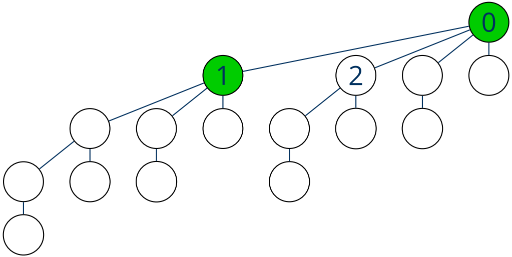

1. Tree phase



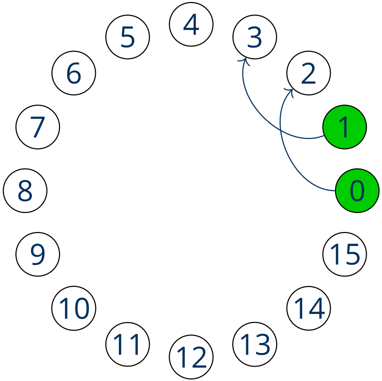
# Corrected Tree-based Broadcast



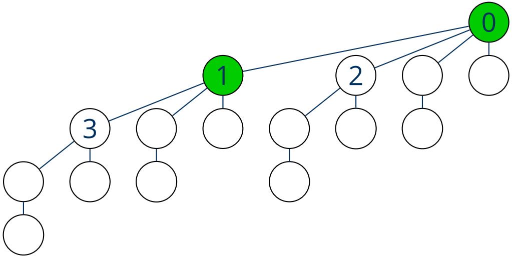

1. Tree phase
   – Interleave parents and children



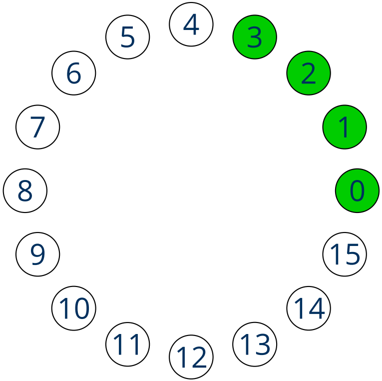
# Corrected Tree-based Broadcast



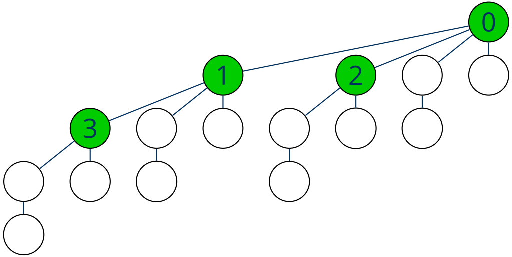

1. Tree phase
   – Interleave parents and children



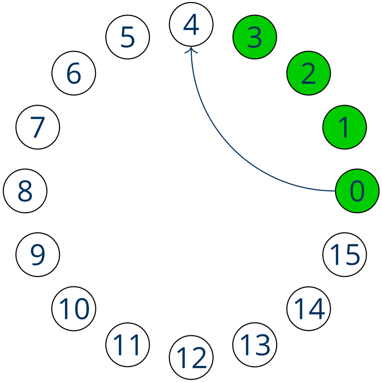
# Corrected Tree-based Broadcast



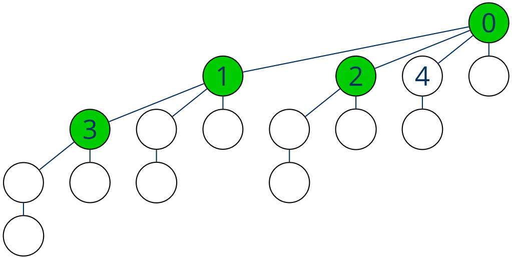

1. Tree phase
   - Interleave parents and children



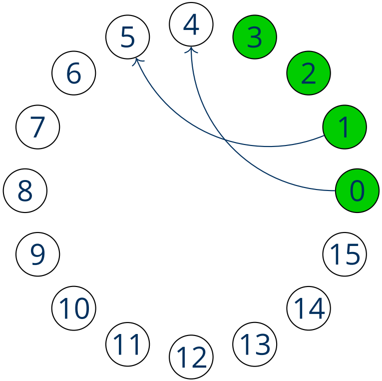
# Corrected Tree-based Broadcast



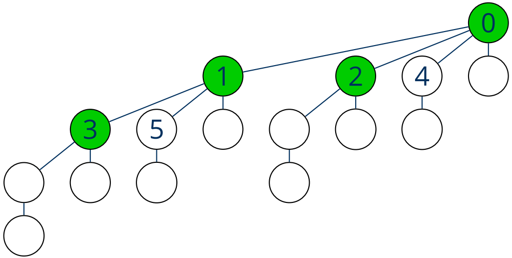

1. Tree phase
   - Interleave parents and children
   - Recursively repeat



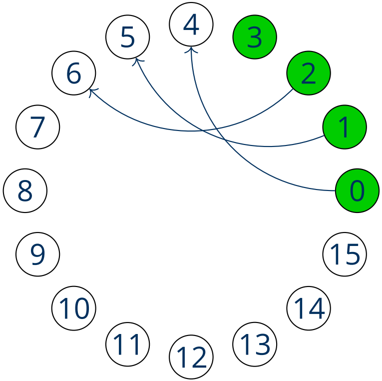
# Corrected Tree-based Broadcast



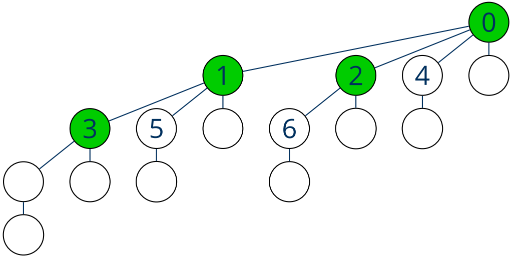

1. Tree phase
   - Interleave parents and children
   - Recursively repeat



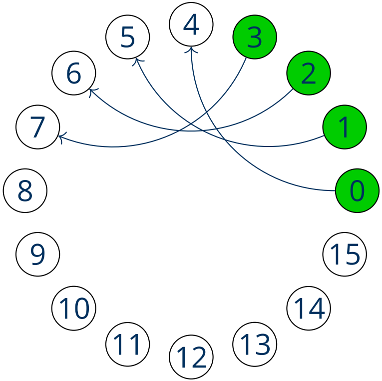
# Corrected Tree-based Broadcast



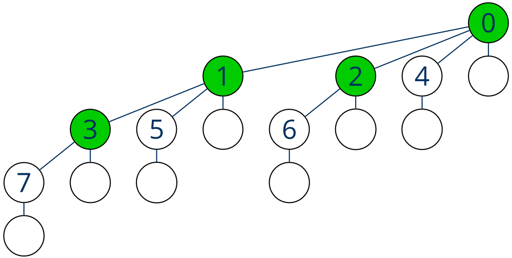

1. Tree phase
   - Interleave parents and children
   - Recursively repeat



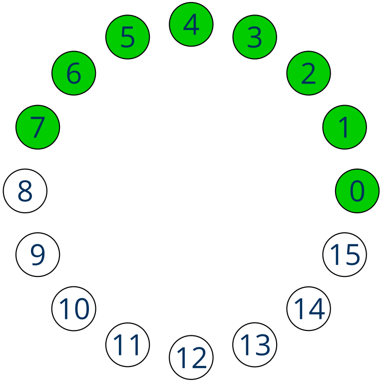
# Corrected Tree-based Broadcast



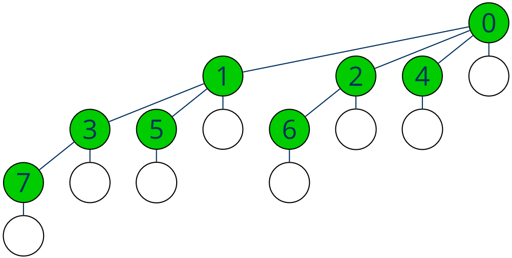

1. Tree phase
   - Interleave parents and children
   - Recursively repeat



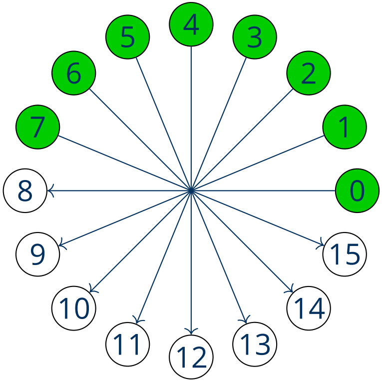
# Corrected Tree-based Broadcast



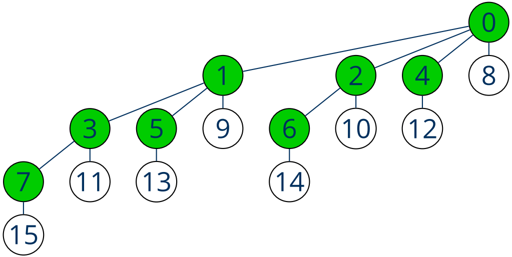

1. Tree phase
   - Interleave parents and children
   - Recursively repeat



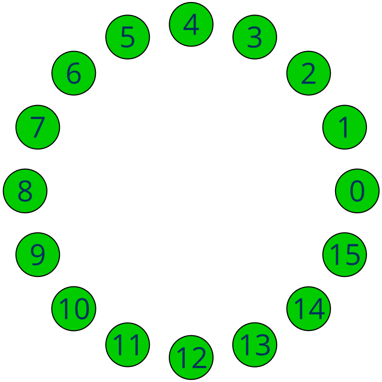
# Corrected Tree-based Broadcast



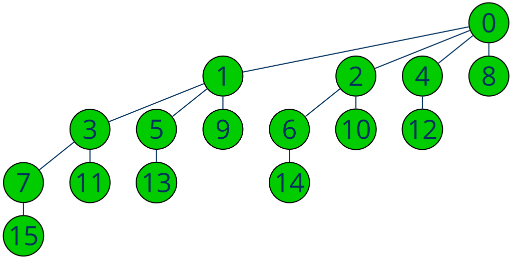

1. Tree phase
   - Interleave parents and children
   - Recursively repeat



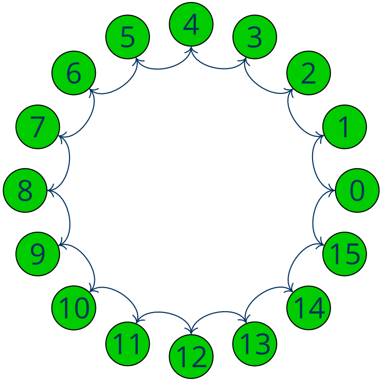
# Corrected Tree-based Broadcast



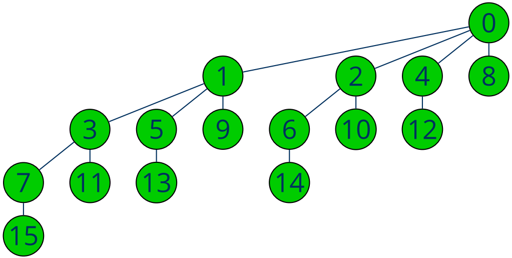

1. Tree phase
   - Interleave parents and children
   - Recursively repeat



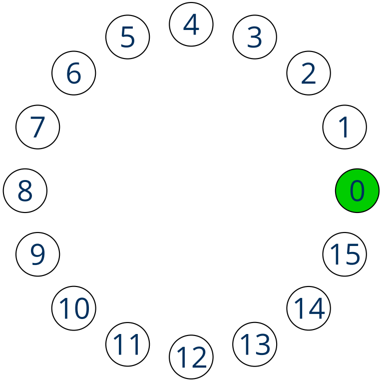
# Corrected Tree-based Broadcast

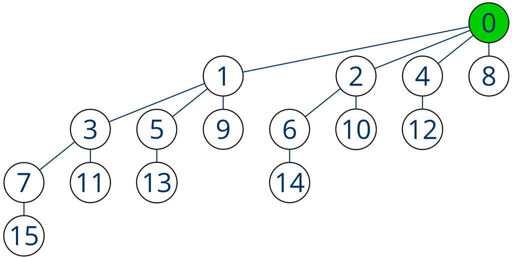



1. Tree phase
   - Interleave parents and children
   - Recursively repeat
2. Correction phase

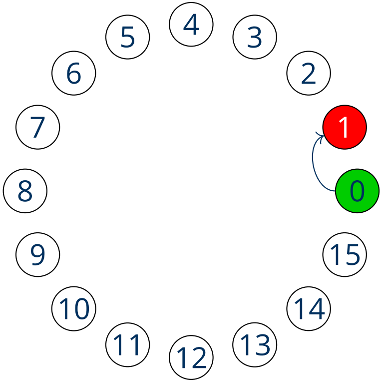


# Corrected Tree-based Broadcast with Failure

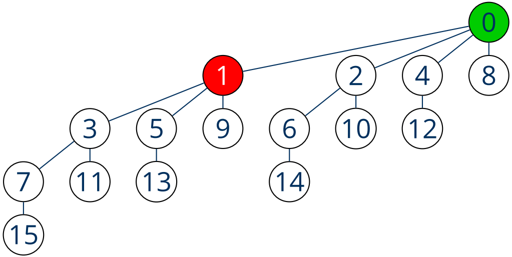


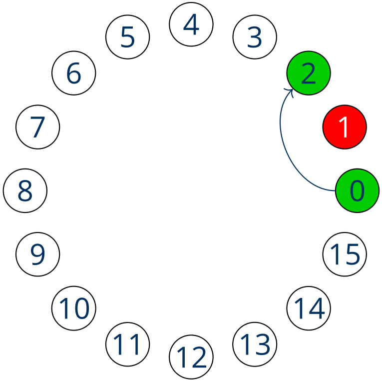
# Corrected Tree-based Broadcast with Failure



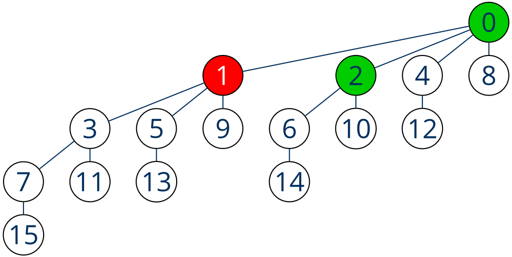

- Non-leaf failure



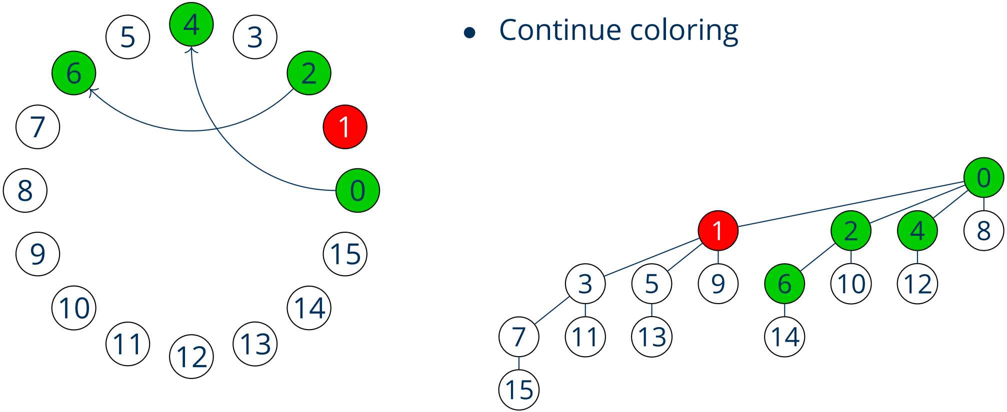
# Corrected Tree-based Broadcast with Failure




- Non-leaf failure
- Continue coloring

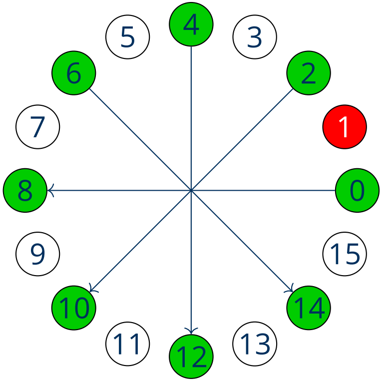


# Corrected Tree-based Broadcast with Failure

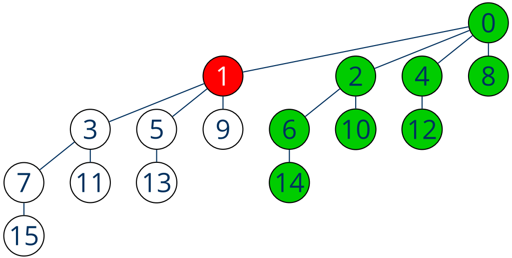



- Non-leaf failure
- Continue coloring

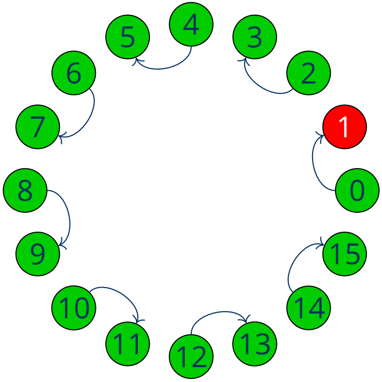
# Corrected Tree-based Broadcast with Failure



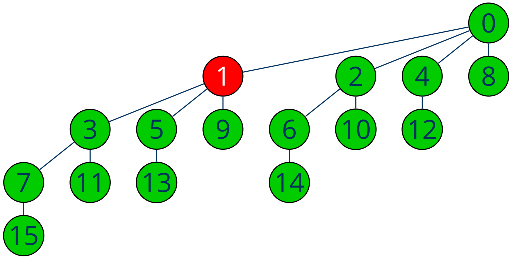

- Non-leaf failure
- Continue coloring
- Small holes



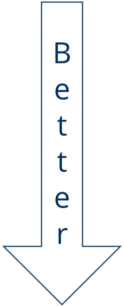
# Corrected Tree-based Broadcast with Failure

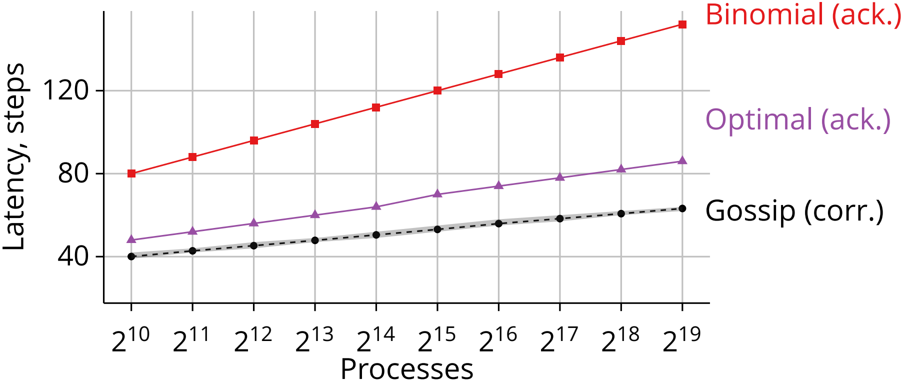



- Non-leaf failure
- Continue coloring
- Small holes
- Correction phase

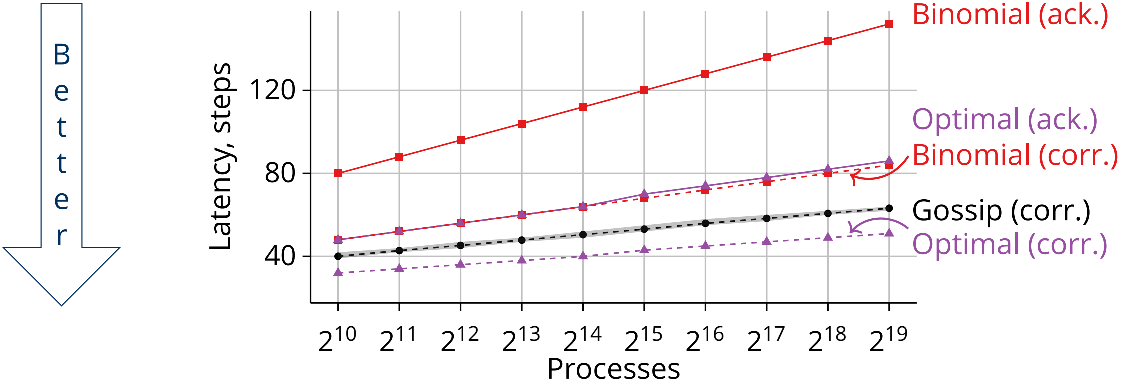


# Simulated Latency: Corrected Trees



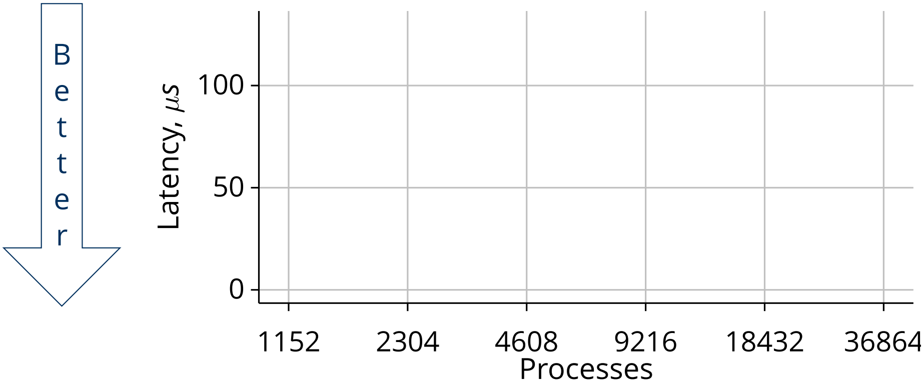
# Simulated Latency: Corrected Trees



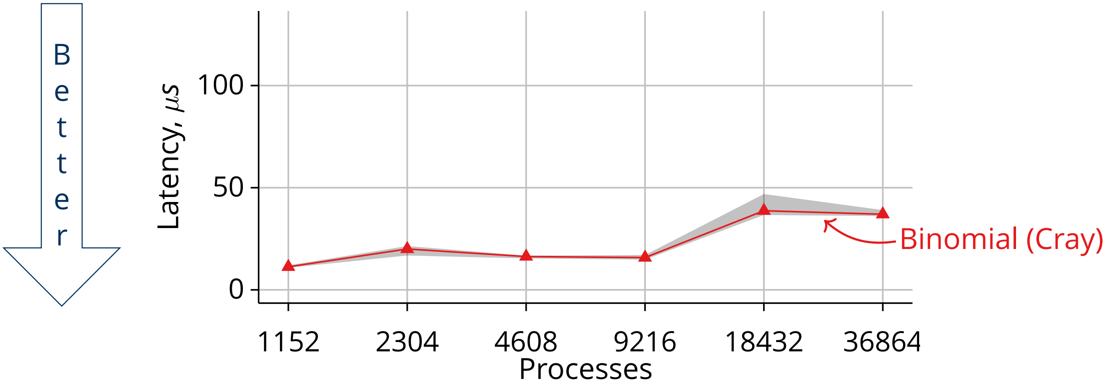
# Evaluation

- OpenMPI-based
- $512 \times 72$-core nodes $= 36\,864$ processes
- Piz Daint (Cray XC 40)

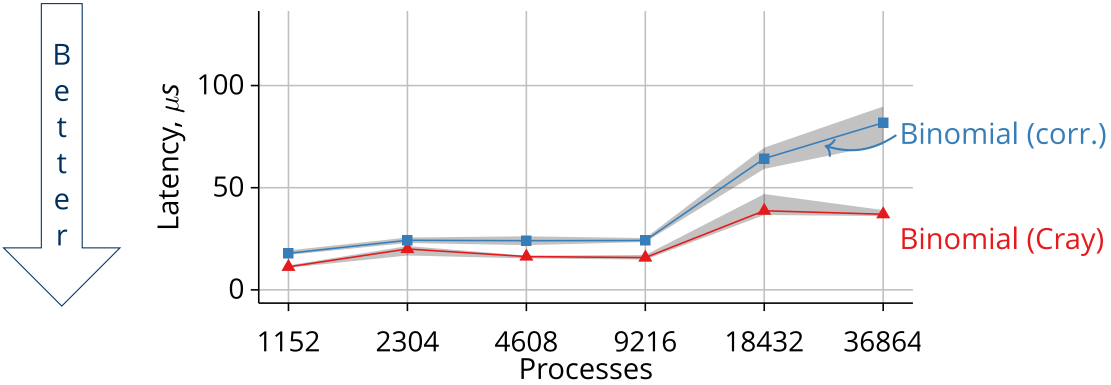
# MPI-based Implementation



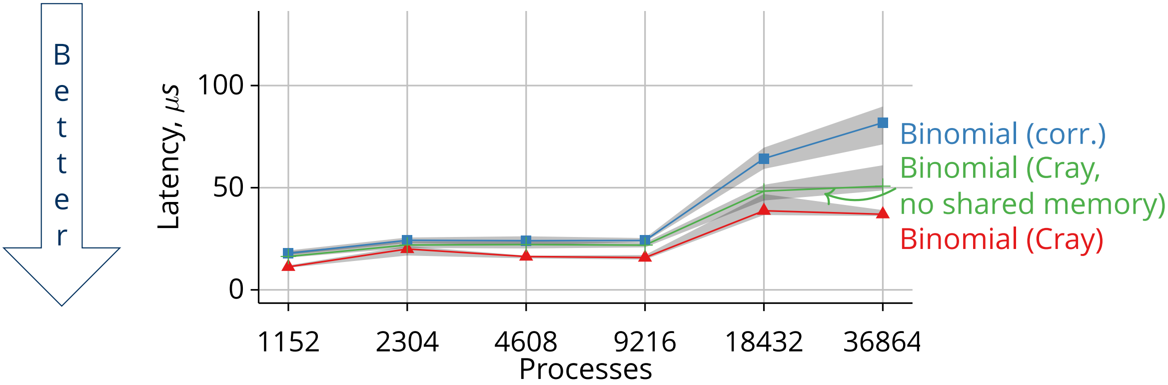
# MPI-based Implementation



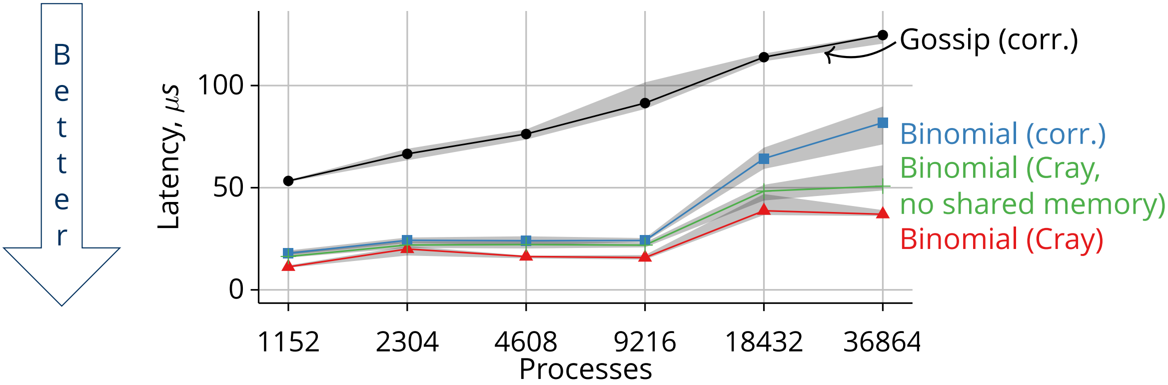
# MPI-based Implementation



# MPI-based Implementation



# MPI-based Implementation



# Conclusion

- Corrected Trees
  - Interleaved Trees
  - Correction Phase

# Conclusion

- Corrected Trees
  - Interleaved Trees
  - Correction Phase
- Properties
  - Reliable broadcast
  - Faster than acknowledged tree
  - Less messages than Corrected Gossip

# Conclusion

- Corrected Trees
  - Interleaved Trees
  - Correction Phase
- Properties
  - Reliable broadcast
  - Faster than acknowledged tree
  - Less messages than Corrected Gossip
- Closed-form expression for binomial
- Simple construction for other practical tree types
- Future work: Other collectives

# Backup Slides

# Correction Types[4]

1. Opportunistic correction
   - Fixed rounds
   - Probabilistic reachability

[4] Hoefler et al., “Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems.”

# Correction Types[4]

1. Opportunistic correction
   - Fixed rounds
   - Probabilistic reachability
2. Checked correction
   - Variable rounds
   - Guaranteed reachability

[4] Hoefler et al., "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems."

# Correction Types[4]

1. Opportunistic correction
   - Fixed rounds
   - Probabilistic reachability
2. Checked correction
   - Variable rounds
   - Guaranteed reachability
3. Failure-proof correction
   - Extended checked correction
   - Tolerates online failures

[4] Hoefler et al., “Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems.”

# Correction Types[4]

1. Opportunistic correction
   - Fixed rounds
   - Probabilistic reachability
2. Checked correction
   - Variable rounds
   - Guaranteed reachability
3. Failure-proof correction
   - Extended checked correction
   - Tolerates online failures

[4] Hoefler et al., "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems."

# Correction Types[4]

1. Opportunistic correction
   - Fixed rounds
   - Probabilistic reachability
2. Checked correction
   - Variable rounds
   - Guaranteed reachability
3. Failure-proof correction
   - Extended checked correction
   - Tolerates online failures

Limitations:

- Only broadcast for now
- Small messages

[4] Hoefler et al., "Corrected Gossip Algorithms for Fast Reliable Broadcast on Unreliable Systems."

# Different Corrected Trees

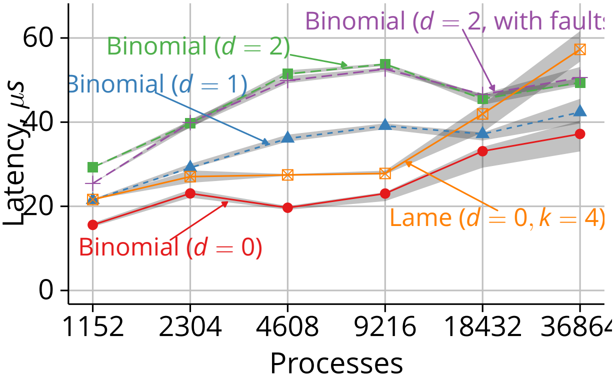
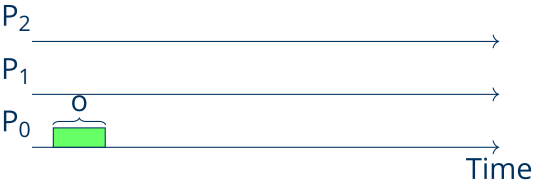


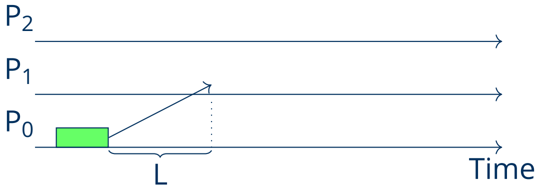

Figure: Broadcast median latency (system X)

# Latency Model



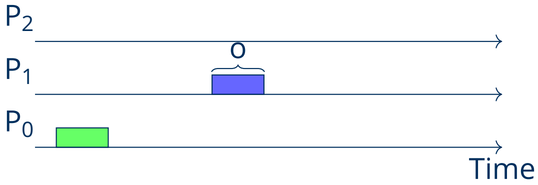

- LogP-like model
  - Send overhead

# Latency Model



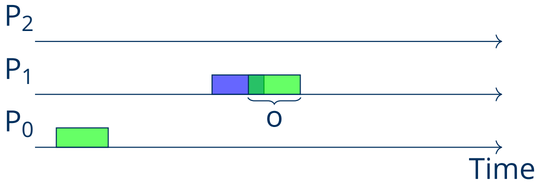

- LogP-like model
  - Send overhead
  - p2p-Latency

# Latency Model



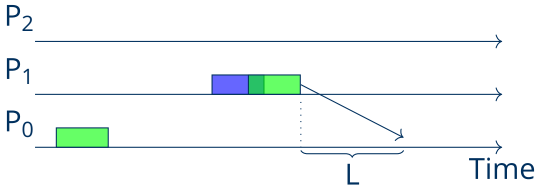

- LogP-like model
  - Send overhead
  - p2p-Latency
  - Receive overhead

# Latency Model



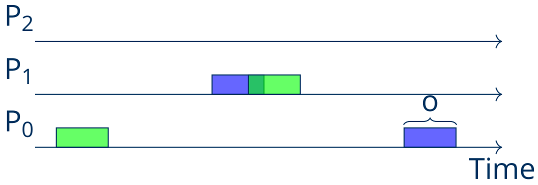

- LogP-like model
  - Send overhead
  - p2p-Latency
  - Receive overhead
  - Parallel send and receive

# Latency Model



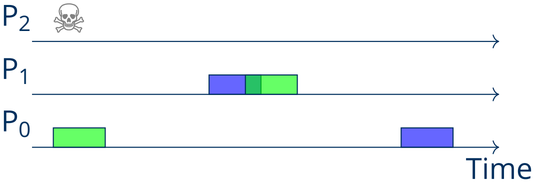

- LogP-like model
  - Send overhead
  - p2p-Latency
  - Receive overhead
  - Parallel send and receive

# Latency Model



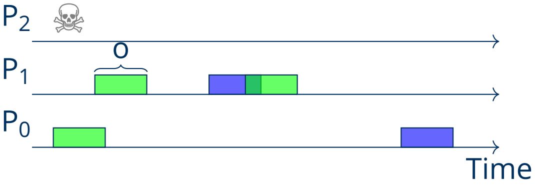

- LogP-like model
  - Send overhead
  - p2p-Latency
  - Receive overhead
  - Parallel send and receive

# Latency Model



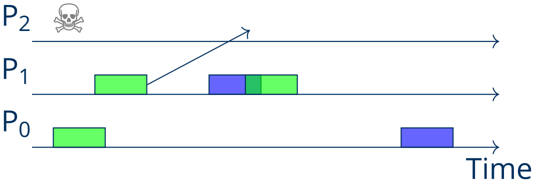

- LogP-like model
  - Send overhead
  - p2p-Latency
  - Receive overhead
  - Parallel send and receive
- Fault model
  - Failed processes

# Latency Model



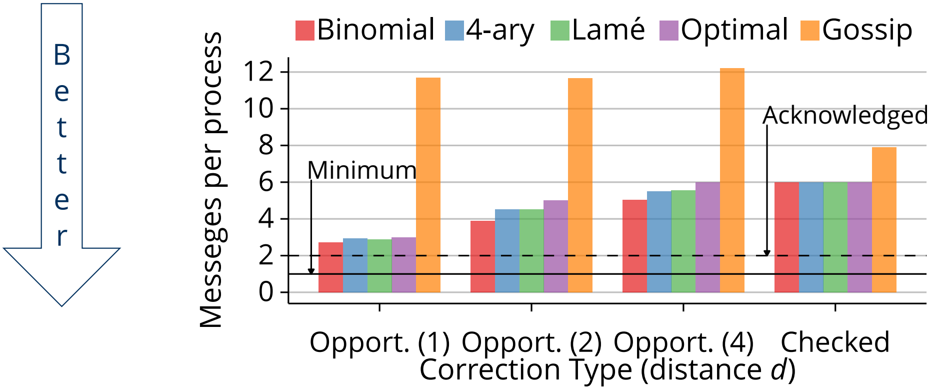

- LogP-like model
  - Send overhead
  - p2p-Latency
  - Receive overhead
  - Parallel send and receive
- Fault model
  - Failed processes

# Latency Model

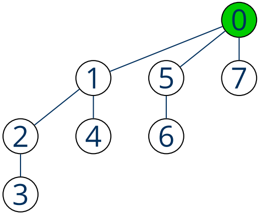



- LogP-like model
  - Send overhead
  - p2p-Latency
  - Receive overhead
  - Parallel send and receive
- Fault model
  - Failed processes
  - Failures not detected

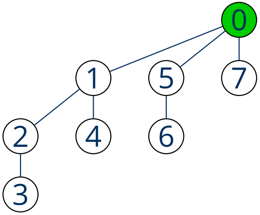
# Simulation: Message Count



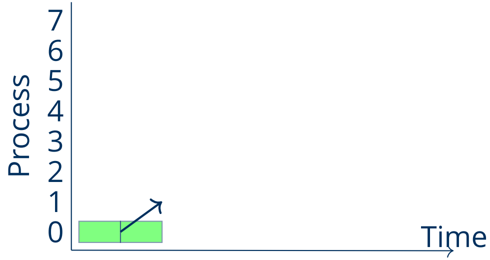
# Broadcast Timeline

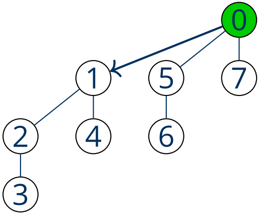



# Broadcast Timeline

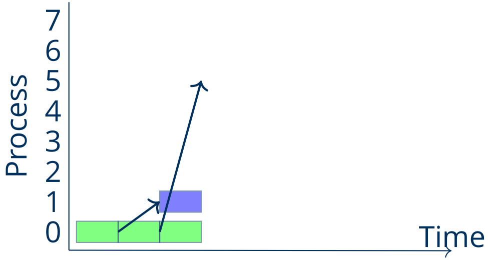



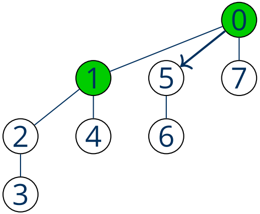
# Broadcast Timeline



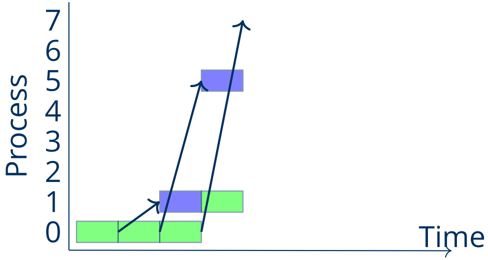



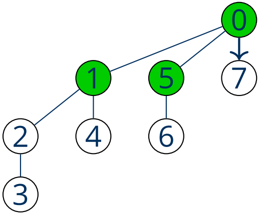
# Broadcast Timeline



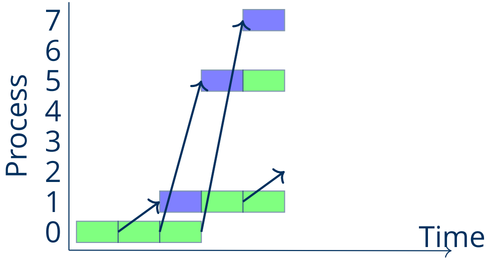



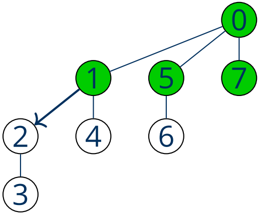
# Broadcast Timeline



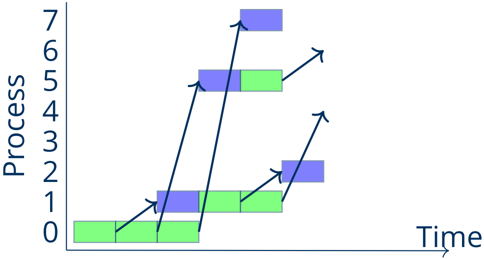



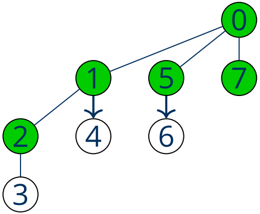
# Broadcast Timeline



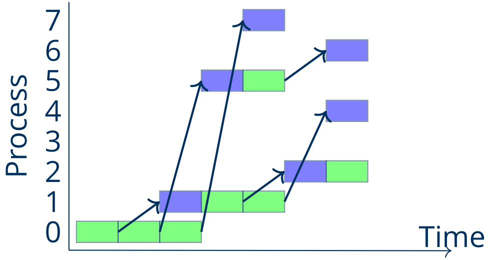



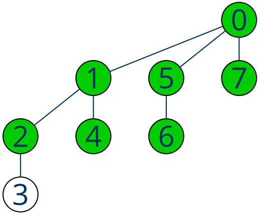
# Broadcast Timeline



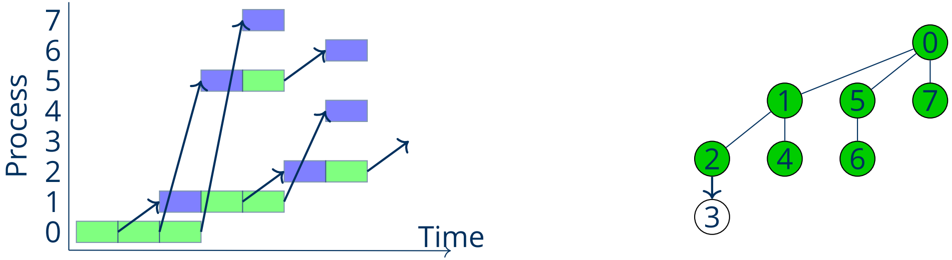



# Broadcast Timeline

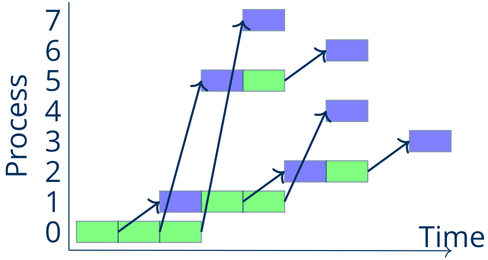


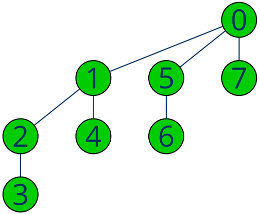



# Broadcast Timeline



# Broadcast Timeline



# Simulation: Latency Sturdiness

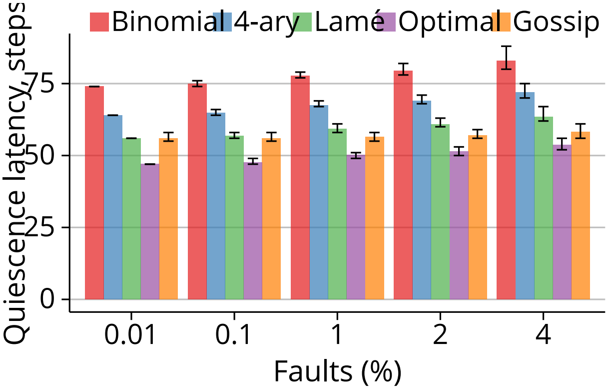



Figure: Average quiescence latency grows with fault rate

# Simulation: Message Count Sturdiness

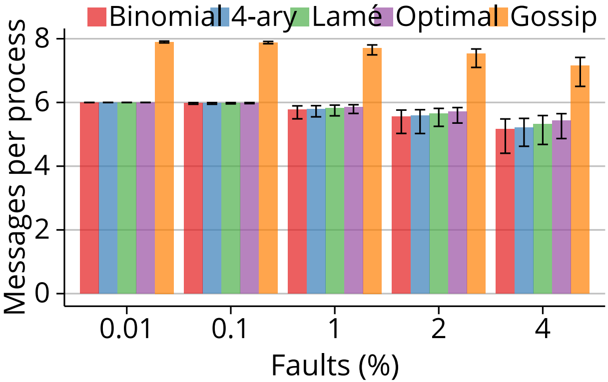



Figure: Average number of messages goes down with higher fault rate

# Component Failures in HPC Systems[5]

| Component | # of Nodes Affected | MTBF |
|---|---|---|
| PFS, core switch | 1,408 | 65.10 days |
| Rack | 32 | 86.90 days |
| Edge switch | 16 | 17.37 days |
| PSU | 4 | 28.94 days |
| Compute node | 1 | 15.8 hours |

Data gathered between 2010-11-01 and 2012-04-06 on TSUBAME 2.0

[5] Sato et al., “Design and Modeling of a Non-blocking Checkpointing System.”