UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

FlipBack: Automatic Target Protection Against Soft Errors

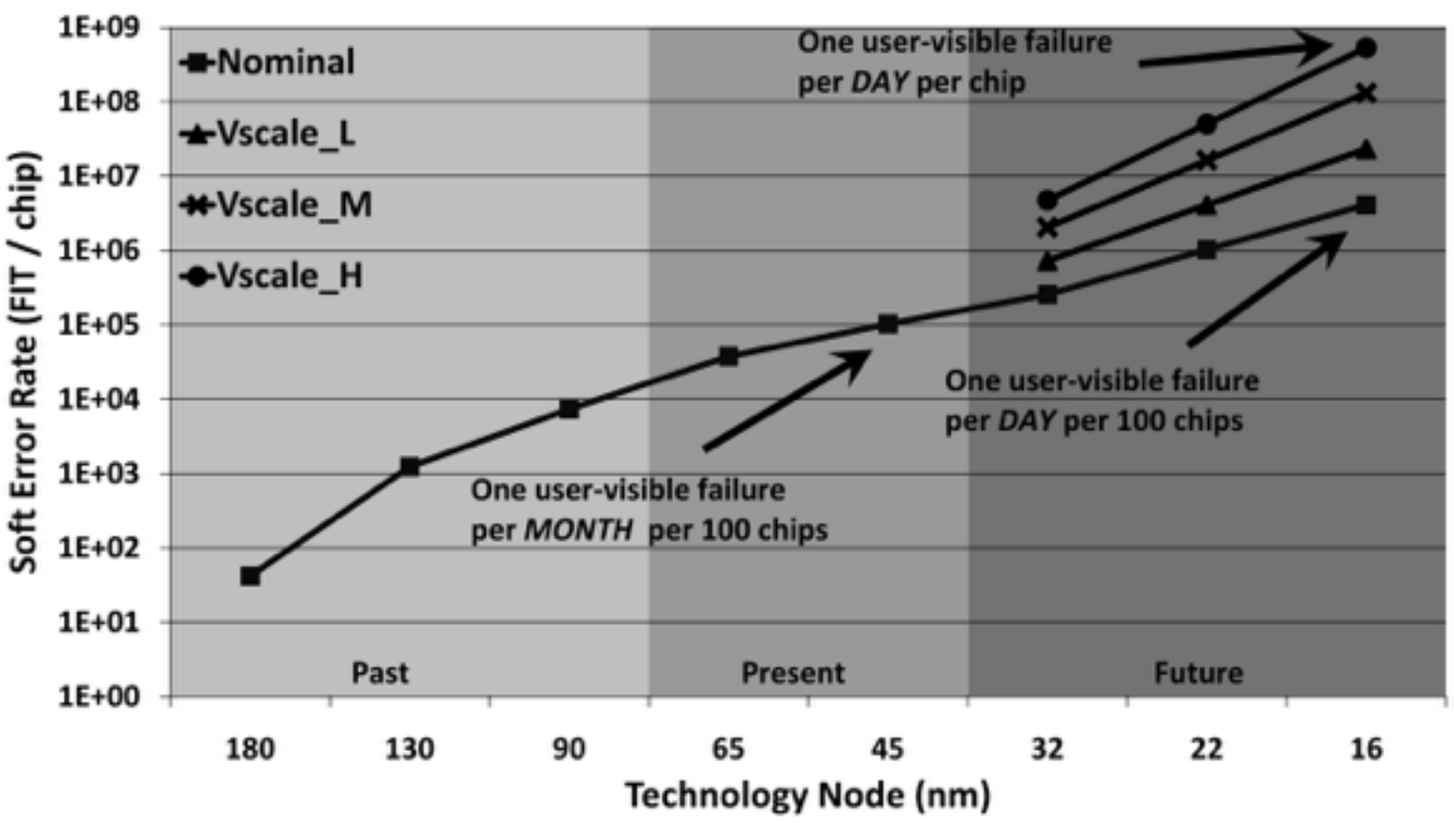
Xiang Ni

Parallel Programming Lab

illinois.edu

Soft Errors

- Common source of soft errors
  - Electrical noise
  - External radiation
  - Manufacturing fault
- Data corruption: we may or may not know

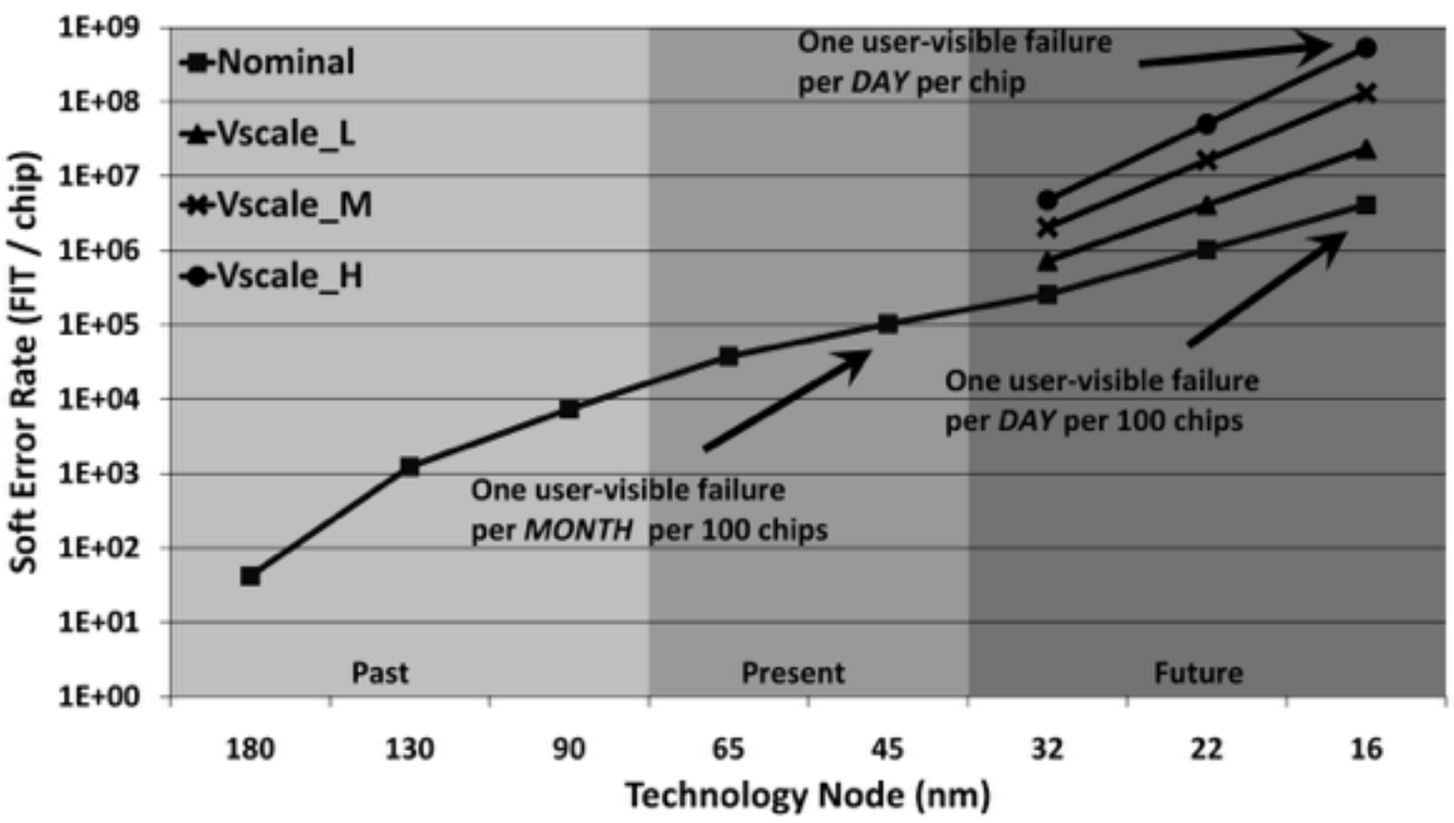


Shrinking chip size

- More energy efficient
- Higher soft error rate

Soft Errors

- Common source of soft errors
  - Electrical noise
  - External radiation
  - Manufacturing fault
- Data corruption: we may or may not know

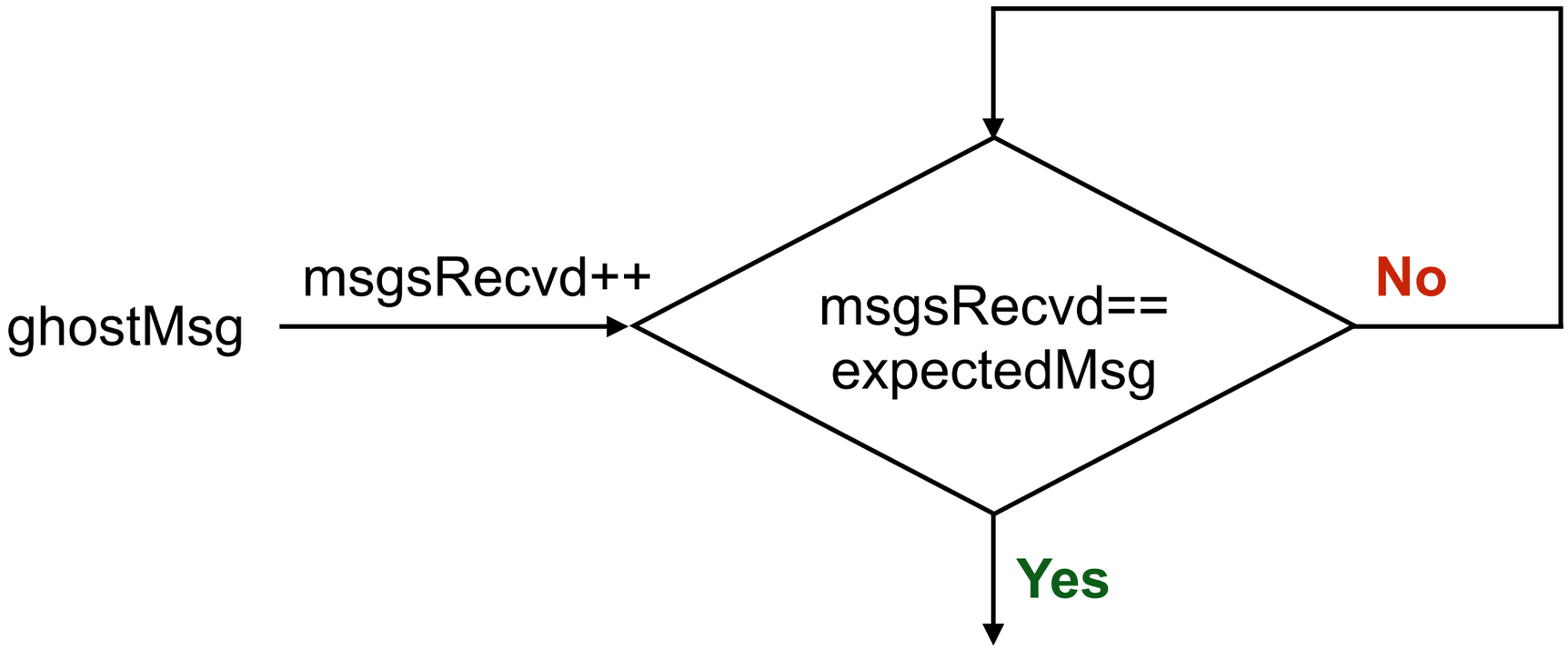


Shrinking chip size

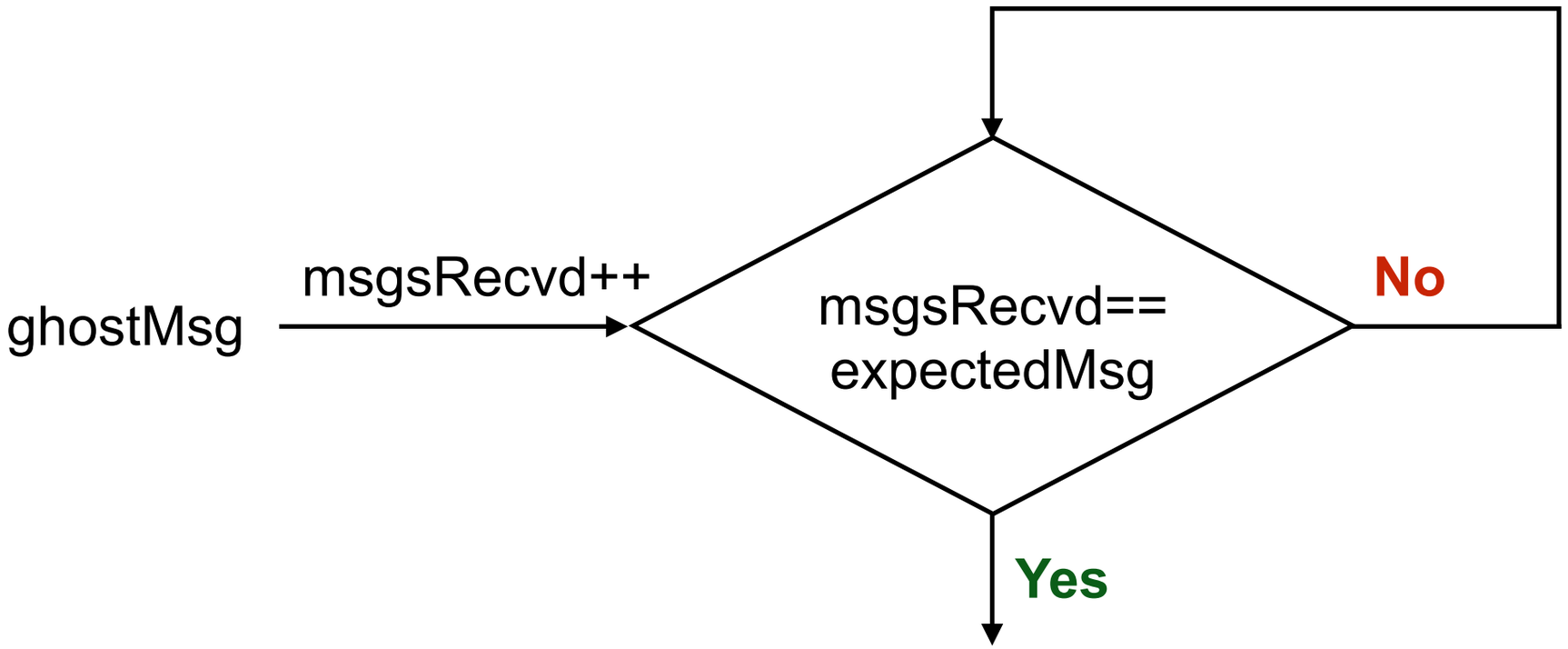
- More energy efficient
- Higher soft error rate

Less energy required to corrupt data

Motivation Example



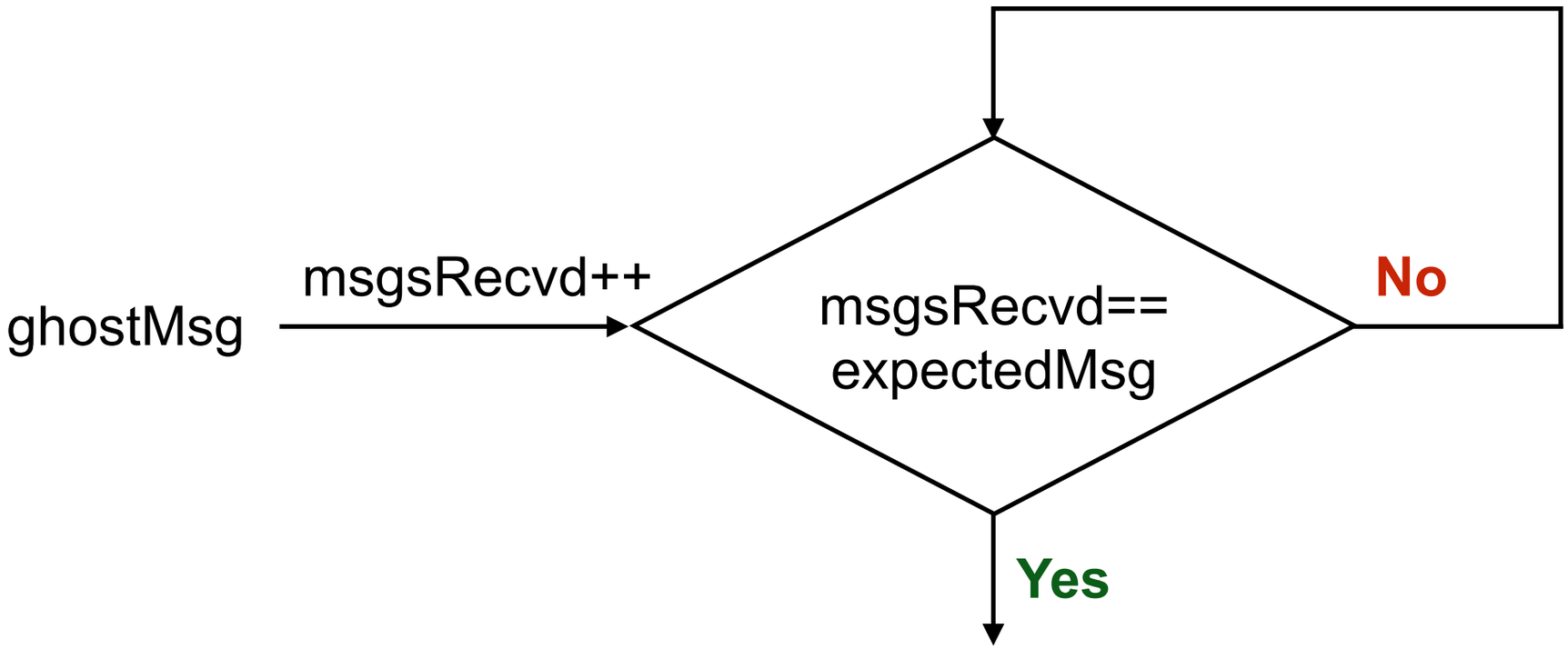
Motivation Example



expectedMsg

00000111

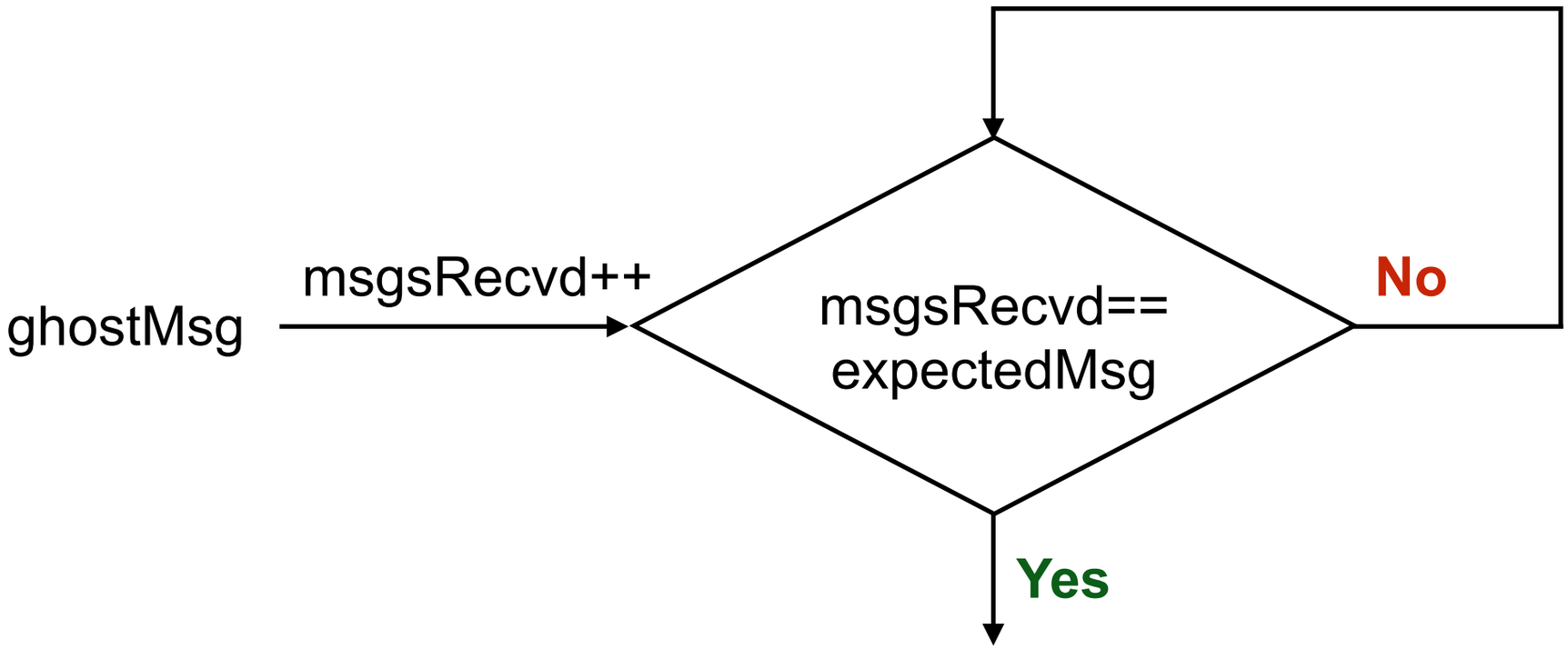
Motivation Example



expectedMsg

00001111

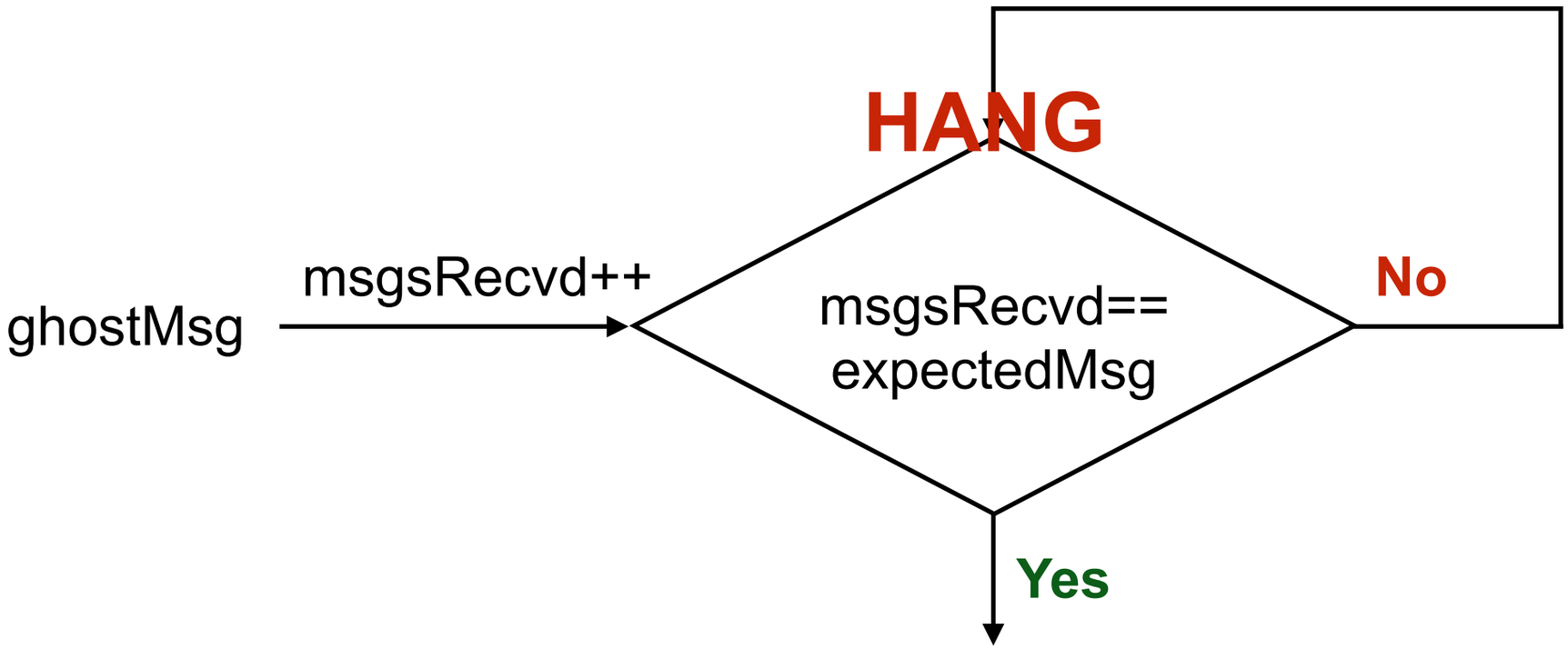
Motivation Example



expectedMsg 7 —> 15

00001111

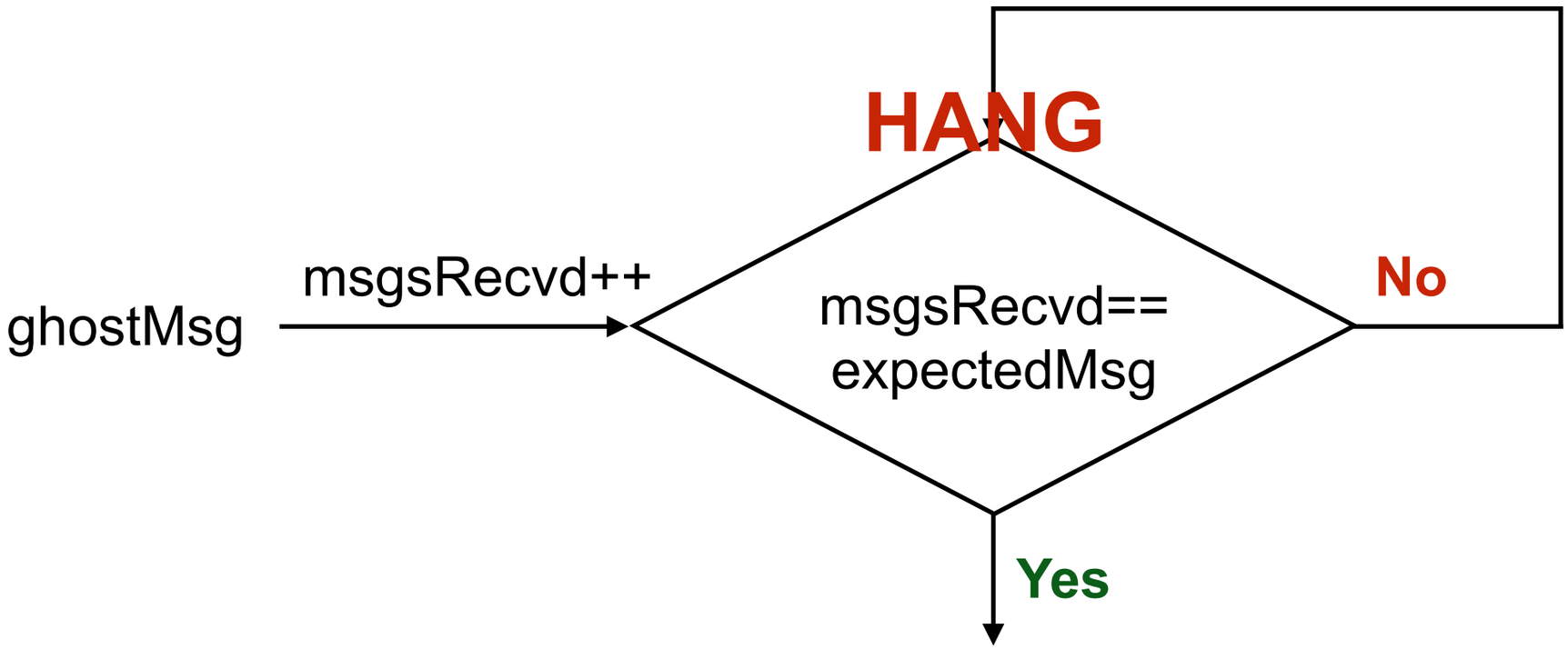
Motivation Example



expectedMsg 7 —> 15

00001111

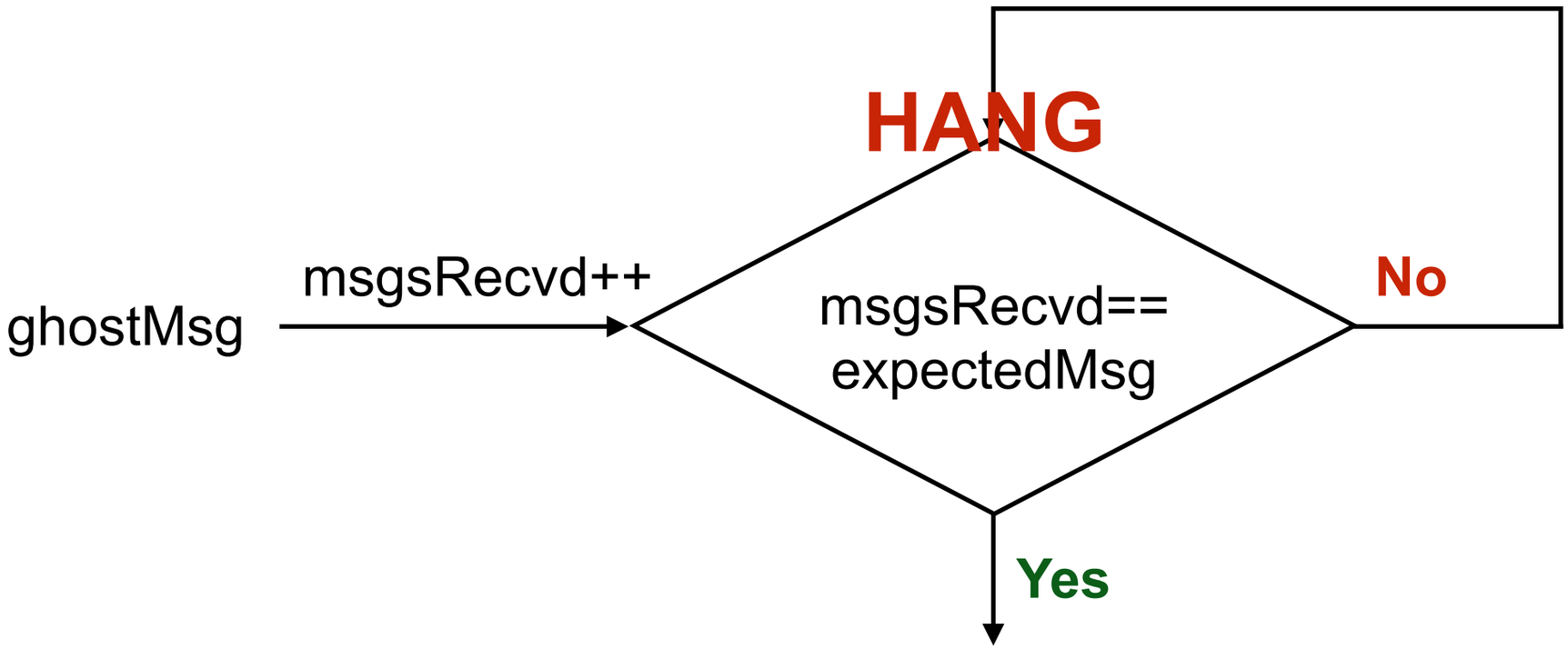
Motivation Example



expectedMsg 7 —> 15

00000011

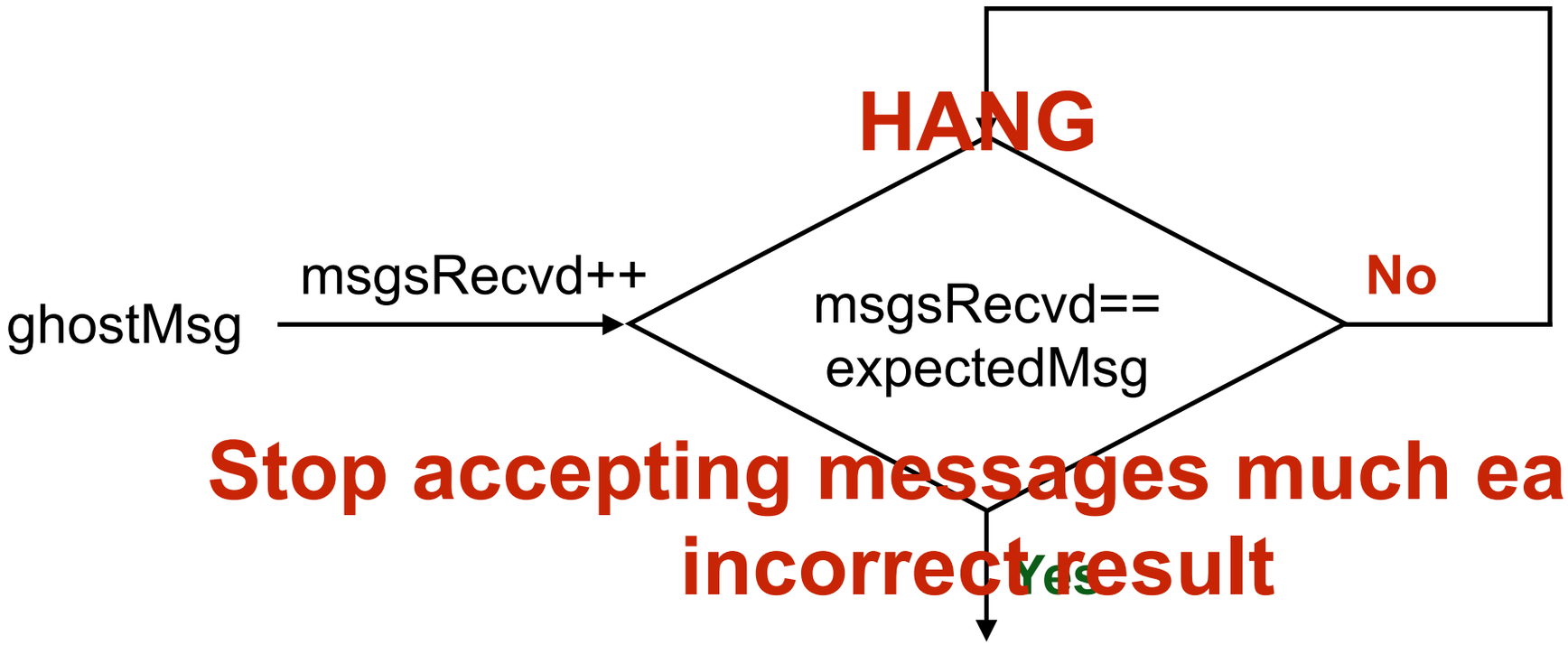
Motivation Example



expectedMsg 7 —> 3

00000011

Motivation Example



Stop accepting messages much earlier: incorrect result

expectedMsg 7 —> 3

00000011

Runtime Guided Replication

Runtime Guided Replication

- Control Variables
  - msgsRecvd, expectedMsg
  - Affecting program flow

Runtime Guided Replication

- Control Variables
  - msgsRecvd, expectedMsg
  - Affecting program flow
- How do we ensure the program control flow is correct?
- Fully duplication is expensive: less than 50% resource utilization or at least twice the running time

Runtime Guided Replication

- Control Variables
  - msgsRecvd, expectedMsg
  - Affecting program flow
- How do we ensure the program control flow is correct?
  - Fully duplication is expensive: less than 50% resource utilization or at least twice the running time
- What about only duplicating the computation that affects program flow?
  - Leverage a compiler slicing pass
  - Reduce computation time
  - Avoid doubling the memory

Compiler Slicing Pass

```
Input:
f: the targeted function to perform slicing on
c: set of control variables
Output: slices: the program slice for recomputation
// search for slicing criterions
1  foreach Instruction I in f do
2      if Defs(I) ⊂ c or I sends messages then
3          criterions.push(I);
4      end
5  end
6  while !criterions.empty() do
7      I←criterions.top(); criterions.pop();
8      if !I.processed() then
9          slices.push(I);
           // data flow analysis
10         foreach Values I′ in Uses(I) do
11             foreach Instruction I″ in Defs(I′) do
12                 if I″ may lead to I then
13                     criterions.push(I″);
14                 end
15             end
16         end
           // control flow analysis
17         foreach BasicBlock B that may lead to I do
18             criterions.push(B.getTerminator());
19         end
20     end
21 end
```

Compiler Slicing Pass

```
   Input:
   f: the targeted function to perform slicing on
   c: set of control variables
   Output: slices: the program slice for recomputation
   // search for slicing criterions
1  foreach Instruction I in f do
2  |  if Defs(I) ⊂ c or I sends messages then
3  |  |  criterions.push(I);
4  |  end
5  end
6  while !criterions.empty() do
7  |  I←criterions.top(); criterions.pop();
8  |  if !I.processed() then
9  |  |  slices.push(I);
   |  |  // data flow analysis
10 |  |  foreach Values I′ in Uses(I) do
11 |  |  |  foreach Instruction I″ in Defs(I′) do
12 |  |  |  |  if I″ may lead to I then
13 |  |  |  |  |  criterions.push(I″);
14 |  |  |  |  end
15 |  |  |  end
16 |  |  end
   |  |  // control flow analysis
17 |  |  foreach BasicBlock B that may lead to I do
18 |  |  |  criterions.push(B.getTerminator());
19 |  |  end
20 |  end
21 end
```

```
void Stencil::beginNextIter() {

   iterCount++;
   if(iterCount >= totalIter){

      mainProxy.done(); //program exits

   }else{

      for(int i = 0; i < totalDirections; i++) {

         ghostMsg * m = createGhostMsg(dirs[i]);
         copy(m->data, boundary[i]);
         int sendTo = myIdx+dirs[i];

         stencilProxy(sendTo).receiveMessage(m);

      }

   }

}
```

Compiler Slicing Pass

```
   Input:
   f: the targeted function to perform slicing on
   c: set of control variables
   Output: slices: the program slice for recomputation
   // search for slicing criterions
1  foreach Instruction I in f do
2  |  if Defs(I) ⊂ c or I sends messages then
3  |  |  criterions.push(I);
4  |  end
5  end
6  while !criterions.empty() do
7  |  I←criterions.top(); criterions.pop();
8  |  if !I.processed() then
9  |  |  slices.push(I);
   |  |  // data flow analysis
10 |  |  foreach Values I′ in Uses(I) do
11 |  |  |  foreach Instruction I″ in Defs(I′) do
12 |  |  |  |  if I″ may lead to I then
13 |  |  |  |  |  criterions.push(I″);
14 |  |  |  |  end
15 |  |  |  end
16 |  |  end
   |  |  // control flow analysis
17 |  |  foreach BasicBlock B that may lead to I do
18 |  |  |  criterions.push(B.getTerminator());
19 |  |  end
20 |  end
21 end
```

```
void Stencil::beginNextIter() {

   iterCount++;
   if(iterCount >= totalIter){

      mainProxy.done(); //program exits

   }else{

      for(int i = 0; i < totalDirections; i++) {

         ghostMsg * m = createGhostMsg(dirs[i]);
         copy(m->data, boundary[i]);
         int sendTo = myIdx+dirs[i];

         stencilProxy(sendTo).receiveMessage(m);

      }

   }

}
```

Compiler Slicing Pass

```
   Input:
   f: the targeted function to perform slicing on
   c: set of control variables
   Output: slices: the program slice for recomputation
   // search for slicing criterions
1  foreach Instruction I in f do
2  |  if Defs(I) ⊂ c or I sends messages then
3  |  |  criterions.push(I);
4  |  end
5  end
6  while !criterions.empty() do
7  |  I←criterions.top(); criterions.pop();
8  |  if !I.processed() then
9  |  |  slices.push(I);
   |  |  // data flow analysis
10 |  |  foreach Values I′ in Uses(I) do
11 |  |  |  foreach Instruction I″ in Defs(I′) do
12 |  |  |  |  if I″ may lead to I then
13 |  |  |  |  |  criterions.push(I″);
14 |  |  |  |  end
15 |  |  |  end
16 |  |  end
   |  |  // control flow analysis
17 |  |  foreach BasicBlock B that may lead to I do
18 |  |  |  criterions.push(B.getTerminator());
19 |  |  end
20 |  end
21 end
```

```
void Stencil::beginNextIter() {
  iterCount++;
  if(iterCount >= totalIter){
    mainProxy.done(); //program exits
  }else{
    for(int i = 0; i < totalDirections; i++) {
      ghostMsg * m = createGhostMsg(dirs[i]);
      copy(m->data, boundary[i]);
      int sendTo = myIdx+dirs[i];
      stencilProxy(sendTo).receiveMessage(m);
    }
  }
}
```

Compiler Slicing Pass

```
   Input:
   f: the targeted function to perform slicing on
   c: set of control variables
   Output: slices: the program slice for recomputation
   // search for slicing criterions
1  foreach Instruction I in f do
2  |  if Defs(I) ⊂ c or I sends messages then
3  |  |  criterions.push(I);
4  |  end
5  end
6  while !criterions.empty() do
7  |  I←criterions.top(); criterions.pop();
8  |  if !I.processed() then
9  |  |  slices.push(I);
   |  |  // data flow analysis
10 |  |  foreach Values I′ in Uses(I) do
11 |  |  |  foreach Instruction I″ in Defs(I′) do
12 |  |  |  |  if I″ may lead to I then
13 |  |  |  |  |  criterions.push(I″);
14 |  |  |  |  end
15 |  |  |  end
16 |  |  end
   |  |  // control flow analysis
17 |  |  foreach BasicBlock B that may lead to I do
18 |  |  |  criterions.push(B.getTerminator());
19 |  |  end
20 |  end
21 end
```

```
void Stencil::beginNextIter() {
  iterCount++;
  if(iterCount >= totalIter){
    mainProxy.done(); //program exits
  }else{
    for(int i = 0; i < totalDirections; i++) {
      ghostMsg * m = createGhostMsg(dirs[i]);
      copy(m->data, boundary[i]);
      int sendTo = myIdx+dirs[i];
      stencilProxy(sendTo).receiveMessage(m);
    }
  }
}
```

The Role of Runtime System

The Role of Runtime System

- Creation of shadow chares

The Role of Runtime System

- Creation of shadow chares
  - Initialize with the same control variables from the original chare

The Role of Runtime System

- Creation of shadow chares
  - Initialize with the same control variables from the original chare
  - Share the same pointers of the non-control variables

The Role of Runtime System

- Creation of shadow chares
  - Initialize with the same control variables from the original chare
  - Share the same pointers of the non-control variables
  - Compare the values of control variables and outgoing messages at the end of entry method

Runtime Guided Replication

```
   Input: o: the original full fledged chare
   s: the shadow chare
   // RTS receives a message M for o
1  checkpointControl(o);
2  checkpointControl(s);
3  restart←true;
4  while restart do
   |  // buffering outgoing messages
5  |  o.invoke(M); s.invoke(M);
6  |  if compareControl(o, s) and compareMsgs(o, s) then
7  |  |  restart←false;
8  |  |  sendMsgs(o); deleteMsgs(s);
9  |  end
10 |  else
11 |  |  restartControl(o); restartControl(s);
12 |  end
13 end
```

Another Example

```
void Stencil:invokeCompution() {
    //computation routine
   for(int i = 0; i < size; ++i){
      temperature[i] = ...
   }
}
```

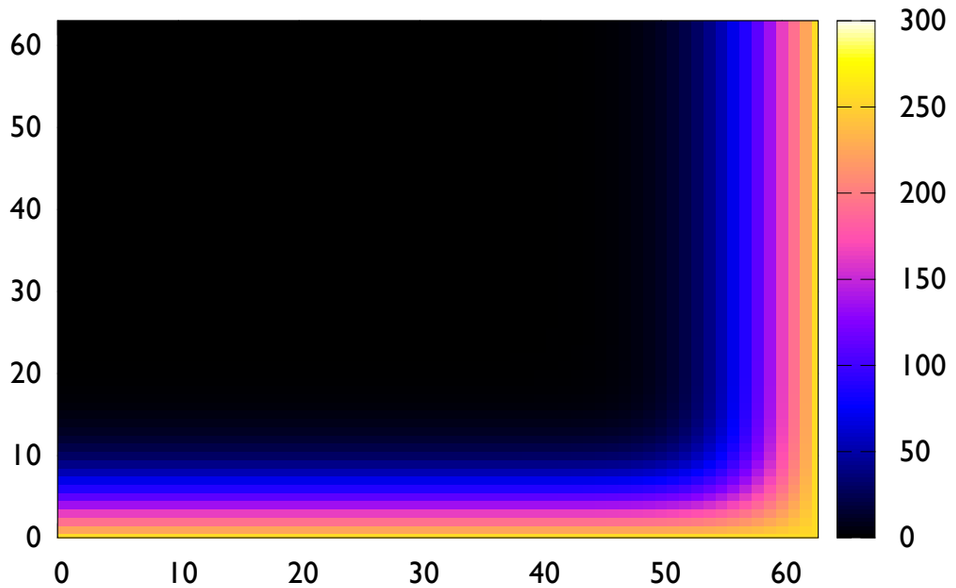
- The previous method fails to protect loop index **i**
  - Lifetime ends before the end of the entry method
  - However, if bit flip occurs to **i**: incorrect data to be used or program crashes

Selective Instruction Duplication

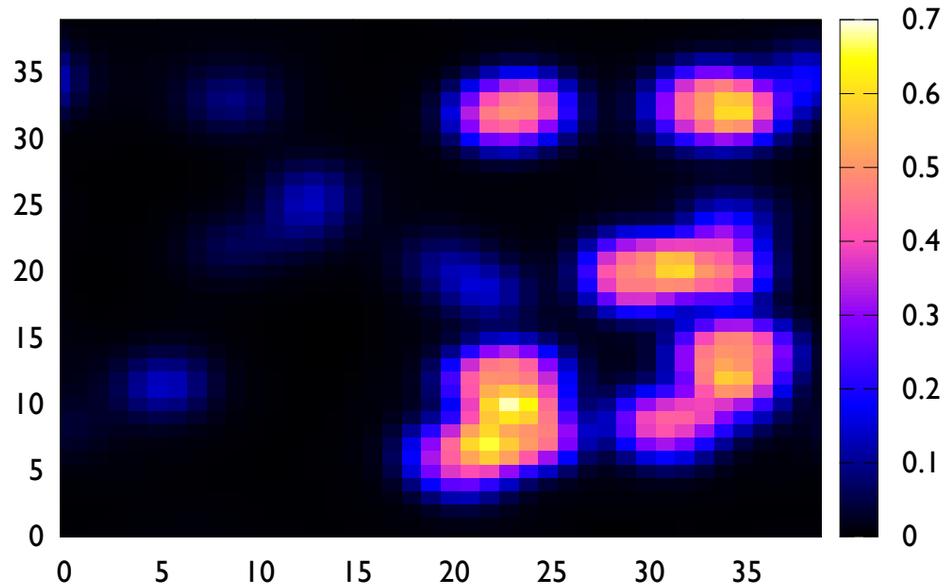
```
1  ;label 0
2  %1 = add i, 1
3  %2 = add i, 1
4  %3 = icmp eq %1, %2
5  br %3, label %4, label %6
6  ;label 4
7  5 = add i, 1
8  br label %6
9  ;label 6
10 %7 = phi [%1, label %0], [%5, label %4]
```

Protection for Field Data

- The rule holds in nature also be held in scientific programs



Stencil2d



OpenAtom

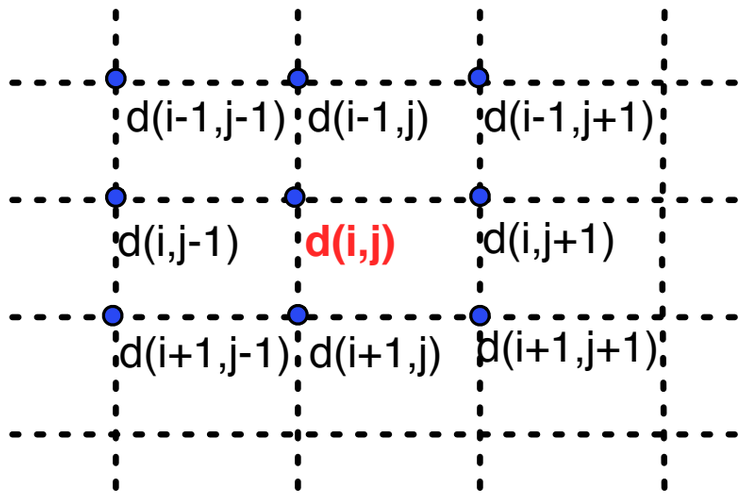
Protection for Field Data

Protection for Field Data

- Spatial similarity

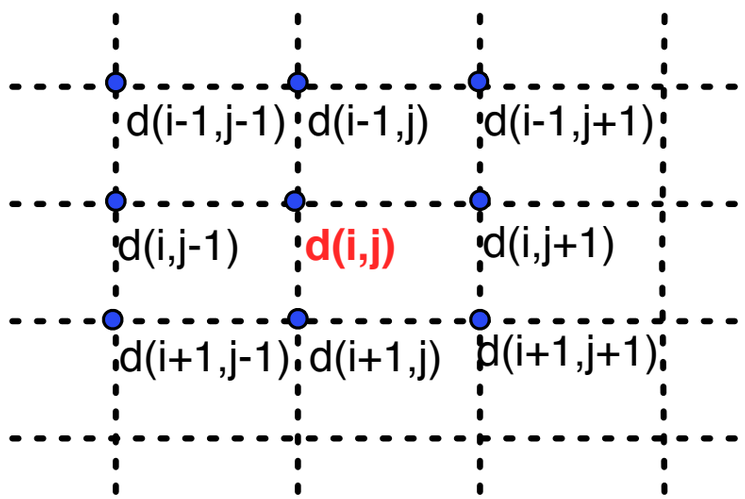
Protection for Field Data

- Spatial similarity



Protection for Field Data

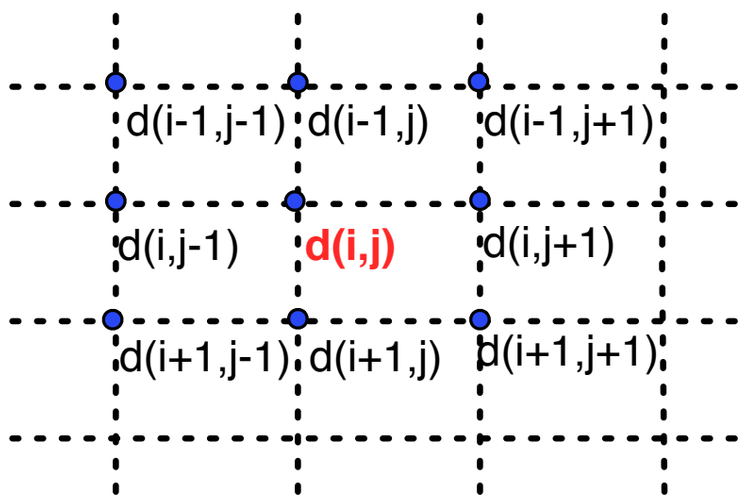
- Spatial similarity



- Temporal similarity

Protection for Field Data

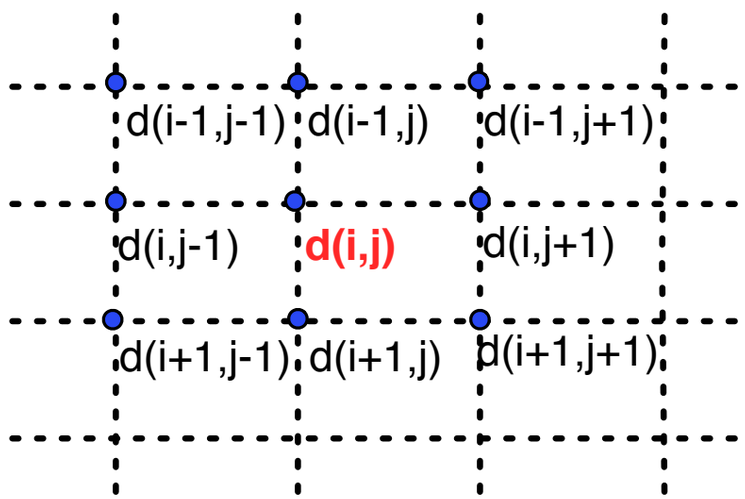
- Spatial similarity



- Temporal similarity
  - data at time step t-2k, t-k, t

Protection for Field Data

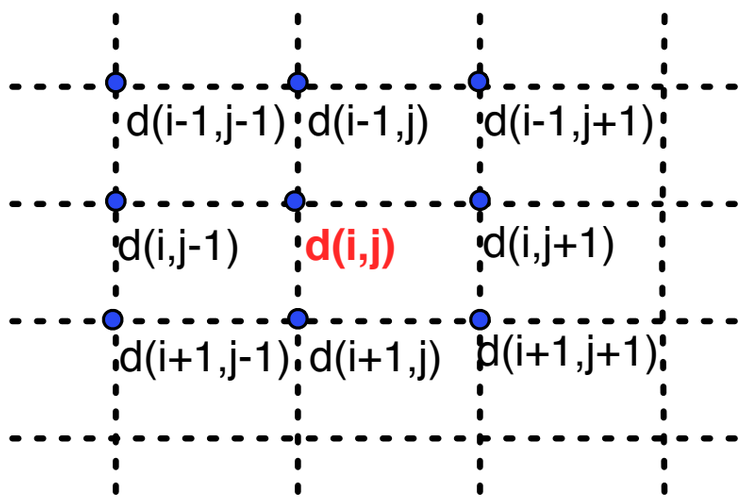
- Spatial similarity



- Temporal similarity
  - data at time step t-2k, t-k, t
- Spatial temporal similarity

Protection for Field Data

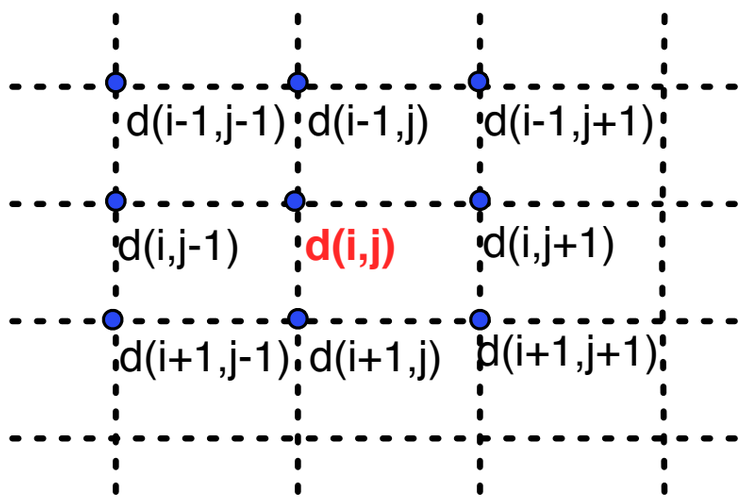
- Spatial similarity



- Temporal similarity
  - data at time step t-2k, t-k, t
- Spatial temporal similarity
  - spatial similarity of temporal updates

Protection for Field Data

- Spatial similarity



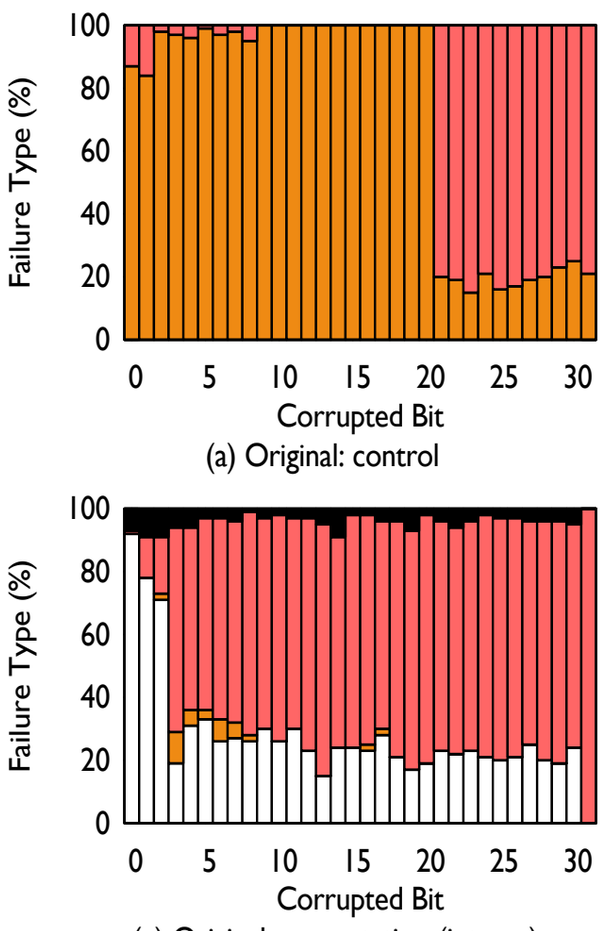
- Temporal similarity
  - data at time step t-2k, t-k, t
- Spatial temporal similarity
  - spatial similarity of temporal updates
  - temporal similarity of spatial differences

Evaluation

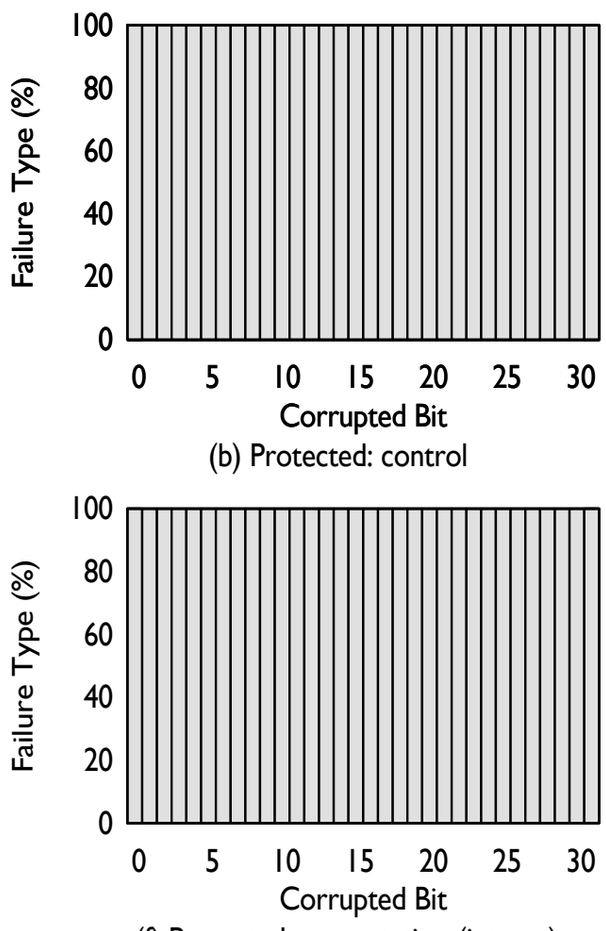
- **Miniaero**
  - Mantevo mini-applications suite
  - compressible Navier-Stokes equations using explicit RK4 method
- **Particle-in-cell**
  - Intel PRK benchmark suite
  - Charm++ implementation
    - Particles are distributed within a fixed grid of charges. At each time step, PIC calculates the impact of the Coulomb potential of particles with related grid points.
- **Stencil3d**
  - 7-point stencil-based computation on a 3D-structured mesh
- Fault Injection with **LLFI**
  - random time
  - random processor

Evaluation

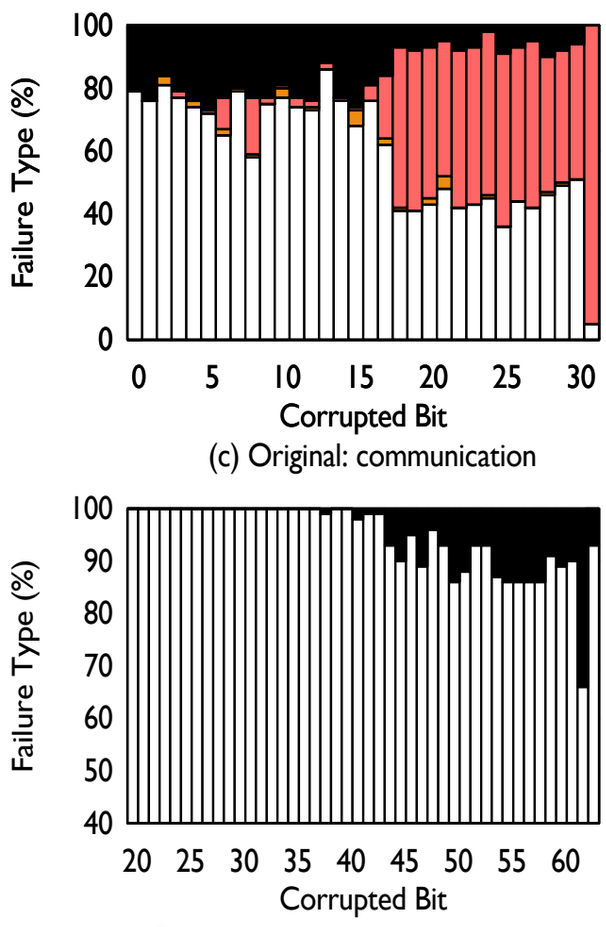
Miniaero



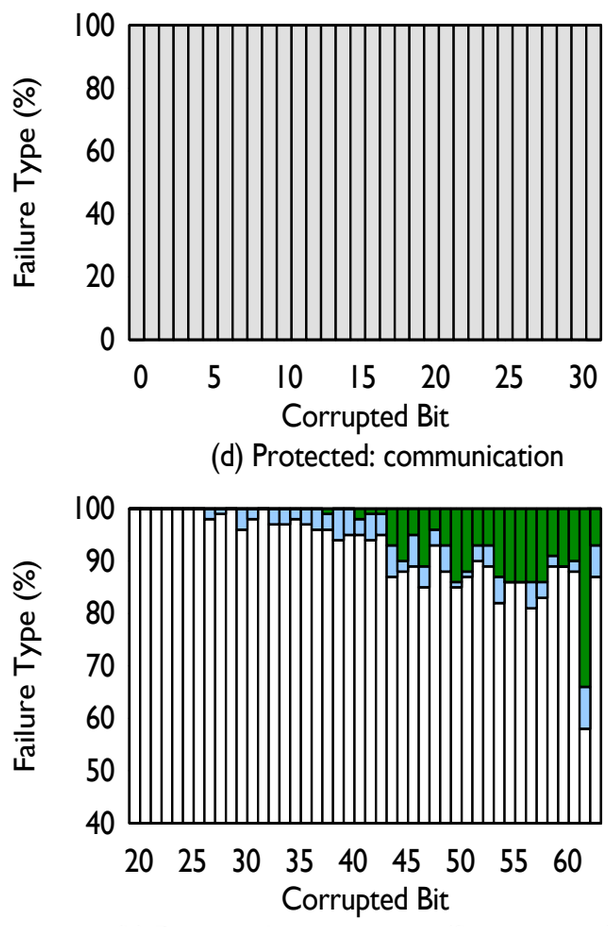
(a) Original: control



(b) Protected: control



(c) Original: communication



(d) Protected: communication

(e) Original: computation (integer)

(f) Protected: computation (integer)

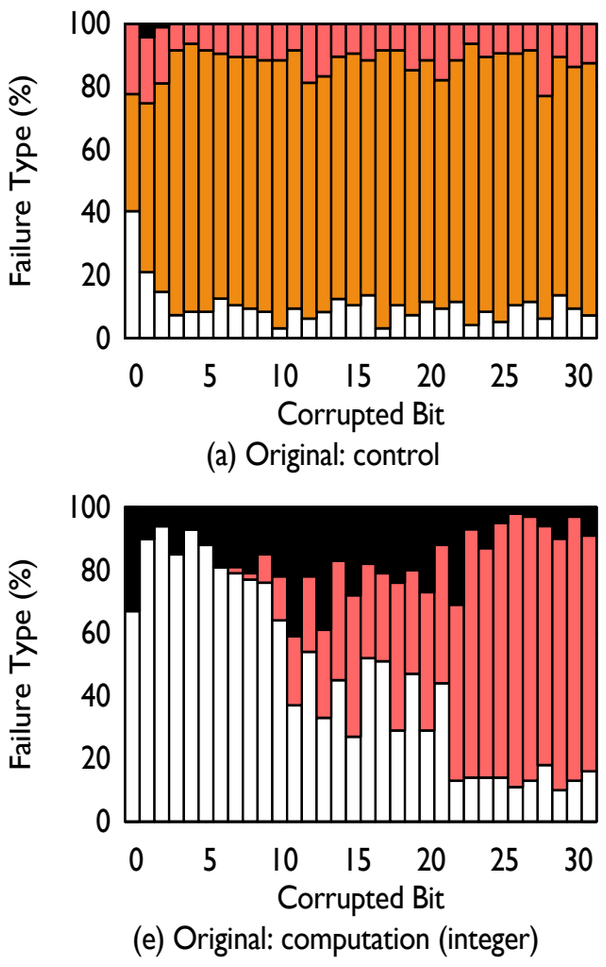
(g) Original: computation (floating point)

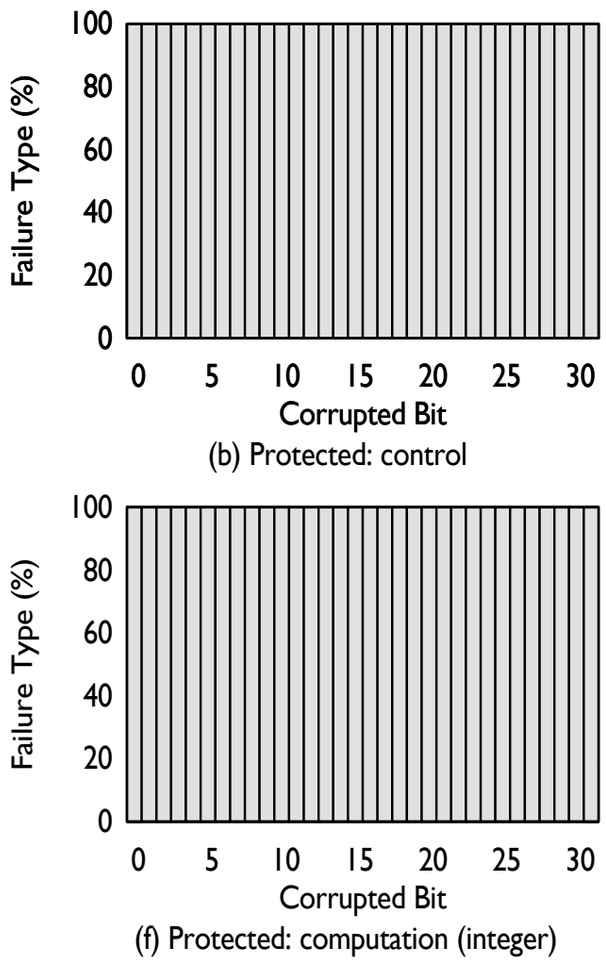
(h) Protected: computation (floating point)

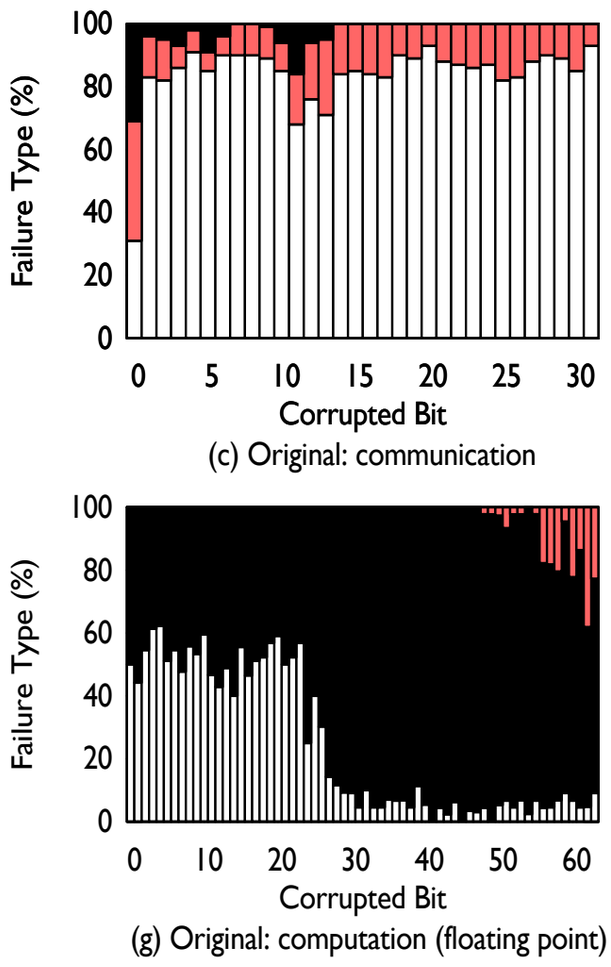
Hang Crash Masked SOC Detected Detected & Masked Detected & Corrected

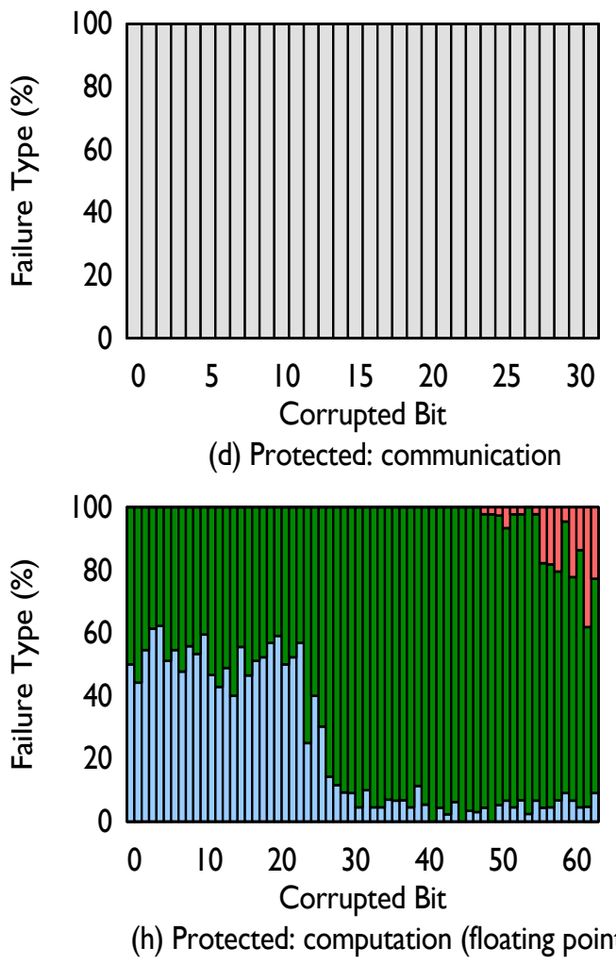
Evaluation

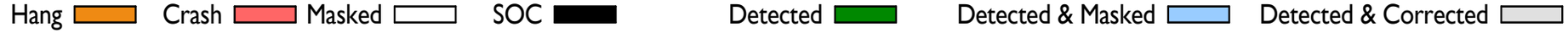
Particle-in-cell





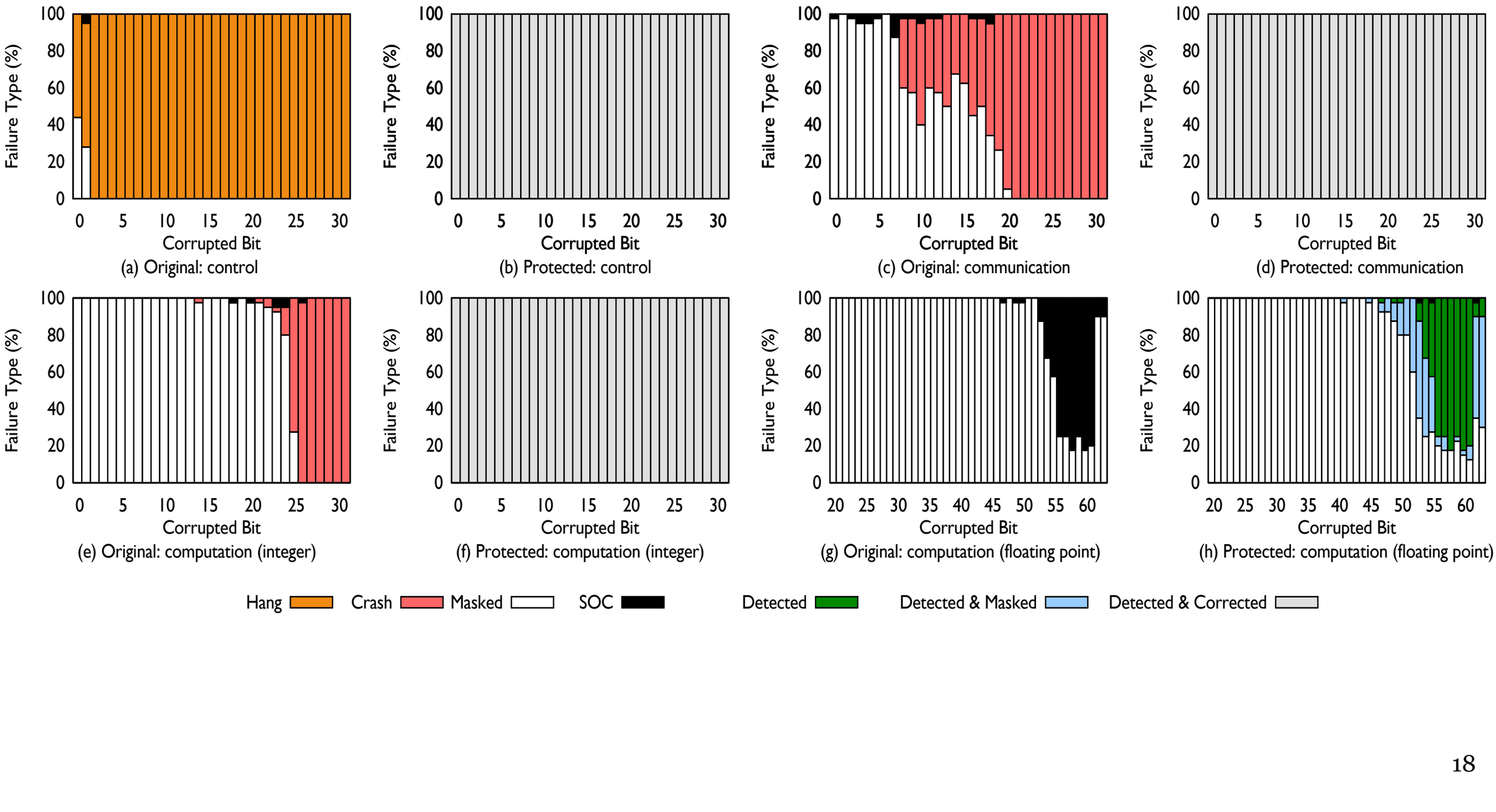




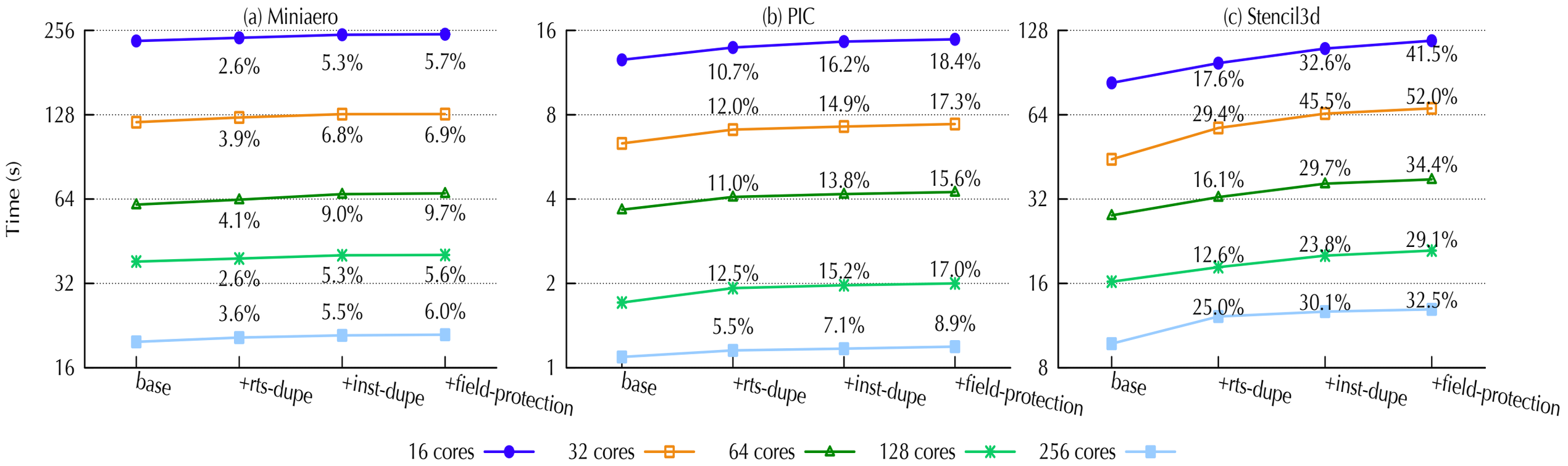


Evaluation

Stencil3d



Performance



Discussion

Discussion

- Compare with traditional checkpoint/restart strategy

Discussion

- Compare with traditional checkpoint/restart strategy
  - For bit-flips induced crashes/hangs, rolling back to previous checkpoint is another solution

Discussion

- Compare with traditional checkpoint/restart strategy
  - For bit-flips induced crashes/hangs, rolling back to previous checkpoint is another solution
  - At the cost of global restart

Discussion

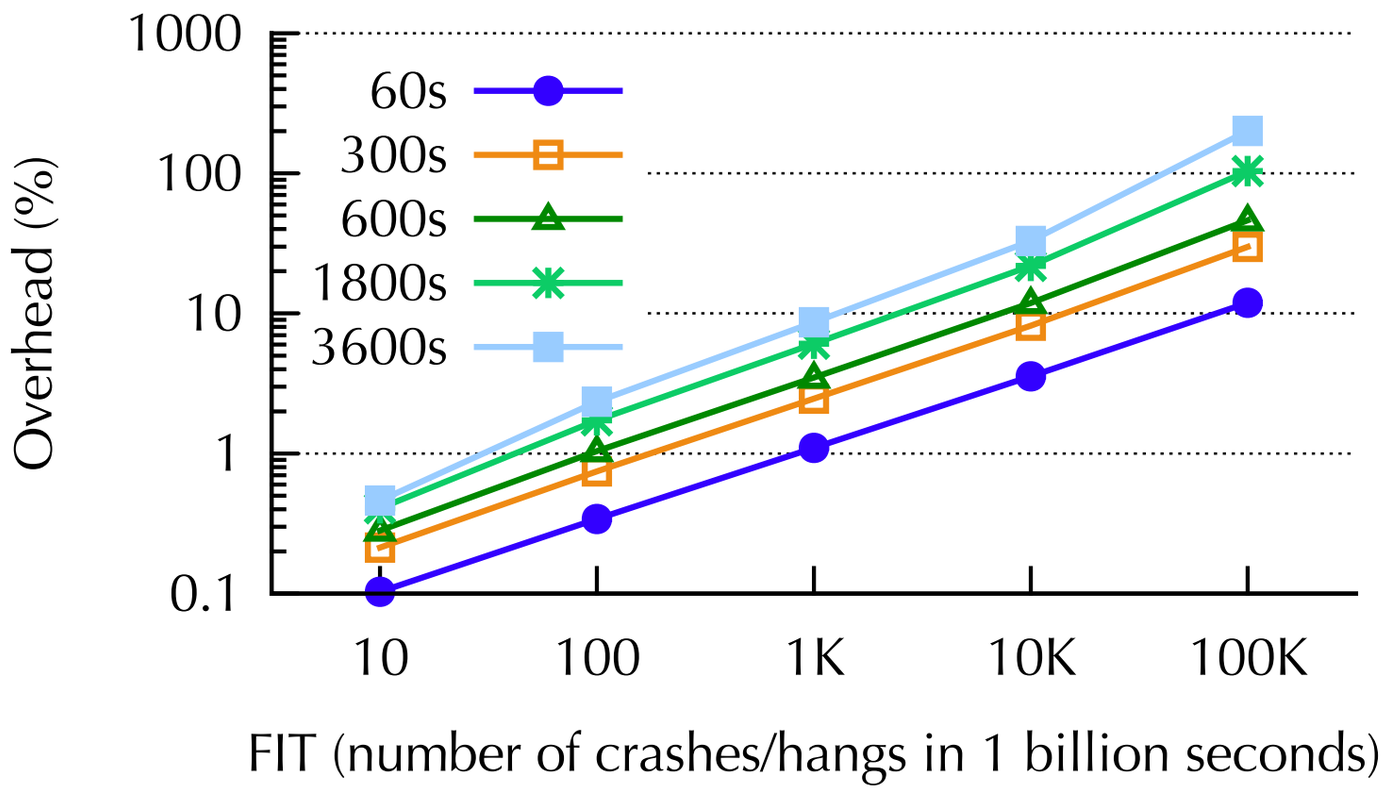
- Compare with traditional checkpoint/restart strategy
  - For bit-flips induced crashes/hangs, rolling back to previous checkpoint is another solution
  - At the cost of global restart
  - With FlipBack, overhead of local restart is minimal

Discussion

- Compare with traditional checkpoint/restart strategy
  - For bit-flips induced crashes/hangs, rolling back to previous checkpoint is another solution
  - At the cost of global restart
  - With FlipBack, overhead of local restart is minimal

Discussion

- Compare with traditional checkpoint/restart strategy
  - For bit-flips induced crashes/hangs, rolling back to previous checkpoint is another solution
  - At the cost of global restart
  - With FlipBack, overhead of local restart is minimal



Conclusion

- Leverage compiler and runtime techniques for a cheaper way to protect applications against silent data corruptions

- Almost 100% coverage

- 6-20% overhead