Practical Hardening of Crash-Tolerant Systems

Marco Serafini (Yahoo! Research BCN)

Joint work with: Miguel Correia (IST-UTL / INESC-ID)

Daniel Gómez Ferro and Flavio Junqueira (Yahoo! Research BCN)



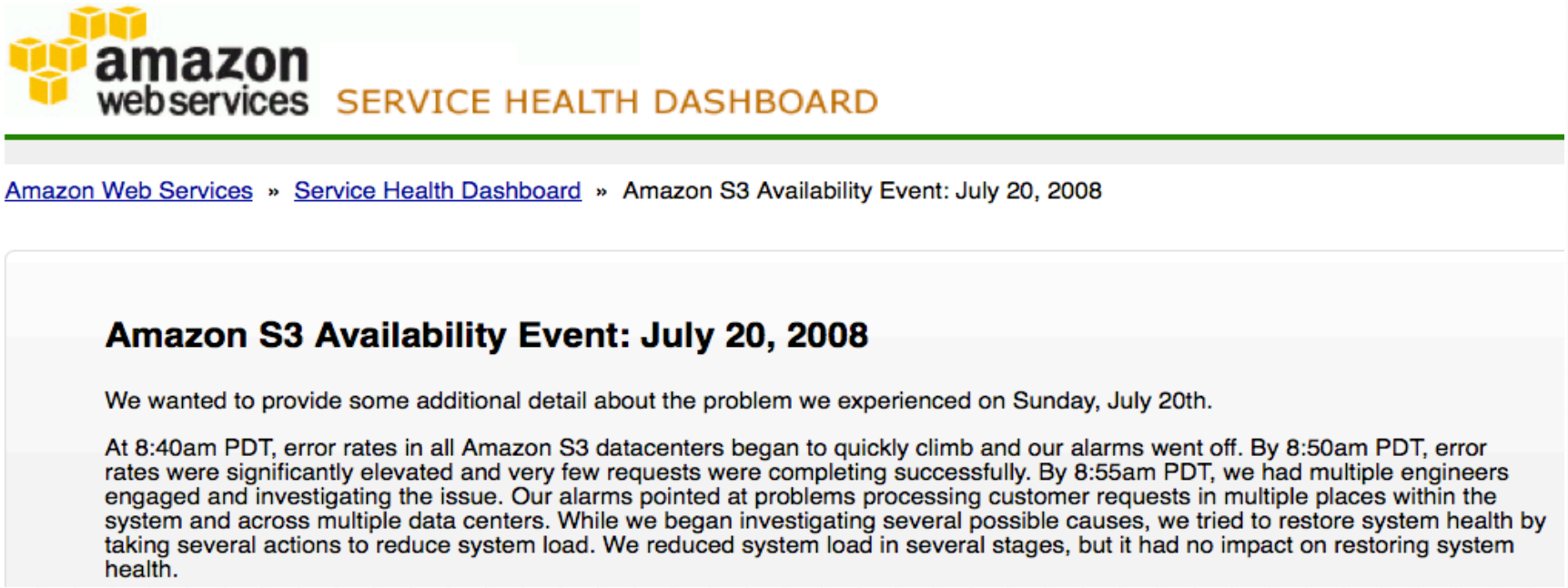
Dependability in data centers

- Crashes are commonplace…
- … but scarier faults **do** occur



A horror story

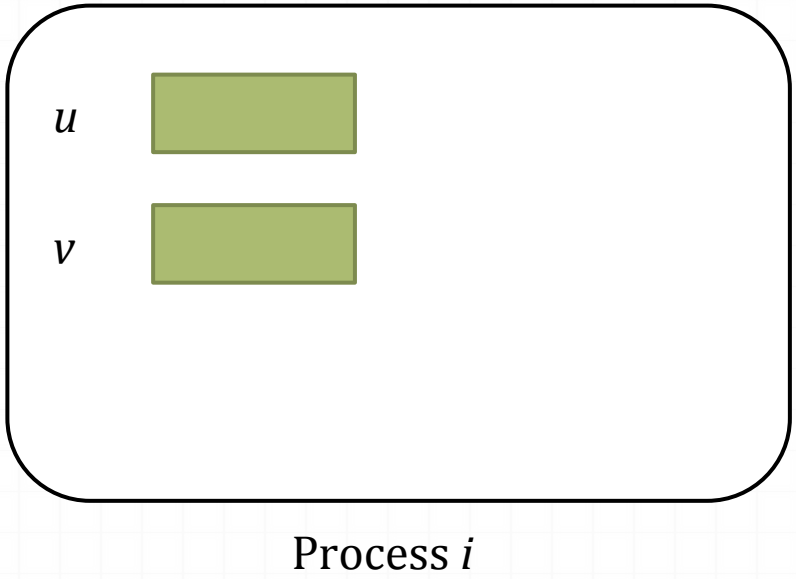
An 8-hour system-wide outage due to a single hardware fault



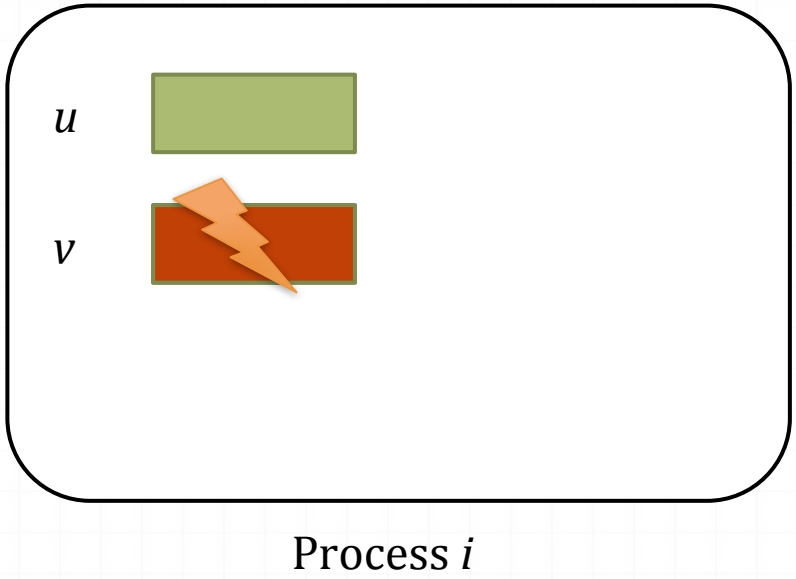
What happened?

- Quoted from the Amazon service health dashboard
  - *"A handful of messages had a single bit corrupted"*
  - *"The message was still intelligible, but the system state information was incorrect"*
  - *"We used MD5 checksums throughout the system (but not) for this particular internal state information"*
  - *"(The corruption) spread throughout the system causing the symptoms described above"*

Error propagation

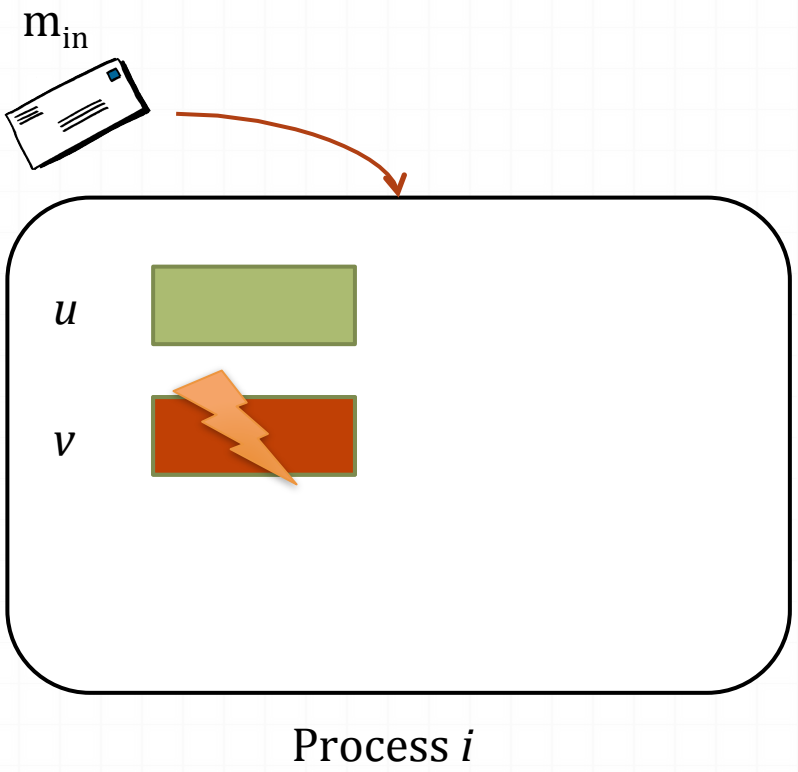


Error propagation

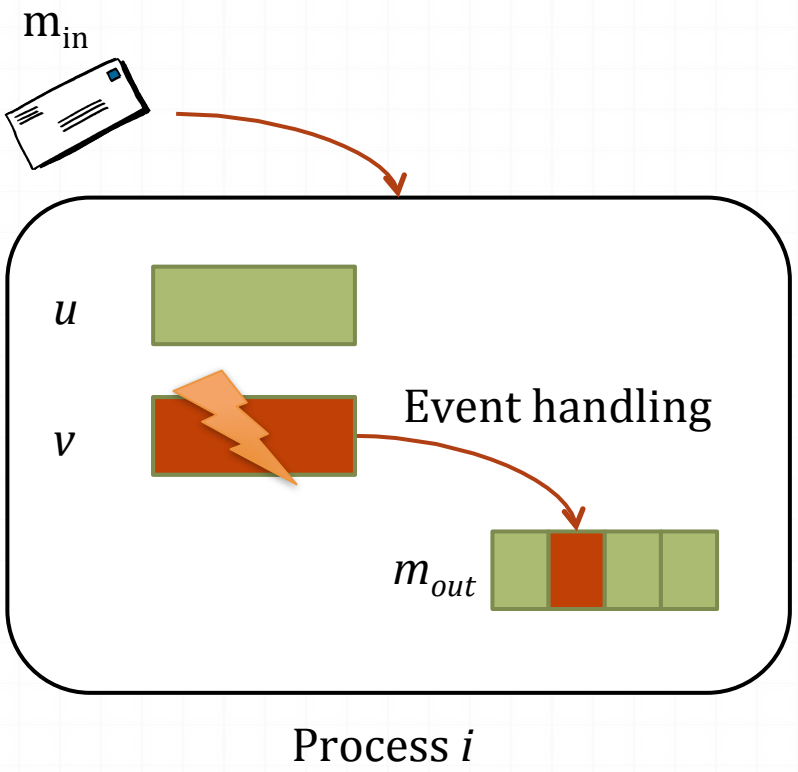


Process *i*

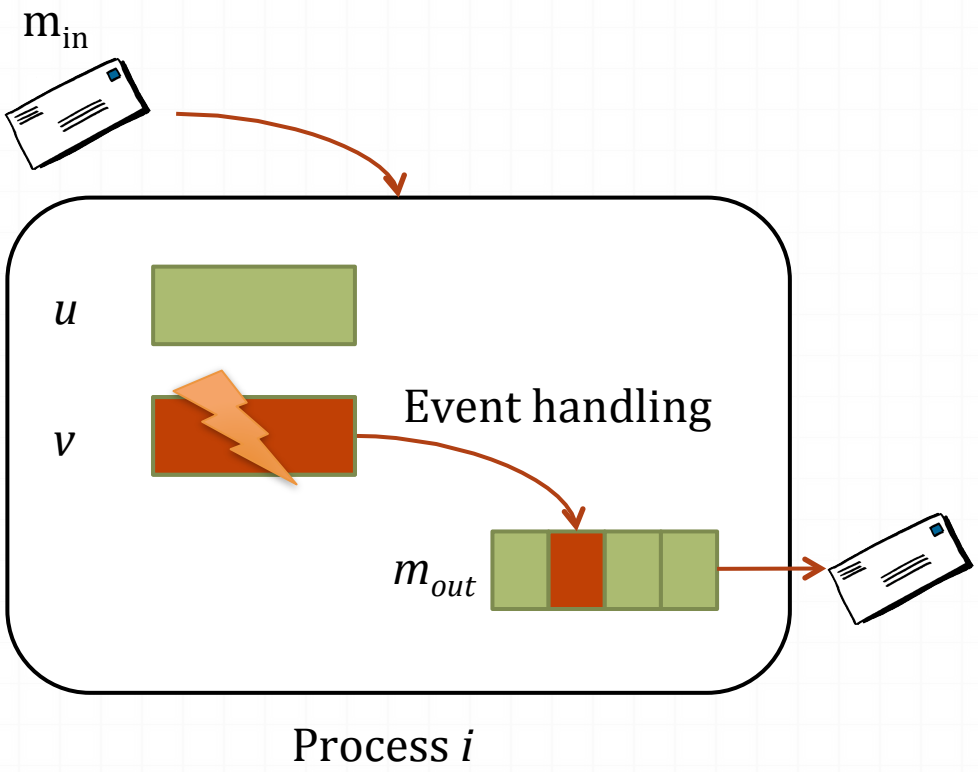
Error propagation



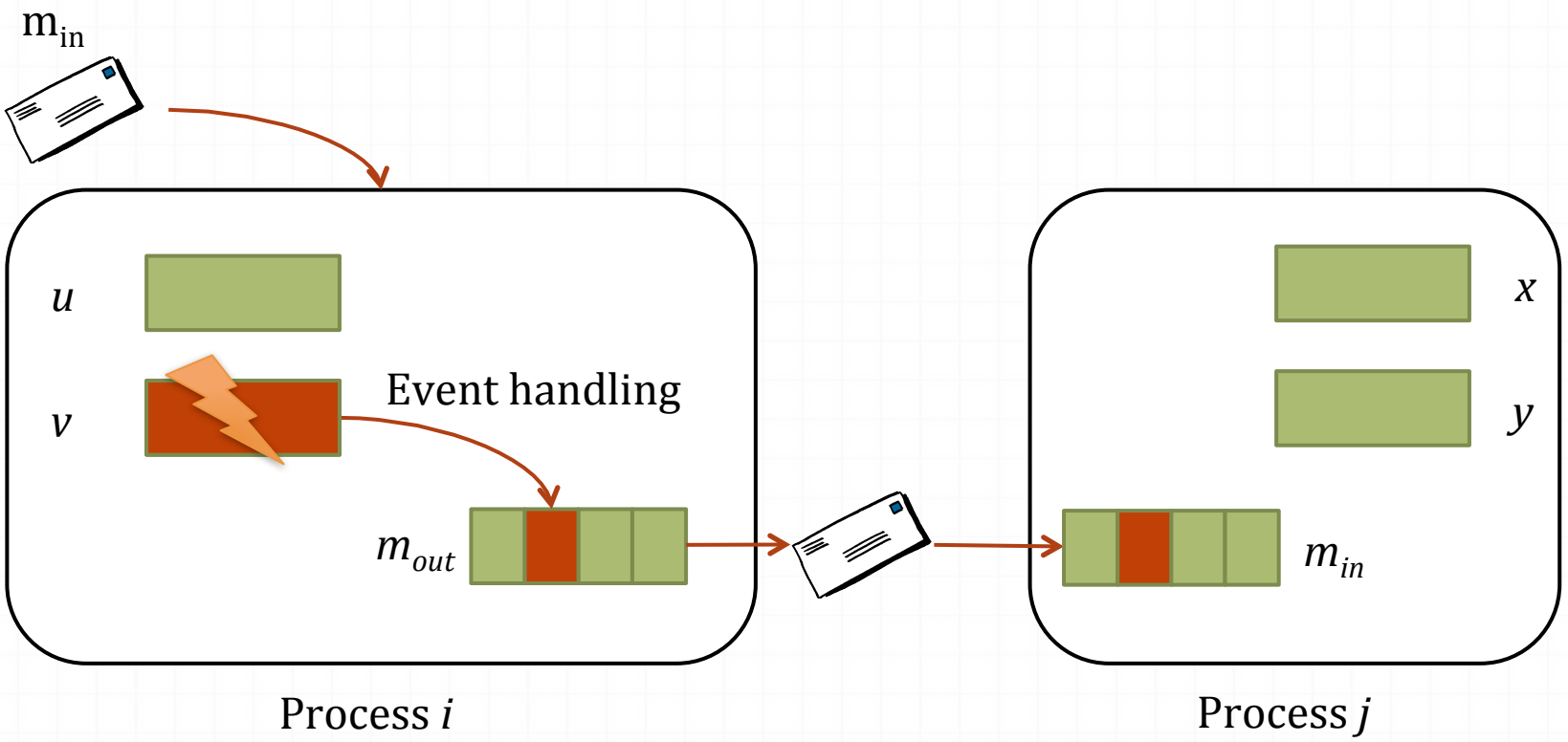
Error propagation



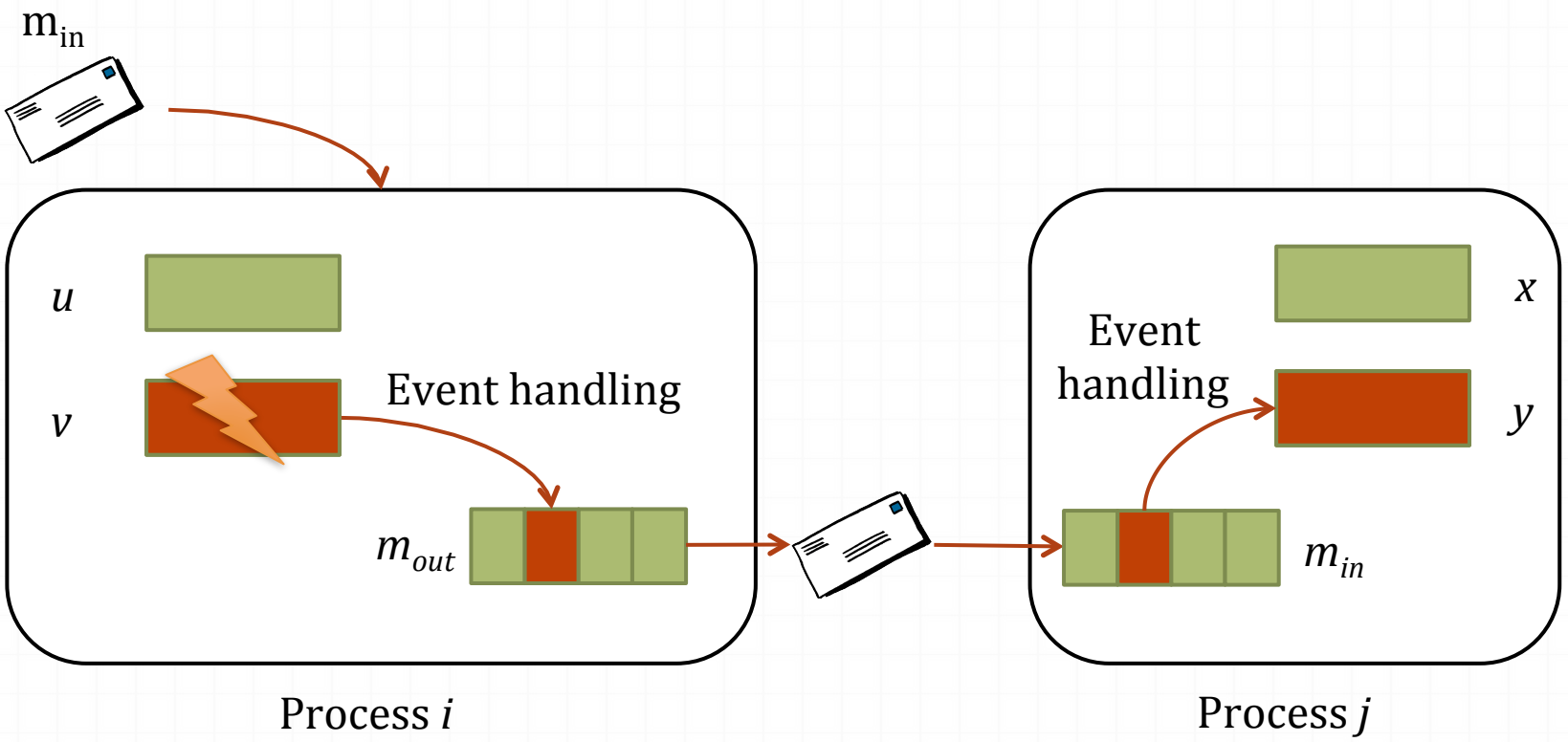
Error propagation



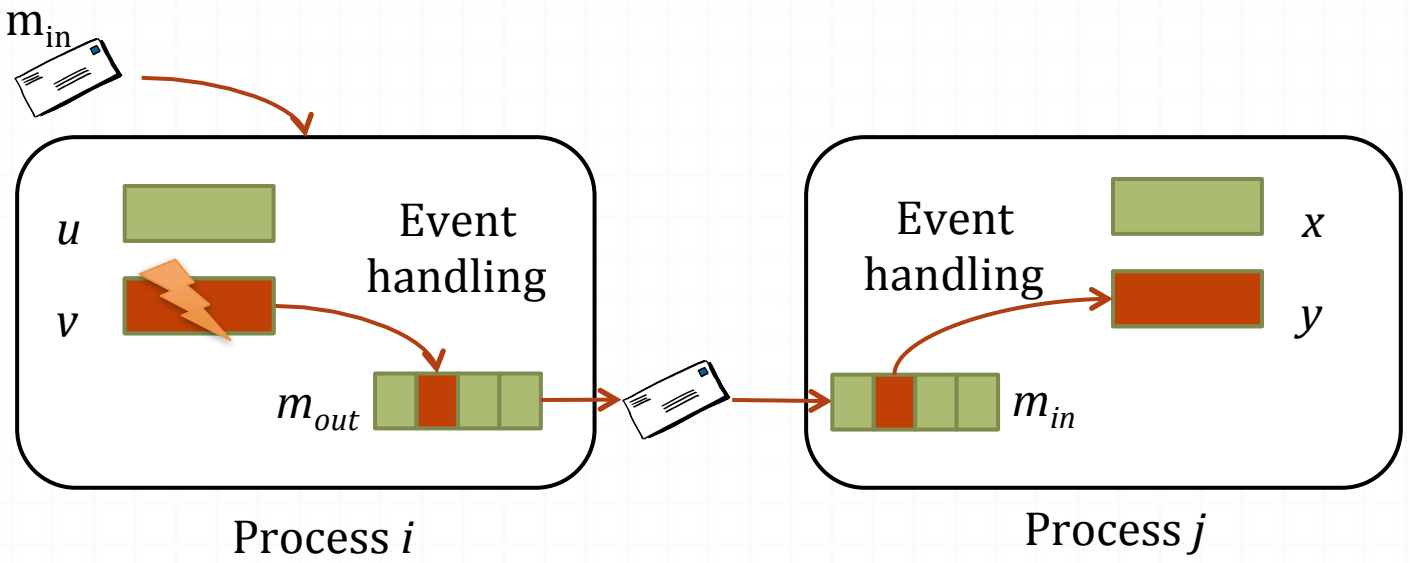
Error propagation



Error propagation

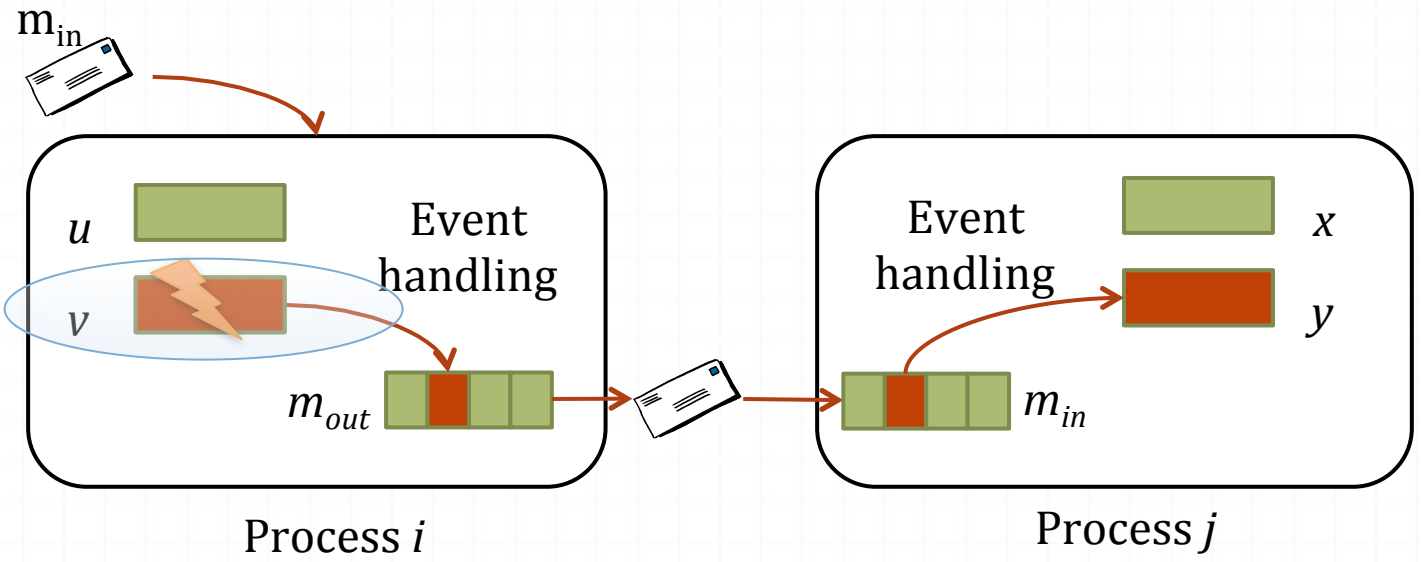


A new approach to error isolation



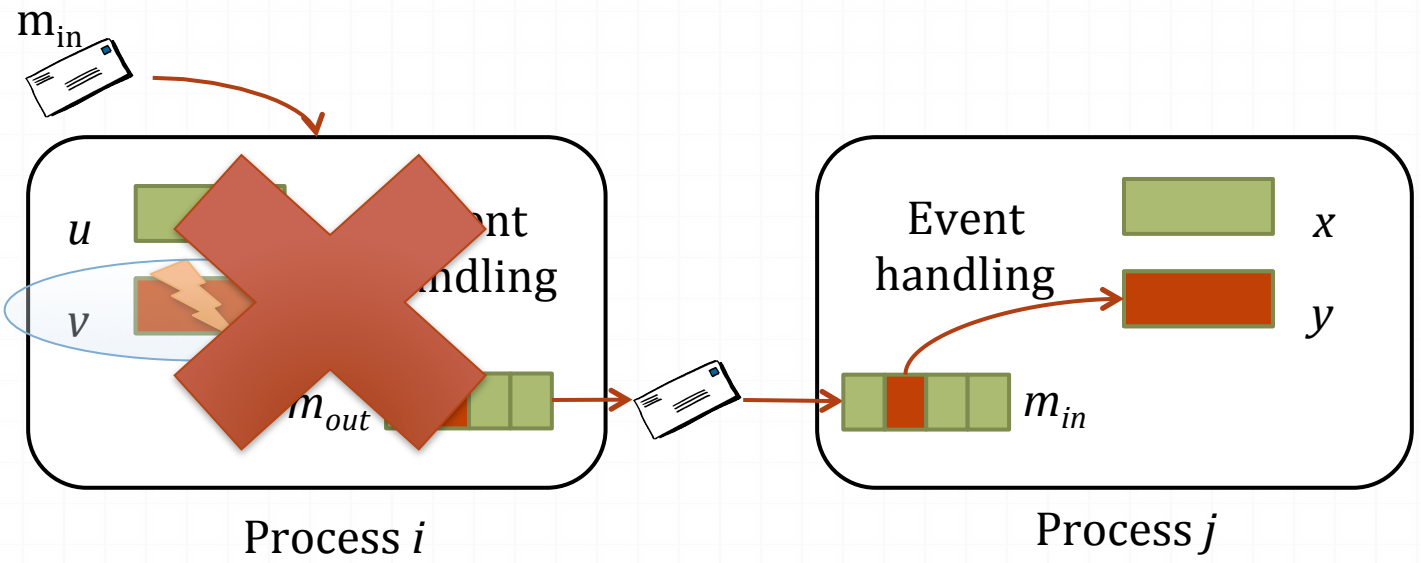
1. General model of **process** behavior
2. **Arbitrary State Corruption (ASC)** fault model
3. **Guarantee error isolation** through hardening

A new approach to error isolation



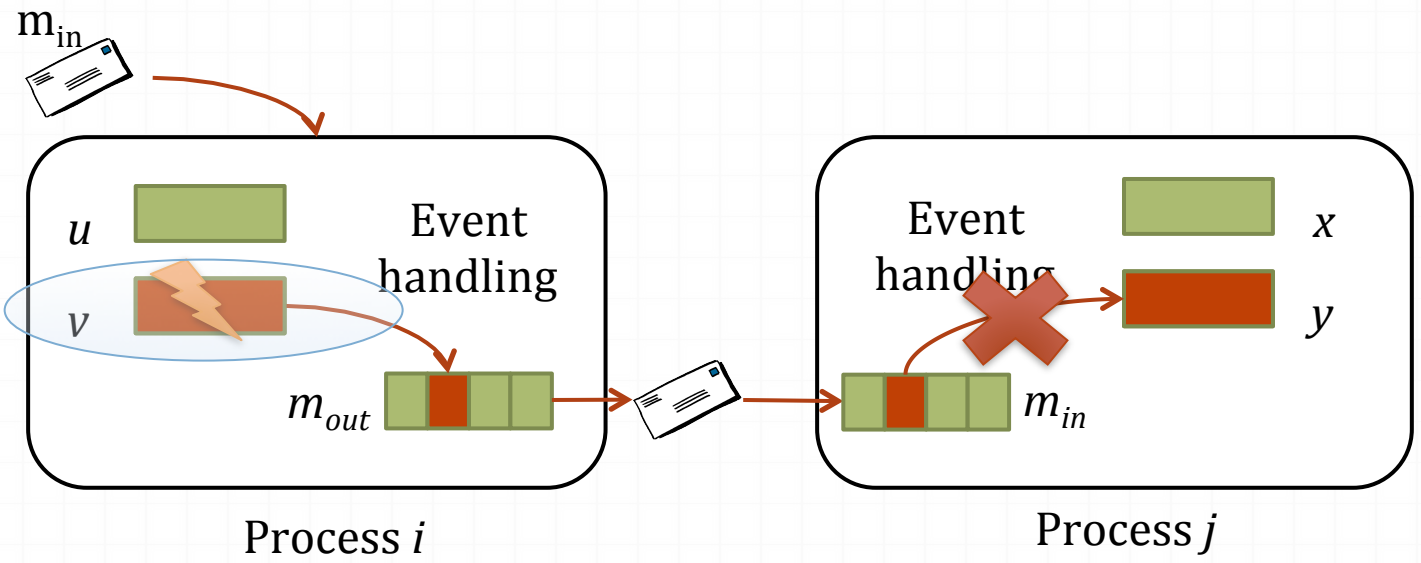
1. General model of **process** behavior
2. **Arbitrary State Corruption (ASC)** fault model
3. **Guarantee error isolation** through hardening

A new approach to error isolation



1. General model of **process** behavior
2. **Arbitrary State Corruption (ASC)** fault model
3. **Guarantee error isolation** through hardening

A new approach to error isolation

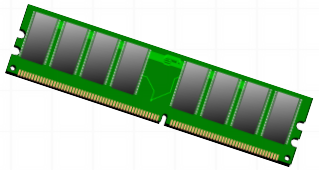


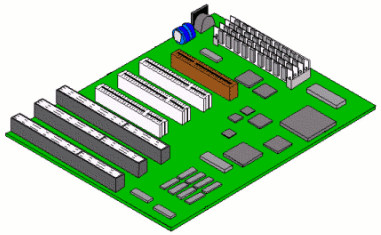
1. General model of **process** behavior
2. **Arbitrary State Corruption (ASC)** fault model
3. **Guarantee error isolation** through hardening

Data corruptions

- Commodity disks are known to be unreliable
  - Faulty firmware is the first reason
- RAM: ECC errors are frequent
  - Production machines only see detected errors
    → Coverage not known
- Interconnects and CPUs also fail
  - Faulty drivers or bit flips







Existing approaches

Background

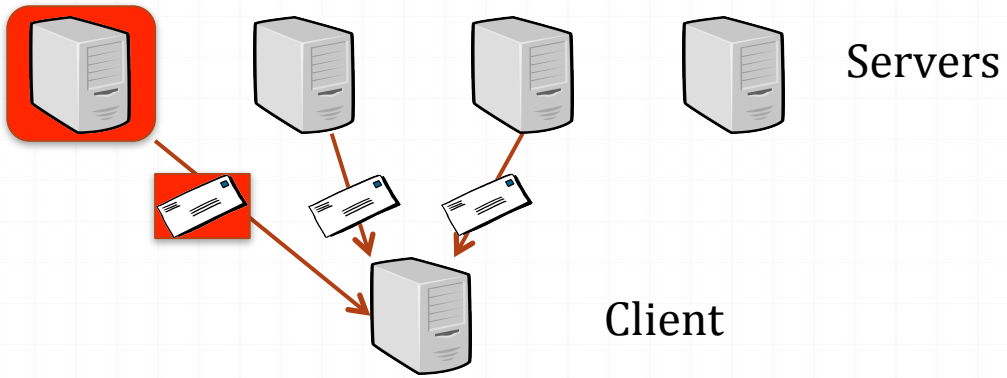
Common practice

- Manual placement of error detection checks
  - Application knowledge
  - Time consuming
- Hard to structure without fault model
- No error isolation guarantee



Byzantine fault model

- **Black-box model** of faulty processes: adversarial
- Hardening for error isolation [Nysiad NSDI 2008]
  - Based on state machine replication
  - Replication and performance costs



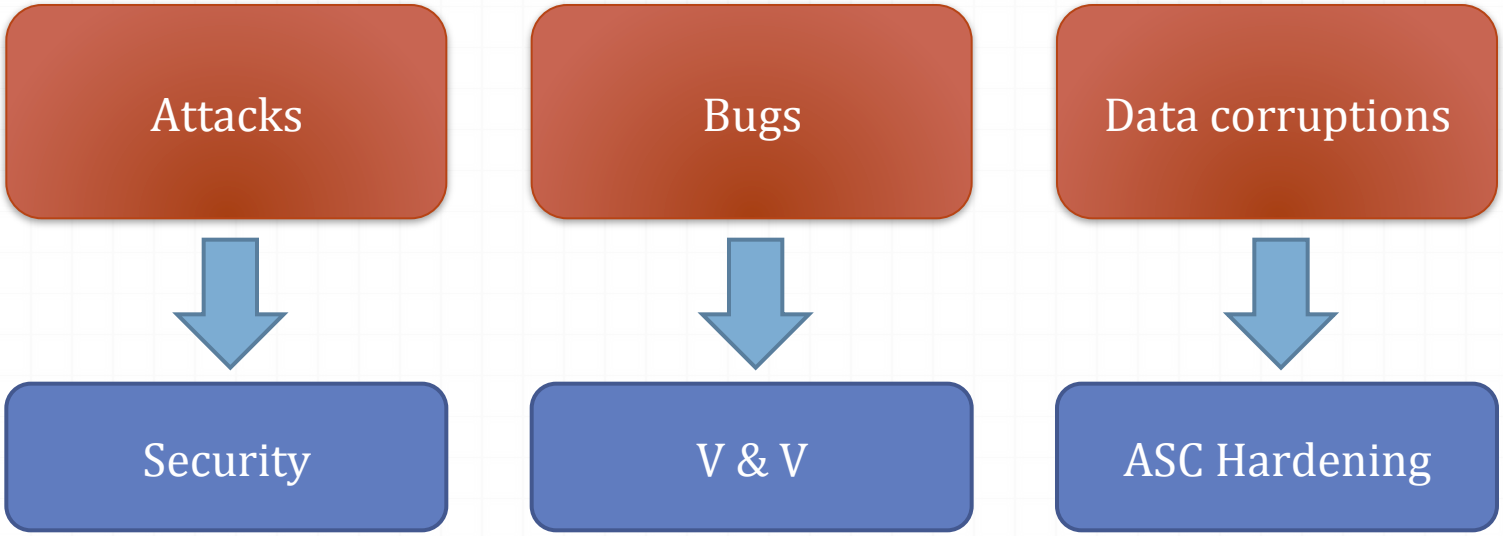
Byzantine faults

- Byzantine hardening covers **attacks** and **bugs**…
- … assuming, e.g., **design diversity** of replicas
  - Unpractical in most systems



Byzantine faults

- Byzantine hardening covers **attacks** and **bugs**…
- … assuming, e.g., **design diversity** of replicas
  - Unpractical in most systems



Process and fault models

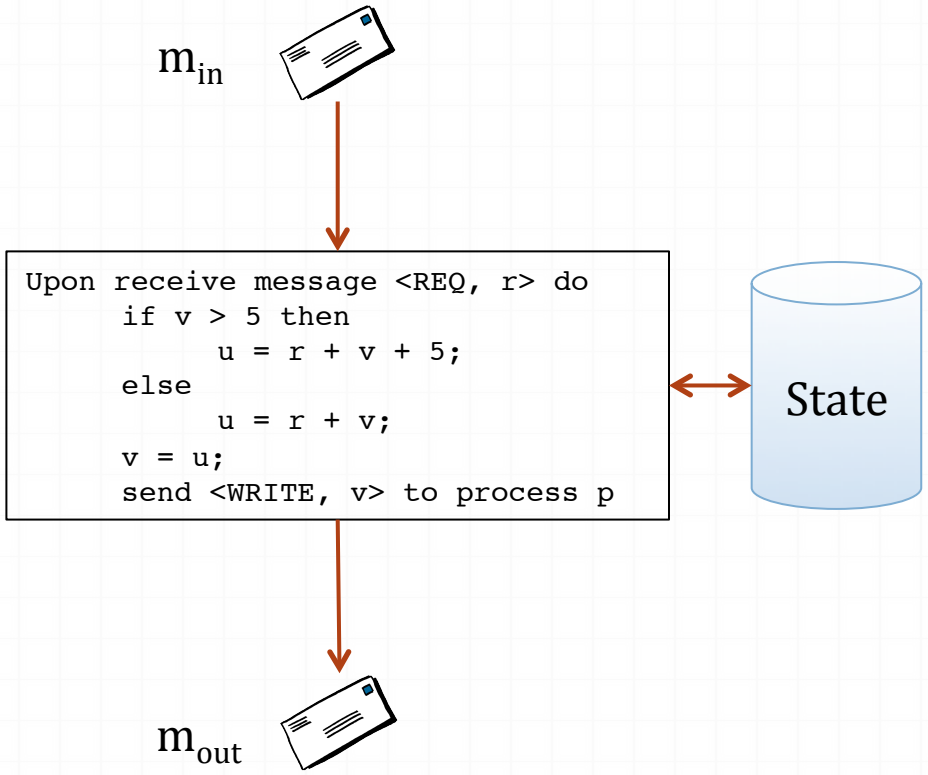
Defining Arbitrary State Corruptions

Process model

1) Event Dispatching

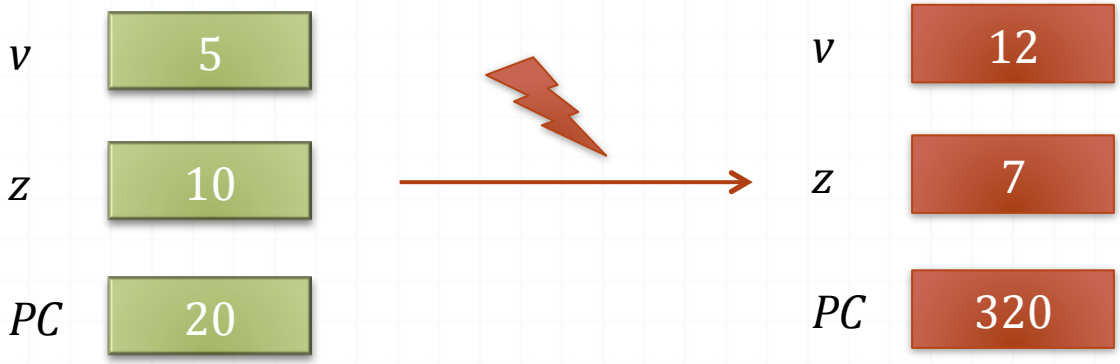
2) Event Handling

3) Message sending



ASC fault model

- An **Arbitrary State Corruption** can make a process
  - Crash
  - Assign an arbitrary value to any variable
  - Start the execution from an arbitrary instruction



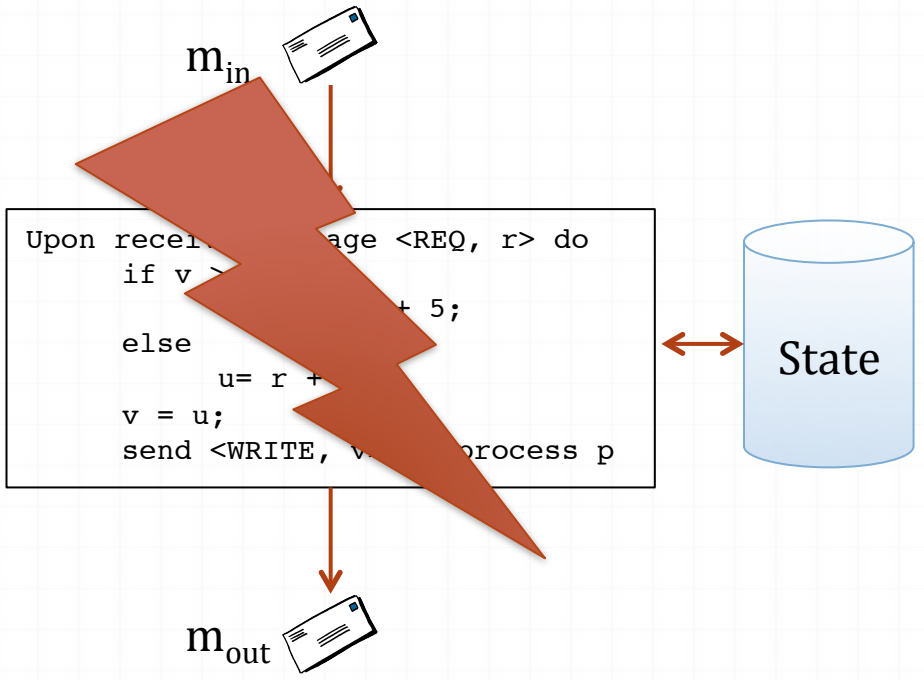
Fault frequency

- One fault for every processed input message

1) Event Dispatching

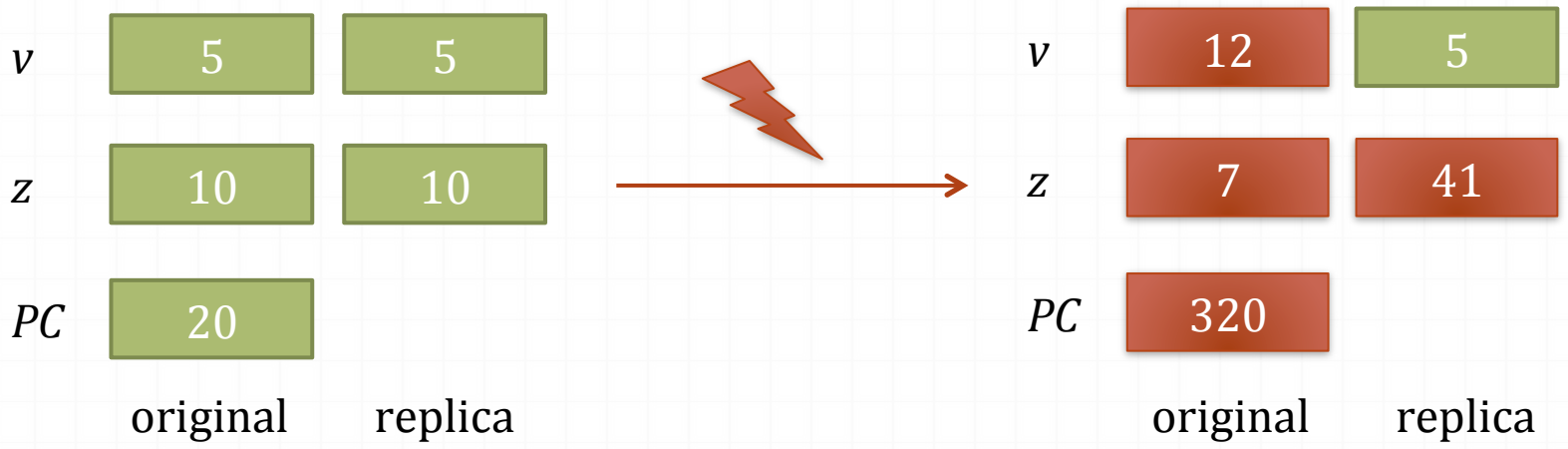
2) Event Handling

3) Message sending



Fault diversity

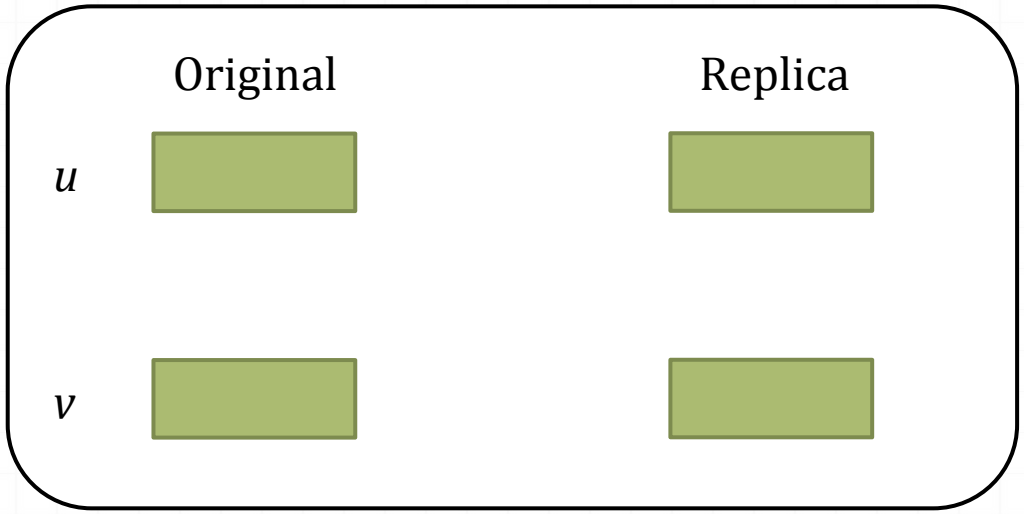
- A corrupted variable is different from its replica



- Only holds **immediately after the fault**
  - Can be invalidated if instructions modify the variable

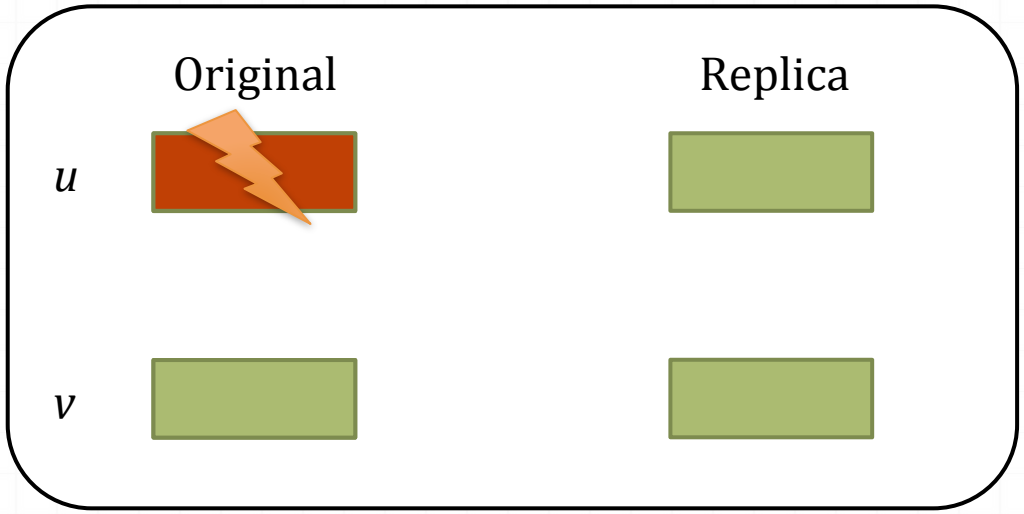
Error propagation

- Fault diversity does not hold
- Hardening preserves diversity



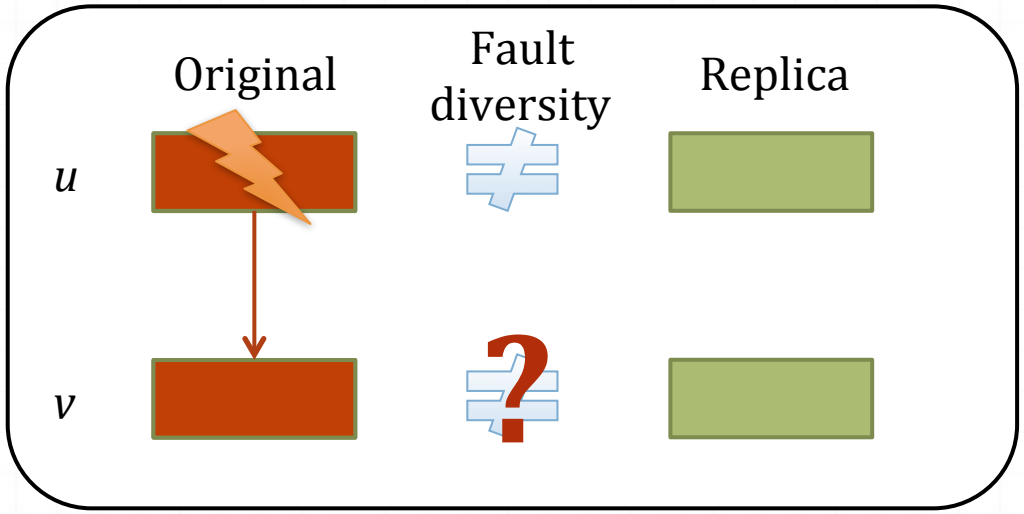
Error propagation

- 0 Fault diversity does not hold
- 0 Hardening preserves diversity



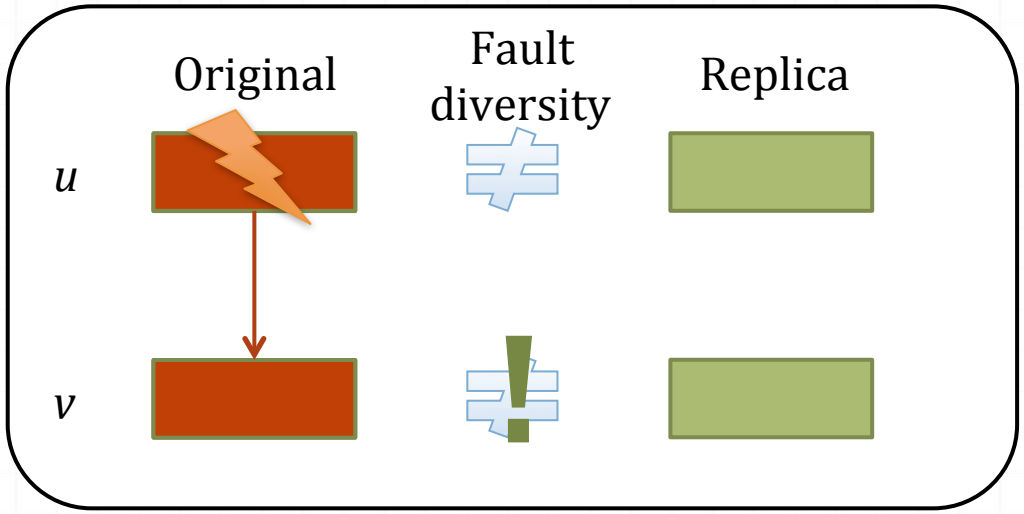
Error propagation

- Fault diversity does not hold
- Hardening preserves diversity



Error propagation

- Fault diversity does not hold
- Hardening preserves diversity

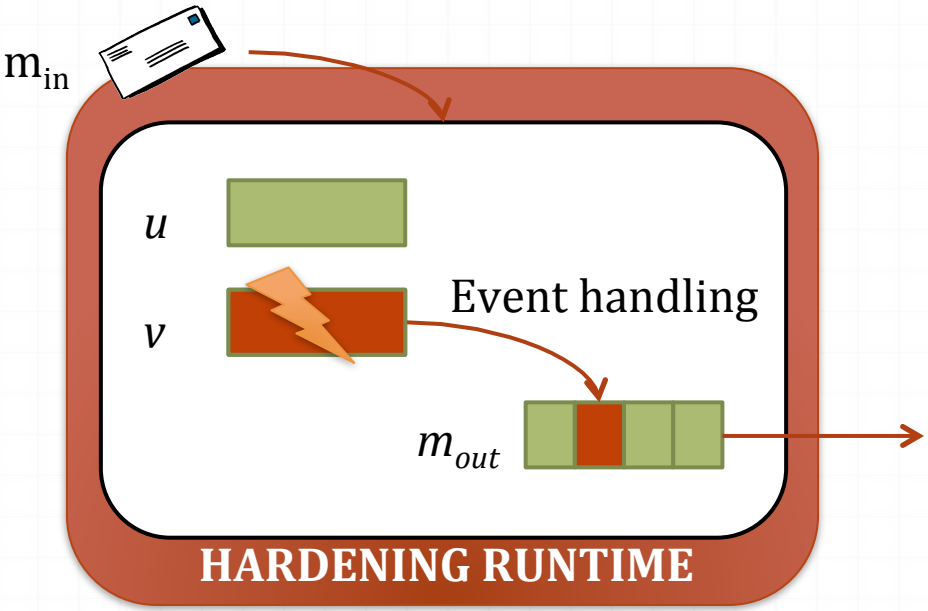


ASC hardening

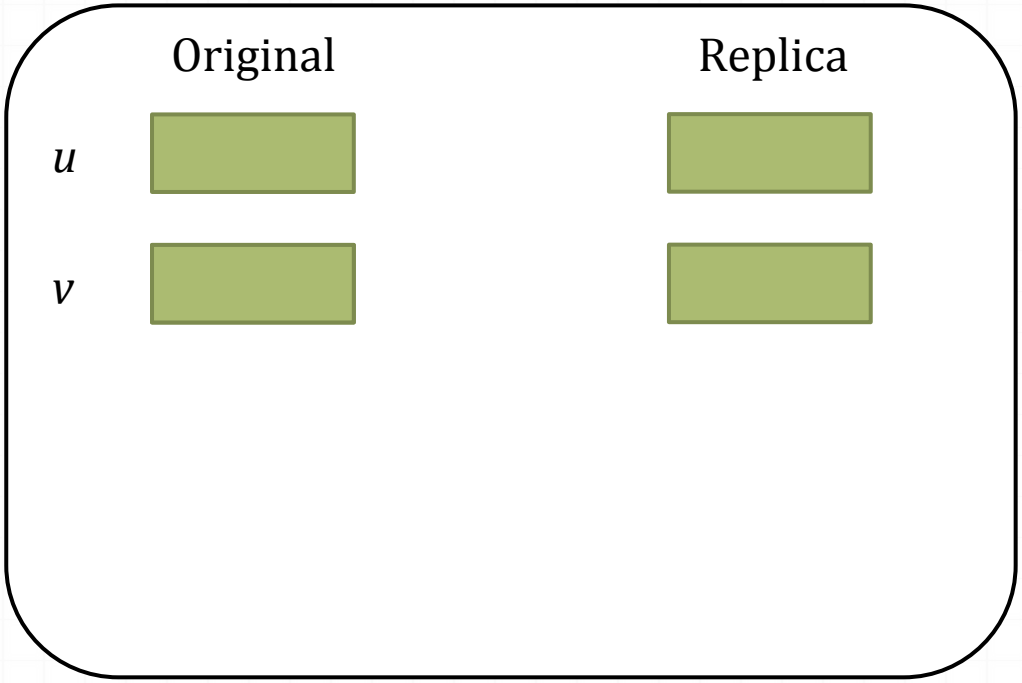
From ASC faults to crashes and message omissions

From ASC to crashes

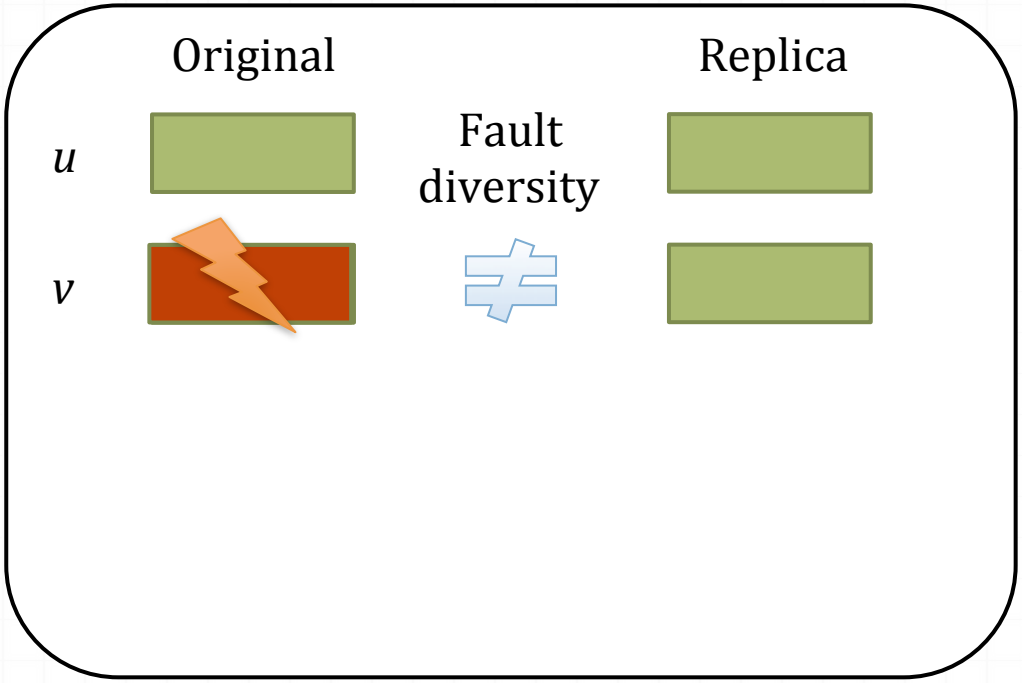
- **Transparent**: to the hardened process
- **Local**: no process replication on multiple machines
- **Untrusted**: can have faults while executing hardening



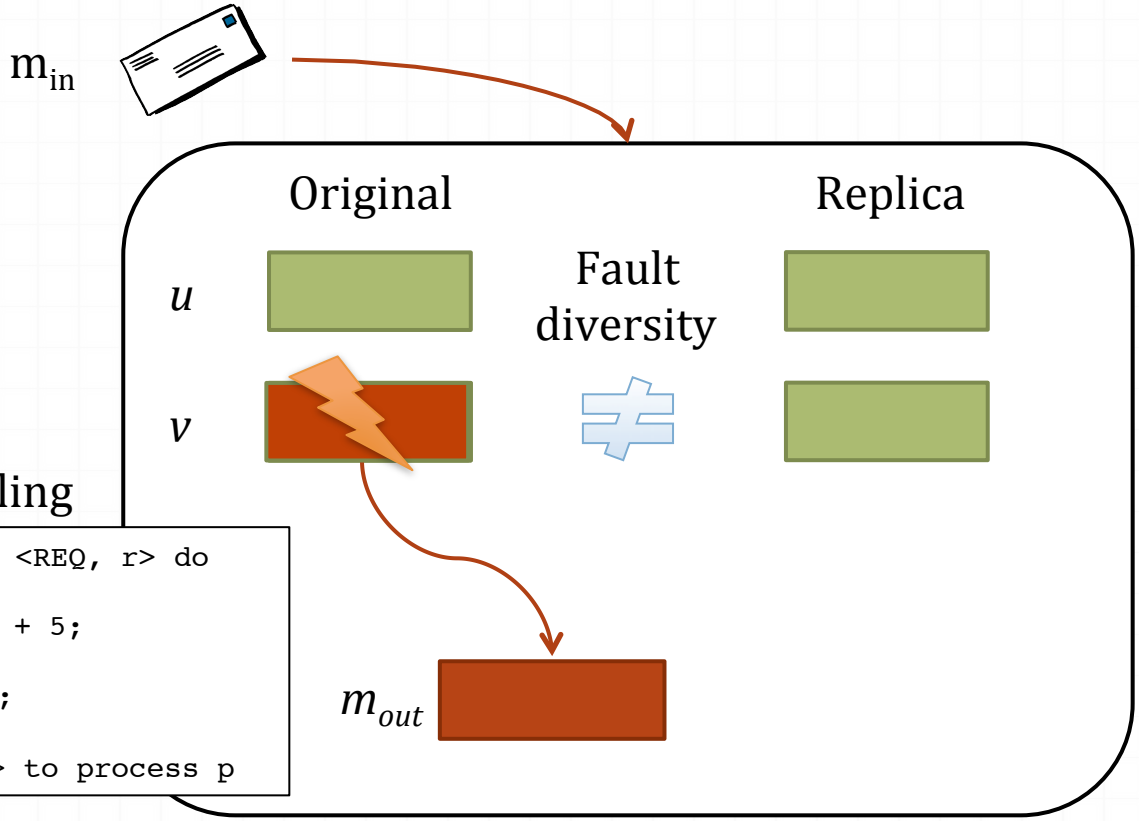
Simple hardening



Simple hardening



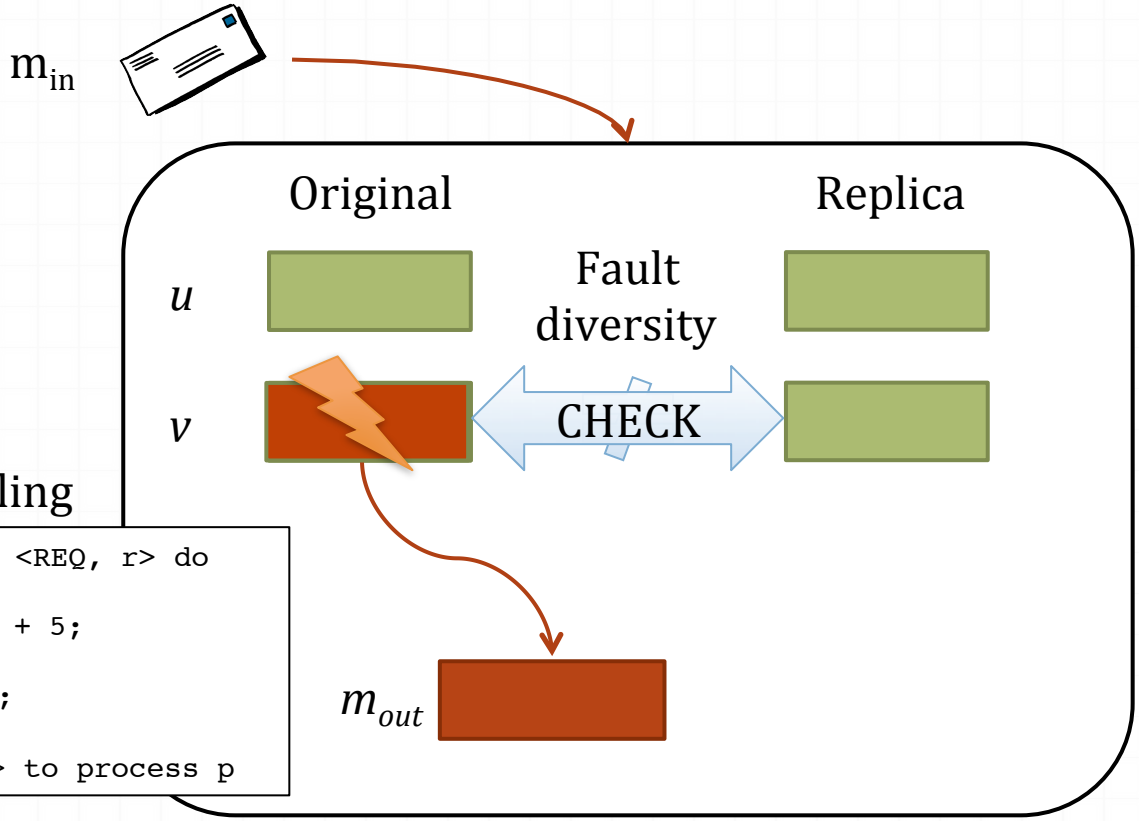
Simple hardening



Event handling

```
Upon receive message <REQ, r> do
      if v > 5 then
            u = r + v + 5;
      else
            u = r + v;
      v = u;
      send <WRITE, v> to process p
```

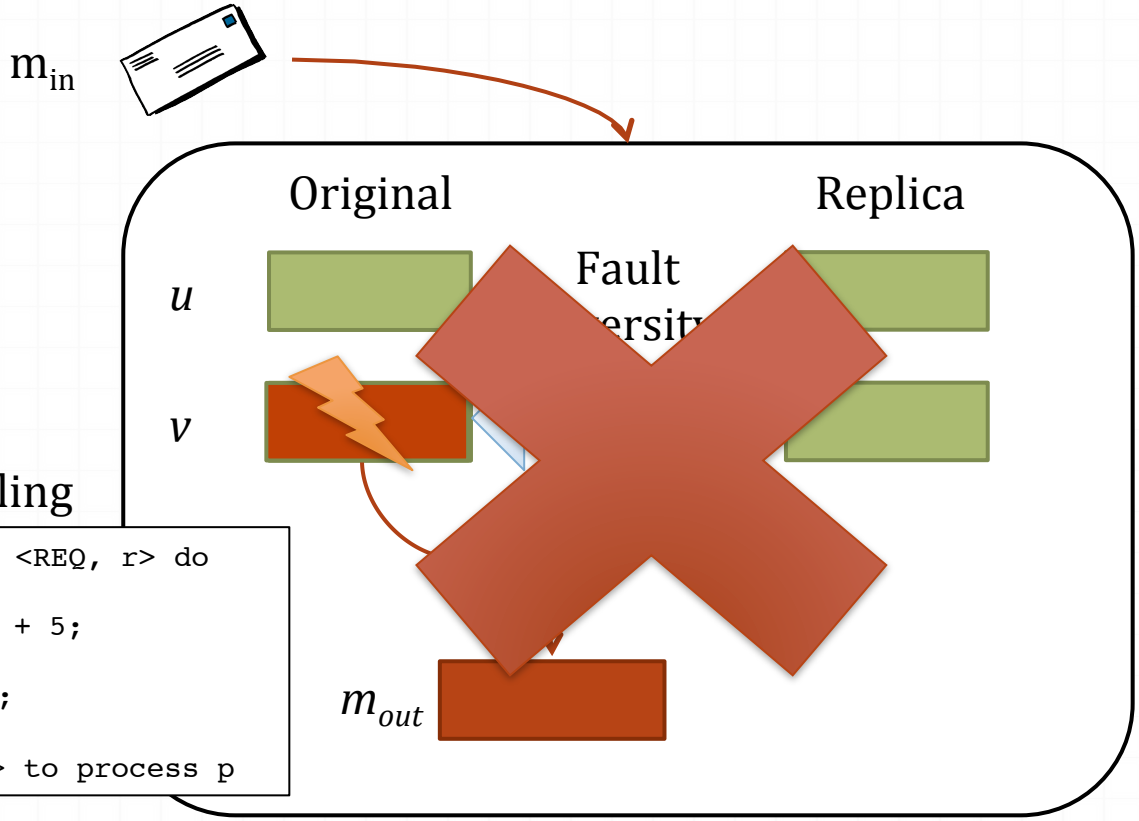
Simple hardening



Event handling

```
Upon receive message <REQ, r> do
      if v > 5 then
            u = r + v + 5;
      else
            u = r + v;
      v = u;
      send <WRITE, v> to process p
```

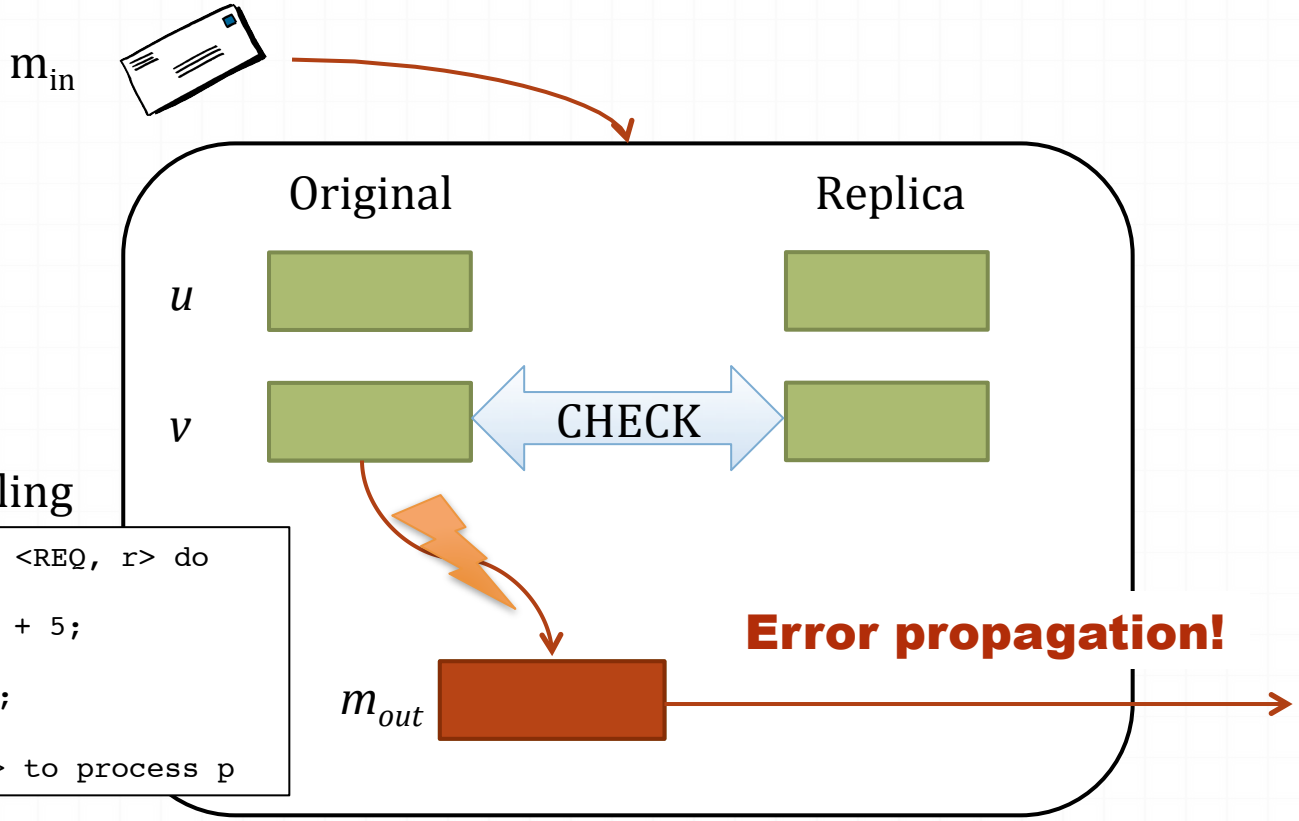
Simple hardening



Event handling

```
Upon receive message <REQ, r> do
      if v > 5 then
            u = r + v + 5;
      else
            u = r + v;
      v = u;
      send <WRITE, v> to process p
```

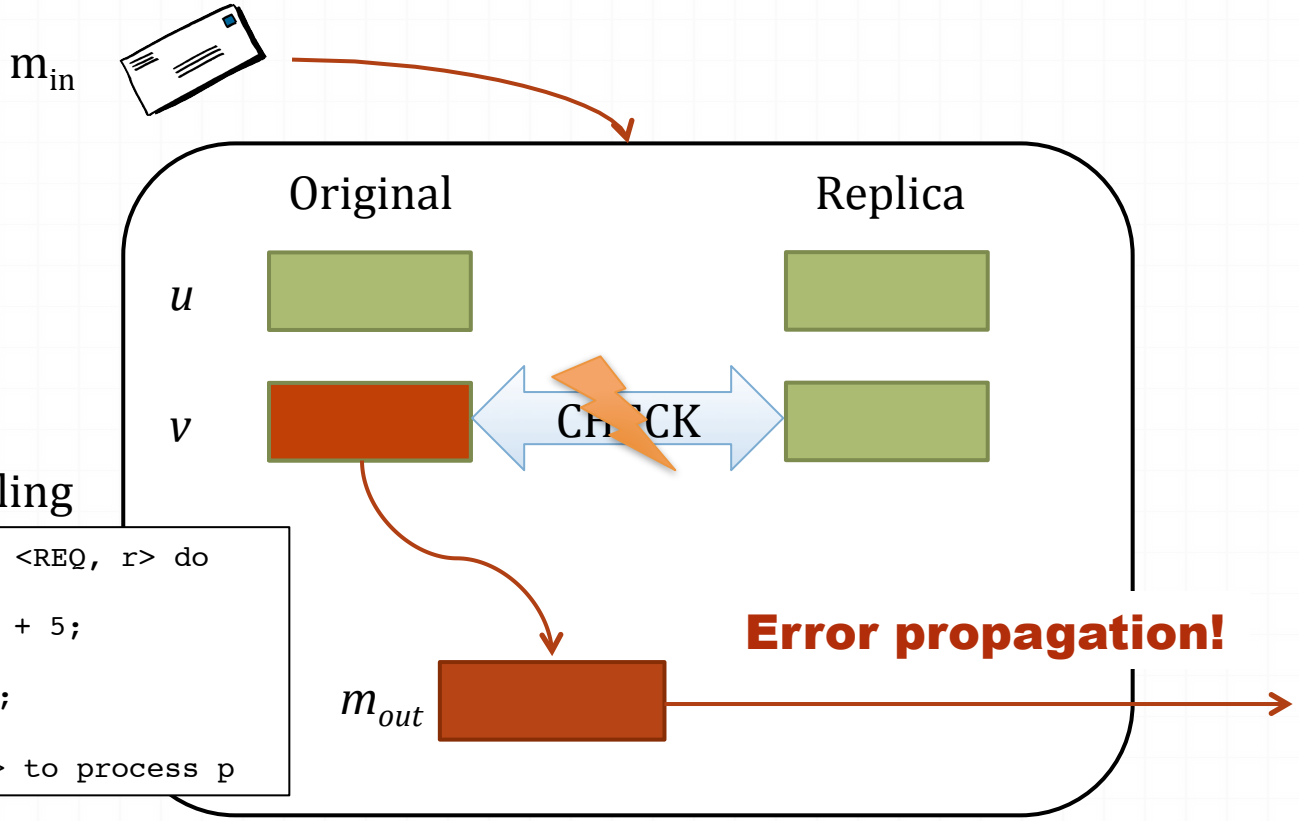
Simple hardening



Event handling

```
Upon receive message <REQ, r> do
      if v > 5 then
            u = r + v + 5;
      else
            u = r + v;
      v = u;
      send <WRITE, v> to process p
```

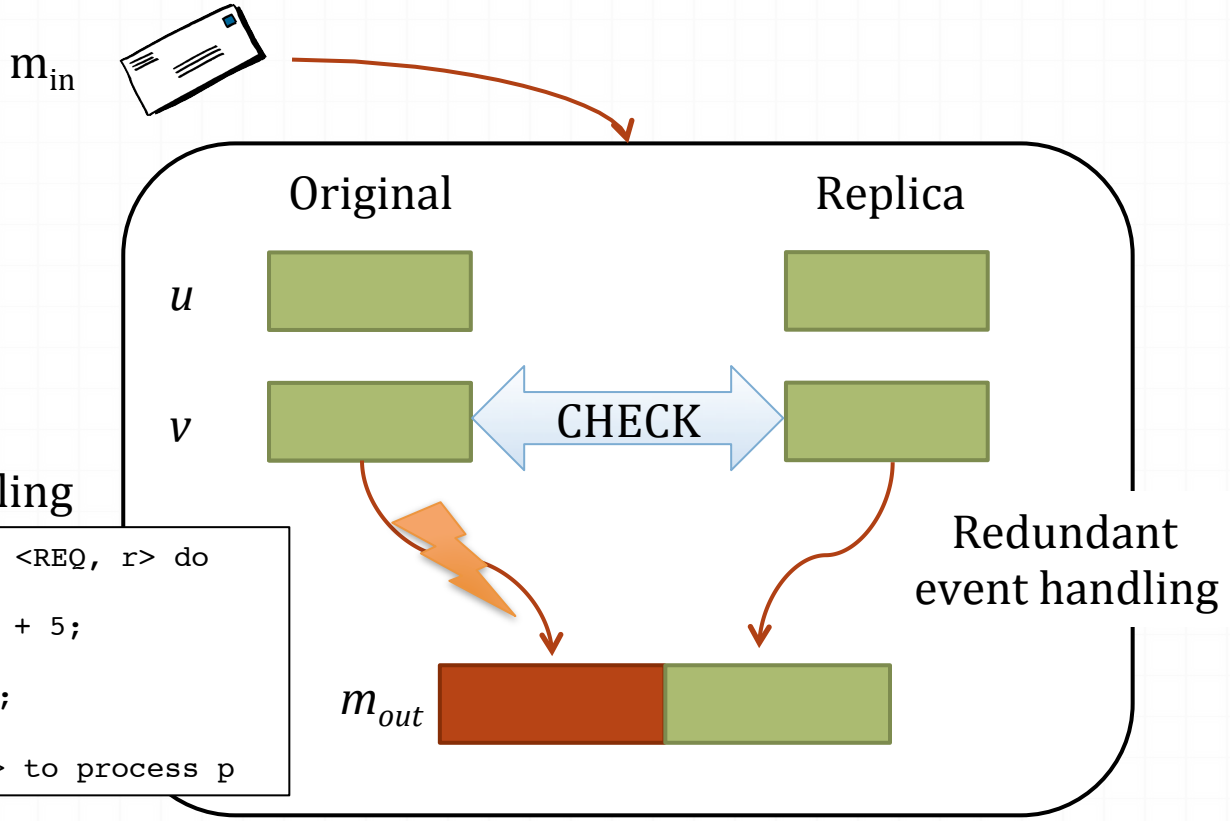
Simple hardening



Event handling

```
Upon receive message <REQ, r> do
      if v > 5 then
            u = r + v + 5;
      else
            u = r + v;
      v = u;
      send <WRITE, v> to process p
```

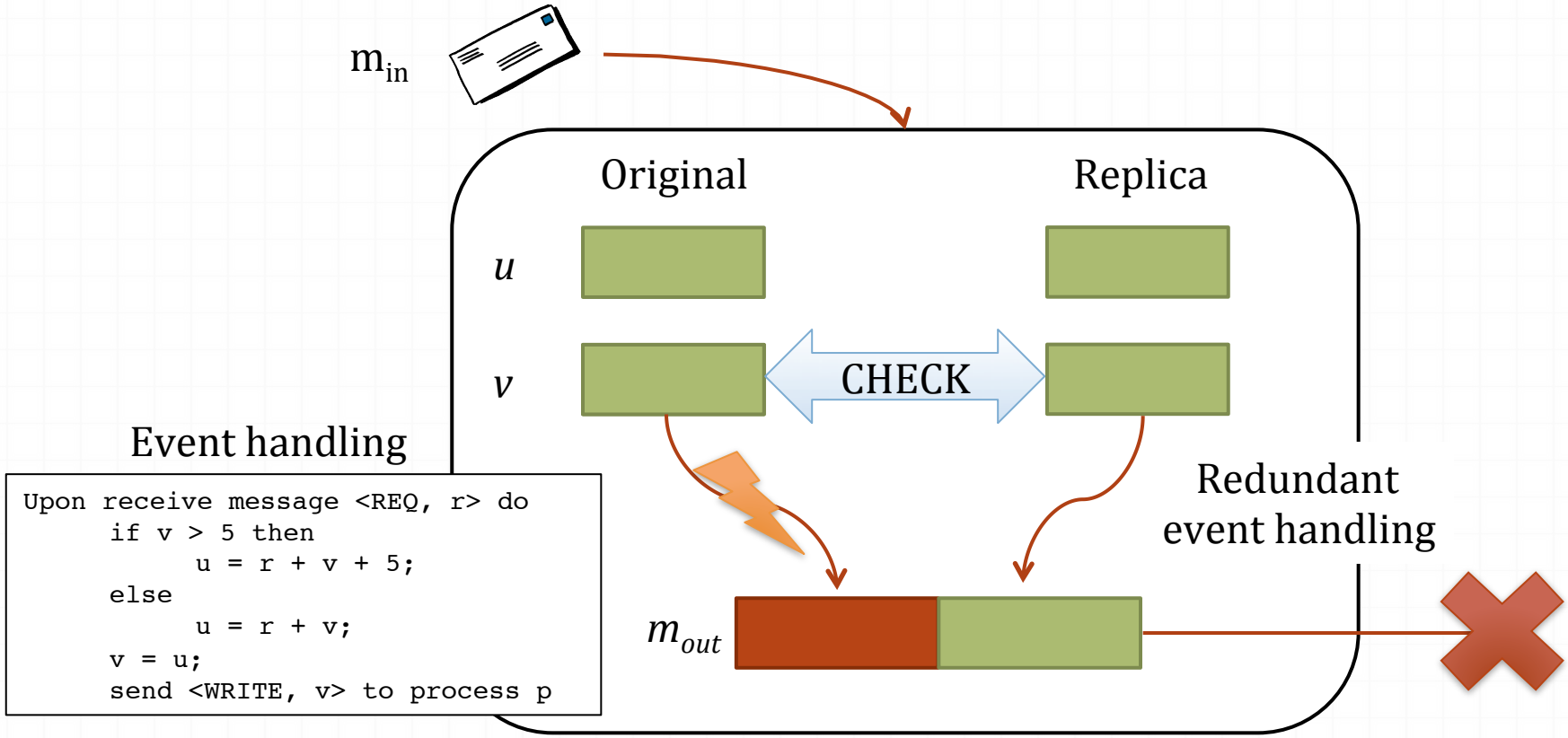
Protecting computation



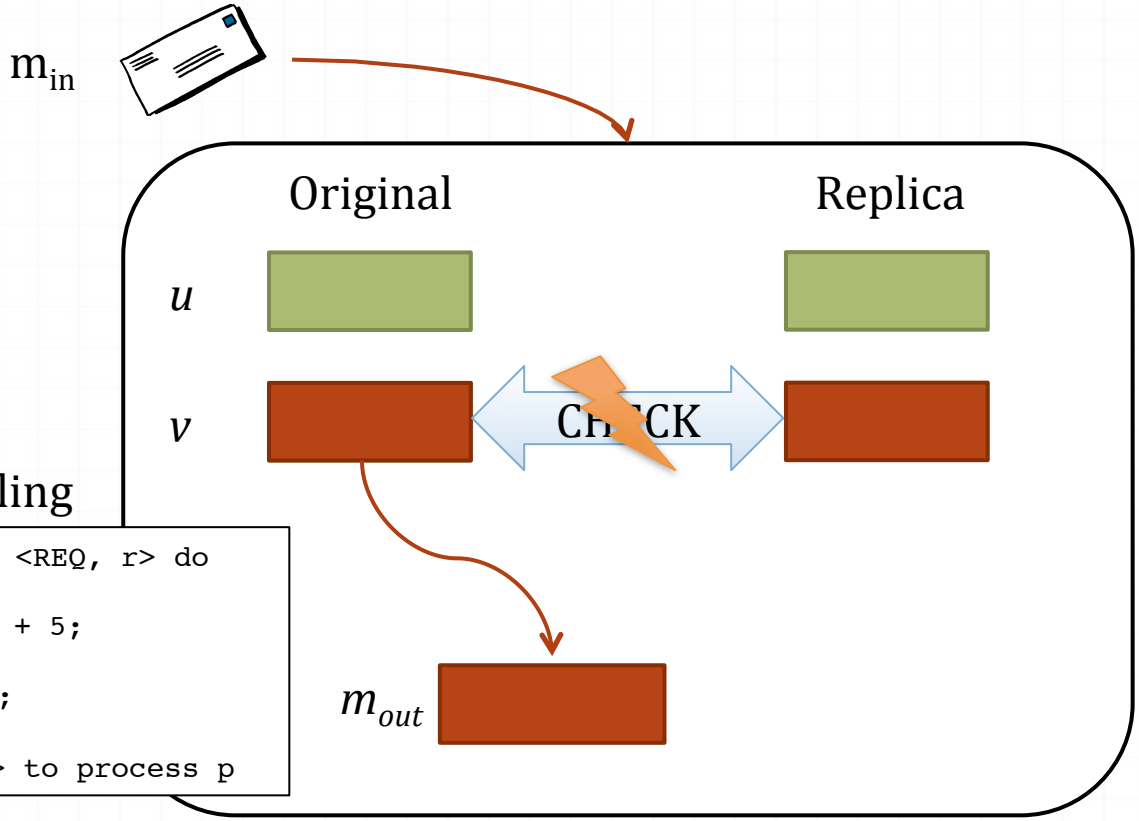
Event handling

```
Upon receive message <REQ, r> do
      if v > 5 then
            u = r + v + 5;
      else
            u = r + v;
      v = u;
      send <WRITE, v> to process p
```

Protecting computation



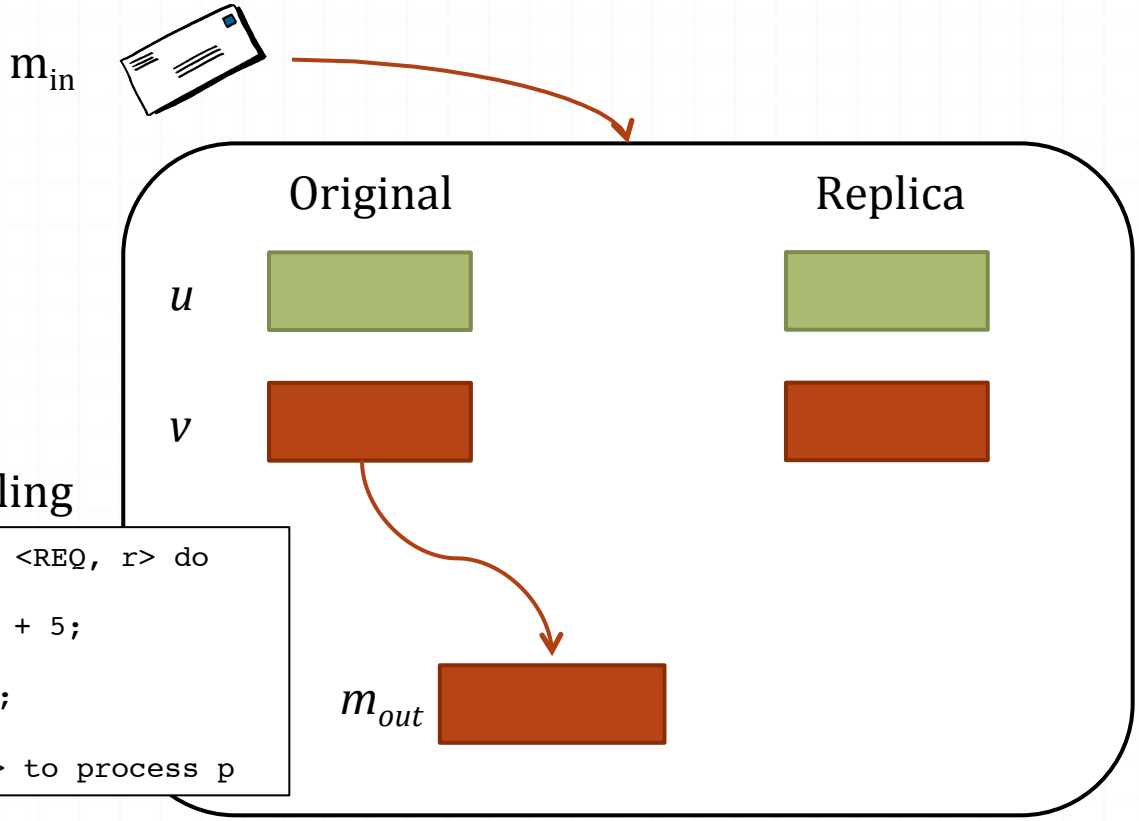
Checking the checkers



Event handling

```
Upon receive message <REQ, r> do
      if v > 5 then
            u = r + v + 5;
      else
            u = r + v;
      v = u;
      send <WRITE, v> to process p
```

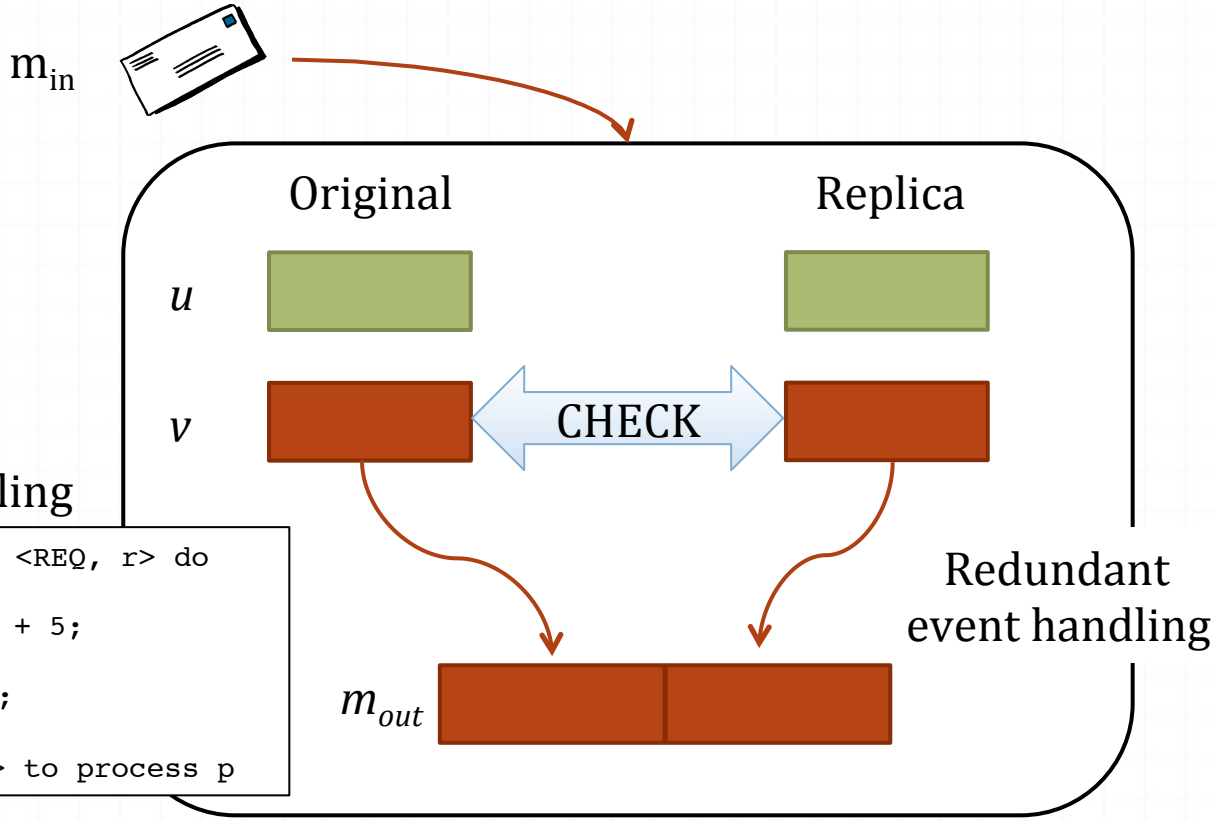
Checking the checkers



Event handling

```
Upon receive message <REQ, r> do
      if v > 5 then
            u = r + v + 5;
      else
            u = r + v;
      v = u;
      send <WRITE, v> to process p
```

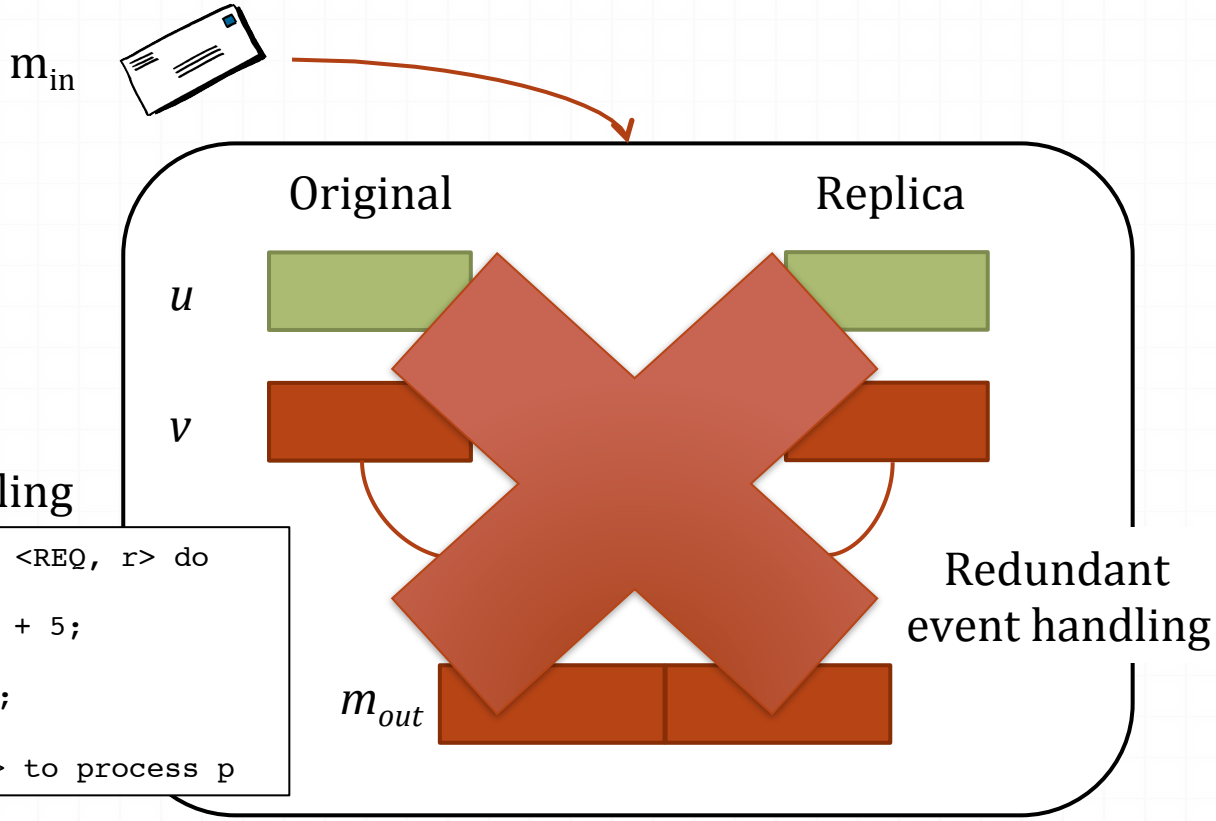
Checking the checkers



Event handling

```
Upon receive message <REQ, r> do
      if v > 5 then
            u = r + v + 5;
      else
            u = r + v;
      v = u;
      send <WRITE, v> to process p
```

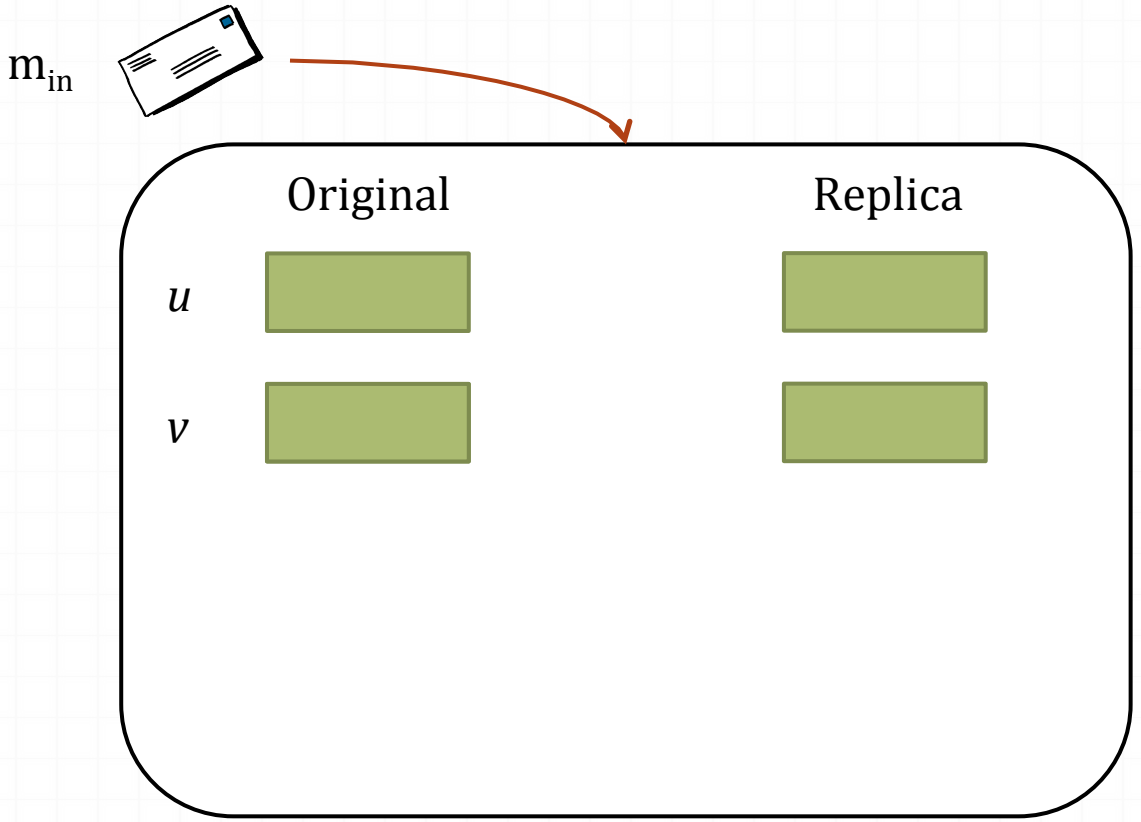
Checking the checkers



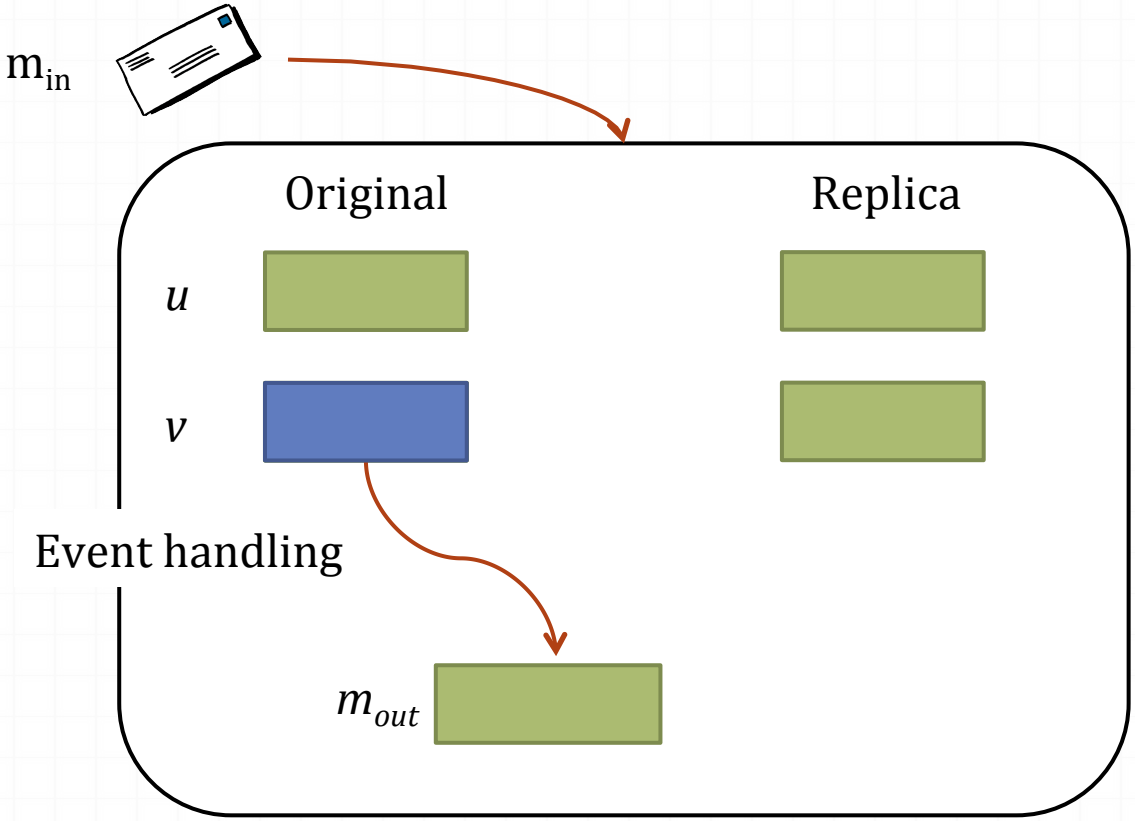
Event handling

```
Upon receive message <REQ, r> do
      if v > 5 then
            u = r + v + 5;
      else
            u = r + v;
      v = u;
      send <WRITE, v> to process p
```

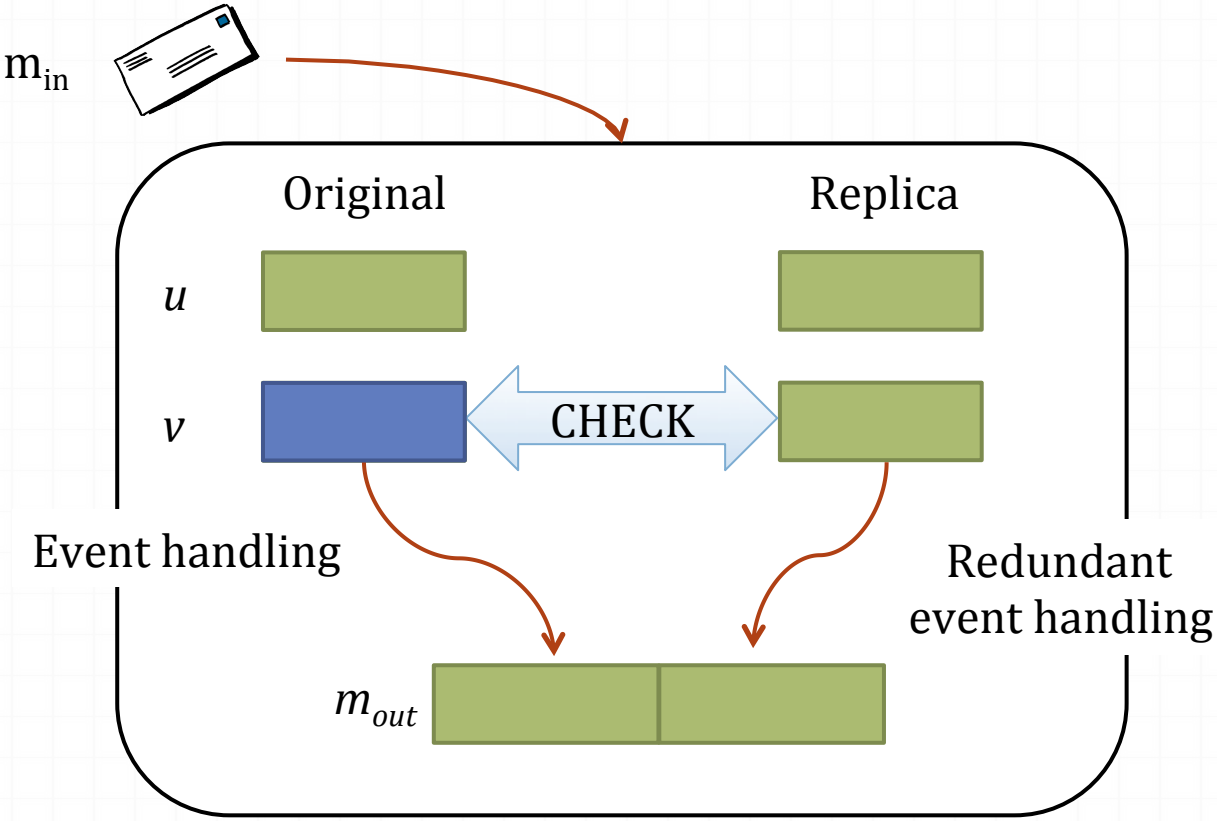
Issue with redundant checks



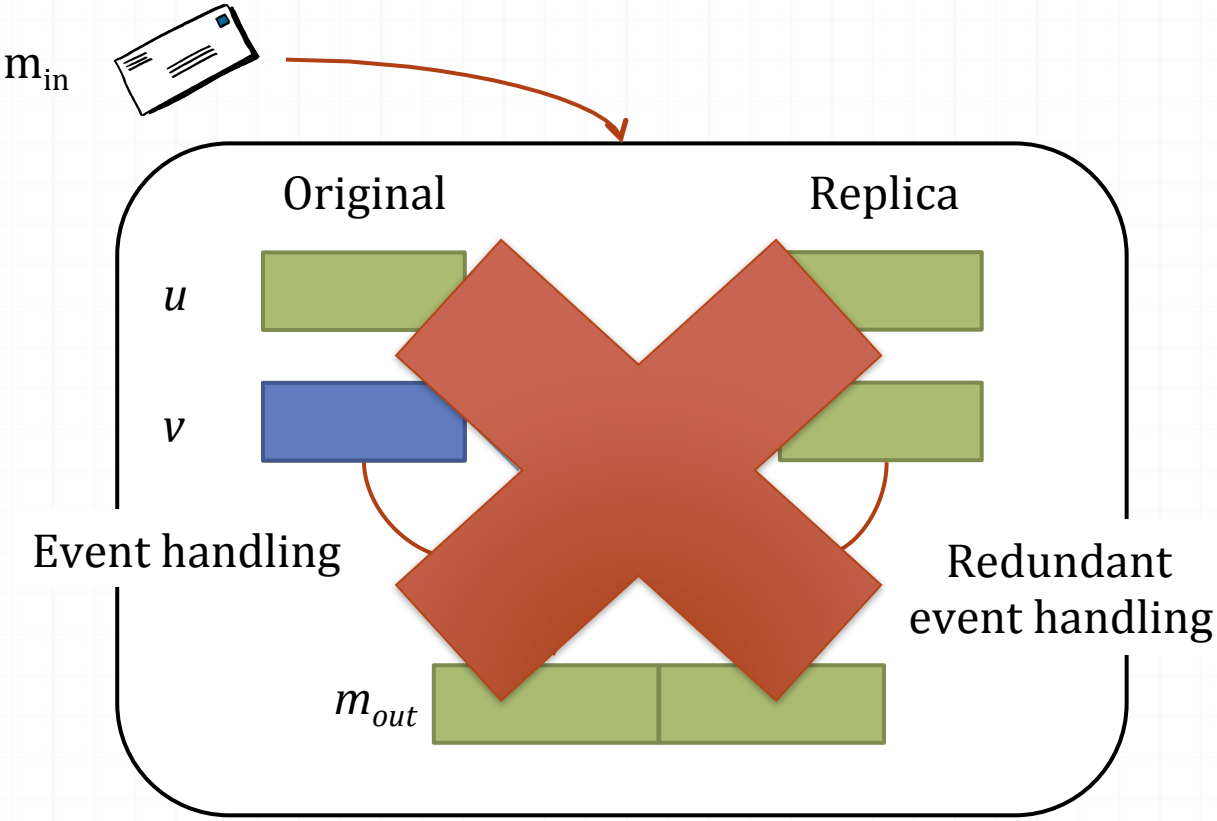
Issue with redundant checks



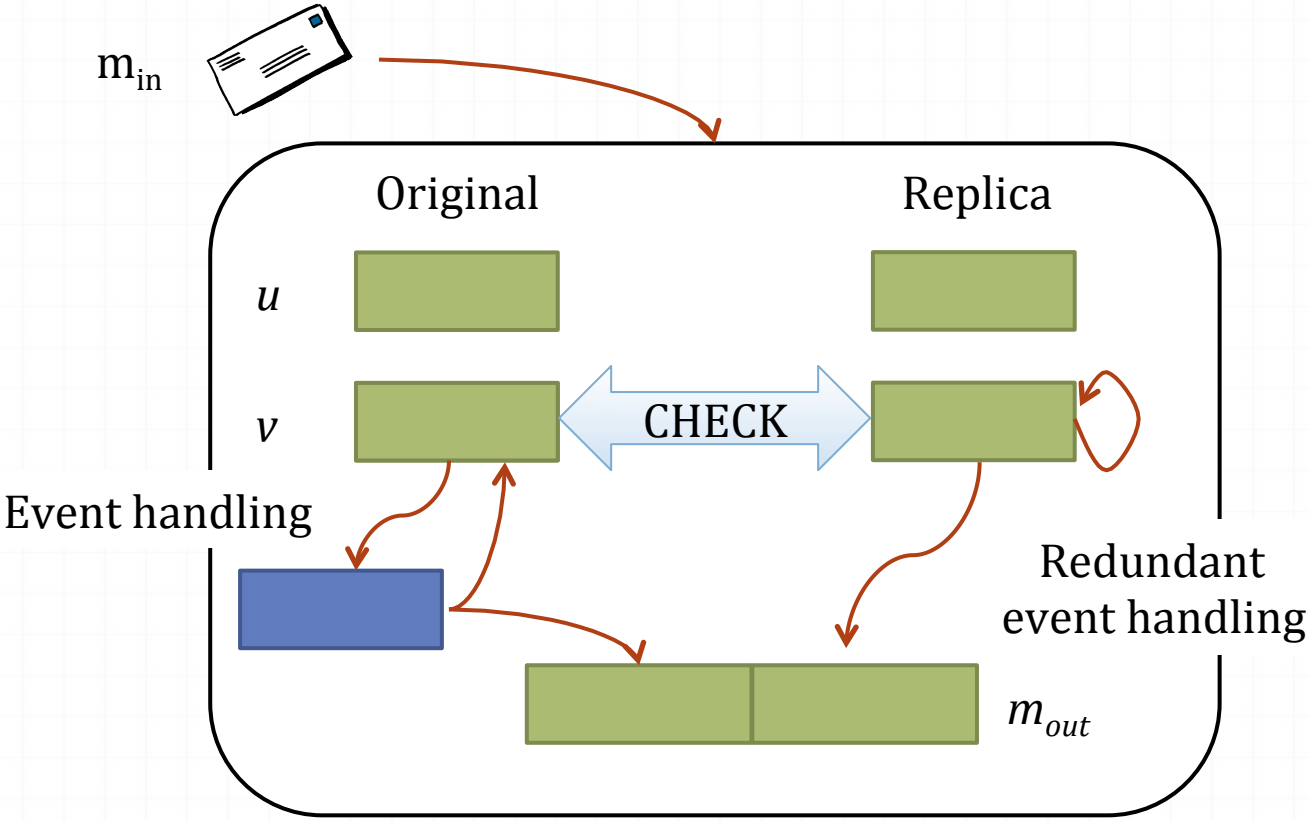
Issue with redundant checks



Issue with redundant checks



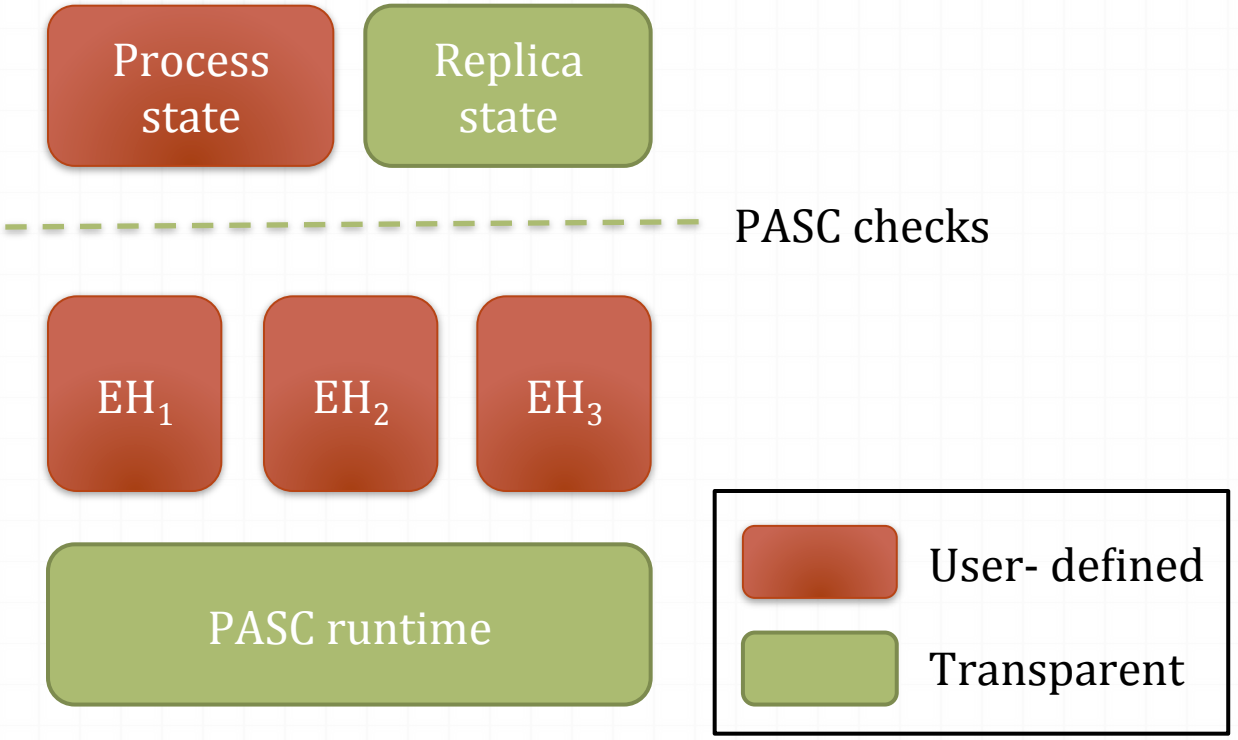
Incremental buffer



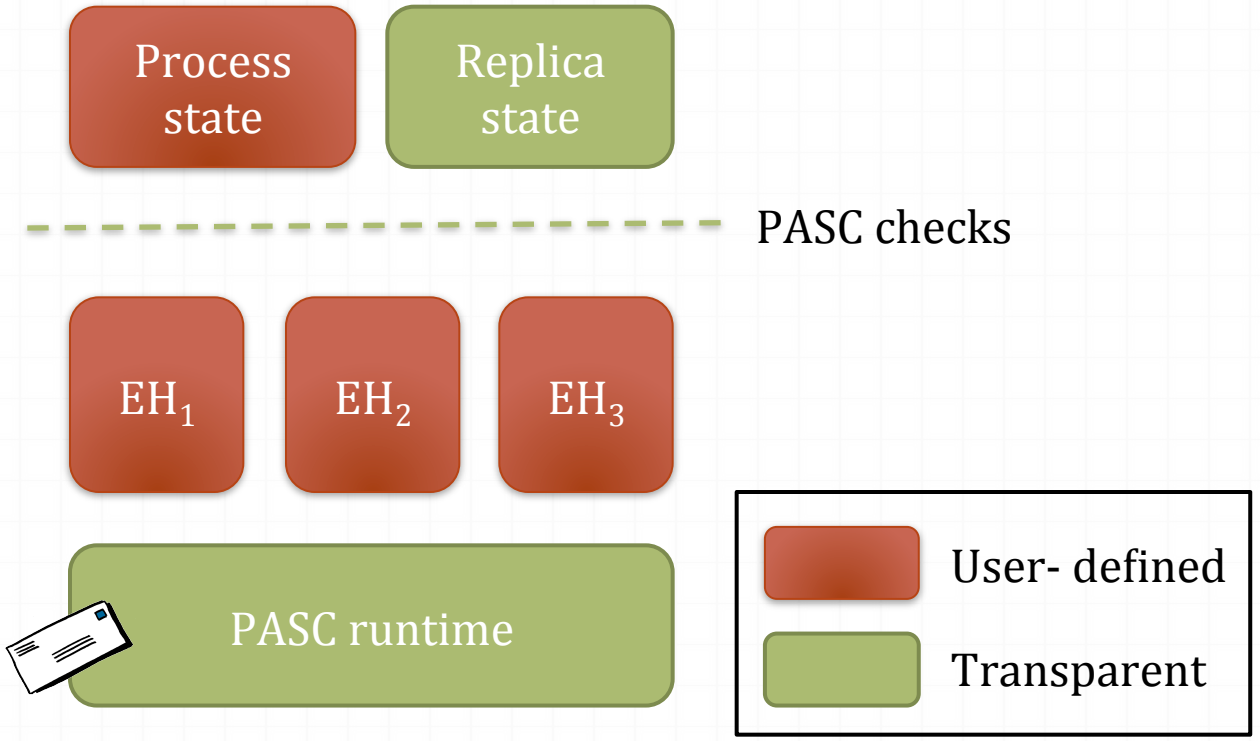
Control flow errors

- A control flow error may subvert the execution
  - An event handler could be executed twice
  - Event handling may be skipped or incomplete
- Requires **control flow checks**
  - Use flags to control the control flow
  - Very lightweight

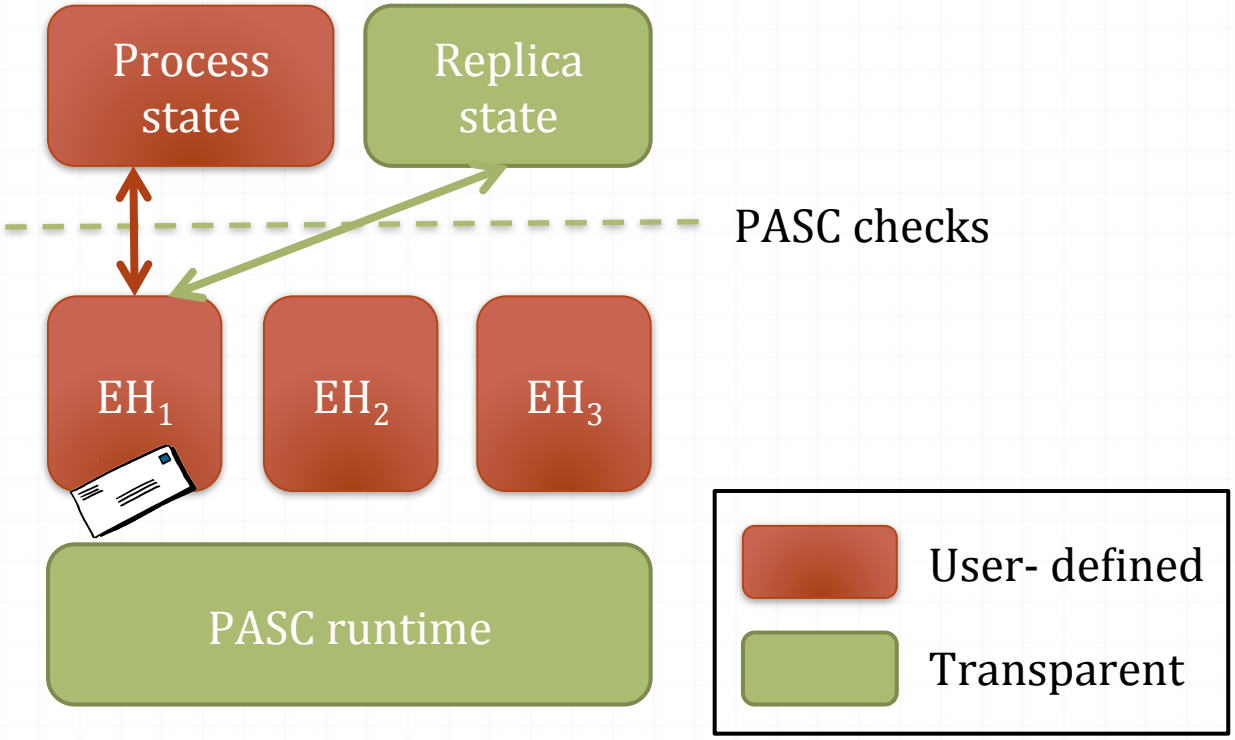
PASC library



PASC library



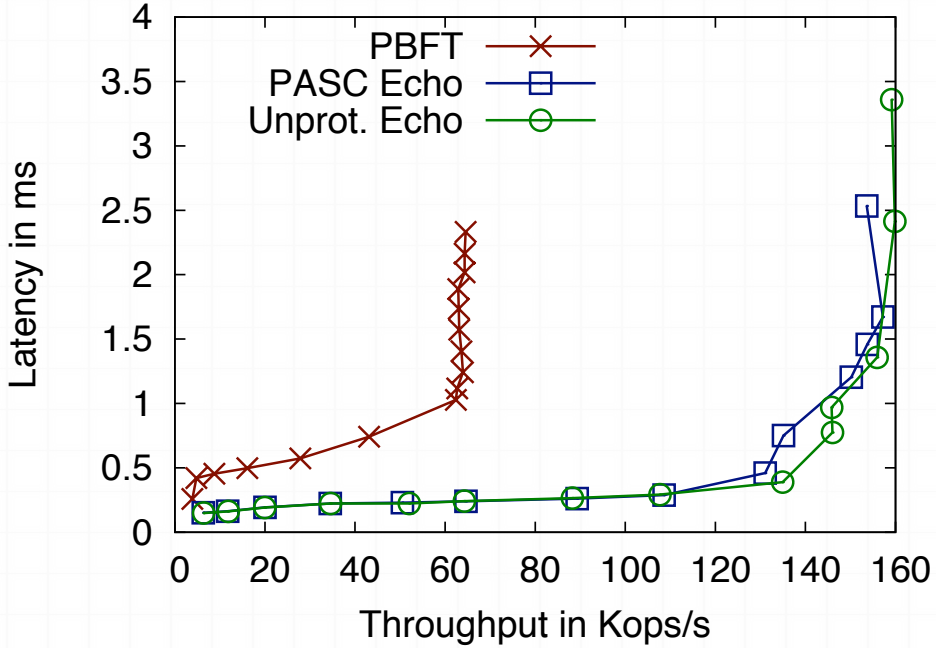
PASC library



github.com/yahoo/pasc

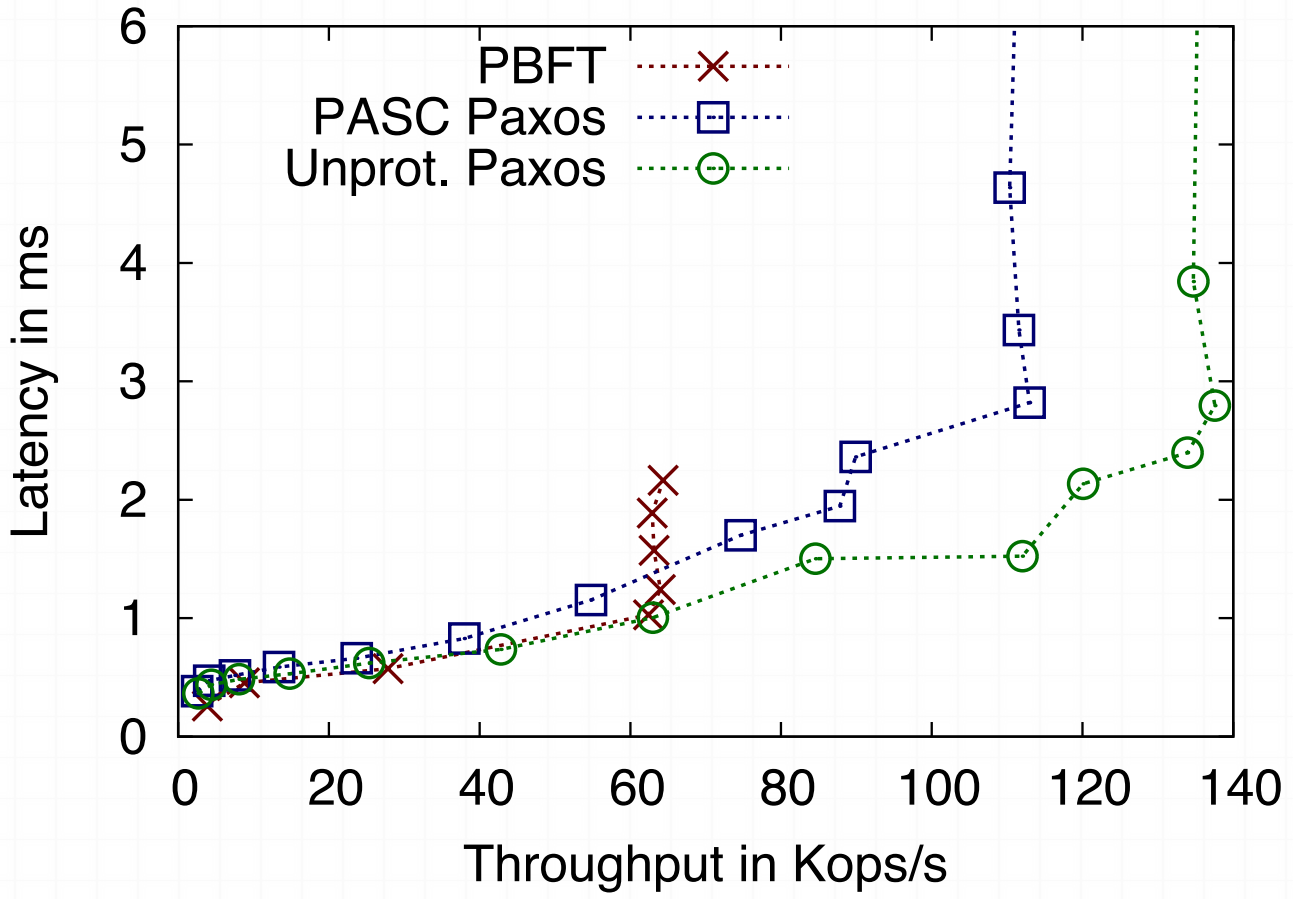
Evaluation

Hardening an echo server

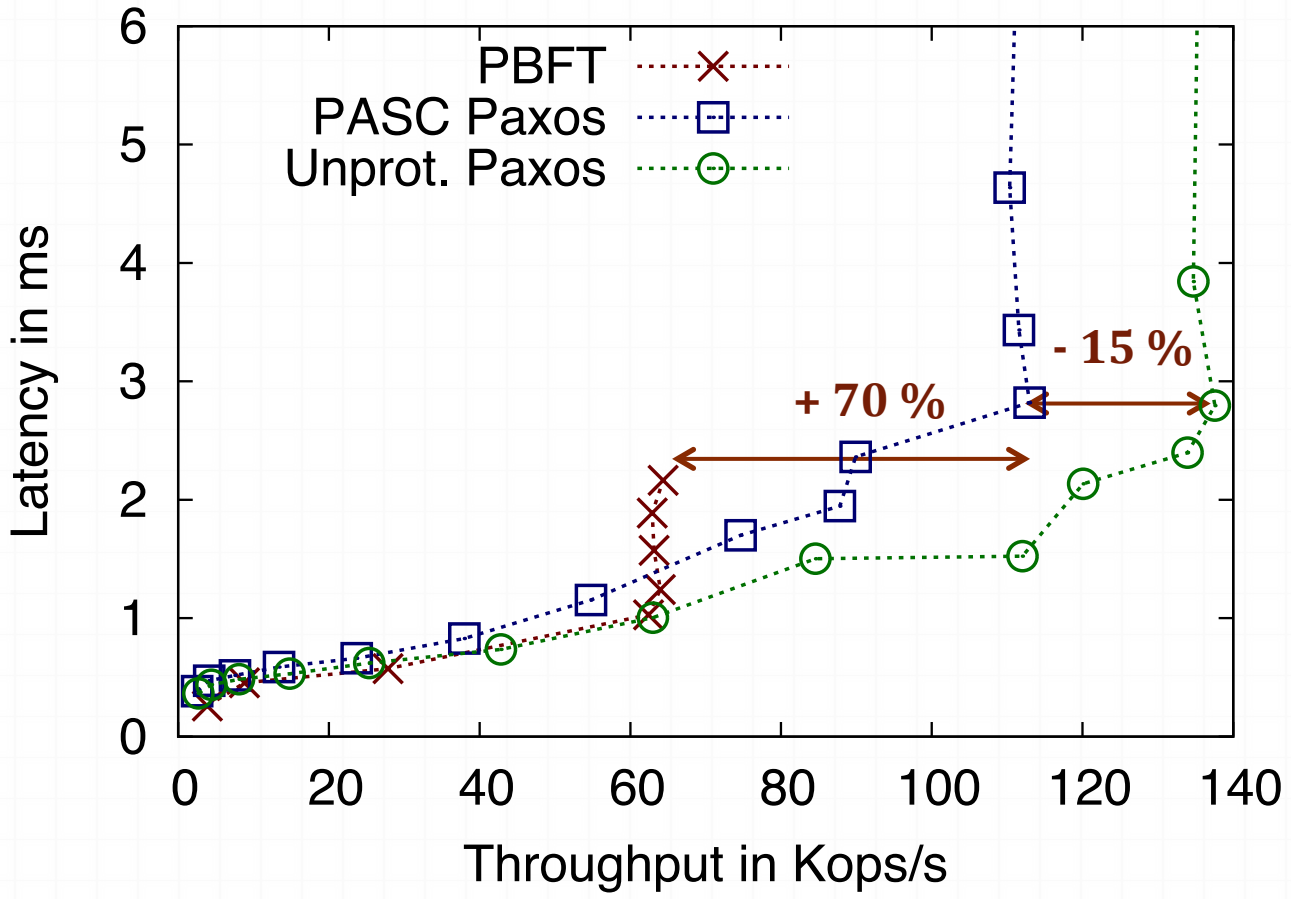


- Little computation, network bound, no overhead
- PBFT is a reference (Nysiad not available)

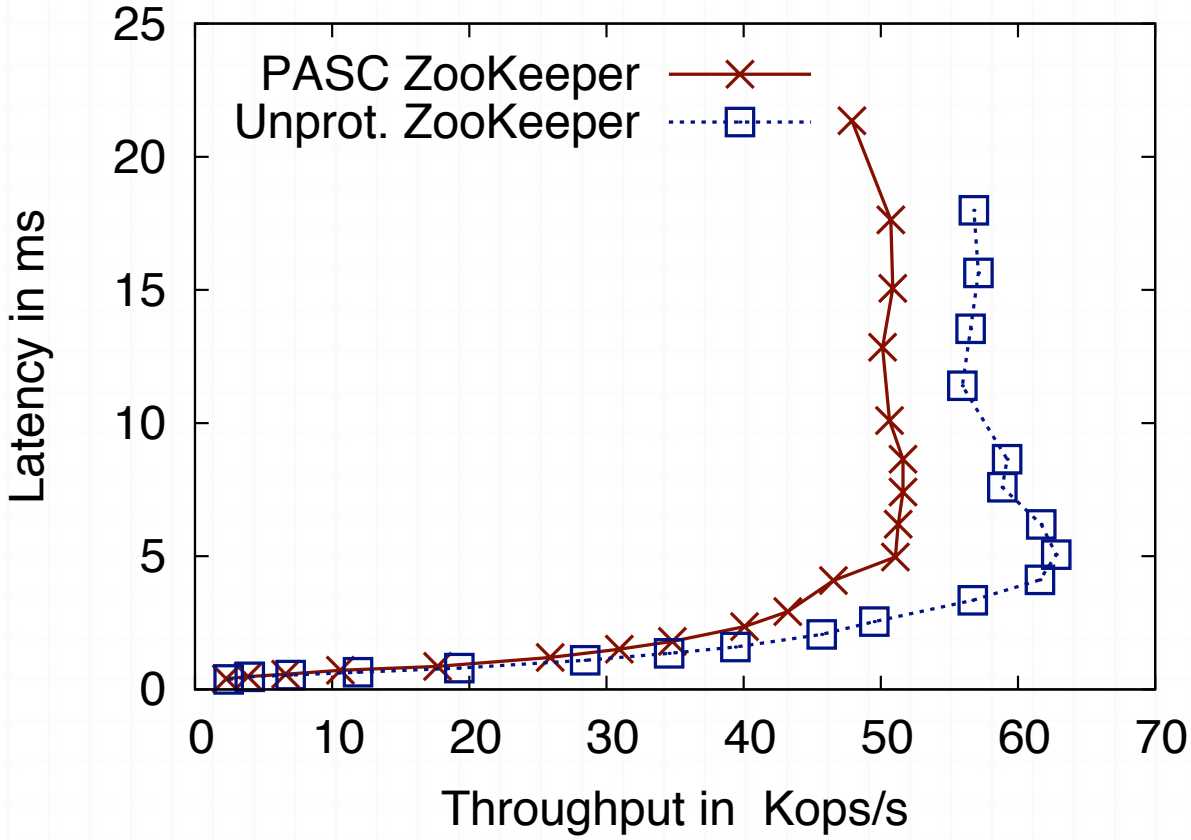
Hardening State Machine Replication



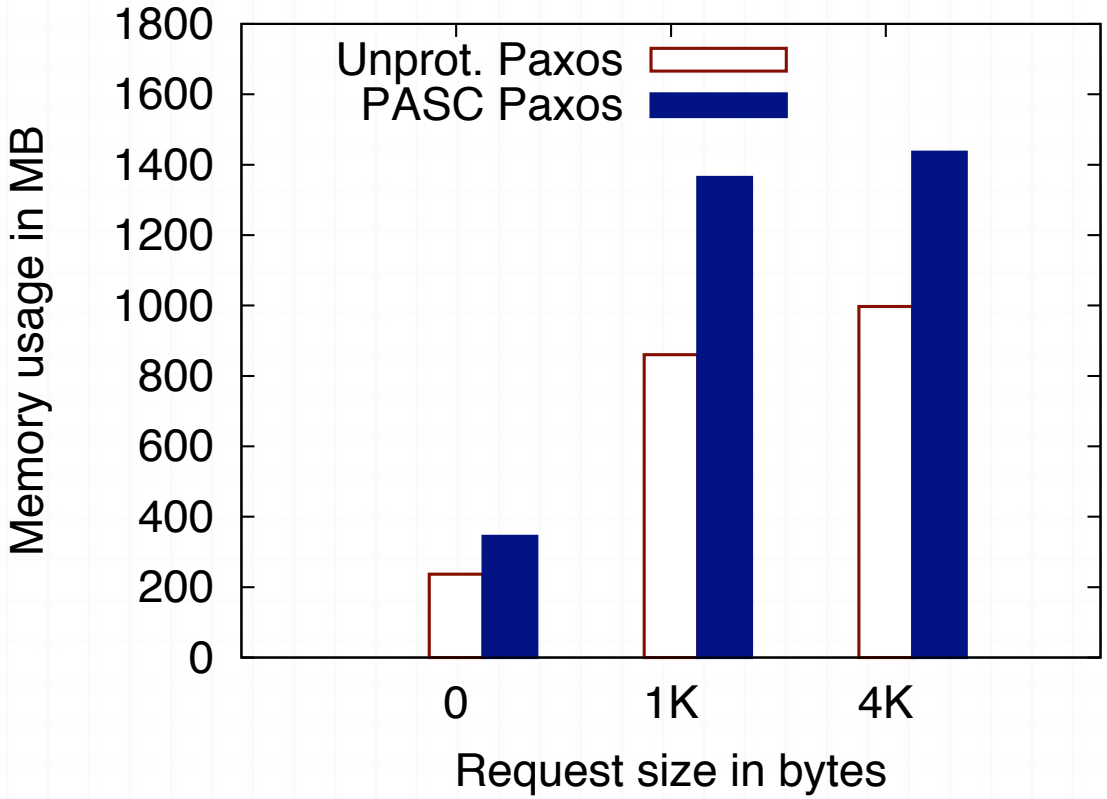
Hardening State Machine Replication



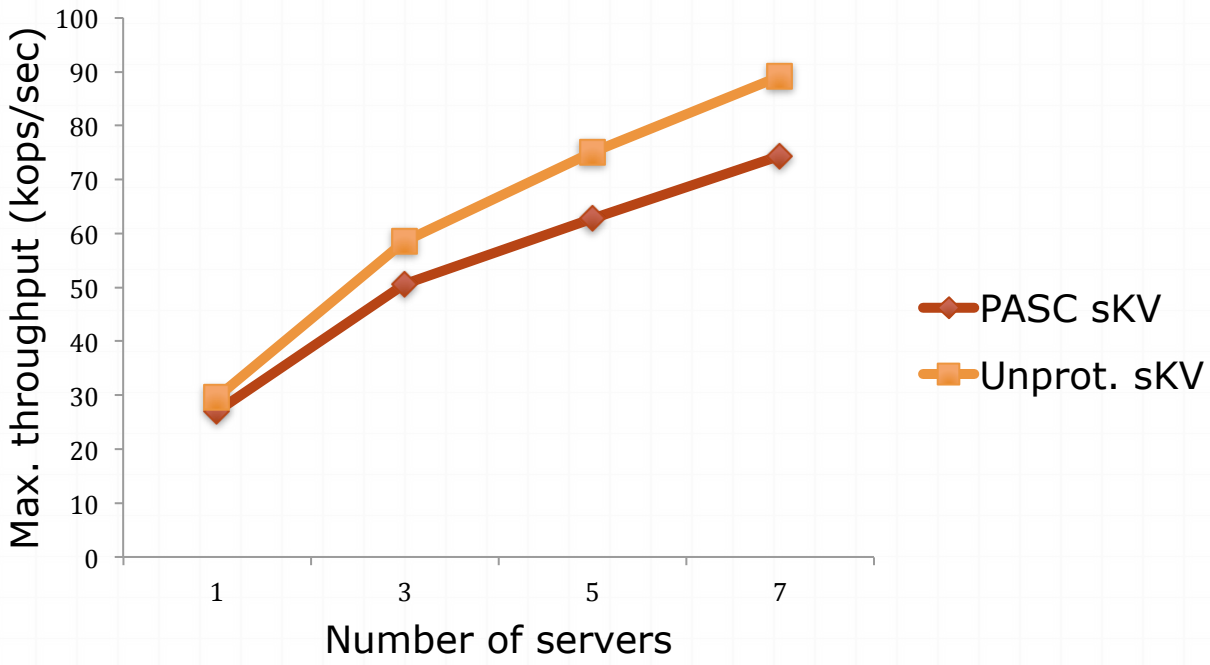
Zookeeper


PASC ZooKeeper
Unprot. ZooKeeper
Latency in ms
Throughput in Kops/s
0
5
10
15
20
25
0
10
20
30
40
50
60
70

Memory overhead



Scalability



- SimpleKV: eventually consistent store, no replication
  - Scales similarly with hardening
  - No server "wasted" for replication

PASC fault coverage

- Injected random bit flips in Paxos
  - Code corruptions: bytecode and binary code
  - State corruptions: pointers and primitive values

	Code corruptions		**State corruptions**	
	Unprot	**PASC**	**Unprot**	**PASC**
Undet.	3	**0**	93	**0**
Det.	-	1	-	330
Crash	1640	1663	2301	2066
Total	2856	2856	5237	5237

Conclusions

- Hardware data corruptions occur in data centers
- Model as **Arbitrary State Corruptions**
- ASC-hardening algorithm for **error isolation**
  - Local: does not require replication
- PASC: ASC-hardening library
  - Efficient: PASC-Paxos has up to 70% more throughput than PBFT
  - High fault coverage

Thank you

serafini@yahoo-inc.com

github.com/yahoo/pasc